

© А.В. Фролов, Г.В. Фролов, 1997

Microsoft Visual J++

Создание приложений и апплетов на языке Java
Часть 1

АННОТАЦИЯ

Книга представляет собой первую часть практического пособия по созданию автономных приложений и апплетов, работающих под управлением навигаторов WWW, на языке программирования Java.

Описаны основные отличия языка программирования Java от C++, среда выполнения приложений Java, приемы работы в интегрированной среде Microsoft Visual J++, основные библиотеки классов Java и методики их использования с иллюстрацией на примерах конкретных приложений. Читатель научится создавать сложные интерактивные апплеты для страниц серверов WWW.

Эта книга будет полезна всем, кто начинает самостоятельно осваивать новый язык программирования Java, и может быть использована в качестве учебного пособия для студентов учебных заведений.

ВВЕДЕНИЕ

Казалось бы, на сегодняшний день изобрели уже все языки программирования, какие только можно придумать. Но нет - появился еще один, с названием Java. Этот язык сумел завоевать весьма заметную популярность за последние несколько лет, так как он ориентирован на самую популярную компьютерную среду - сеть Internet и серверы WWW.

Язык Java произошел от языка программирования Oak (а не от C++, как думают многие). Oak был приспособлен для работы в Internet и затем переименован в Java. Изучая Java, вы будете приятно удивлены тем, что его синтаксис близок к синтаксису языка C++. Унаследовав самое лучшее от языка программирования C++, язык Java при этом избавился от некоторых недостатков C++, в результате чего на нем стало проще программировать. В этом языке нет, например, указателей, которые сложны в использовании и потенциально могут послужить причиной доступа программы к не принадлежащей ей области памяти. Нет множественного наследования и шаблонов, хотя функциональные возможности языка Java от этого не пострадали. Если вы умеете программировать на C++, для вас не составит особого труда изучить язык Java.

Огромное преимущество Java заключается в том, что на этом языке можно создавать приложения, способные работать на различных платформах. К сети Internet подключены компьютеры самых разных типов - совместимые с IBM PC, компьютеры фирмы Apple, рабочие станции Sun и так далее. Даже в рамках компьютеров, созданных на базе процессоров Intel, существует несколько платформ, например, Microsoft Windows версии 3.1, Microsoft Windows 95, Microsoft Windows NT, IBM OS/2, Solaris, различные разновидности операционной системы UNIX с графической оболочкой X-Windows. Между тем, создавая сервер WWW в сети Internet, вы бы наверняка хотели, чтобы им могло пользоваться как можно большее число людей. В этом случае вас выручат мультиплатформные приложения Java, не зависящие от конкретного типа процессора и операционной системы.

Программы, составленные на языке программирования Java, можно разделить по своему назначению на две большие группы.

К первой группе относятся приложения Java, предназначенные для автономной работы под управлением специальной интерпретирующей машины Java. Реализации этой машины созданы для всех основных компьютерных платформ.

Вторая группа - это так называемые апплеты (applets). Апплеты представляют собой разновидность приложений Java, которые интерпретируются виртуальной машиной Java, встроенной в браузеры WWW, такие как Microsoft Internet Explorer или Netscape Navigator.

Приложения, относящиеся к первой группе (в нашей книге мы будем называть их просто приложениями Java), это обычные автономные программы. Так как они не содержат машинного кода и работают под управлением специального интерпретатора, их производительность заметно ниже, чем у обычных программ, составленных, например, на языке программирования C++. Однако не следует забывать, что программы Java без перетрансляции способны работать на любой платформе, что само по себе имеет большое значение в плане разработок для Internet.

Апплеты Java встраиваются в документы HTML, хранящиеся на сервере WWW. С помощью апплетов вы можете сделать страницы сервера WWW динамичными и интерактивными. Апплеты позволяют выполнять сложную локальную обработку данных, полученных от сервера WWW и введенных пользователем с клавиатуры. Из соображений безопасности апплеты (в отличие от обычных приложений Java) не имеют никакого доступа к файловой системе локального компьютера. Все данные для обработки они могут получить только от сервера WWW. Более сложную обработку данных можно выполнять, организовав взаимодействие между апплетами и расширениями сервера WWW - приложениями CGI и ISAPI.

Для повышения производительности приложений Java в браузере Microsoft Internet Explorer использована технология с названием Just-in-Time Compilation, или JIT. При первой загрузке апплета его код транслируется в обычную исполнимую программу, которая сохраняется на диске и запускается. В результате общая скорость выполнения апплета Java увеличивается в несколько раз.

Язык Java является объектно-ориентированным и поставляется с достаточно объемной библиотекой классов. Так же как и библиотеки классов систем разработки приложений на языке C++, такие как Microsoft Foundation Classes (MFC), библиотеки классов Java значительно упрощают разработку приложений, представляя в распоряжение программиста мощные средства решения распространенных задач. Поэтому программист может больше внимания уделить решению прикладных задач, а не таких, как, например, организация динамических массивов, взаимодействие с операционной системой или реализация элементов пользовательского интерфейса.

Первоначально средства разработки приложений и апплетов Java были созданы фирмой Sun и до сих пор эти средства пользуются популярностью. В сети Internet по адресу <http://www.sun.com> есть сервер фирмы Sun, с которого можно бесплатно получить набор Java Development Kit (JDK). В JDK входят пакетные программы для компиляции исходных текстов приложений Java, виртуальная машина, программа автоматизированного создания документации по классам, справочник по классам Java и другие необходимые средства.

Для тех, кто привык пользоваться средствами разработки корпорации Microsoft, мы можем порекомендовать SDK-Java - пакетное средство разработки приложений и апплетов Java, расположенное на сервере <http://microsoft.com>. В составе этого средства есть также комплект документации по классам Java в виде набора документов HTML. Вы можете переписать себе SDK-Java бесплатно с указанного сервера Microsoft.

Если вы привыкли к интегрированным средствам разработки приложений, таким как Microsoft Visual C++, инструментарий JDK и SDK-Java могут показаться вам неудобными. В этом случае вы можете

воспользоваться такими системами разработки приложений Java, как, например, Symantec Cafe или Microsoft Visual J++.

Мы рассмотрим недорогое, но удобное и мощное интегрированное средство разработки приложений Microsoft Visual J++. Оценочную версию этого средства можно бесплатно получить с сервера WWW корпорации Microsoft. Заметим, что эта оценочная версия не содержит внутри себя “бомбы”, уничтожающей программу через заданное время, и вы сможете оценивать ее достаточно долго. Стоимость коммерческой версии Microsoft Visual J++ невелика, особенно с учетом того, что в коробке вы найдете прекрасную книгу Стефана Дэвиса “Learn Java Now” (на английском языке) издательства Microsoft Press. Если вы работаете с Microsoft Visual C++, то после установки Microsoft Visual J++ вы получите единую среду для разработки на языках программирования C++ и Java, что очень удобно.

На прилавках книжных магазинов вы можете найти несколько переводных книг, посвященных программированию на языке Java. Практически все они ориентированы на инструментарий JDK, созданный фирмой Sun, и содержат более или менее подробное описание классов Java. В нашей книге мы научим вас работать с Microsoft Visual J++ и приведем все сведения, необходимые для разработки как автономных приложений, так и апплетов Java.

При изложении материала мы будем предполагать, что вы знакомы с языком программирования C++. Наша книга не является учебником по языку Java. Мы будем рассказывать о том, как использовать язык Java и систему разработки Microsoft Visual J++ для создания автономных приложений и апплетов Java, как организовать взаимодействие апплетов с сервером WWW (во второй части этой книги). При этом мы будем считать, что с основами языка Java вы знакомы.

Что вам потребуется для работы с книгой?

Очень хорошо, если вы подключены к сети Internet или имеете возможность хотя бы эпизодической работы в этой сети. В этом случае вам будет доступна бесплатная оценочная версия Microsoft Visual J++, другие бесплатные средства разработки приложений и апплетов Java, грандиозные запасы документации и примеров программ.

Для проверки работы апплетов вам следует установить навигатор Internet, способный запускать апплеты Java. Это Microsoft Internet Explorer версии 3.01 и Netscape Navigator версии 3.0. Заметим, что для запуска апплетов вам не нужно обязательно подключаться к Internet - вы можете встраивать апплеты в документы HTML, расположенные на локальном диске вашего компьютера и просматривать эти документы навигатором просто как локальные файлы.

Автономные приложения Java работают под управлением специального интерпретатора (виртуальной машины Java), поэтому для их отладки вам также не потребуется сеть Internet.

Если вы собираетесь проверять работу приложений и апплетов Java, взаимодействующих с сервером WWW, вы можете воспользоваться собственным сервером в Internet или в корпоративной сети Intranet (если они у вас есть). Примеры таких апплетов мы приведем в следующей книге “Библиотеки системного программиста”, посвященной Java. Можно также установить сервер WWW, входящий в комплект операционной системы Microsoft Windows NT Workstation версии 4.0, или Personal Web Service для операционной системы Microsoft Windows 95. Последний доступен для бесплатной загрузки с сервера <http://microsoft.com>.

Для того чтобы успешно работать с апплетами Java, вы должны иметь некоторое представление о серверах WWW и языке гипертекстовой разметки документов HTML. Поэтому перед чтением этой книги мы рекомендуем вам ознакомиться с 29 томом “Библиотеки системного программиста”, который называется “Сервер Web своими руками”. В этой книге мы рассказали о том, как сделать собственный сервер WWW в сети Internet и Intranet, а также привели необходимую информацию о расширениях сервера WWW, реализованных как приложения CGI и ISAPI.

Базовые знания об Internet, а также описание конкретных способов подключения к ней в российских условиях вы найдете в 23 томе “Библиотеки системного программиста”, который называется “Глобальные сети компьютеров. Практическое введение в Internet, E-Mail, FTP, WWW и HTML, программирование для Windows Sockets”. Мы рекомендуем эту книгу для тех, кто только планирует подключиться к Internet, но еще не вполне осознает, как это можно сделать и для чего это нужно.

Что еще почитать?

Количество книг, посвященных Java, растет катастрофически, особенно за рубежом. Среди удачных еще раз назовем книгу Стефана Дэвиса с названием “Learn Java Now”, которая может служить учебником по языку Java для тех, кто никогда не программировал на C и C++.

Среди переводных книг, которые можно встретить в продаже, отметим книгу Джона Родли “Создание JAVA-апплетов”. Эта книга рассчитана на серьезных программистов, хорошо знающих язык программирования Java. Однако для тех, кто только начинает изучать язык Java, она может оказаться слишком сложной.

Другая книга, заслуживающая внимание, это книга Криса Джамса с названием “Java”. После небольшого введения, рассчитанного на начинающих, в этой книге приводится описание более чем дюжины достаточно интересных апплетов с исходными текстами и комментариями.

В качестве справочника по языку Java и библиотекам классов вы можете использовать книгу И. Баженовой “Язык программирования Java”, которая вышла в издательстве АО “Диалог-МИФИ”.

Ну и, конечно, вам следует ознакомиться с различными руководствами по языку Java, хранящимися в сети Internet. В качестве отправной точки для поиска вы можете выбрать сервер основного разработчика этого языка - фирмы Sun. Адрес ее сервера мы уже приводили: <http://www.sun.com>. Помимо документации и примеров программ на Java, здесь вы найдете ссылки на другие ресурсы, посвященные этому языку программирования. Попробуйте также воспользоваться поисковыми серверами, такими как Jahoo! и Alta Vista, указав в качестве ключевого слово “Java”.

БЛАГОДАРНОСТИ

В работе над книгой нам помогали сотрудники фирмы Interactive Products Inc. Максим Синев и Сергей Ноженко, у которых мы консультировались по различным вопросам.

Мы признательны генеральному директору АО «ДиалогНаука» Антимонову Сергею Григорьевичу и его заместителю Лященко Юрию Павловичу за возможность размещения информации о наших книгах на сервере Web по адресу <http://www.dials.ccas.ru/frolov>, а также за возможность доступа к сети Internet через сервер АО «ДиалогНаука».

Мы также благодарим корректора Кустова В. С. и сотрудников издательского отдела АО «Диалог-МИФИ» Голубева О. А., Голубева А. О., Дмитриеву Н. В., Виноградову Е. К., Кузьминову О. А.

КАК СВЯЗАТЬСЯ С АВТОРАМИ

Полную информацию о всех наших книгах серий “Библиотека системного программиста” и “Персональный компьютер. Шаг за шагом”, а также дискеты к книгам, статьи и другую информацию вы можете найти в сети Internet на серверах Web по следующим адресам:

<http://www.glasnet.ru/~frolov>
<http://www.dials.ccas.ru/frolov>

Вы можете передать нам свои замечания и предложения по содержанию этой и других наших книг через электронную почту по адресам:

frolov@glas.apc.org
frolov.alexandr@usa.net

Если электронная почта вам недоступна, присылайте ваши отзывы в АО “Диалог-МИФИ” по адресу:
115409, Москва, ул. Москворечье, 31, корп. 2,
тел. 324-43-77

Приносим свои извинения за то что не можем ответить на каждое письмо. Мы также не занимаемся продажей и рассылкой книг, дискет, рекламы, отдельных фрагментов наших книг и исходных текстов к книгам, консультациями через электронную почту. По вопросам приобретения книг и дискет обращайтесь непосредственно в издательство “Диалог-МИФИ”.

1 НОВЫЙ ЯЗЫК ПРОГРАММИРОВАНИЯ

Почему вам нужно изучать новый язык программирования Java?

Если ответить на этот вопрос кратко, то потому, что он специально ориентирован на самые передовые технологии, связанные с сетью Internet. Растущая популярность Internet и, в особенности, серверов WWW, создает для программистов новые возможности для реализации своих способностей. Затратив всего пару-тройку недель на чтение нашей книги, вы сможете проникнуться идеологией Java и, как мы надеемся, сумеете по достоинству оценить возможности этого языка программирования.

В этой главе мы расскажем об основных особенностях Java и библиотеках классов, которые поставляются в составе систем разработки приложений. После этого вы поймете, почему этот язык программирования выглядит весьма привлекательно именно для реализации сетевых проектов в Internet и в корпоративных сетях Intranet.

Мобильность Java

В свое время вы слышали, что язык программирования C является мобильным в том смысле, что имеется принципиальная возможность переноса программ, составленных на этом языке, на различные платформы.

Однако следует отметить, что создание действительно многоплатформных приложений - непростая задача. К сожалению, дело не ограничивается необходимостью перекомпиляции исходного текста программы для работы в другой среде. Много проблем возникает с несовместимостью программных интерфейсов различных операционных систем и графических оболочек, реализующих пользовательский интерфейс.

Вспомните хотя бы проблемы, связанные с переносом 16-разрядных приложений Windows в 32-разрядную среду Microsoft Windows 95 и Microsoft Windows NT. Даже если вы тщательно следовали всем рекомендациям Microsoft, разрабатывая приложения так, чтобы они могли работать в будущих версиях Windows, едва ли вам удастся просто перекомпилировать исходные тексты, не изменив в них ни строчки. Ситуация еще больше ухудшается, если вам нужно, например, перенести исходные тексты приложения Windows в среду операционной системы IBM OS/2 или в оболочку X-Windows операционной системы UNIX. А ведь есть еще компьютеры Apple, рабочие станции с процессором RISC и другие!

Как нетрудно заметить, даже если стандартизовать язык программирования для всех платформ, проблемы совместимости с программным интерфейсом операционной системы значительно усложняют перенос программ на различные платформы. И, конечно, вы не можете мечтать о том, чтобы загрузочный модуль одной и той же программы мог работать без изменений в среде различных операционных систем и на различных платформах. Если программа подготовлена для процессора Intel, она ни за что не согласится работать на процессоре Alpha или каком-либо другом.

В результате создавая приложение, способное работать на различных платформах, вы вынуждены фактически создавать несколько различных приложений и сопровождать их по отдельности.

На рис. 1.1 мы показали, как приложение, изначально разработанное для Microsoft Windows NT, переносится на платформу Apple Macintosh.

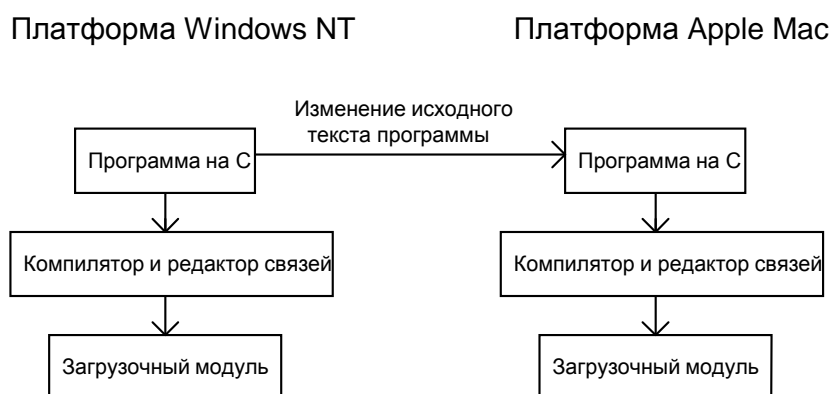


Рис. 1.1. Перенос приложения с платформы Windows NT на платформу Apple Macintosh

Вначале программист готовит исходные тексты приложения для платформы Microsoft Windows NT и отлаживает их там. Для получения загрузочного модуля исходные тексты компилируются и редактируются. Полученный в результате загрузочный модуль может работать на процессоре фирмы Intel в среде операционной системы Microsoft Windows NT.

Для того чтобы перенести приложение в среду операционной системы компьютера Apple Macintosh, программист вносит необходимые изменения в исходные тексты приложения. Эти изменения необходимы из-за различий в программном интерфейсе операционной системы Microsoft Windows NT и операционной системы, установленной в Apple Macintosh. Далее эти исходные тексты транслируются и редактируются, в результате чего получается загрузочный модуль, способный работать в среде Apple Macintosh, но не способный работать в среде Microsoft Windows NT.

Программа на языке Java компилируется в двоичный модуль, состоящий из команд виртуального процессора Java. На настоящий момент созданы только первые модели физического процессора, способного выполнять эти двоичные команды, однако интерпретаторы Java имеются на всех основных компьютерных платформах. Разумеется, на каждой платформе используется свой интерпретатор, или, точнее говоря, свой виртуальный процессор Java.

Если ваше приложение Java (или апплет) должно работать на нескольких платформах, нет необходимости компилировать его исходные тексты несколько раз. Вы можете откомпилировать и отладить приложение Java на одной, наиболее удобной для вас платформе. В результате вы получите двоичный модуль, пригодный для любой платформы, где есть виртуальный процессор Java.

Сказанное иллюстрируется на рис. 1.2.

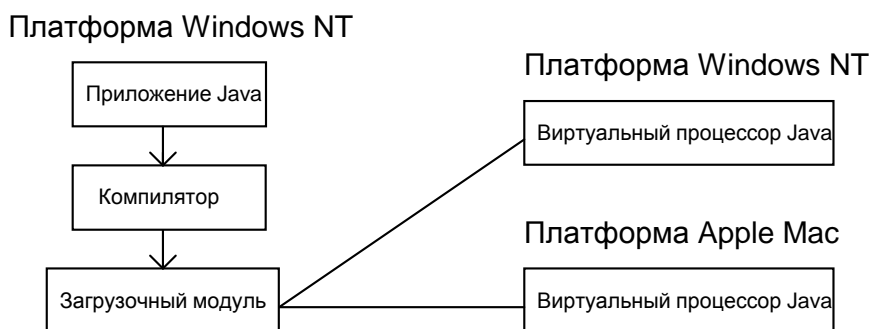


Рис. 1.2. Подготовка приложения Java для работы на разных платформах

Таким образом, приложение Java компилируется и отлаживается только один раз, что уже значительно лучше. Остается, правда, вопрос - как быть с программным интерфейсом операционной системы, который отличается для разных платформ?

Здесь разработчиками Java предлагается достаточно неплохое решение. Приложение Java не обращается напрямую к интерфейсу операционной системы. Вместо этого оно пользуется готовыми стандартными библиотеками классов, содержащими все необходимое для организации пользовательского интерфейса, обращения к файлам, для работы в сети и так далее.

Внутренняя реализация библиотек классов, разумеется, зависит от платформы. Однако все загрузочные модули, реализующие возможности этих библиотек, поставляются в готовом виде вместе с виртуальной машиной Java, поэтому программисту не нужно об этом заботиться. Для операционной системы Microsoft Windows, например, поставляются библиотеки динамической загрузки DLL, внутри которых запрятана вся функциональность стандартных классов Java.

Абстрагируясь от аппаратуры на уровне библиотек классов, программисты могут больше не заботиться о различиях в реализации программного интерфейса конкретных операционных систем. Это позволяет создавать по-настоящему мобильные приложения, не требующие при переносе на различные платформы перетрансляции и изменения исходного текста.

Еще одна проблема, возникающая при переносе программ, составленных на языке программирования C, заключается в том, что размер области памяти, занимаемой переменными стандартных типов, различный на разных платформах. Например, в среде операционной системы Microsoft Windows версии 3.1 переменная типа `int` в программе, составленной на C, занимает 16 бит. В среде Microsoft Windows NT этот размер составляет 32 бита.

Очевидно, что трудно составлять программу, не зная точно, сколько имеется бит в слове или в байте. При переносе программ на платформы с иной разрядностью могут возникать ошибки, которые трудно обнаружить.

В языке Java все базовые типы данных имеют фиксированную разрядность, которая не зависит от платформы. Поэтому программисты всегда знают размеры переменных в своей программе.

Базовые типы данных

В языке Java определено восемь базовых типов данных, перечисленных ниже:

Тип данных	Размер занимаемой области памяти	Значение по умолчанию
<code>boolean</code>	8	<code>false</code>
<code>byte</code>	8	<code>0</code>
<code>char</code>	16	<code>'\0'</code>
<code>short</code>	16	<code>0</code>
<code>int</code>	32	<code>0</code>
<code>long</code>	64	<code>0</code>
<code>float</code>	32	<code>0.0F</code>
<code>double</code>	64	<code>0.0D</code>

Для каждого базового типа данных отводится конкретный размер памяти, который, как мы говорили в предыдущем разделе, не зависит от платформы, на которой выполняется приложение Java. Фактический размер памяти, отведенные для хранения переменной, могут отличаться от приведенных выше,

например, для хранения переменной типа `short` может быть зарезервировано слово размером 32 бита. Однако язык Java сделан таким образом, что это никак не повлияет на мобильность приложения. Забегая вперед, скажем, что в языке Java нет указателей, поэтому вы не можете адресоваться к элементам массива чисел по относительному смещению этих элементов в оперативной памяти. Следовательно, точный размер элемента в данном случае не играет никакой роли.

Все базовые типы данных по умолчанию инициализируются, поэтому программисту не нужно об этом беспокоиться. Вы можете также инициализировать переменные базовых типов в программе или при их определении, как это показано ниже:

```
int nCounter = 0;
int i;
i = 8;
```

Переменные типа `boolean` могут находиться только в двух состояниях - `true` и `false`, причем эти состояния никаким образом нельзя соотнести с целыми значениями. Вы не можете, как это было в языке C, выполнить преобразование типа `boolean`, например, к типу `int` - компилятор выдаст сообщение об ошибке.

Переменная типа `byte` занимает восемь бит памяти и про нее больше нечего сказать.

Что же касается типа `char`, то он используется для хранения символов в кодировке Unicode. Эта кодировка позволяет хранить национальные наборы символов, что очень удобно для интернациональных приложений, предназначенных для работы в Internet.

Переменные типа `byte`, `short`, `int` и `long` являются знаковыми. В языке Java нет беззнаковых переменных, как это было в языке C.

Приложение Java может оперировать числами в формате с плавающей точкой, определенным в спецификации IEEE 754. Тип `float` позволяет хранить числа с одинарной точностью, а формат `double` - с двойной.

Переменные базовых типов могут передаваться функциям в качестве параметров только по значению, но не по ссылке. Поэтому следующий фрагмент кода работать не будет:

```
int x;
void ChangeX(int x)
{
    x = 5;
}
. . .
x = 0;
ChangeX(x);
```

После вызова функции `ChangeX` содержимое переменной `x` останется равным нулю.

Проблему можно решить, если вместо базовых переменных использовать объекты встроенных классов, соответствующие базовым переменным. О встроенных классах вы узнаете из следующего раздела.

Библиотеки классов Java

Если предоставить в распоряжение программиста только язык программирования и не снабдить его набором готовых модулей, предназначенных для решения самых распространенных задач, ему придется отвлекаться на множество мелких деталей. Обычно все профессиональные системы разработки приложений содержат в своем составе набор стандартных библиотечных функций или библиотеки классов, таких как Microsoft MFC или Borland OWL. В комплекте со всеми средствами разработки Java поставляются достаточно развитые библиотеки классов, значительно упрощающие программирование. В этом разделе мы кратко расскажем о составе и назначении библиотек классов Java.

Встроенные классы

В языке Java все классы происходят от класса `Object`, и, соответственно, наследуют методы этого класса. Некоторые библиотеки классов подключаются автоматически, и мы будем называть их встроенными. К таким относится, в частности, библиотека с названием `java.lang`. Другие библиотеки классов вы должны подключать в исходном тексте приложения Java явным образом с помощью оператора `import`, о котором мы еще расскажем.

Замещающие классы

Очень часто в наших приложениях вместо базовых типов переменных мы будем использовать объекты встроенных классов, которые называются замещающими классами (*wrapper classes*). Ниже мы перечислили названия этих классов и названия базовых типов данных, которые они замещают:

Базовый тип данных	Замещающий класс
<code>boolean</code>	<code>Boolean</code>
<code>char</code>	<code>Character</code>
<code>int</code>	<code>Integer</code>
<code>long</code>	<code>Long</code>
<code>float</code>	<code>Float</code>
<code>double</code>	<code>Double</code>

Заметим, что для преобразования базовых типов данных в объекты замещающего класса и обратно вы не можете применять оператор присваивания. Вместо этого необходимо использовать соответствующие методы замещающих классов, которые мы будем рассматривать по ходу дела. Полную информацию о классах и методах вы сможете найти в справочной системе, которая устанавливается вместе с Microsoft Visual J++, поэтому в нашей книге вы не найдете справочника по классам Java.

Класс String

Класс String предназначен для работы с такими часто встречающимися объектами, как текстовые строки. Методы этого класса позволяют выполнять над строками практически все операции, которые вы делали раньше при помощи библиотечных функций C. Это преобразование строки в число и обратно с любым заданным основанием, определение длины строки, сравнение строк, извлечение подстроки и так далее.

Хотя в языке Java не допускается перезагрузка (переопределение) операторов, для объектов класса String и объектов всех произошедших от него классов сделана встроенная перезагрузка операторов "+" и "+=". С помощью этих операторов можно выполнять слияние текстовых строк, например:

```
System.out.println("x = " + x + '\n');
```

Здесь в качестве параметра функции println передается текстовая строка, составленная из трех компонент: строки "x = ", числа x и символа перехода на следующую строку '\n'. Значение переменной x автоматически преобразуется в текстовую строку (что выполняется только для текстовых строк) и полученная таким образом текстовая строка сливается со строкой "x = ".

Другие встроенные классы

Среди других встроенных классов отметим класс Math, предназначенный для выполнения математических операций, таких как взятие синуса, косинуса и тангенса.

Предусмотрены также классы для выполнения запуска процессов и задач, управления системой безопасности, а также для решения прочих системных задач.

Библиотека встроенных классов содержит очень важные классы для работы с исключениями. Эти классы нужны для обработки ошибочных ситуаций, которые могут возникнуть (и возникают!) при работе приложений или апплетов Java.

Подключаемые библиотеки классов

Ниже мы кратко перечислим подключаемые библиотеки классов для того чтобы вы могли оценить возможности набора классов Java. Подробное описание этих классов есть в справочной системе Microsoft Visual J++ и в литературе, список которой приведен в конце книги. Мы же ограничимся описанием тех классов, которые будем использовать в наших примерах приложений.

Библиотека классов java.util

Библиотека классов java.util очень полезна при составлении приложений, так как в ней имеются классы для создания таких структур, как динамические массивы, стеки и словари. Есть классы для работы с генератором псевдослучайных чисел, для разбора строк на составляющие элементы (токены), для работы с календарной датой и временем.

Библиотека классов java.io

В библиотеке классов java.io собраны классы, имеющие отношение к вводу и выводу данных через потоки. Заметим, что с использованием этих классов можно работать не только с потоками байт, но также и с потоками данных других типов, например числами int или текстовыми строками.

Библиотека классов java.net

Язык программирования Java разрабатывался в предположении, что им будут пользоваться для создания сетевых приложений. Поэтому было бы странно, если бы в составе среды разработки приложений Java не поставлялась библиотека классов для работы в сети. Библиотека классов java.net предназначена как раз для этого. Она содержит классы, с помощью которых можно работать с универсальными сетевыми адресами URL, передавать данные с использованием сокетов TCP и UDP, выполнять различные операции с адресами IP. Эта библиотека содержит также классы для выполнения преобразований двоичных данных в текстовый формат, что часто бывает необходимо.

В качестве примера приложения, составленного на языке программирования Java и ориентированного на работу в сети Internet, можно привести игру Java Color Lines (рис. 1.3).

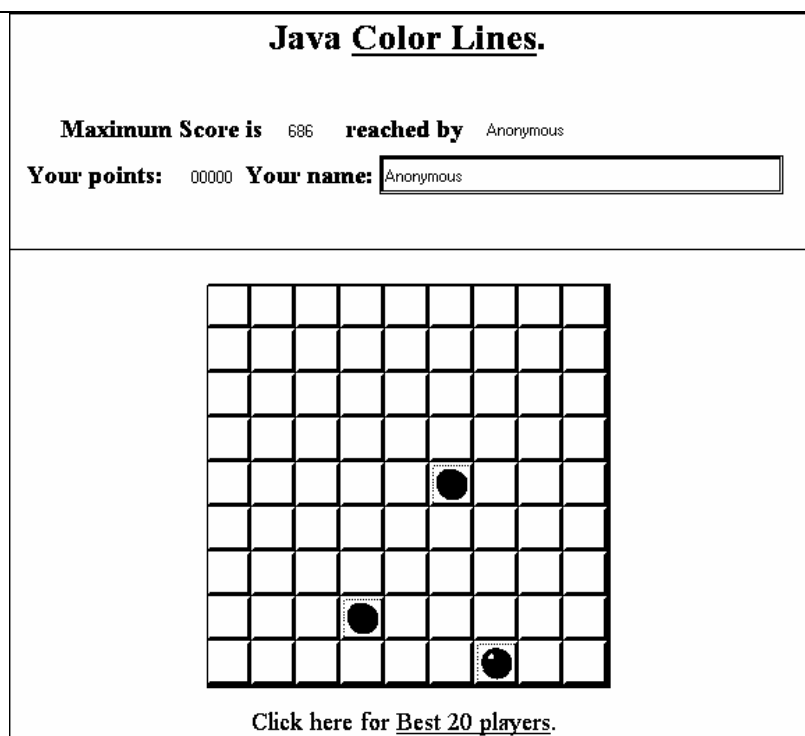


Рис. 1.3. Сетевая игра Java Color Lines, расположенная на сервере <http://spektr.orc.ru>

Это сетевая версия известной игры Lines, которая выполнена в виде нескольких апплетов, взаимодействующих между собой и между сервером WWW с адресом <http://spektr.orc.ru>, на котором они расположены. Так как список имен игроков и достигнутых ими результатов хранится на сервере, вы можете поучаствовать в мировом турнире, сразившись с игроками из разных стран.

В следующем томе “Библиотеки системного программиста”, посвященном Java, мы расскажем о том, как организовать взаимодействие между апплетами и сервером WWW.

Библиотека классов `java.awt`

Для создания пользовательского интерфейса апплеты Java могут и должны использовать библиотеку классов `java.awt`. AWT - это сокращение от Abstract Window Toolkit (инструментарий для работы с абстрактными окнами).

Классы, входящие в состав библиотеки `java.awt`, предоставляют возможность создания пользовательского интерфейса способом, не зависящим от платформы, на которой выполняется апплет Java. Вы можете создавать обычные окна и диалоговые панели, кнопки, переключатели, списки, меню, полосы просмотра, однострочные и многострочные поля для ввода текстовой информации.

Библиотека классов `java.awt.image`

В среде любой операционной системы работа с графическими изображениями является достаточно сложной задачей. В 14 томе “Библиотеки системного программиста”, который называется “Графический интерфейс GDI в MS Windows” мы детально рассмотрели вопросы, связанные с рисованием графики и обработкой графических файлов в среде операционной системы Microsoft Windows. Если вы будете рисовать графические изображения в среде IBM OS/2 или X-Windows, вам, очевидно, придется использовать другие методики и другой программный интерфейс. Большую сложность также вызывает разбор заголовков графических файлов, так как они могут иметь различный формат и иногда содержат неправильную или противоречивую информацию.

Когда вы программируете на Java, рисование и обработка графических изображений выполняется намного проще, так как вам доступна специально предназначенная для этого библиотека классов `java.awt.image`. Помимо широкого разнообразия и удобства определенных в ней классов и методов, отметим способность этой библиотеки работать с графическими изображениями в формате GIF. Этот формат широко используется в Internet, так как он позволяет сжимать файлы графических изображений во много раз без потери качества за счет устранения избыточности.

Библиотека классов `java.awt.peer`

Библиотека классов `java.awt.peer` служит для подключения компонент AWT (например, кнопок, списков, полей редактирования текстовой информации, переключателей и так далее) к реализациям, зависящим от платформы, в процессе создания этих компонент.

Библиотека классов `java.applet`

Как нетрудно догадаться из названия, библиотека классов `java.applet` инкапсулирует поведение апплетов Java. Когда вы будете создавать свои апплеты, вам будет нужен класс `Applet`, расположенный в этой библиотеке классов. Дополнительно в библиотеке классов `java.applet` определены интерфейсы для подключения апплетов к содержащим их документам и классы для проигрывания аудиофрагментов.

Указатели, которых нет

Самая большая и шокирующая новость для тех, кто раньше программировал на C, а теперь занялся изучением Java, это то, что в языке Java нет указателей. Традиционно считалось, что работать с указателями трудно, а их использование приводит к появлению трудно обнаруживаемых ошибок. Поэтому разработчики Java решили отказаться от использования указателей совсем.

Спешим успокоить - вы сможете успешно составлять приложения Java и без указателей, несмотря на то что вам, возможно, придется немного изменить стиль программирования.

Вы можете спросить: как же передавать функциям ссылки на объекты, если нет указателей?

Если вам нужно передать ссылку на переменную базового типа, такого, например, как `int` или `long`, то ничего не получится - мы уже говорили, что переменные базовых типов передаются по значению, а не по ссылке. Поэтому вы не сможете напрямую создать на языке Java эквивалент следующей программы, составленной на языке C:

```
// Некоторая переменная
int nSomeValue;

// Функция, изменяющая значение переменной,
// заданной своим адресом
void StoreValue(int *pVar, int nNewValue)
{
    pVar->nNewValue;
}
...
StoreValue(&nSomeValue, 10);
```

Выход, однако, есть.

Язык Java позволяет использовать вместо указателей ссылки на объекты. Пользуясь этими ссылками, вы можете адресовать объекты по их имени, вызывая методы и изменяя значения данных объектов.

Что же касается данных базовых типов, если вам нужно передавать на них ссылки, то следует заменить базовые типы на соответствующие встроенные классы. Например, вместо типа `int` используйте класс `Integer`, вместо типа `long` - класс `Long` и так далее.

Инициализация таких объектов должна выполняться с помощью конструктора, как это показано ниже:

```
Integer nSomeValue;
nSomeValue = new Integer(10);
```

Первая строка создает неинициализированную ссылку с именем `nSomeValue` и типом `Integer`. При попытке использования такой ссылки возникнет исключение.

Вторая строка создает объект класса `Integer`, вызывая конструктор. Этот конструктор определяет начальное значение. После выполнения оператора присваивания ссылка `nSomeValue` будет ссылаться на реальный объект класса `Integer` и ее можно будет использовать.

Имя объекта `nSomeValue` типа `Integer` вы можете передавать функциям в качестве параметра, причем это будет ссылкой на объект.

Составляя программы на языке C, вы часто использовали указатели для адресации элементов массивов, созданных статически или динамически в оперативной памяти. Зная начальный адрес такого массива и тип хранящихся в нем элементов, вы могли адресоваться к отдельным элементам массива.

В языке Java реализован механизм массивов, исключающих необходимость использования указателей.

Массивы в Java

Для создания массива вы можете пользоваться квадратными скобками, расположив их справа от имени массива или от типа объектов, из которых составлен массив, например:

```
int nNumbers[];
int[] nAnotherNumbers;
```

Допустимы оба варианта, поэтому вы можете выбрать тот, который вам больше нравится.

При определении массивов в языке Java нельзя указывать их размер. Приведенные выше две строки не вызывают резервирования памяти для массива. Здесь просто создаются ссылки на массивы, которые без инициализации использовать нельзя.

Для того чтобы заказать память для массива, вы должны создать соответствующие объекты с помощью ключевого слова `new`, например:

```
int[] nAnotherNumbers;
nAnotherNumbers = new int[15];
```

Как выполнить инициализацию ячеек таблицы?

Такую инициализацию можно выполнить либо статически, либо динамически. В первом случае вы просто перечисляете значения в фигурных скобках, как это показано ниже:

```
int[] nColorRed = {255, 255, 100, 0, 10};
```

Динамическая инициализация выполняется с использованием индекса массива, например, в цикле:

```
int nInitialValue = 7;
int[] nAnotherNumbers;
nAnotherNumbers = new int[15];
for(int i = 0; i < 15; i++)
{
    nAnotherNumbers[i] = nInitialValue;
}
```

Вы можете создавать массивы не только из переменных базовых типов, но и из произвольных объектов. Каждый элемент такого массива должен инициализироваться оператором `new`.

Массивы могут быть многомерными и, что интересно, несимметричными.

Ниже создается массив массивов. В нулевом и первом элементе создается массив из четырех чисел, а во втором - из восьми:

```
int[][] nDim = new int[5][10];
nDim[0] = new int [4];
nDim[1] = new int [4];
nDim[2] = new int [8];
```

Заметим, что во время выполнения приложения Java виртуальная машина Java проверяет выход за границы массива. Если приложение пытается выйти за границы массива, происходит исключение.

Массивы в языке Java являются объектами некоторого встроенного класса. Для этого класса существует возможность определить размер массива, обратившись к элементу данных класса с именем `length`, например:

```
int[] nAnotherNumbers;
nAnotherNumbers = new int[15];
for(int i = 0; i < nAnotherNumbers.length; i++)
{
    nAnotherNumbers[i] = nInitialValue;
}
```

Для определения размера массива вам не нужен такой оператор как `sizeof` из языка программирования C, так как существует другой способ определения размера массива.

Сборка мусора

Одна из интереснейших особенностей языка программирования Java и среды выполнения приложений Java заключается в наличии специального процесса сборки мусора, предназначенного для удаления ненужных объектов из памяти. Эта система избавляет программиста от необходимости внимательно следить за использованием памяти, освобождая ненужные более области явным образом.

Создавая объекты в Java, вы можете руководствоваться принципом “Создай и забудь”, так как система сборки мусора позаботится об удалении ваших объектов. Объект будет удален из памяти, как только на него не останется ни одной ссылки из других объектов.

Приоритет процесса сборки мусора очень низкий, поэтому “уборка” среды выполнения приложений Java не отнимает ресурсы у самих приложений.

Особенности реализации классов в Java

Если вы умеете программировать на языке C++, у вас не возникнет никаких проблем с программированием на Java, так как эти языки очень похожи. Однако есть и некоторые отличия, которые следует учитывать. Мы приведем только краткое перечисление основных отличий. Более подробную информацию вы найдете в литературе, список которой есть в конце книги.

Определение класса

Для создания классов вы можете использовать только ключевое слово `class`. Что же касается `union`, то это ключевое слово в Java не применяется для создания классов.

В языке программирования C++ описание класса может быть отделено от его определения. Для Java это не так - описание класса не допустимо. Все методы должны быть определены внутри определения класса.

Недопустимо определение вложенных классов.

В Java также нет шаблонов. Вы можете создавать классы только на базе других классов.

Объект класса создается при помощи ключевого слова `new`, однако вы не можете удалить объект явным образом, так как ключевое слово `delete` языка программирования C++ в Java не используется.

При определении класса вы не можете указать деструктор. Функции удаления объектов Java из памяти выполняет система сборки мусора.

Внутри одного исходного файла вы можете определить только один общедоступный класс `public`.

Все классы в Java наследуются от класса `Object`, поэтому для любого объекта вы можете использовать методы этого класса.

Определение методов

Вы не можете определять методы вне тела класса, создавая таким образом глобальные функции. Нет также возможности определения вне класса глобальных данных. Тем не менее, внутри класса можно определять статические методы и поля (с помощью ключевого слова `static`), которые будут играть роль глобальных методов и данных.

Пользуясь ключевыми словами `static` и `final`, вы можете определять внутри классов глобальные константы.

Если в базовом классе метод определен с ключевым словом `final`, его нельзя переопределить в дочернем классе, созданном на базе данного метода.

Методы не могут быть определены как `inline`.

Методы Java могут создавать исключения, вызванные возникновением ошибок или других событий. Все создаваемые исключения должны либо обрабатываться внутри метода, либо описываться в определении метода после ключевого слова `throws`.

Переопределение операторов

В языке C++ вы могли переопределить операторы, такие как +, -, ++ и так далее. Язык Java не допускает такое переопределение, что сделано для упрощения программирования. Тем не менее, операторы "+" и "+=" перегружены по умолчанию для выполнения операции слияния текстовых строк класса String.

Интерфейсы

Интерфейсы создаются при помощи ключевого слова interface таким же образом, что и классы. Однако в отличие от последних, интерфейсы являются аналогом абстрактных базовых классов без полей данных и предназначены только для определений набора методов для решения каких-либо задач, например, добавления компонент в контейнеры, организации списков, сортировки и так далее.

Вы можете создать свой класс на базе другого класса, указав при этом с помощью ключевого слова implements, что он реализует тот или иной интерфейс. При этом наряду с методами базового класса в созданном таким образом классе будут доступны методы, определенные в интерфейсе.

Ссылки на методы класса

Так как в Java нет указателей, нет возможности ссылаться на методы с помощью оператора ->. Для ссылки на метод класса используется только оператор "точка".

Оператор "::" также не определен в Java. Если вам необходимо вызвать метод из базового класса, следует использовать ключевое слово super.

Наследование

С помощью ключевого слова extends вы можете унаследовать один класс (дочерний) от другого (базового).

Множественное наследование не допускается. Таким образом, для каждого дочернего класса может быть только один базовый класс. При необходимости, однако, этот дочерний класс может реализовывать произвольное количество интерфейсов.

Для ссылки на методы базового класса вы должны использовать ключевое слово super.

При необходимости вы можете вызвать в первой исполняемой строке конструктор дочернего класса конструктор базового класса (опять же с помощью ключевого слова super).

2 ПЕРВОЕ ПРИЛОЖЕНИЕ И ПЕРВЫЙ АПЛЕТ

Не стремясь быть оригинальными, начнем программирование на Java с составления простейшей программы, которая выводит текстовую строку "Hello, Java!". Для этого вам сначала нужно установить среду разработки Microsoft Visual J++, запустив программу setup.exe, расположенную в корневом каталоге дистрибутивного компакт-диска.

Процесс установки не вызывает затруднений. Вы просто должны следовать инструкциям, появляющимся на экране. Заметим только, что если у вас уже была установлена система Microsoft Visual C++, то имеет смысл для установки Microsoft Visual J++ использовать тот же каталог, что был использован для установки Microsoft Visual C++. При этом формируется единая среда разработки приложений на языках программирования C++ и Java, что очень удобно.

Приложение Hello

Как мы уже говорили, приложения Java могут выполняться под управлением специального интерпретатора, работающего в рамках отдельного процесса, либо под управлением навигатора Internet, такого как Microsoft Internet Explorer или Netscape Navigator. В последнем случае приложение называется апплетом.

Первое приложение, которое мы рассмотрим, относится к простым приложениям, второе будет апплетом, встроенным в документ HTML.

Подготовка и запуск приложения

Итак, запустите среду Microsoft Developer Studio, сделав двойной щелчок левой клавишей мыши по соответствующей пиктограмме.

Выберите из меню File строку New. На экране появится диалоговая панель New, показанная на рис. 2.1.

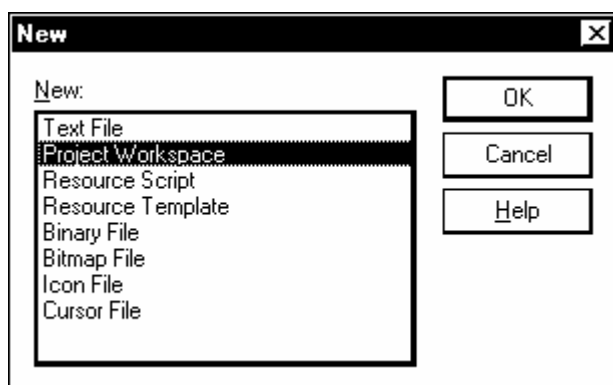


Рис. 2.1. Диалоговая панель New

В этой диалоговой панели вам нужно выбрать строку Project Workspace и нажать кнопку OK. Сразу вслед за этим вы увидите диалоговую панель New Project Workspace (рис. 2.2).

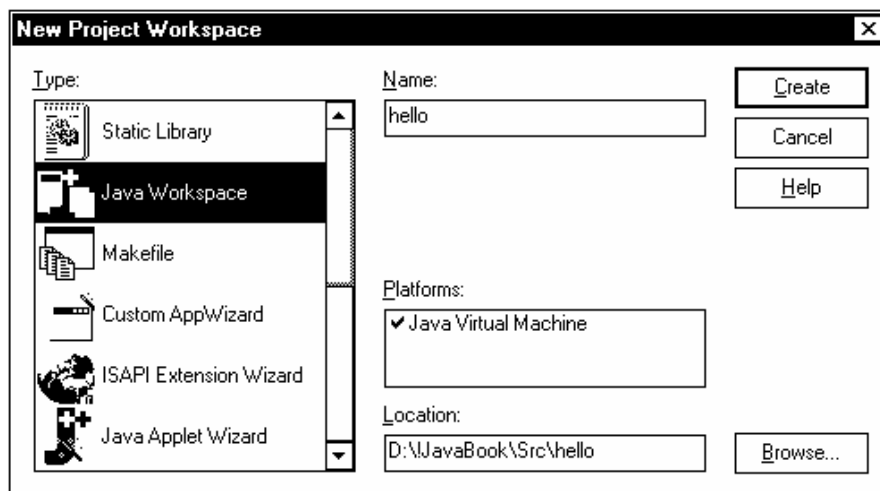


Рис. 2.2. Диалоговая панель New Project Workspace

В левой части этой диалоговой панели есть список различных типов проектов, которые можно создать. К теме нашей книги относятся проекты типа Java Workspace и Java Applet Wizard. Первый из них предназначен для ручного создания приложений и апплетов Java, второй позволяет создавать заготовки апплетов Java в полуавтоматическом режиме.

Выберите проект типа Java Workspace. В поле Location укажите каталог, в котором будут создаваться проекты, а в поле Name - имя проекта. Переключатель Java Virtual Machine должен находиться во включенном состоянии.

После заполнения диалоговой панели New Project Workspace нажмите кнопку Create. Будет создан проект, в котором пока нет ни одного исходного файла.

Далее из меню Insert среды разработки Microsoft Developer Studio выберите строку Files into Project. В поле Filename наберите имя файла hello.java и нажмите кнопку Add. На экране появится сообщение о том, что файл с указанным именем не существует. Вы, однако, можете добавить ссылку на этот файл в проект, нажав кнопку Yes.

На следующем этапе вы должны открыть папку файлов проекта, отмеченную как hello files на средней странице блокнота, расположенного в левой части окна системы разработки (рис. 2.3).

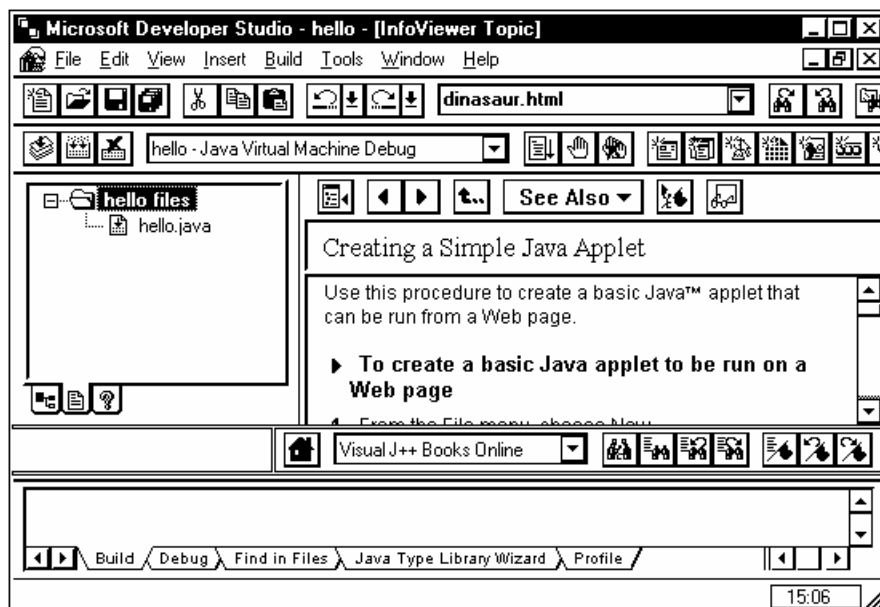


Рис. 2.3. В проект добавлен файл hello.java

Сделайте двойной щелчок по имени файла hello.java. На экране появится сообщение о том, что файла с указанным именем не существует. Для того чтобы создать его, нажмите кнопку Yes.

В правой части главного окна системы разработки появится окно редактирования, в котором вам нужно ввести исходный текст нашей программы, приведенный в листинге 2.1.

Листинг 2.1. Файл hello\hello.java

```
public class hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello, Java!\n");
    }
}
```

Затем выберите из меню Build строку Execute. На экране появится сообщение о том, что запускаемый файл hello.class не существует (рис. 2.4).



Рис. 2.4. Сообщение о том, что файл hello.class не существует

Для создания файла нажмите кнопку Yes. Исходный текст программы будет откомпилирован. Если вы ввели его правильно, сообщения об ошибках не появятся.

Затем вы увидите на экране диалоговую панель Information For Running Class, показанную на рис. 2.5.

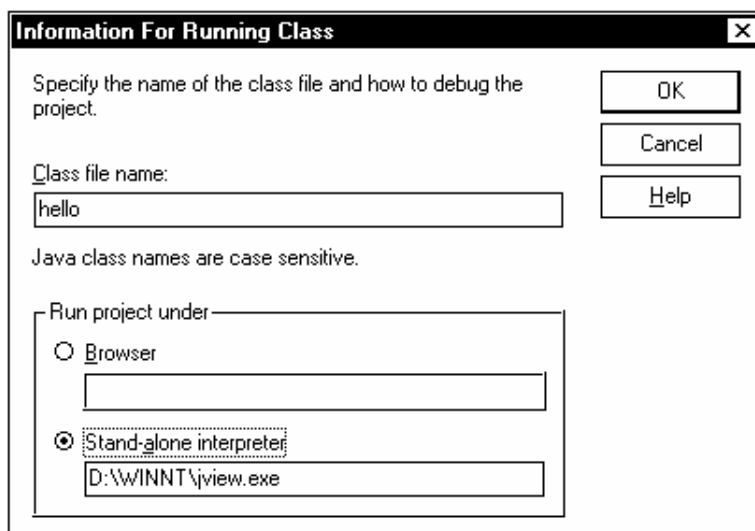


Рис. 2.5. Диалоговая панель Information For Running Class

В поле Class file name вам нужно ввести строку hello. Это имя класса и оно должно в точности соответствовать имени класса, указанному в определении класса (листинг 2.1), а также имени файла, в котором определен класс.

Заметим, что для каждого класса типа public вы должны создавать отдельный файл. Имя этого файла должно быть таким же, что и имя класса (с учетом строчных и прописных букв), а расширение имени файла необходимо указать как class.

Указав имя класса, включите переключатель Stand-alone interpreter в поле Run project under. При этом ваша программа будет выполняться под управлением автономного интерпретатора Java jview.exe, который находится в каталоге Windows.

После того как вы нажмете кнопку OK, ваше приложение будет запущено. На короткое время вы увидите окно интерпретатора Java, в котором появится сообщение Hello, Java! (рис. 2.6).

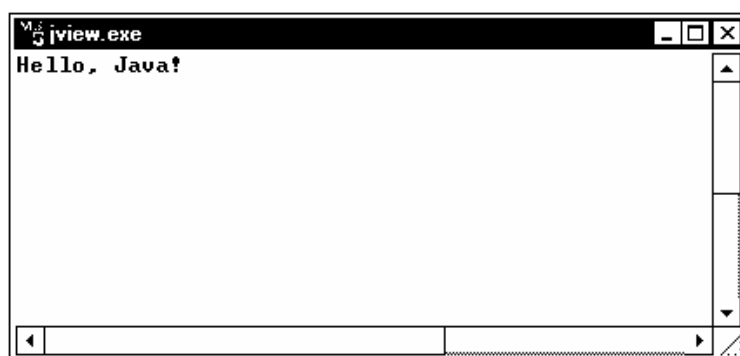


Рис. 2.6. Окно интерпретатора Java с сообщением, выведенным нашим приложением

Не огорчайтесь, что это окно быстро исчезло - вы можете запустить программу под отладкой и наслаждаться видом окна сколько угодно, остановив работу программы после оператора вывода сообщения. Давайте попробуем сделать это.

Установите курсор на оператор `System.out.println("Hello, Java!\n")` и нажмите комбинацию клавиш `<Ctrl+F10>`. Работа программы будет остановлена в момент достижения указанной строки. На экране появится окно интерпретатора jview.exe, не содержащее пока никаких сообщений.

Далее выполняйте программу по шагам, нажимая клавишу `<F10>`. После первого раза в окне интерпретатора появится сообщение Hello, Java!. На второй раз приложение завершит свою работу и окно интерпретатора Java исчезнет с экрана.

Взгляд на исходный текст приложения Hello

Давайте теперь взглянем еще раз на исходный текст приложения Hello и посмотрим, что там к чему. Так как этот текст велик, для удобства приведем его снова:

```
public class hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello, Java!\n");
    }
}
```

В приложении определен один класс `public` с именем `hello`. Исходный файл приложения Java может содержать только один класс `public`, причем имя файла должно в точности соответствовать имени такого класса. В данном случае исходный файл называется `hello.java`. Если бы вы назвали файл `Hello.java`, компилятор выдал бы сообщение об ошибке.

В классе `hello` мы определили один статический метод с именем `main`. Статическая функция с этим именем является точкой входа приложения Java, если она определена в классе `public` с именем, таким же как и имя файла.

В качестве параметра функции `main` передается ссылка на массив строк класса `String`. Через эти строки вы можете передавать приложению Java параметры запуска.

Как наше приложение выводит текстовую строку на консоль?

В классе `System` определена переменная класса `PrintStream` с именем `out`. В классе `PrintStream` определен метод `println`, при помощи которой наше приложение выводит сообщение "Hello, Java!" на консоль.

Но где же объект, для которого вызывается метод `println`? В классе `System` поле `PrintStream` определено как статическое, поэтому методы этого класса можно вызывать, не создавая объектов класса, чем мы и воспользовались.

Как видите, текст простейшего приложения Java по своей сложности не намного превосходит исходный текст программы аналогичного назначения, составленной на языке программирования C.

Простейший апплет

Апплетами называются приложения Java, которые выполняются под управлением виртуальной машины Java, встроенной в браузер, такой как Microsoft Internet Explorer или Netscape Navigator. Апплет встраивается в документ HTML и выглядит как окно заранее заданного размера. Он может рисовать в своем окне (и только в нем) произвольные изображения или текст.

Двоичный файл с исполняемым (а точнее говоря, интерпретируемым) кодом Java располагается на сервере WWW. В документе HTML с помощью оператора `<APPLET>` организуется ссылка на этот двоичный файл.

Когда пользователь загружает в браузер документ HTML с апплетом, файл апплета переписывается с сервера WWW на рабочую станцию пользователя. После этого браузер начинает его выполнение.

Возможно, вам не понравится такая идея, как запуск чужого апплета на своем компьютере - мало ли чего этот апплет может там сделать. Однако апплеты, в отличие от обычных приложений Java, сильно ограничены в своих правах. Например, они не могут читать локальные файлы и тем более в них писать. Есть также ограничения и на передачу данных через сеть: апплет может передавать данные только тому серверу WWW, с которого он загружен. В крайнем случае вы можете совсем отказаться от использования апплетов, отключив возможность их загрузки соответствующей настройкой браузера. Но мы пока не будем этого делать, так как апплеты являются предметом изучения в нашей книге.

Давайте создадим простейший апплет, воспользовавшись для этого системой автоматизированной разработки шаблонов апплета Java Applet Wizard, встроенной в Microsoft Visual J++.

Запустите систему Microsoft Visual J++ и выберите из меню `File` строку `New`. В появившейся на экране диалоговой панели `New` выберите строку `New Project Workspace`. Затем вам нужно выбрать тип проекта Java Applet Wizard, как это показано на рис. 2.7.

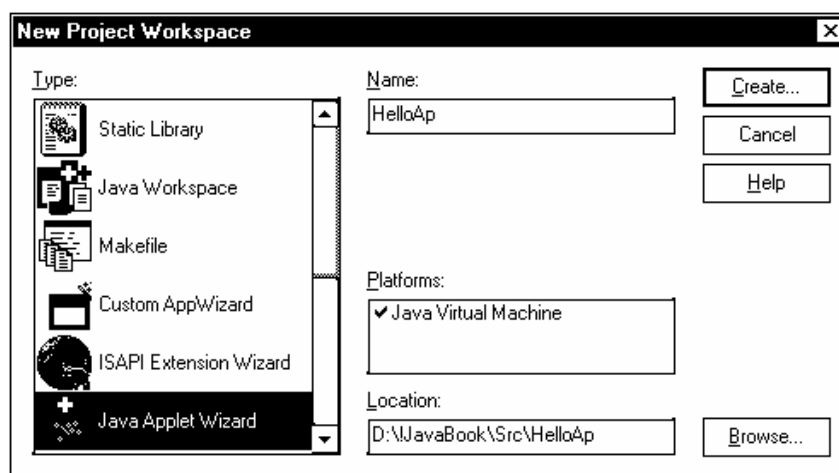


Рис. 2.7. Выбор типа проекта Java Applet Wizard

В поле `Name` введите имя приложения `HelloAp`, а в каталоге `Location` укажите путь к каталогу, в котором будут созданы файлы проекта. Затем нажмите кнопку `Create`. Вслед за этим на экране появится по очереди несколько диалоговых панелей, в которых вы должны описать создаваемый апплет.

Первая такая диалоговая панель показана на рис. 2.8.

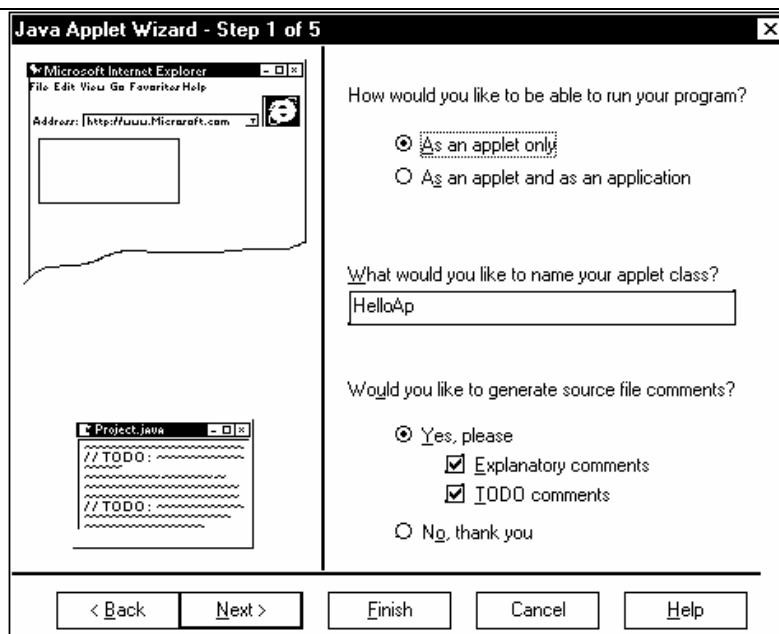


Рис. 2.8. Выбор типа приложения, названия класса и степени подробности создаваемых комментариев к исходному тексту

В поле How would you like to be able to run your program включите переключатель As an applet only. При этом создаваемое приложение сможет работать только под управлением навигатора.

Имя класса апплета нужно указать в поле What would you like to name your applet class. Оставьте имя HelloAp, которое там есть по умолчанию.

Состояние переключателей в поле Would you like to generate source file comments влияет на то, насколько подробно будут комментироваться создаваемый исходный текст приложения, и будет ли он комментироваться вообще.

Если включить переключатель Yes, please, в исходный текст будут добавлены комментарии. Если же включить переключатель No, thank you, никаких комментариев не будет.

При включении переключателя Explanatory comments в исходный текст будут включены комментарии, объясняющие назначение отдельных фрагментов кода. Переключатель TODO влияет на то, будут ли отмечены места исходного текста, в который вы должны вставить свой код, наполняющий апплет реальной жизнью.

Завершив заполнение первой диалоговой панели, нажмите кнопку Next и переходите к следующей панели, показанной на рис. 2.9.

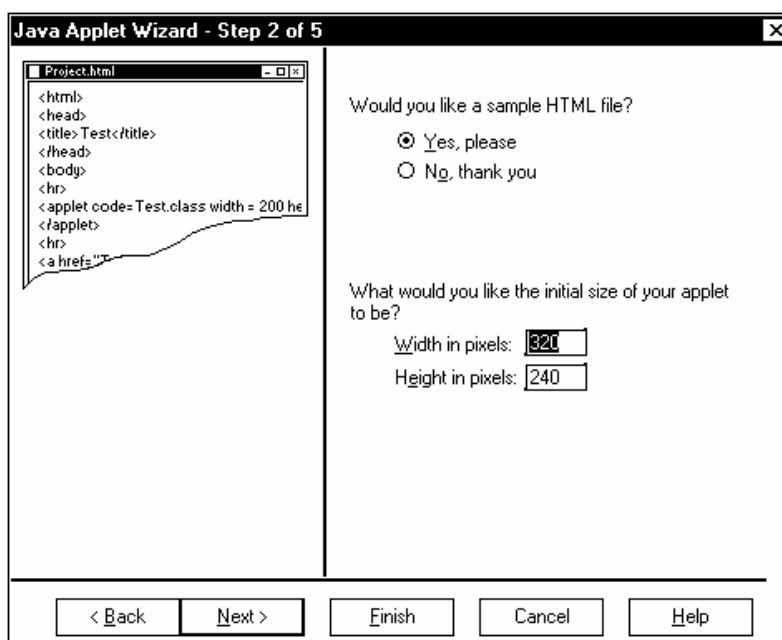


Рис. 2.9. Запрос на создание документа HTML и определение размера окна апплета

Система Java Applet Wizard может создать для вас образец документа HTML, в который будет включен разрабатываемый вами апплет. Для этого во второй диалоговой панели вы должны включить переключатель Yes, please, расположенный в поле Would you like a sample HTML file.

Начальные размеры окна, создаваемого в документе HTML для апплета, определяются в полях Width in pixels и Height in pixels (соответственно, ширина и высота). Заметим, что апплет может изменять размеры своего окна, о чем мы еще будем говорить.

Третья диалоговая панель показана на рис. 2.10.

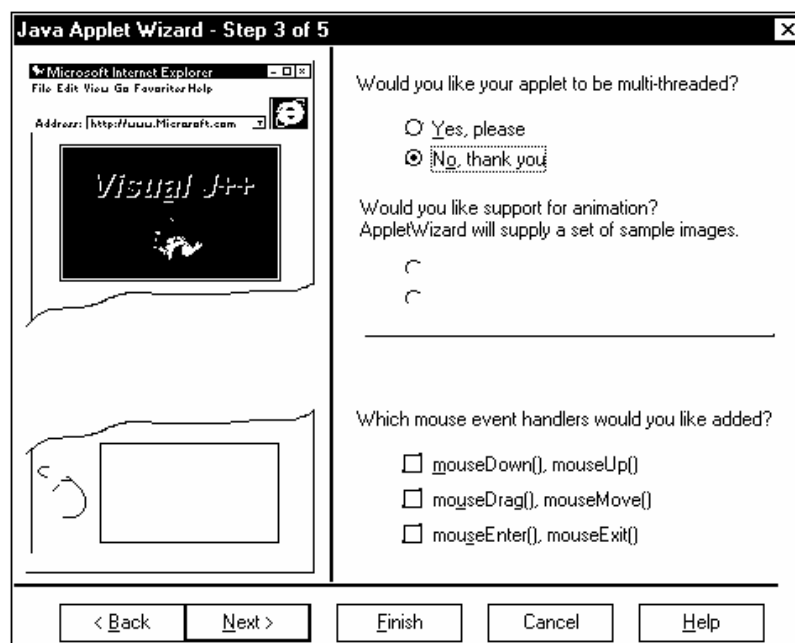


Рис. 2.10. Вопросы, связанные с мультизадачностью, анимацией и обработкой сообщений от мыши

В этой панели вы должны указать, будет ли ваш апплет создавать задачи. Наш первый апплет однозадачный, поэтому в поле Would you like your applet to be multi-threaded вы должны включить переключатель No, thank you.

На вопрос Would you like support for animation вы сможете ответить утвердительно только в том случае, если ваш апплет мультизадачный.

Три переключателя, расположенные в поле Which mouse event handlers would you like added, позволят вам автоматически добавит обработчики сообщений от мыши. Пока не включайте их, так как мышью мы займемся позже.

Следующая, четвертая диалоговая панель показана на рис. 2.11.

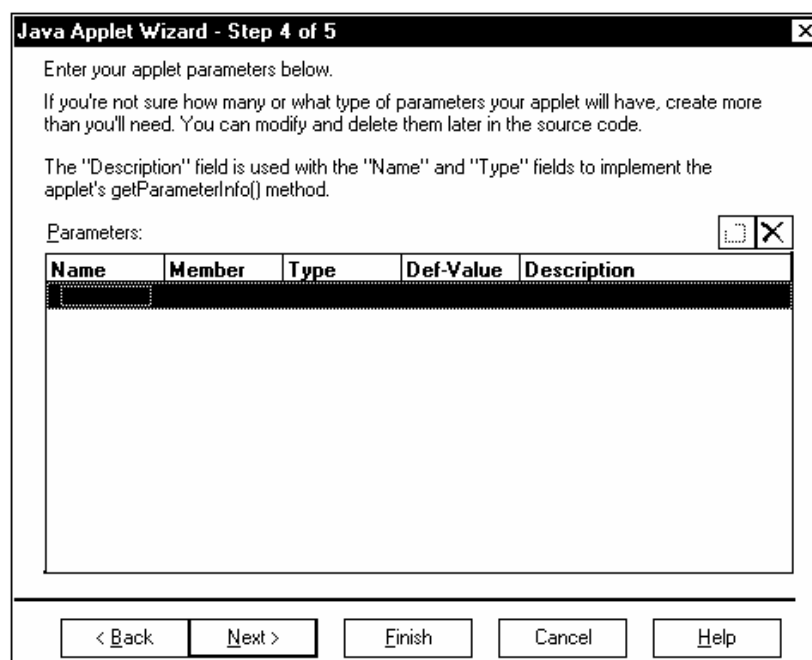


Рис. 2.11. Определение параметров, передаваемых апплету

С помощью этой диалоговой панели вы можете указать, какие параметры должны передаваться апплету через документ HTML при запуске. Нажмите здесь кнопку Next, не добавляя никаких параметров.

В пятой диалоговой панели (рис. 2.12) вам дается возможность отредактировать информацию, описывающую ваш апплет.

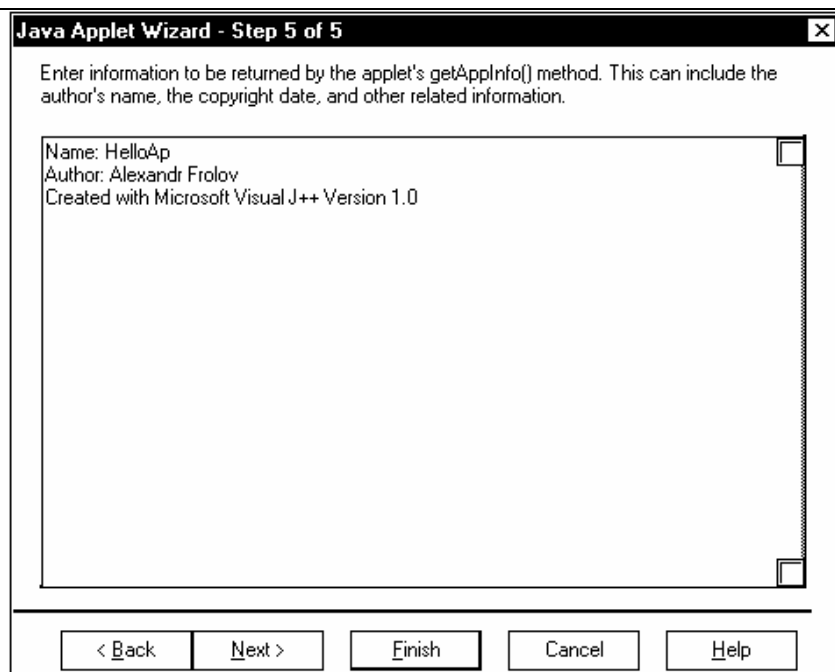


Рис. 2.12. Редактирование информации, описывающей апплет

Эта информация будет возвращаться методом `getAppInfo`, определенным в классе апплета. При необходимости измените строки описания и нажмите кнопку **Next**.
Финальная диалоговая панель показана на рис. 2.13.

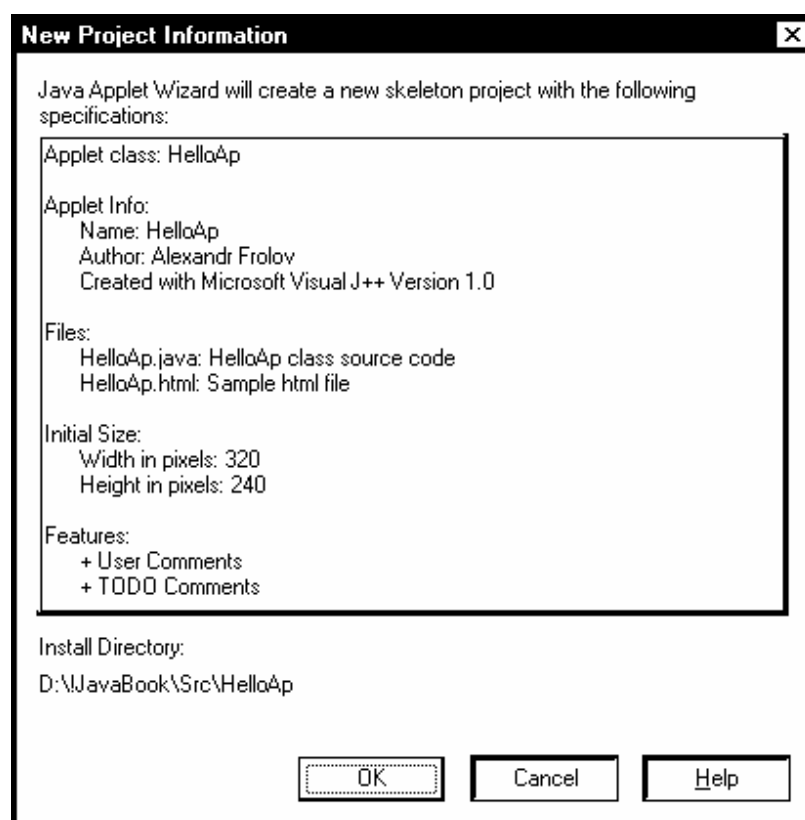


Рис. 2.13. Финальная диалоговая панель

Здесь вы можете последний раз перед созданием файлов проекта посмотреть на заданные вами параметры. Если нажать кнопку **OK**, проект будет создан. Для отказа от создания проекта нажмите кнопку **Cancel**.

В результате работы системы Java Applet Wizard будет создано два файла (не считая файла проекта). Это исходный текст апплета `HelloAp.java` (листинг 2.2) и исходный текст документа HTML `HelloAp.html`, в который включен создаваемый апплет (листинг 2.3).

Листинг 2.2. Файл `HelloAp\HelloAp.java` (комментарии переведены на русский язык)
// *****

```
// HelloAp.java:    Applet
//
//*****
import java.applet.*;
import java.awt.*;

//=====
// Основной класс для апплета HelloAp
//
//=====
public class HelloAp extends Applet
{
    // Конструктор класса HelloAp
    //-----
    public HelloAp()
    {
        // Сделать: Добавьте сюда код конструктора
    }

    // Обеспечение информации об апплете:
    //
    // Метод getAppletInfo возвращает строку, которая
    // описывает апплет. Вы можете привести такую информацию,
    // как имя автора и дата создания, а так же любые другие
    // сведения об апплете
    //-----
    public String getAppletInfo()
    {
        return "Name: HelloAp\r\n" +
            "Author: Alexandr Frolov\r\n" +
            "Created with Microsoft Visual J++ Version 1.0";
    }

    // Метод init вызывается системой AWT при первой загрузке
    // или перезагрузке апплета. Вы можете переопределить этот
    // метод для выполнения еобходимой инициализации апплета,
    // например, инициализации структур данных, загрузку
    // изображений или шрифтов, создание окон фреймов,
    // установку системы управления внешним видом или
    // добавление элементов пользовательского интерфейса
    //-----
    public void init()
    {
        // Если для размещения в окне апплета органов управления
        // вы используете класс "control creator", созданный
        // системой ResourceWizard, из метода init можно
        // вызывать метод CreateControls. Удалите вызов функции
        // resize перед добавлением вызова функции
        // CreateControls, так как эта функция выполняет
        // изменение размера окна апплета самостоятельно
        //-----
        resize(320, 240);

        // Сделать: Добавьте сюда дополнительный код
        // инициализации
    }

    // Разместите здесь дополнительный код, необходимый
    // для "чистого" завершения работы апплета. Метод
    // destroy вызывается, когда апплет завершает работу
    // и будет выгружен из памяти
    //-----
    public void destroy()
    {
        // Сделать: Добавьте сюда код завершения работы апплета
    }

    // Обработчик процедуры рисования окна апплета HelloAp
    //-----
    public void paint(Graphics g)
    {
        g.drawString(
            "Created with Microsoft Visual J++ Version 1.0",
            10, 20);
    }

    // Метод start вызывается при первом появлении на
```

```
// экране страницы HTML с апплетом
//-----
public void start()
{
    // Сделать: Добавьте сюда дополнительный код,
    //    который должен работать при запуске апплета
}

//    Метод stop вызывается когда страница HTML с
//    апплетом исчезает с экрана
//-----
public void stop()
{
    // Сделать: Добавьте сюда дополнительный код,
    //    который должен работать при остановке апплета
}
// Сделать: Добавьте сюда дополнительный код
}
```

Листинг 2.3. Файл HelloAp\HelloAp.html

```
<html>
<head>
<title>HelloAp</title>
</head>
<body>
<hr>
<applet
    code=HelloAp.class
    id=HelloAp
    width=320
    height=240 >
</applet>
<hr>
<a href="HelloAp.java">The source.</a>
</body>
</html>
```

Исходные файлы апплета HelloAp

Как мы уже говорили, в результате работы системы Java Applet Wizard созданы файлы HelloAp.java и HelloAp.html. Рассмотрим их по отдельности.

Файл HelloAp.java

Исходный текст апплета HelloAp начинается с двух строк, подключающих оператором import библиотеки классов:

```
import java.applet.*;
import java.awt.*;
```

Оператор import должен располагаться в файле исходного текста перед другими операторами (за исключением операторов комментария). В качестве параметра оператору import передается имя подключаемого класса из библиотеки классов. Если же необходимо подключить все классы данной библиотеки (как в нашем случае), вместо имени класса указывается символ “*”.

Мы уже перечисляли библиотеки классов Java. Напомним, что библиотека java.applet содержит классы, необходимые для создания апплетов, то есть разновидности приложений Java, встраиваемых в документы HTML и работающих под управлением навигатора Internet. С помощью классов библиотеки java.awt апплет может выполнять в своем окне рисование различных изображений или текста, причем данный метод рисования не зависит от платформы, на которой работает апплет.

Далее в исходном тексте апплета определяется класс типа public с именем HelloAp, которое должно обязательно совпадать с именем файла, содержащего исходный текст этого класса:

```
public class HelloAp extends Applet
{
    . . .
}
```

Определенный нами класс HelloAp с помощью ключевого слова extends наследуется от класса Applet. При этом методам класса HelloAp становятся доступными все методы и данные класса, за исключением определенных как private. Класс Applet определен в библиотеке классов java.applet, которую мы подключили оператором import.

Создавая файл HelloAp.java, система Java Applet Wizard определили в классе HelloAp конструктор и несколько методов, заменив некоторые методы базового класса Applet.

Конструктор HelloAp

Конструктор класса HelloAp находится в одноименном методе и вызывается при создании объекта класса:

```
public HelloAp()
{
```

```
// Сделать: Добавьте сюда код конструктора
}
```

По умолчанию тело конструктора, создаваемого системой Java Applet Wizard, не содержит никакого кода. Однако вы можете добавить сюда строки, выполняющие инициализацию апплета при его создании как объекта.

Метод getAppletInfo

Базовый класс Applet содержит определение метода getAppletInfo, возвращающее значение null. В нашем классе HelloAp, который является дочерним по отношению к классу Applet, система Java Applet Wizard переопределила метод getAppletInfo из базового класса следующим образом:

```
public String getAppletInfo()
{
    return "Name: HelloAp\r\n" +
        "Author: Alexandr Frolov\r\n" +
        "Created with Microsoft Visual J++ Version 1.0";
}
```

Теперь этот метод возвращает текстовую информацию об апплете в виде объекта класса String.

Заметьте, что здесь возвращается достаточно длинная строка. Она разделена на три части, но способ разделения отличен от принятого в языке программирования C: части строки объединены оператором "+". Для объектов класса String этот оператор в языке Java переопределен и имеет очевидное назначение - слияние строк.

Метод init

Метод init, так же как и метод getAppletInfo, определен в базовом классе Applet, от которого наследуются все апплеты. Определение его таково, что этот метод ровным счетом ничего не делает.

Когда вызывается метод init и зачем он нужен?

Метод init вызывается тогда, когда навигатор Internet загружает в свое окно документ HTML с оператором <APPLET>, ссылающимся на данный апплет. В этот момент апплет может выполнять инициализацию, например, создавать задачи, если он работает в мультизадачном режиме.

Существует контрпара для метода init - метод destroy. О нем мы расскажем немного позже.

Система Java Applet Wizard переопределяет метод init следующим образом:

```
public void init()
{
    resize(320, 240);

    // Сделать: Добавьте сюда дополнительный код
    // инициализации
}
```

Здесь вызывается метод resize, который изменяет размер окна апплета. Этот метод определен в базовом классе Applet. В нашем классе вы можете вызывать его потому, что мы образовали этот класс от класса Applet.

Забегая вперед, скажем, что параметры оператора <APPLET>, с помощью которого апплет встраивается в документ HTML, допускают установку размеров окна апплета. Пользуясь методом resize, вы можете изменить эти размеры.

Если же вы желаете изменять размеры окна, редактируя параметры оператора <APPLET> в документе HTML, вы должны удалить вызов метода resize из исходного текста метода init.

Метод destroy

Перед удалением апплета из памяти вызывается метод destroy, который определен в базовом классе Applet как пустая заглушка. Система Java Applet Wizard добавляет в исходный текст класса переопределение метода destroy, которое выглядит следующим образом:

```
public void destroy()
{
    // Сделать: Добавьте сюда код завершения работы апплета
}
```

Здесь вы можете выполнить все необходимые операции, которые следует выполнить перед удалением апплета. Например, если в методе init вы создавали какие-либо задачи, в методе destroy вы можете их завершить.

Метод start

Метод start вызывается после метода init в момент, когда пользователь начинает просматривать документ HTML с встроенным в него апплетом. Система Java Applet Wizard создает заглушку, переопределяющую метод start из базового класса:

```
public void start()
{
    // Сделать: Добавьте сюда дополнительный код,
    // который должен работать при запуске апплета
}
```

Вы можете модифицировать текст этого метода, если при каждом посещении пользователем страницы с апплетом необходимо выполнять какую-либо инициализацию.

Метод stop

Дополнением к методу start служит метод stop. Он вызывается, когда пользователь покидает страницу с апплетом и загружает в окно навигатора другую страницу. Заметим, что метод stop вызывается перед методом destroy.

По умолчанию система Java Applet Wizard переопределяет метод stop базового класса Applet следующим образом:

```
public void stop()
{
    // Сделать: Добавьте сюда дополнительный код,
    //    который должен работать при остановке апплета
}
```

Метод paint

Наиболее интересен для нас метод paint, который выполняет рисование в окне апплета. Вот его исходный текст, созданный системой Java Applet Wizard:

```
public void paint(Graphics g)
{
    g.drawString(
        "Created with Microsoft Visual J++ Version 1.0",
        10, 20);
}
```

Если посмотреть определение класса Applet, то в нем нет метода paint. В каком же классе определен этот метод?

Взглянем на определение класса Applet:

```
public class java.applet.Applet
    extends java.awt.Panel
{
    . . .
}
```

Во-первых, вы видите, что полное имя класса Applet есть java.applet.Applet. Включая оператором import библиотеку классов java.applet.*, мы включали и определение класса Applet.

Во-вторых, из определения класса можно заключить, что класс java.applet.Applet произошел от класса java.awt.Panel. Напомним, что определение классов java.awt.* также было включено в исходный текст нашего апплета оператором import.

Если класс java.applet.Applet был создан на базе класса java.awt.Panel, то нет ли в базовом классе определения метода paint?

Изучив исходный текст класса java.awt.Panel, убеждаемся, что такого метода там нет, однако сам класс java.awt.Panel произошел от класса java.awt.Container:

```
public class java.awt.Panel
    extends java.awt.Container
{
    . . .
}
```

Продолжим наши исследования. В классе java.awt.Container снова нет метода paint, но сам этот класс создан на базе класса java.awt.Component:

```
public abstract class java.awt.Container
    extends java.awt.Component
{
    . . .
}
```

Но и здесь метода paint нет. Этот метод определен в классе java.awt.Component, который, в свою очередь, произошел от класса java.lang.Object и реализует интерфейс java.awt.image.ImageObserver:

```
public abstract class java.awt.Component
    extends java.lang.Object
    implements java.awt.image.ImageObserver
{
    . . .
}
```

Мы проследили иерархию классов от класса java.applet.Applet, на базе которого создан наш апплет, до класса java.lang.Object, который является базовым для всех классов в Java.

Схематически эту иерархию можно изобразить так:

```
java.lang.Object (корневой класс)
-> java.awt.Component
   -> java.awt.Container
       -> java.awt.Panel
           -> java.applet.Applet
```

Метод paint определен в классе java.awt.Component, но так как этот класс является базовым для класса Applet и для нашего класса HelloAp, мы можем переопределить метод paint.

Теперь о том, когда вызывается метод paint.

Этот метод вызывается, когда необходимо перерисовать окно апплета. Если вы создавали приложения для операционной системы Microsoft Windows, то наверняка знакомы с сообщением WM_PAINT, которое поступает в функцию окна приложения при необходимости его перерисовки.

Перерисовка окна приложения Windows и окна апплета обычно выполняется асинхронно по отношению к работе приложения или апплета. В любой момент времени апплет должен быть готов перерисовать содержимое своего окна.

Такая техника отличается от той, к которой вы, возможно, привыкли, создавая обычные программы для MS-DOS. Программы MS-DOS сами определяют, когда им нужно рисовать на экране, причем рисование может выполняться из разных мест программы. Апплеты, так же как и приложения Windows, выполняют рисование в своих окнах централизованно. Апплет делает это в методе paint, а приложение Windows - при обработке сообщения WM_PAINT.

Обратите внимание, что методу paint в качестве параметра передается ссылка на объект Graphics:

```
public void paint(Graphics g)
{
    . . .
}
```

По своему смыслу этот объект напоминает контекст отображения, с которым хорошо знакомы создатели приложений Windows. Контекст отображения - это как бы холст, на котором апплет может рисовать изображение или писать текст. Многочисленные методы класса Graphics позволяют задавать различные параметры холста, такие, например, как цвет или шрифт.

Наше приложение вызывает метод drawString, который рисует текстовую строку в окне апплета:

```
g.drawString(
    "Created with Microsoft Visual J++ Version 1.0", 10, 20);
```

Вот прототип этого метода:

```
public abstract void
drawString(String str, int x, int y);
```

Через первый параметр методу drawString передается текстовая строка в виде объекта класса String. Второй и третий параметр определяют, соответственно, координаты точки, в которой начнется рисование строки.

В какой координатной системе?

Апплеты используют систему координат, которая соответствует режиму отображения MM_TEXT, знакомому тем, кто создавал приложения Windows. Начало этой системы координат расположено в левом верхнем углу окна апплета, ось X направлена слева направо, а ось Y - сверху вниз (рис. 2.14).



Рис. 2.14. Система координат, используемая методом drawString

На этом же рисунке показано, как метод drawString нарисует текстовую строку с координатами (xCoord, yCoord). Более подробно вопросы рисования в окне апплета мы рассмотрим в третьей главе нашей книги, которая так и называется - "Рисование в окне апплета".

Файл HelloApp.html

Файл HelloApp.html автоматически создается системой Java Applet Wizard, если это было указано во второй диалоговой панели, задающей параметры нового проекта.

Нас интересует в этом файле оператор <APPLET>, который используется в паре с оператором </APPLET> и предназначен для встраивания окна апплета в документ HTML.

Вот как выглядит фрагмент документа HTML, созданного для нашего проекта, в котором встраивается апплет:

```
<applet
    code=HelloAp.class
    id=HelloAp
    width=320
    height=240 >
</applet>
```

Рассмотрим параметры оператора <APPLET>, указанные в этом фрагменте кода, а также некоторые другие.

Параметр	Описание
----------	----------

ALIGN	Выравнивание окна апплета относительно окружающего его текста. Возможны следующие значения: LEFT выравнивание влево относительно окружающего текста; CENTER центрирование; RIGHT выравнивание вправо относительно окружающего текста; TOP выравнивание по верхней границе; MIDDLE центрирование по вертикали; BOTTOM выравнивание по нижней границе
ALT	С помощью этого параметра можно задать текст, который будет отображаться в окне апплета в том случае, если навигатор не может работать с апплетами Java
CODE	Имя двоичного файла, содержащего код апплета. По умолчанию путь к этому файлу указывается относительно каталога с файлом HTML, в который встроен апплет. Такое поведение может быть изменено параметром CODEBASE
CODEBASE	Базовый адрес URL апплета, то есть путь к каталогу, содержащему апплет
HEIGHT	Начальная ширина окна апплета в пикселах
WIDTH	Начальная высота окна апплета в пикселах
HSPACE	Зазор слева и справа от окна апплета
VSPACE	Зазор сверху и снизу от окна апплета
NAME	Идентификатор апплета, который может быть использован другими апплетами, расположенными в одном и том же документе HTML
TITLE	Строка заголовка

Дополнительно между операторами `<APPLET>` и `</APPLET>` вы можете задать параметры апплета. Для этого используется оператор `<PARAM>`, который мы рассмотрим позже.

Внешний вид окна навигатора, в котором отображается созданный документ HTML с апплетом, показан на рис. 2.15.

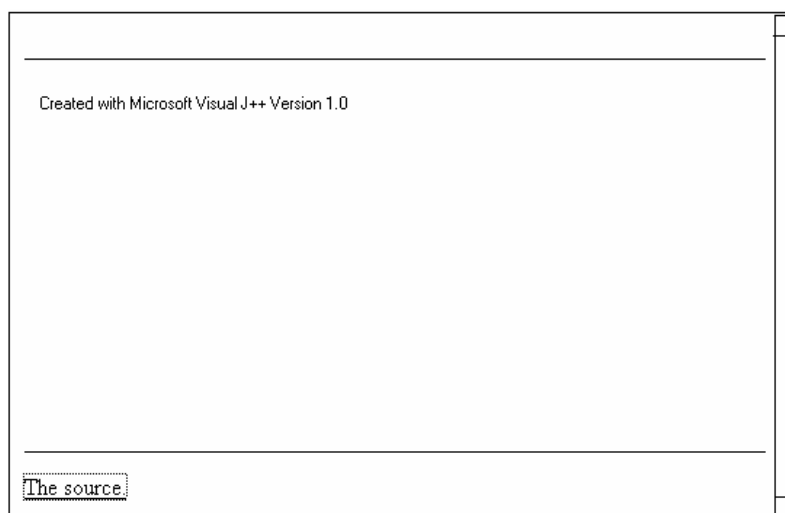


Рис. 2.15. Окно апплета в документе HTML

Обратите внимание, что границы окна апплета никак не выделяются, а цвет его фона совпадает с цветом фона документа HTML. В следующей главе мы научим вас изменять цвет фона окна апплета и текста, отображаемого в этом окне.

В нижней части окна под разделительной линией находится ссылка на исходный текст апплета:

```
<a href="HelloAp.java">The source.</a>
```

Вы можете использовать эту ссылку для просмотра содержимого файла HelloAp.java.

Упрощаем исходный текст апплета

Если вам показалось, что исходный текст апплета слишком сложный, вы можете его упростить, как это показано ниже:

```
//*****
// HelloAp.java:    Applet
//
//*****
import java.applet.*;
import java.awt.*;
```

```
//=====
// Основной класс для апплета HelloAp
//
//=====
public class HelloAp extends Applet
{
    // Обработчик процедуры рисования окна апплета HelloAp
    //-----
    public void paint(Graphics g)
    {
        g.drawString(
            "Created with Microsoft Visual J++ Version 1.0",
            10, 20);
    }
}
```

Мы выбросили определения всех методов, которые не выполняют никакой полезной работы, а также удалили методы `getAppletInfo` и `init`. Апплет будет работать также, как и раньше, потому что методы `init`, `start`, `stop`, `destroy`, `getAppletInfo`, удаленные нами, определены в базовом классе `Applet`.

Однако метод `paint` необходимо переопределить в любом случае, так как именно в нем выполняется рисование строки, то есть то, что делает наш апплет.

Почему же система `Java Applet Wizard` создает пустые определения методов? Просто для того, чтобы вы при необходимости наполнили их чем-нибудь полезным. Если вам не нужны эти определения, вы можете их удалить.

3 РИСОВАНИЕ В ОКНЕ АПЛЕТА

В предыдущей главе мы привели простейший пример апплета, который выполняет рисование текстовой строки в своем окне. Теперь мы более подробно расскажем вам о том, что и как может рисовать апплет.

Контекст отображения

Способ, которым апплет выполняет рисование в своем окне, полностью отличается от того, которым пользуются программы MS-DOS. Вместо того чтобы обращаться напрямую или через драйвер к регистрам видеоконтроллера, апплет пользуется методами из класса Graphics. Эти методы инкапсулируют все особенности аппаратуры, предоставляя в распоряжение программиста платформно-независимое средство рисования.

Для окна апплета создается объект класса Graphics, ссылка на который передается методу paint. В предыдущей главе мы уже пользовались этим объектом, вызывая для него метод drawString, рисующий в окне текстовую строку. Объект, ссылка на который передается методу paint, и есть контекст отображения. Сейчас мы займемся контекстом отображения вплотную.

Полотно для рисования

Проще всего представить себе контекст отображения как полотно, на котором рисует художник. Точно так же как художник может выбирать для рисования различные инструменты, программист, создающий апплет Java, может выбирать различные методы класса Graphics и задавать различные атрибуты контекста отображения.

Методы класса Graphics

В качестве базового для класса Graphics (полное название класса java.awt.Graphics) выступает класс java.lang.Object. В виду важности класса Graphics мы приведем его определение с комментариями:

```
public abstract class java.awt.Graphics
    extends java.lang.Object
{
    // -----
    // Конструктор
    // -----

    protected Graphics();

    // -----
    // Методы
    // -----

    // Стирание содержимого прямоугольной области
    public abstract void
        clearRect(int x, int y, int width, int height);

    // Задание области ограничения вывода
    public abstract void
        clipRect(int x, int y, int width, int height);

    // Копирование содержимого прямоугольной области
    public abstract void
        copyArea(int x, int y, int width,
            int height, int dx, int dy);

    // Создание контекста отображения
    public abstract Graphics create();

    // Создание контекста отображения
    public Graphics create(int x, int y,
        int width, int height);

    // Удаление контекста отображения
    public abstract void dispose();

    // Рисование прямоугольной области с трехмерным
    // выделением
    public void draw3DRect(int x, int y, int width,
        int height, boolean raised);

    // Рисование сегмента
    public abstract void drawArc(int x, int y,
```

```
int width, int height, int startAngle, int arcAngle);

// Рисование текста из массива байт
public void drawBytes(byte data[], int offset,
    int length, int x, int y);

// Рисование текста из массива символов
public void drawChars(char data[], int offset,
    int length, int x, int y);

// Рисование растрового изображения
public abstract boolean
    drawImage(Image img, int x, int y, Color bgcolor,
        ImageObserver observer);

// Рисование растрового изображения
public abstract boolean
    drawImage(Image img, int x, int y,
        ImageObserver observer);

// Рисование растрового изображения
public abstract boolean
    drawImage(Image img, int x, int y,
        int width, int height, Color bgcolor,
        ImageObserver observer);

// Рисование растрового изображения
public abstract boolean
    drawImage(Image img, int x, int y,
        int width, int height, ImageObserver observer);

// Рисование линии
public abstract void drawLine(int x1, int y1,
    int x2, int y2);

// Рисование овала
public abstract void drawOval(int x, int y,
    int width, int height);

// Рисование многоугольника
public abstract void
    drawPolygon(int xPoints[], int yPoints[], int nPoints);

// Рисование многоугольника
public void drawPolygon(Polygon p);

// Рисование прямоугольника
public void drawRect(int x, int y,
    int width, int height);

// Рисование прямоугольника с круглыми углами
public abstract void
    drawRoundRect(int x, int y, int width,
        int height, int arcWidth, int arcHeight);

// Рисование текстовой строки
public abstract void
    drawString(String str, int x, int y);

// Рисование заполненного прямоугольника с
// трехмерным выделением
public void
    fill3DRect(int x, int y, int width,
        int height, boolean raised);

// Рисование заполненного сегмента круга
public abstract void
    fillArc(int x, int y, int width,
        int height, int startAngle, int arcAngle);

// Рисование заполненного овала
public abstract void
    fillOval(int x, int y, int width, int height);

// Рисование заполненного многоугольника
public abstract void
    fillPolygon(int xPoints[], int yPoints[], int nPoints);
```

```

// Рисование заполненного многоугольника
public void fillPolygon(Polygon p);

// Рисование заполненного прямоугольника
public abstract void
    fillRect(int x, int y, int width, int height);

// Рисование заполненного прямоугольника
// с круглыми углами
public abstract void
    fillRoundRect(int x, int y, int width, int height,
        int arcWidth, int arcHeight);

// Прослеживание вызова метода dispose
public void finalize();

// Определение границ области ограничения вывода
public abstract Rectangle getClipRect();

// Определение цвета, выбранного в контекст отображения
public abstract Color getColor();

// Определение шрифта, выбранного в контекст отображения
public abstract Font getFont();

// Определение метрик текущего шрифта
public FontMetrics getFontMetrics();

// Определение метрик заданного шрифта
public abstract FontMetrics getFontMetrics(Font f);

// Установка цвета для рисования в контексте отображения
public abstract void setColor(Color c);

// Установка текущего шрифта в контексте отображения
public abstract void setFont(Font font);

// Установка режима рисования
public abstract void setPaintMode();

// Установка маски для рисования
public abstract void setXORMode(Color cl);

// Получение текстовой строки, представляющей
// данный контекст отображения
public String toString();

// Сдвиг начала системы координат
// в контексте отображения
public abstract void translate(int x, int y);
}

```

Рассмотрим назначение основных методов, сгруппировав их по выполняемым функциям.

Установка атрибутов контекста отображения

Изменяя атрибуты контекста отображения, приложение Java может установить цвет для рисования графических изображений, таких как линии и многоугольники, шрифт для рисования текста, режим рисования и маску. Возможен также сдвиг начала системы координат.

Выбор цвета

Изменение цвета, выбранного в контекст отображения, выполняется достаточно часто. В классе Graphics для изменения цвета определен метод setColor, прототип которого представлен ниже:

```
public abstract void setColor(Color c);
```

В качестве параметра методу setColor передается ссылка на объект класса Color, с помощью которого можно выбрать тот или иной цвет.

Как задается цвет?

Для этого можно использовать несколько способов.

Прежде всего, вам доступны статические объекты, определяющие фиксированный набор основных цветов:

Объект	Цвет
public final static Color black;	черный
public final static Color blue;	голубой

<code>public final static Color cyan;</code>	циан
<code>public final static Color darkGray;</code>	темно-серый
<code>public final static Color gray;</code>	серый
<code>public final static Color green;</code>	зеленый
<code>public final static Color lightGray;</code>	светло-серый
<code>public final static Color magenta;</code>	малиновый
<code>public final static Color orange;</code>	оранжевый
<code>public final static Color pink;</code>	розовый
<code>public final static Color red;</code>	красный
<code>public final static Color white;</code>	белый
<code>public final static Color yellow;</code>	желтый

Этим набором цветов пользоваться очень просто:

```
public void paint(Graphics g)
{
    // Выбираем в контекст отображения желтый цвет
    g.setColor(Color.yellow);
    g.drawString("Привет из апплета!", 10, 20);
    . . .
}
```

Здесь мы привели фрагмент исходного текста метода paint, в котором в контексте отображения устанавливается желтый цвет. После этого метод drawString выведет текстовую строку "Привет из апплета!" желтым цветом.

Если необходима более точная установка цвета, вы можете воспользоваться одним из трех конструкторов объекта Color:

```
public Color(float r, float g, float b);
public Color(int r, int g, int b);
public Color(int rgb);
```

Первые два конструктора позволяют задавать цвет в виде совокупности значений трех основных цветовых компонент - красной, желтой и голубой (соответственно, параметры r, g и b). Для первого конструктора диапазон возможных значений компонент цвета находится в диапазоне от 0.0 до 1.0, а для второго - в диапазоне от 0 до 255.

Третий конструктор также позволяет задавать отдельные компоненты цвета, однако они должны быть скомбинированы в одной переменной типа int. Голубая компонента занимает биты от 0 до 7, зеленая - от 8 до 15, красная - от 16 до 23.

Ниже мы привели пример выбора цвета с помощью конструктора, передав ему три целочисленных значения цветовых компонент:

```
g.setColor(new Color(0, 128, 128));
```

В классе Color определено еще несколько методов, которые могут оказаться вам полезными:

Метод	Описание
<code>public Color brighter();</code>	Установка более светлого варианта того же цвета
<code>public Color darker();</code>	Установка более темного варианта того же цвета
<code>public boolean equals(Object obj);</code>	Проверка равенства цветов текущего объекта и объекта, заданного параметром
<code>public int getBlue();</code>	Определение голубой компоненты цвета (в диапазоне от 0 до 255)
<code>public int getRed();</code>	Определение красной компоненты цвета (в диапазоне от 0 до 255)
<code>public int getGreen();</code>	Определение зеленой компоненты цвета (в диапазоне от 0 до 255)
<code>getHSBColor(float h, float s, float b);</code>	Определение компонент оттенка, насыщенности и яркости (схема HSB)
<code>public int getRGB();</code>	Определение компонент RGB для цвета, выбранного в контекст отображения
<code>public static int HSBtoRGB(float hue, float saturation, float brightness);</code>	Преобразование цветового представления из схемы HSB в схему RGB
<code>public static float[] RGBtoHSB(int r, int g, int b, float hsbvals[]);</code>	Преобразование, обратное выполняемому предыдущей функцией
<code>public String toString();</code>	Получение текстовой строки названия цвета

Второй способ установки цвето фона и изображения заключается в вызове методов setBackground и setForeground, например:

```
setBackground(Color.yellow);
```

```
setBackground(Color.black);
```

Здесь мы устанавливаем для окна апплета желтый цвет фона и черный цвет изображения.

Выбор шрифта

С помощью метода `setFont` из класса `Graphics` вы можете выбрать в контекст отображения шрифт, который будет использоваться методами `drawString`, `drawBytes` и `drawChars` для рисования текста. Вот прототип метода `setFont`:

```
public abstract void setFont(Font font);
```

В качестве параметра методу `setFont` следует передать объект класса `Font`:

```
public class java.awt.Font
    extends java.lang.Object
{
    // -----
    // Поля класса
    // -----
    protected String name;
    protected int size;
    protected int style;

    // Битовые маски стиля шрифта
    public final static int BOLD;
    public final static int ITALIC;
    public final static int PLAIN

    // -----
    // Конструктор
    // -----
    public Font(String name, int style, int size);

    // -----
    // Методы
    // -----

    // Сравнение шрифтов
    public boolean equals(Object obj);

    // Определение названия семейства шрифтов
    public String getFamily();

    // Получение шрифта по его характеристикам
    public static Font getFont(String nm);
    public static Font getFont(String nm, Font font);

    // Определение названия шрифта
    public String getName();

    // Определение размера шрифта
    public int getSize();

    // Определение стиля шрифта
    public int getStyle();

    // Получение хэш-кода шрифта
    public int hashCode();

    // Определение жирности шрифта
    public boolean isBold();

    // Проверка, является ли шрифт наклонным
    public boolean isItalic();

    // Проверка, есть ли шрифтовое выделение
    public boolean isPlain();

    // Плучение текстовой строки для объекта
    public String toString();
}
```

Создавая шрифт конструктором `Font`, вы должны указать имя, стиль и размер шрифта.

В качестве имени можно указать, например, строки `Arial` или `Courier`. Учтите, что в системе удаленного пользователя, загрузившего ваш апплет, может не найтись шрифта с указанным вами именем. В этом случае навигатор заменит его на наиболее подходящий (с его точки зрения).

Стиль шрифта задается масками `BOLD`, `ITALIC` и `PLAIN`, которые можно комбинировать при помощи логической операции "ИЛИ":

Маска	Описание
BOLD	Утолщенный шрифт
ITALIC	Наклонный шрифт
PLAIN	Шрифтовое выделение не используется

Что же касается размера шрифта, то он указывается в пикселах.

Установка режима рисования

Метод `setPaintMode` устанавливает в контексте отображения режим рисования, при котором выполняется замещение изображения текущим цветом, установленным в контексте отображения.

Прототип метода `setPaintMode` приведен ниже:

```
public abstract void setPaintMode();
```

Установка маски для рисования

Задавая маску для рисования при помощи метода `setXORMode`, вы можете выполнить при рисовании замещение текущего цвета на цвет, указанный в параметре метода, и наоборот, цвета, указанного в параметре метода, на текущий.

Все остальные цвета изменяются непредсказуемым образом, однако эта операция обратима, если вы нарисуете ту же самую фигуру два раза на одном и том же месте.

Прототип метода `setXORMode`:

```
public abstract void setXORMode(Color c1);
```

Сдвиг начала системы координат

Метод `translate` сдвигает начало системы координат в контексте отображения таким образом, что оно перемещается в точку с координатами (x, y), заданными через параметры метода:

```
public abstract void translate(int x, int y);
```

Определение атрибутов контекста отображения

Ряд методов класса `Graphics` позволяет определить различные атрибуты контекста отображения, например, цвет, выбранный в контекст отображения или метрики текущего шрифта, которым выполняется рисование текста.

Рассмотрим методы, позволяющие определить атрибуты контекста отображения.

Определение границ области ограничения вывода

С помощью метода `clipRect`, о котором мы расскажем чуть позже, вы можете определить в окне апплета область ограничения вывода прямоугольной формы. Вне этой области рисование графических изображений и текста не выполняется.

Метод `getClipRect` позволяет вам определить координаты текущей области ограничения, заданной в контексте отображения:

```
public abstract Rectangle getClipRect();
```

Метод возвращает ссылку на объект класса `Rectangle`, который, в частности, имеет поля класса с именами `x`, `y`, `height` и `width`. В этих полях находится, соответственно, координаты верхнего левого угла, высота и ширина прямоугольной области.

Определение цвета, выбранного в контекст отображения

Метод `getColor` возвращает ссылку на объект класса `Color`, представляющий текущий цвет, выбранный в контекст отображения:

```
public abstract Color getColor();
```

Определение шрифта, выбранного в контекст отображения

С помощью метода `getFont`, возвращающего ссылку на объект класса `Font`, вы можете определить текущий шрифт, выбранный в контекст отображения:

```
public abstract Font getFont();
```

Определение метрик текущего шрифта

Несмотря на то что вы можете заказать шрифт с заданным именем и размером, не следует надеяться, что навигатор выделит вам именно такой шрифт, какой вы попросите. Для правильного размещения текста и других изображений в окне апплета вам необходимо знать метрики реального шрифта, выбранного навигатором в контекст отображения.

Метрики текущего шрифта в контексте отображения вы можете узнать при помощи метода `getFontMetrics`, прототип которого приведен ниже:

```
public FontMetrics getFontMetrics();
```

Метод `getFontMetrics` возвращает ссылку на объект класса `FontMetrics`. Ниже мы привели список наиболее важных методов этого класса, предназначенных для получения отдельных параметров шрифта:

Метод	Описание
<code>public Font getFont();</code>	Определение шрифта, который описывается

	данной метрикой
<code>public int bytesWidth(byte data[], int off, int len);</code>	Метод возвращает ширину строки символов, расположенных в массиве байт data. Параметры off и len задают, соответственно, смещение начала строки в массиве и ее длину
<code>public int charsWidth(char data[], int off, int len);</code>	Метод возвращает ширину строки символов, расположенных в массиве символов data. Параметры off и len задают, соответственно, смещение начала строки в массиве и ее длину
<code>public int charWidth(char ch);</code>	Метод возвращает ширину заданного символа
<code>public int charWidth(int ch);</code>	Метод возвращает ширину заданной строки символов
<code>public int getAscent();</code>	Определение расстояния от базовой линии до верхней выступающей части символов
<code>public int getDescent();</code>	Определение расстояния от базовой линии до нижней выступающей части символов
<code>public int getLeading();</code>	Расстояние между строками текста
<code>public int getHeight();</code>	Определение полной высоты символов, выполняется по формуле: $getLeading() + getAscent() + getDescent()$
<code>public int getMaxAdvance();</code>	Максимальная ширина символов в шрифте
<code>public int getMaxAscent();</code>	Максимальное расстояние от базовой линии до верхней выступающей части символов для символов данного шрифта
<code>public int getMaxDescent();</code>	Максимальное расстояние от базовой линии до нижней выступающей части символов для символов данного шрифта
<code>public int[] getWidths();</code>	Массив ширин первых 256 символов в шрифте
<code>public int stringWidth(String str);</code>	Ширина строки, передаваемой методу в качестве параметра
<code>public String toString();</code>	Текстовая строка, которая представляет данную метрику шрифта

Обратите внимание на метод `stringWidth`, позволяющий определить ширину текстовой строки. Заметим, что без этого метода определение ширины текстовой строки было бы непростой задачей, особенно если шрифт имеет переменную ширину символов.

Для определения полной высоты строки символов вы можете воспользоваться методом `getHeight`.

Определение метрик заданного шрифта

Метод `getFontMetrics` с параметром типа `Font` позволяет определить метрики любого шрифта, передаваемого ему в качестве параметра:

```
public abstract FontMetrics getFontMetrics(Font f);
```

В отличие от нее метод `getFontMetrics` без параметров возвращает метрики текущего шрифта, выбранного в контекст отображения.

Рисование геометрических фигур

В этом разделе мы опишем методы класса `Graphics`, предназначенные для рисования элементарных геометрических фигур, таких как линии, прямоугольники, окружности и так далее.

Линии

Для того чтобы нарисовать прямую тонкую сплошную линию, вы можете воспользоваться методом `drawLine`, прототип которого приведен ниже:

```
public abstract void drawLine(int x1, int y1,  
int x2, int y2);
```

Концы линии имеют координаты (x_1, y_1) и (x_2, y_2) , как это показано на рис. 3.1.

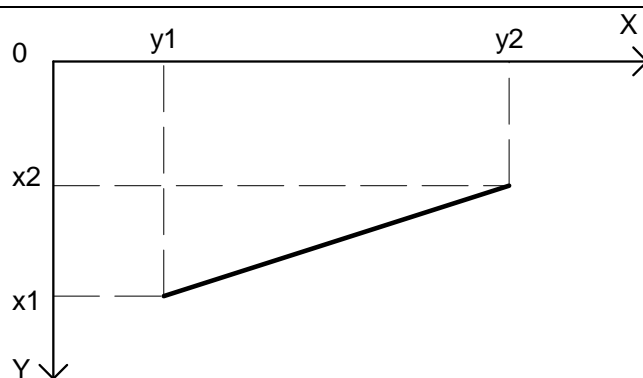


Рис. 3.1. Рисование прямой линии

К сожалению, в контексте отображения не предусмотрены никакие атрибуты, позволяющие нарисовать пунктирную линию или линию увеличенной толщины.

Прямоугольники и квадраты

Среди методов класса Graphics есть несколько, предназначенных для рисования прямоугольников. Первый из них, с именем drawRect, позволяет нарисовать прямоугольник, заданный координатами своего левого верхнего угла, шириной и высотой:

```
public void drawRect(int x, int y,
    int width, int height);
```

Параметры x и y задают, соответственно, координаты верхнего левого угла, а параметры width и height - высоту и ширину прямоугольника (рис. 3.2).

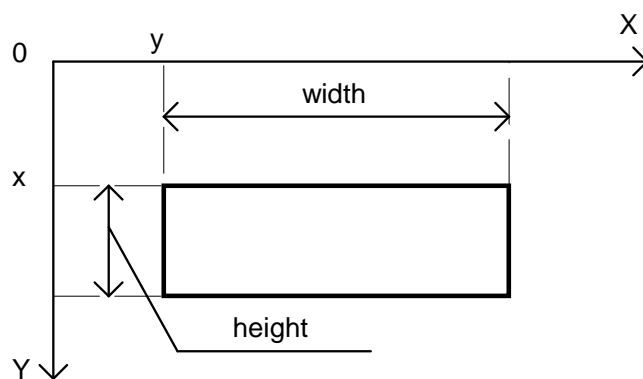


Рис. 3.2. Рисование прямоугольника

В отличие от метода drawRect, рисующего только прямоугольную рамку, метод fillRect рисует заполненный прямоугольник. Для рисования и заполнения прямоугольника используется цвет, выбранный в контекст отображения (рис. 3.3).

Прототип метода fillRect приведен ниже:

```
public abstract void
    fillRect(int x, int y, int width, int height);
```

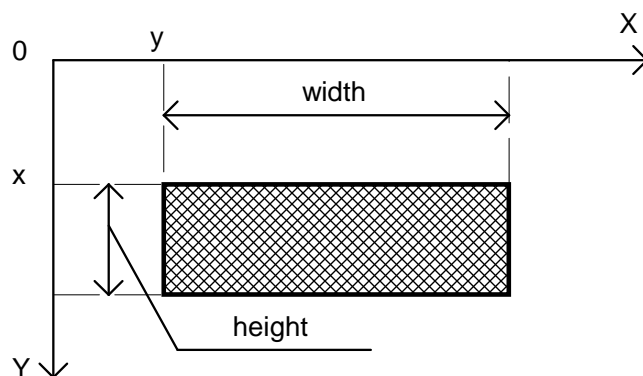


Рис. 3.3. Рисование заполненного прямоугольника

Метод drawRoundRect позволяет нарисовать прямоугольник с закругленными углами:

```
public abstract void
    drawRoundRect(int x, int y, int width,
        int height, int arcWidth, int arcHeight);
```

Параметры x и y определяют координаты верхнего левого угла прямоугольника, параметры $width$ и $height$ задают, соответственно его ширину и высоту.

Размеры эллипса, образующего закругления по углам, вы можете задать с помощью параметров $arcWidth$ и $arcHeight$. Первый из них задает ширину эллипса, а второй - высоту (рис. 3.4).

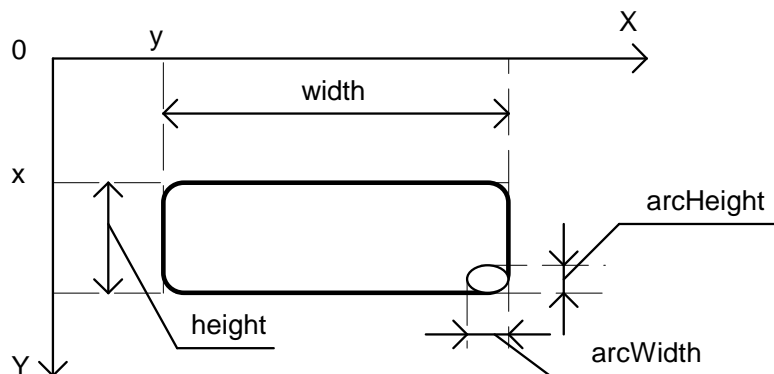


Рис. 3.4. Рисование прямоугольника с закругленными углами

Метод `fillRoundRect` позволяет нарисовать заполненный прямоугольник с закругленными углами (рис. 3.5). Назначение параметров этого метода аналогично назначению параметров только что рассмотренного метода `drawRoundRect`:

```
public abstract void
    fillRoundRect(int x, int y, int width, int height,
                  int arcWidth, int arcHeight);
```

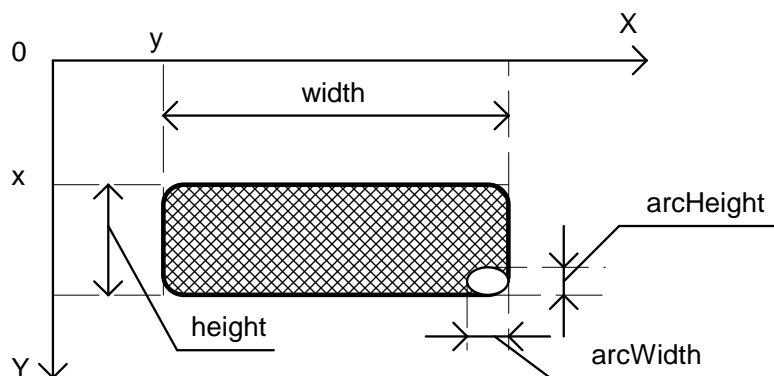


Рис. 3.5. Рисование заполненного прямоугольника с закругленными углами

Метод `fill3DRect` предназначен для рисования выступающего или западающего прямоугольника:

```
public void
    fill3DRect(int x, int y, int width,
               int height, boolean raised);
```

Если значение параметра `raised` равно `true`, рисуется выступающий прямоугольник, если `false` - западающий. Назначение остальных параметров аналогично назначению параметров метода `drawRect`.

Многоугольники

Для рисования многоугольников в классе `Graphics` предусмотрено четыре метода, два из которых рисуют незаполненные многоугольники, а два - заполненные.

Первый метод рисует незаполненный многоугольник, заданный массивами координат по осям X и Y :

```
public abstract void
    drawPolygon(int xPoints[], int yPoints[], int nPoints);
```

Через параметры `xPoints` и `yPoints` передаются, соответственно, ссылки на массивы координат по осям X и Y . Параметр `nPoints` задает количество точек в массивах.

На рис. 3.6 показан многоугольник, нарисованный методом `drawPolygon`.

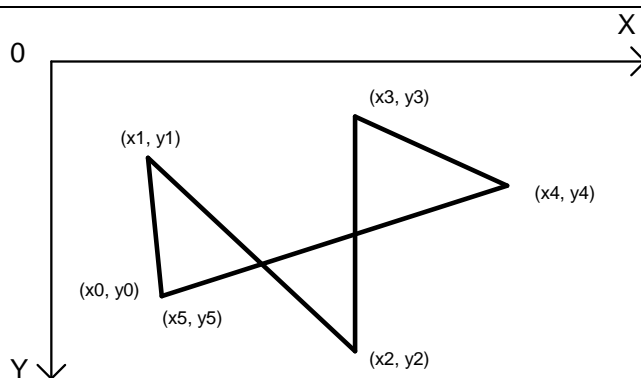


Рис. 3.6. Многоугольник, нарисованный методом drawPolygon

В этом многоугольнике шесть вершин с координатами от (x_0, y_0) до (x_5, y_5) , причем для того чтобы он стал замкнутым, координаты первой и последней вершины совпадают.

Второй метод также рисует незаполненный многоугольник, однако в качестве параметра методу передается ссылка на объект Polygon:

```
public void drawPolygon(Polygon p);
```

Класс Polygon достаточно прост, поэтому мы приведем его описание полностью:

```
public class java.awt.Polygon
    extends java.lang.Object
{
    // -----
    // Поля класса
    // -----
    public int npoints;    // количество вершин
    public int xpoints[];  // массив координат по оси X
    public int ypoints[];  // массив координат по оси Y

    // -----
    // Конструкторы
    // -----
    public Polygon();
    public Polygon(int xpoints[], int ypoints[], int npoints);

    // -----
    // Методы
    // -----

    // Добавление вершины
    public void addPoint(int x, int y);

    // Получение координат охватывающего прямоугольника
    public Rectangle getBoundingBox();

    // Проверка, находится ли точка внутри многоугольника
    public boolean inside(int x, int y);
}
```

Ниже мы показали фрагмент кода, в котором создается многоугольник, а затем в него добавляется несколько точек. Многоугольник рисуется методом drawPolygon:

```
Polygon p = new Polygon();
p.addPoint(270, 239);
p.addPoint(350, 230);
p.addPoint(360, 180);
p.addPoint(390, 160);
p.addPoint(340, 130);
p.addPoint(270, 239);
g.drawPolygon(p);
```

Если вам нужно нарисовать заполненный многоугольник (рис. 3.7), то для этого вы можете воспользоваться методами, приведенными ниже:

```
public abstract void
    fillPolygon(int xPoints[], int yPoints[], int nPoints);
public void fillPolygon(Polygon p);
```

Первый из этих методов рисует многоугольник, координаты вершин которого заданы в массивах, второй - получая объект класса Polygon в качестве параметра.

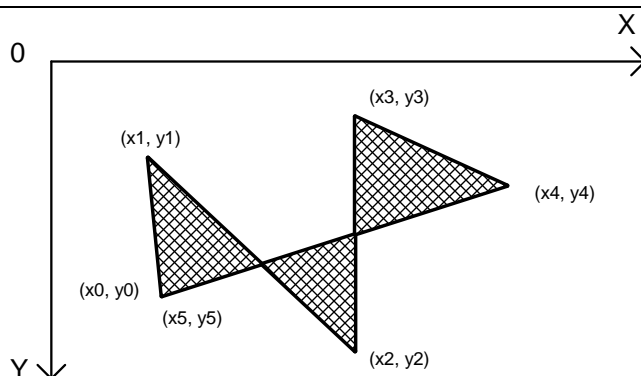


Рис. 3.6. Многоугольник, нарисованный методом *drawPolygon*

Овалы и круги

Для рисования окружностей и овалов вы можете воспользоваться методом *drawOval*:

```
public abstract void drawOval(int x, int y,
    int width, int height);
```

Параметры этого метода задают координаты и размеры прямоугольника, в который вписывается рисуемый овал (рис. 3.7).

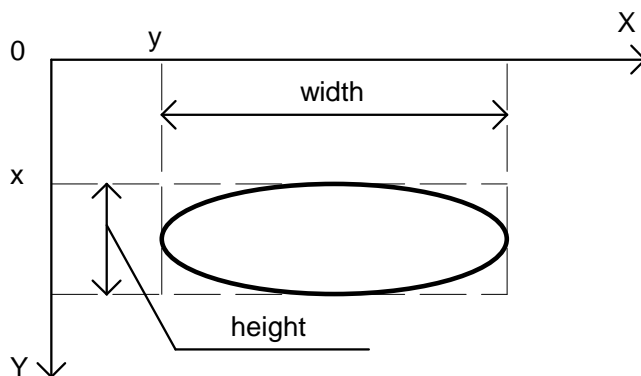


Рис. 3.7. Рисование овала

Метод *fillOval* предназначен для рисования заполненного овала (рис. 3.8). Назначение его параметров аналогично назначению параметров метода *drawOval*:

```
public abstract void
    fillOval(int x, int y, int width, int height);
```

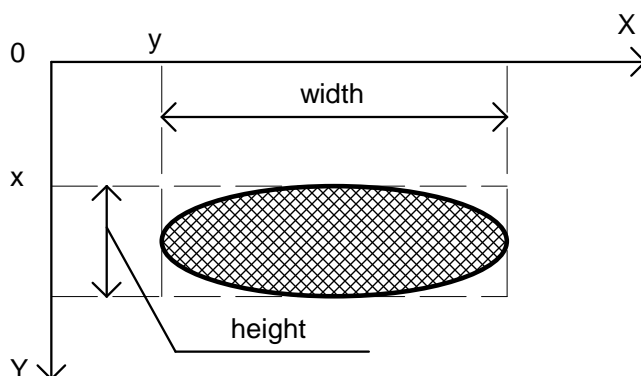


Рис. 3.7. Рисование заполненного овала

Сегменты

Метод *drawArc* предназначен для рисования незаполненного сегмента (рис. 3.8). Прототип этого метода приведен ниже:

```
public abstract void drawArc(int x, int y,
    int width, int height, int startAngle, int arcAngle);
```

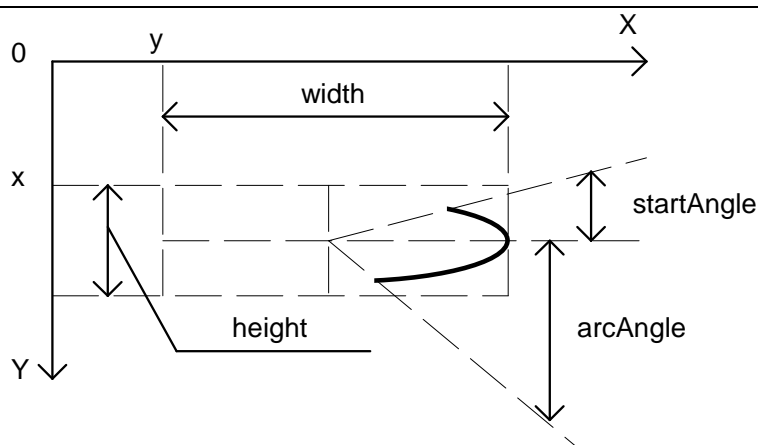


Рис. 3.8. Рисование незаполненного сегмента

Параметры *x*, *y*, *width* и *height* задают координаты прямоугольника, в который вписан сегмент.

Параметры *startAngle* и *arcAngle* задаются в градусах. Они определяют, соответственно, начальный угол и угол разворота сегмента.

Для того чтобы нарисовать заполненный сегмент, вы можете воспользоваться методом `fillArc`:

```
public abstract void
    fillArc(int x, int y, int width,
            int height, int startAngle, int arcAngle);
```

Задание области ограничения

Если для окна апплета задать область ограничения, то рисование будет возможно только в пределах этой области. Область ограничения задается методом `clipRect`, прототип которого мы привели ниже:

```
public abstract void
    clipRect(int x, int y, int width, int height);
```

Параметры *x*, *y*, *width* и *height* задают координаты прямоугольной области ограничения.

Копирование содержимого прямоугольной области

Метод `copyArea` позволяет скопировать содержимое любой прямоугольной области окна апплета:

```
public abstract void
    copyArea(int x, int y, int width,
             int height, int dx, int dy);
```

Параметры *x*, *y*, *width* и *height* задают координаты копируемой прямоугольной области. Область копируется в другую прямоугольную область такого же размера, причем параметры *dx* и *dy* определяют координаты последней.

Приложение Painter

В этом разделе мы рассмотрим исходные тексты апплета `Painter`, в которых демонстрируется использование большинства только что описанных нами функций рисования.

Внешний вид окна апплета при просмотре соответствующего документа HTML навигатором Microsoft Internet Explorer показано на рис. 3.9.

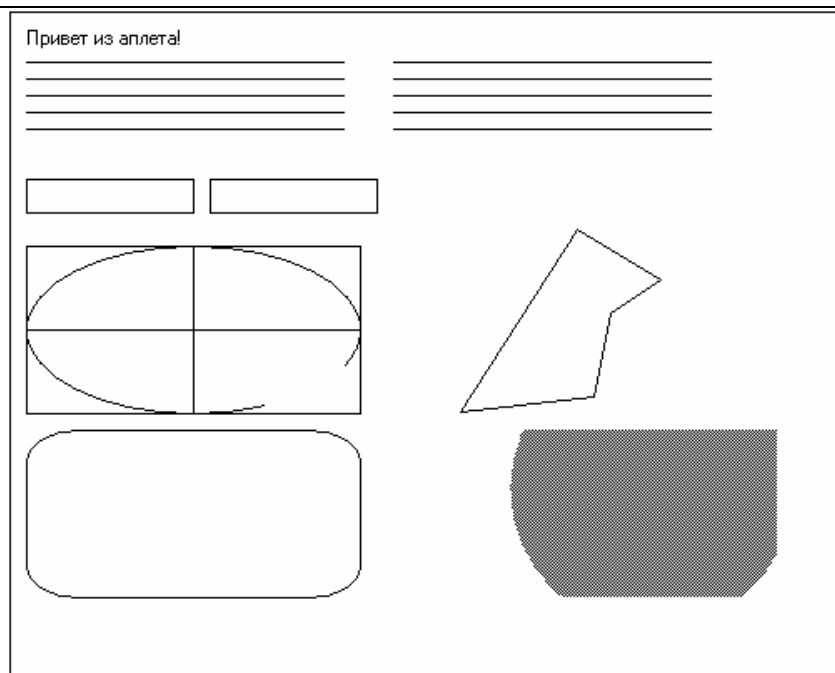


Рис. 3.9. Окно апплета Painter

Здесь мы написали текстовую строку, нарисовали несколько горизонтальных линий, скопировав это линии в другое место окна, а также изобразили несколько простейших геометрических фигур.

Исходные файлы приложения Painter

В листинге 3.1 мы привели исходный текст апплета Painter.

Листинг 3.1. Файл Painter\Painter.java

```
// =====
// Апплет, демонстрирующий использование различных
// функций рисования
//
// (C) Фролов А.В, 1997
//
// E-mail: frolov@glas.apc.org
// WWW:   http://www.glasnet.ru/~frolov
//        или
//        http://www.dials.ccas.ru/frolov
// =====

import java.applet.*;
import java.awt.*;

public class Painter extends Applet
{
    // -----
    // Painter
    // Конструктор не используется
    // -----
    public Painter()
    {
    }

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: Painter\r\n" +
            "Author: Alexandr Frolov\r\n" +
            "E-mail: frolov@glas.apc.org" +
            "WWW:   http://www.glasnet.ru/~frolov" +
            "Created with Microsoft Visual J++ Version 1.0";
    }

    // -----
    // init
```

```

// Вызывается во время инициализации апплета
// -----
public void init()
{
    // Для того чтобы размеры окна апплета можно было
    // задавать в документе HTML, закрываем следующую
    // строку символом комментария

    // resize(320, 240);
}

// -----
// destroy
// Метод destroy не используется
// -----
public void destroy()
{
}

// -----
// paint
// Метод paint, выполняющий рисование в окне апплета
// -----
public void paint(Graphics g)
{
    // Определяем текущие размеры окна апплета
    Dimension dimAppWndDimension = size();

    // Эта строка будет удалена из окна апплета
    // методом clearRect
    g.drawString("Невидимая строка", 10, 20);

    // Стираем содержимое окна апплета. Цвет окна
    // становится таким же, как и цвет фона
    // окна навигатора
    g.clearRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);

    // Выбираем в контекст отображения желтый цвет
    g.setColor(Color.yellow);

    // Закрашиваем внутреннюю область окна апплета
    g.fillRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);

    // Выбираем в контекст отображения черный цвет
    g.setColor(Color.black);

    // Рисуем рамку вокруг окна апплета
    g.drawRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);

    // Пишем строку в верхнем левом углу окна апплета
    g.drawString("Привет из апплета!", 10, 20);

    // Рисуем в цикле пять горизонтальных линий
    for(int i = 0; i < 5; i++)
    {
        g.drawLine(10, 30 + (i * 10), 200, 30 + (i * 10));
    }

    // Копируем область окна, занятую
    // нарисованными линиями
    g.copyArea(10, 30, 200, 50, 220, 0);

    // Выбираем в контекст отображения белый цвет
    g.setColor(Color.white);

    // Закрашиваем белым цветом нижнюю часть окна
    g.fillRect(1, 80,
        dimAppWndDimension.width - 2,
        dimAppWndDimension.height - 81);

    // Выбираем в контекст отображения черный цвет

```

```

g.setColor(Color.black);

// Рисуем два трехмерных прямоугольника
g.draw3DRect(10, 100, 100, 20, true);
g.draw3DRect(120, 100, 100, 20, false);

// Выбираем в контекст отображения красный цвет
g.setColor(Color.red);

// Рисуем рамку, в которую будет вписан сегмент
g.drawRect(10, 140, 200, 100);
g.drawLine(10, 190, 210, 190);
g.drawLine(110, 140, 110, 240);

// Выбираем в контекст отображения черный цвет
g.setColor(Color.black);

// Рисуем сегмент
g.drawArc(10, 140, 200, 100, -25, 320);

// Создаем многоугольник
Polygon p = new Polygon();

// Добавляем в многоугольник несколько вершин
p.addPoint(270, 239);
p.addPoint(350, 230);
p.addPoint(360, 180);
p.addPoint(390, 160);
p.addPoint(340, 130);
p.addPoint(270, 239);

// Рисуем многоугольник
g.drawPolygon(p);

// Рисуем прямоугольник с закругленными углами
g.drawRoundRect(10, 250, 200, 100, 60, 40);

// Выбираем в контекст отображения красный цвет
g.setColor(Color.red);

// Задаем область ограничения вывода
g.clipRect(260, 250, 200, 100);

// Рисуем круг, пересекающий область ограничения
g.fillOval(300, 200, 170, 170);
}

// -----
// start
// Метод start не используется
// -----
public void start()
{
}

// -----
// stop
// Метод stop не используется
// -----
public void stop()
{
}
}

```

Листинг 3.2 содержит исходный текст документа HTML, в который встроен данный апплет.

Листинг 3.2. Файл Painter\Painter.html

```

<html>
<head>
<title>Painter</title>
</head>
<body>
<hr>
<applet
  code=Painter.class
  id=Painter
  width=500

```

```

        height=400>
</applet>
<hr>
<a href="Painter.java">The source.</a>
</body>
</html>

```

Мы внесли небольшие изменения в файл Painter.html, подготовленный системой Java Applet Wizard. Эти изменения заключаются в увеличении размеров окна апплета, задаваемых параметрами WIDTH и HEIGHT оператора <APPLET>.

Метод init

Апплет Painter был создан с помощью системы Java Applet Wizard. Мы выполнили изменения методов init, getAppletInfo и paint.

Изменения метода init заключаются в том, что мы закрыли символом комментария строку установки размеров окна апплета:

```

public void init()
{
    // resize(320, 240);
}

```

Это позволяет задавать размеры окна апплета не в исходном тексте приложения, а в параметрах оператора <APPLET>, с помощью которого апплет встраивается в документ HTML.

Метод getAppletInfo

В методе getAppletInfo изменения невелики: в описание апплета был добавлен наш почтовый адрес E-mail и адрес нашего сервера WWW:

```

public String getAppletInfo()
{
    return "Name: Painter\r\n" +
        "Author: Alexandr Frolov\r\n" +
        "E-mail: frolov@glas.apc.org" +
        "WWW:      http://www.glasnet.ru/~frolov" +
        "Created with Microsoft Visual J++ Version 1.0";
}

```

Метод paint

В обработчик метода paint мы добавили функции рисования.

В самом начале своей работы метод определяет текущие размеры окна апплета, вызывая для этого метод size:

```

Dimension dimAppWndDimension = size();

```

Метод size определен в классе Component, от которого в конечном счете наследуется класс Applet и класс нашего приложения Painter. Этот метод возвращает ссылку на объект класса Dimension, хранящего высоту и ширину объекта:

```

public class java.awt.Dimension
    extends java.lang.Object
{
    // -----
    // Поля класса
    // -----
    public int height; // высота
    public int width;  // ширина

    // -----
    // Конструкторы
    // -----
    public Dimension();
    public Dimension(Dimension d);
    public Dimension(int width, int height);

    // -----
    // Метод
    // -----
    public String toString();
}

```

На следующем шаге после определения размеров окна наше приложение рисует в окне строку, а затем стирает содержимое всего окна:

```

g.drawString("Невидимая строка", 10, 20);
g.clearRect(0, 0,
    dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);

```

В качестве начальных координат стираемой области мы указали точку (0, 0) - это верхний левый угол окна апплета. Ширина и высота стираемой области задана исходя из размеров апплета, полученных от метода size.

Для того чтобы изменить цвет фона окна, мы его закрашиваем (хотя могли бы воспользоваться и методом setBackground). Это можно сделать методом fillRect. Вначале при помощи метода setColor мы выбираем в контекст отображения желтый цвет, а затем закрашиваем всю внутреннюю область окна апплета методом fillRect:

```
g.setColor(Color.yellow);
g.fillRect(0, 0,
    dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);
```

На следующем этапе метод paint выбирает в контекст отображения черный цвет и рисует черную рамку вокруг окна апплета, вызывая для этого метод drawRect:

```
g.setColor(Color.black);
g.drawRect(0, 0,
    dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);
```

Далее при помощи метода drawString обработчик метода paint пишет в верхней части окна апплета строку, которая останется на экране:

```
g.drawString("Привет из апплета!", 10, 20);
```

Для того чтобы продемонстрировать работу функции copyArea, копирующей прямоугольную область окна апплета, мы нарисовали в окне пять горизонтальных прямых линий, а затем скопировали результат в правый верхний угол окна апплета:

```
for(int i = 0; i < 5; i++)
{
    g.drawLine(10, 30 + (i * 10), 200, 30 + (i * 10));
}
g.copyArea(10, 30, 200, 50, 220, 0);
```

Для рисования линий мы здесь вызываем метод drawLine.

Далее вызывая метод fillRect после предварительного выбора в контекст отображения белого цвета мы закрашиваем белым цветом всю нижнюю часть окна апплета, подготавливая фон для дальнейших упражнений в рисовании:

```
g.setColor(Color.white);
g.fillRect(1, 80,
    dimAppWndDimension.width - 2,
    dimAppWndDimension.height - 81);
```

Ширина и высота закрашиваемой области здесь указана с учетом наличия черной рамки толщиной в 1 пиксел вокруг окна апплета.

В верхней части полученной таким образом области белого цвета приложение рисует два прямоугольника черного цвета с трехмерным выделением:

```
g.setColor(Color.black);
g.draw3DRect(10, 100, 100, 20, true);
g.draw3DRect(120, 100, 100, 20, false);
```

Затем мы приступаем к рисованию сегмента.

Вначале в контекст отображения выбирается красный цвет. Этим цветом мы рисуем прямоугольную рамку, в которую будет вписан сегмент:

```
g.setColor(Color.red);
g.drawRect(10, 140, 200, 100);
g.drawLine(10, 190, 210, 190);
g.drawLine(110, 140, 110, 240);
```

Рамка разделена по горизонтали и по вертикали красными линиями.

Далее мы выбираем в контекст отображения черный цвет и рисуем сегмент:

```
g.setColor(Color.black);
g.drawArc(10, 140, 200, 100, -25, 320);
```

Обратите внимание, что начальный угол сегмента имеет отрицательное значение. Угол сегмента составляет 320 градусов.

Следующий шаг - создание и рисование многоугольника.

Многоугольник создается как объект класса Polygon. К этому объекту с помощью метода addPoint мы добавляем несколько точек:

```
Polygon p = new Polygon();
p.addPoint(270, 239);
p.addPoint(350, 230);
p.addPoint(360, 180);
p.addPoint(390, 160);
p.addPoint(340, 130);
p.addPoint(270, 239);
```

Для того чтобы многоугольник был образован замкнутой ломаной линией, первая и последняя точки имеют одинаковые координаты.

После подготовки многоугольника он рисуется при помощи метода drawPolygon:

```
g.drawPolygon(p);
```

Наш апплет рисует также прямоугольник с закругленными углами, вызывая метод drawRoundRect:

```
g.drawRoundRect(10, 250, 200, 100, 60, 40);
```

Ширина и высота эллипсов закругления составляет, соответственно, 60 и 40 пикселей.

Для демонстрации действия области ограничения вывода мы создаем такую область, вызывая метод clipRect:

```
g.clipRect(260, 250, 200, 100);
```

Затем мы рисуем круг, пересекающий эту область ограничения, в результате чего будет нарисована только та часть круга, которая находится внутри области ограничения:

```
g.fillOval(300, 200, 170, 170);
```

Приложение FontList

Наше следующее приложение отображает список шрифтов, доступных в системе. Окно апплета этого приложения, запущенного в среде Microsoft Internet Explorer, работающего в Microsoft Windows NT версии 4.0, показано на рис. 3.10.

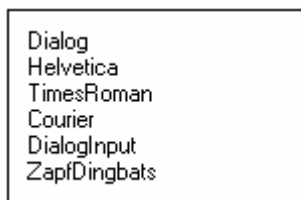


Рис. 3.10. Окно апплета со списком доступных шрифтов

Возможно, этот список покажется вам несколько необычным. В самом деле, давайте заглянем в папку Fonts, которую можно найти в папке Control Panel. Беглого взгляда достаточно для того, чтобы убедиться - список шрифтов, доступных апплету, не совпадает со списком шрифтов, установленных в системе (рис. 3.11).



Рис. 3.11. Список шрифтов, установленных в системе Microsoft Windows NT

Задавая имена шрифтов в конструкторе класса Font, вы должны использовать имена шрифтов, доступные апплету, а не имена шрифтов, установленных в системе. Наш апплет FontList извлекает и отображает список доступных для него шрифтов.

Исходный текст приложения

Исходный текст приложения представлен в листинге 3.3.

Листинг 3.3. Файл FontList\FontList.java

```
// =====  
// Просмотр списка доступных шрифтов
```

```

//
// (C) Фролов А.В, 1997
//
// E-mail: frolov@glas.apc.org
// WWW:   http://www.glasnet.ru/~frolov
//        или
//        http://www.dials.ccas.ru/frolov
// =====

import java.applet.*;
import java.awt.*;

public class FontList extends Applet
{
    // -----
    // Поля класса
    // -----
    Toolkit toolkit; // ссылка на Toolkit
    String fntlist[]; // список шрифтов
    int yStart = 20; // координата Y начала области вывода
    int yStep; // шаг вывода строк с названиями шрифтов

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: FontList\r\n" +
            "Author: Alexandr Frolov\r\n" +
            "E-mail: frolov@glas.apc.org" +
            "WWW:   http://www.glasnet.ru/~frolov" +
            "Created with Microsoft Visual J++ Version 1.0";
    }

    // -----
    // init
    // Вызывается во время инициализации апплета
    // -----
    public void init()
    {
        // Получаем ссылку на Toolkit
        toolkit = Toolkit.getDefaultToolkit();

        // Получаем список доступных шрифтов
        fntlist = toolkit.getFontList();
    }

    // -----
    // paint
    // Метод paint, выполняющий рисование в окне апплета
    // -----
    public void paint(Graphics g)
    {
        // Определяем метрики шрифта
        FontMetrics fm = g.getFontMetrics();

        // Устанавливаем шаг вывода строк по вертикали
        // равным полной высоте символов текущего шрифта
        yStep = fm.getHeight();

        // Устанавливаем новую высоту апплета исходя
        // из количества элементов в списке шрифтов
        resize(150, 20 + yStep * fntlist.length);

        // Определяем текущие размеры окна апплета
        Dimension dimAppWndDimension = size();

        // Выбираем в контекст отображения желтый цвет
        g.setColor(Color.yellow);

        // Закрашиваем внутреннюю область окна апплета
        g.fillRect(0, 0,
            dimAppWndDimension.width - 1,
            dimAppWndDimension.height - 1);

        // Выбираем в контекст отображения черный цвет

```

```

g.setColor(Color.black);

// Рисуем рамку вокруг окна апплета
g.drawRect(0, 0,
    dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);

// Выводим в цикле список установленных шрифтов
for(int i = 0; i < fntlist.length; i++)
{
    g.drawString(fntlist[i], 10, yStart + yStep * i);
}
}
}

```

В листинге 3.4 вы найдете исходный текст документа HTML, в который встроен наш апплет.

Листинг 3.4. Файл FontList\FontList.html

```

<html>
<head>
<title>FontList</title>
</head>
<body>
<hr>
<applet
    code=FontList.class
    id=FontList
    width=320
    height=240 >
</applet>
<hr>
<a href="FontList.java">The source.</a>
</body>
</html>

```

Описание исходного текста

Наиболее важными для нас являются методы init и paint.

Метод init

Процедура извлечения списка доступных шрифтов, использованная в нашем апплете, достаточно проста и выполняется в методе init, как это показано ниже:

```

Toolkit toolkit; // ссылка на Toolkit
String fntlist[]; // список шрифтов
. . .
public void init()
{
    toolkit = Toolkit.getDefaultToolkit();
    fntlist = toolkit.getFontList();
}

```

Апплет вызывает статический метод getDefaultToolkit из класса Toolkit и затем, пользуясь полученной ссылкой, извлекает список шрифтов, записывая его в массив fntlist.

Для чего еще можно использовать класс Toolkit?

Класс Toolkit является абстрактным суперклассом для всех реализаций AWT. Порожденные от него классы используются для привязки различных компонент конкретных реализаций.

Создавая свои апплеты, вы будете редко прибегать к услугам этого класса. Однако в нем есть несколько полезных методов, прототипы которых мы перечислим ниже:

```

public abstract class java.awt.Toolkit
    extends java.lang.Object
{
    // -----
    // Конструктор
    // -----
    public Toolkit();

    // -----
    // Методы (сокращенный список)
    // -----
    . . .

    // Получение ссылки на Toolkit
    public static Toolkit getDefaultToolkit();

    // Определение текущей цветовой модели,
    // выбранной в контекст отображения
    public abstract ColorModel getColorModel();
}

```



```
// Получение списка шрифтов, доступных апплету
public abstract String[] getFontList();

// Получение метрик заданного шрифта
public abstract FontMetrics getFontMetrics(Font font);

// Получение растрового изображения по имени файла
public abstract Image getImage(String filename);

// Получение растрового изображения по адресу URL
public abstract Image getImage(URL url);

// Определение разрешения экрана в точках на дюйм
public abstract int getScreenResolution();

// Размеры экрана в пикселах
public abstract Dimension getScreenSize();

// Подготовка растрового изображения для вывода
public abstract boolean
    prepareImage(Image image, int width, int height,
        ImageObserver observer);

// Синхронизация состояния Toolkit
public abstract void sync();
}
```

Наиболее интересны, с нашей точки зрения, методы `getFontList`, `getScreenResolution` и `getScreenSize`, с помощью которых апплет может, соответственно, получить список шрифтов, определить разрешение и размер экрана. Последние два параметра позволяют сформировать содержимое окна апплета оптимальным образом исходя из объема информации, который может в нем разместиться.

Метод `paint`

В методе `paint` прежде всего мы определяем полную высоту символов шрифта, которая будет использована при выводе строк. Высота шрифта определяется следующим образом:

```
FontMetrics fm = g.getFontMetrics();
yStep = fm.getHeight();
```

Зная высоту шрифта и количество элементов в списке доступных шрифтов, мы можем изменить размер окна апплета по вертикали таким образом, чтобы в нем поместились все строки. Количество элементов в массиве `fontlist` равно `fontlist.length`, а полную высоту шрифта мы только что определили. Для изменения высоты окна апплета мы используем метод `resize`:

```
resize(150, 20 + yStep * fontlist.length);
```

Далее мы определяем новые размеры окна апплета, закрашиваем фон окна желтым цветом и обводим окно тонкой рамкой черного цвета:

```
Dimension dimAppWndDimension = size();
g.setColor(Color.yellow);
g.fillRect(0, 0,
    dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);
g.setColor(Color.black);
g.drawRect(0, 0,
    dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);
```

Эту операцию мы уже выполняли в предыдущем приложении.

Список установленных шрифтов выводится достаточно просто в цикле:

```
for(int i = 0; i < fontlist.length; i++)
{
    g.drawString(fontlist[i], 10, yStart + yStep * i);
}
```

Здесь содержимое параметра цикла (переменной `i`) меняется от 0 до количества элементов в массиве `length`. Каждая новая строка рисуется со смещением, равным полной высоте символов текущего шрифта, выбранного в контекст отображения.

Приложение `TextOut`

До сих пор наши апплеты не получали параметров из документов HTML, в которые мы их встраивали. Конечно, все константы, текстовые строки, адреса URL и другую информацию можно закодировать непосредственно в исходном тексте апплета, однако, очевидно, это очень неудобно.

Пользуясь операторами `<PARAM>`, расположенными в документе HTML сразу после оператора `<APPLET>`, можно передать апплету произвольное количество параметров, например, в виде текстовых строк:

```
<applet
```

```

code=TextOut.class
id=TextOut
width=320
height=240 >
<param name=ParamName1 value="Param Value 1">
<param name=ParamName2 value="Param Value 2">
<param name=ParamName3 value="Param Value 3">
<param name=ParamName4 value="Param Value 4">
</applet>

```

Здесь через параметр NAME оператора <PARAM> передается имя параметра апплета, а через параметр VALUE - значение соответствующего параметра.

Как параметр может получить значение параметров?

Для получения значения любого параметра апплет должен использовать метод `getParameter`. В качестве единственного параметра этому методу передается имя параметра апплета в виде строки типа `String`, например:

```

private String m_ParamName1;
private final String PARAM_ ParamName1= "ParamName1";
String param;

```

```

param = getParameter(PARAM_ParamName1);
if (param != null)
    m_ParamName1 = param;

```

Если вы создаете апплет с помощью системы Java Applet Wizard, то в четвертой диалоговой панели вам предоставляется возможность определить все параметры, передаваемые апплету (рис. 3.12).

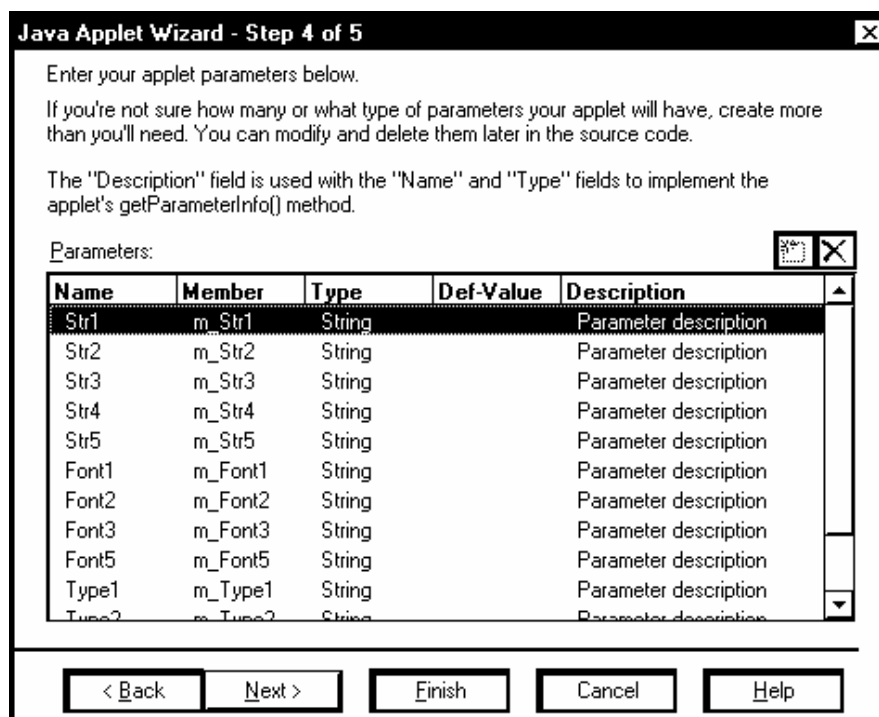


Рис. 3.12. Определение параметров апплета (список параметров уже заполнен)

Первоначально список параметров, отображаемых в четвертой диалоговой панели системы Java Applet Wizard, пуст. Такую панель мы показывали раньше на рис. 2.11.

Для добавления нового параметра сделайте щелчок левой клавишей мыши в столбце Name по свободному полю, отмеченному пунктирным прямоугольником. На месте этого прямоугольника появится поле редактирования, в котором вы должны ввести имя параметра. После ввода сделайте щелчок вне поля, после чего в списке параметров появится новая строка.

Создавая проект `TextOut`, мы выполнили эту операцию для всех параметров, за исключением параметра `Font4`. Этот параметр мы добавили позже в ручном режиме, когда все файлы проекта уже были созданы.

Обратите внимание, что в столбце Member при заполнении списка автоматически появляются имена полей класса, в которые попадут значения параметров.

После завершения формирования списка параметров мы заполнили столбцы Def-Value и Description (рис. 3.13).

Name	Member	Type	Def-Value	Description
Str3	m_Str3	String	Hello 3	Text string to write
Str4	m_Str4	String	Hello 4	Text string to write
Str5	m_Str5	String	Hello 5	Text string to write
Font1	m_Font1	String	Arial	Text font
Font2	m_Font2	String	Courier	Text font
Font3	m_Font3	String	Times	Text font
Font5	m_Font5	String	Undefined	Text font
Type1	m_Type1	String	Bold	Font type
Type2	m_Type2	String	Italic	Font type
Type3	m_Type3	String	Plain	Font type

Рис. 3.13. Заполнение столбцов Def-Value и Description

Значения из столбца Def-Value будут использованы для инициализации соответствующих полей класса. Что же касается столбца описаний Description, о эта информация может быть извлечена апплетом и проанализирована. Если в документе HTML находится несколько апплетов (что вполне допустимо), другие апплеты также могут получить описание параметров нашего апплета.

Какие параметры получает наш апплет и что он делает, кроме получения значения параметров?

Через параметры с именами Str1 - Str5 передается пять строк, который апплет отображает в своем окне (рис. 3.14).

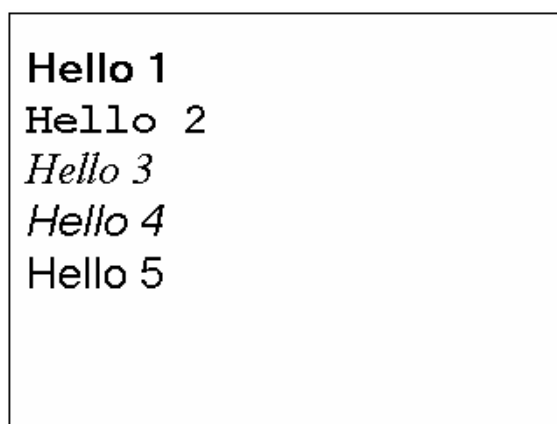


Рис. 3.14. Отображение строк в окне апплета TextOut

Параметры Font1 - Font5 задают имена шрифтов для отображения этих строк. С помощью параметра Type1 можно задать стиль шрифта первой и второй строки, с помощью параметра Type2 - третьей и четвертой, а с помощью параметра Type3 - стиль шрифта для пятой строки.

Рассмотрим исходный текст приложения TextOut.

Исходные тексты приложения TextOut

Файл исходного текста приложения TextOut представлен в листинге 3.5.

Листинг 3.5. Файл TextOut\TextOut.java

```
// =====
// Установка различных шрифтов.
// Демонстрация способов передачи параметров в апплет
//
// (C) Фролов А.В., 1997
//
// E-mail: frolov@glas.apc.org
// WWW:   http://www.glasnet.ru/~frolov
//        или
//        http://www.dials.ccas.ru/frolov
// =====

import java.applet.*;
import java.awt.*;

public class TextOut extends Applet
{
    // -----
    // Поля класса.
    // Создаются автоматически для всех параметров апплета
    // -----
    private String m_Str1 = "Hello 1";
```

```

private String m_Str2 = "Hello 2";
private String m_Str3 = "Hello 3";
private String m_Str4 = "Hello 4";
private String m_Str5 = "Hello 5";
private String m_Font1 = "Arial";
private String m_Font2 = "Courier";
private String m_Font3 = "Times";
private String m_Font4 = "Helvetica";
private String m_Font5 = "Undefined";
private String m_Type1 = "Bold";
private String m_Type2 = "Italic";
private String m_Type3 = "Plain";

// -----
// Имена параметров
// -----
private final String PARAM_Str1 = "Str1";
private final String PARAM_Str2 = "Str2";
private final String PARAM_Str3 = "Str3";
private final String PARAM_Str4 = "Str4";
private final String PARAM_Str5 = "Str5";
private final String PARAM_Font1 = "Font1";
private final String PARAM_Font2 = "Font2";
private final String PARAM_Font3 = "Font3";
private final String PARAM_Font4 = "Font4";
private final String PARAM_Font5 = "Font5";
private final String PARAM_Type1 = "Type1";
private final String PARAM_Type2 = "Type2";
private final String PARAM_Type3 = "Type3";

// -----
// getAppletInfo
// Метод, возвращающей строку информации об апплете
// -----
public String getAppletInfo()
{
    return "Name: TextOut\r\n" +
        "Author: Alexandr Frolov\r\n" +
        "E-mail: frolov@glas.apc.org" +
        "WWW:      http://www.glasnet.ru/~frolov" +
        "Created with Microsoft Visual J++ Version 1.0";
}

// -----
// getParameterInfo
// Метод, возвращающий описание параметров
// -----
public String[][] getParameterInfo()
{
    String[][] info =
    {
        { PARAM_Str1, "String", "Text string to write" },
        { PARAM_Str2, "String", "Text string to write" },
        { PARAM_Str3, "String", "Text string to write" },
        { PARAM_Str4, "String", "Text string to write" },
        { PARAM_Str5, "String", "Text string to write" },
        { PARAM_Font1, "String", "Text font" },
        { PARAM_Font2, "String", "Text font" },
        { PARAM_Font3, "String", "Text font" },
        { PARAM_Font4, "String", "Text font" },
        { PARAM_Font5, "String", "Text font" },
        { PARAM_Type1, "String", "Font type" },
        { PARAM_Type2, "String", "Font type" },
        { PARAM_Type3, "String", "Font type" },
    };
    return info;
}

// -----
// init
// Вызывается во время инициализации апплета
// -----
public void init()
{
    // Рабочая переменная для получения параметров
    String param;

```

```

// Получение параметров и сохранение
// их значений в полях класса

// Строки, которые будут выведены в окно апплета
param = getParameter(PARAM_Str1);
if (param != null)
    m_Str1 = param;

param = getParameter(PARAM_Str2);
if (param != null)
    m_Str2 = param;

param = getParameter(PARAM_Str3);
if (param != null)
    m_Str3 = param;

param = getParameter(PARAM_Str4);
if (param != null)
    m_Str4 = param;

param = getParameter(PARAM_Str5);
if (param != null)
    m_Str5 = param;

// Шрифты для отображения строк
param = getParameter(PARAM_Font1);
if (param != null)
    m_Font1 = param;

param = getParameter(PARAM_Font2);
if (param != null)
    m_Font2 = param;

param = getParameter(PARAM_Font3);
if (param != null)
    m_Font3 = param;

param = getParameter(PARAM_Font4);
if (param != null)
    m_Font4 = param;

param = getParameter(PARAM_Font5);
if (param != null)
    m_Font5 = param;

// Начертание шрифтов
param = getParameter(PARAM_Type1);
if (param != null)
    m_Type1 = param;

param = getParameter(PARAM_Type2);
if (param != null)
    m_Type2 = param;

param = getParameter(PARAM_Type3);
if (param != null)
    m_Type3 = param;
}

// -----
// paint
// Метод paint, выполняющий рисование в окне апплета
// -----
public void paint(Graphics g)
{
    // Начальная координата для вывода по вертикали
    int yStart = 20;

    // Текущая координата для вывода строки
    int yCurrent = 20;

    // Определяем текущие размеры окна апплета
    Dimension dimAppWndDimension = size();

    // Выбираем в контекст отображения желтый цвет
    g.setColor(Color.yellow);

```

```

// Закрашиваем внутреннюю область окна апплета
g.fillRect(0, 0,
    dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);

// Выбираем в контекст отображения черный цвет
g.setColor(Color.black);

// Рисуем рамку вокруг окна апплета
g.drawRect(0, 0,
    dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);

// Получаем стиль шрифта и выбираем шрифт
// в соответствии с этим стилем
if(m_Type1.equals("Bold"))
    g.setFont(new Font(m_Font1, Font.BOLD, 25));

else if(m_Type1.equals("Italic"))
    g.setFont(new Font(m_Font1, Font.ITALIC, 25));

else if(m_Type1.equals("Plain"))
    g.setFont(new Font(m_Font1, Font.PLAIN, 25));

// Отступ для первой строки
yCurrent += yStart;

// Рисуем первую строку
g.drawString(m_Str1, 10, yCurrent);

// Определяем метрики шрифта
FontMetrics fm = g.getFontMetrics();

// Устанавливаем новую текущую позицию для
// вывода очередной строки
yCurrent += fm.getHeight();

// Выбираем шрифт в контекст отображения
if(m_Type1.equals("Bold"))
    g.setFont(new Font(m_Font2, Font.BOLD, 25));

else if(m_Type1.equals("Italic"))
    g.setFont(new Font(m_Font2, Font.ITALIC, 25));

else if(m_Type1.equals("Plain"))
    g.setFont(new Font(m_Font2, Font.PLAIN, 25));

// Рисуем вторую строку
g.drawString(m_Str2, 10, yCurrent);

// Устанавливаем новую текущую позицию для
// вывода очередной строки
fm = g.getFontMetrics();
yCurrent += fm.getHeight();

// Выбираем шрифт в контекст отображения
if(m_Type2.equals("Bold"))
    g.setFont(new Font(m_Font3, Font.BOLD, 25));

else if(m_Type2.equals("Italic"))
    g.setFont(new Font(m_Font3, Font.ITALIC, 25));

else if(m_Type2.equals("Plain"))
    g.setFont(new Font(m_Font3, Font.PLAIN, 25));

// Рисуем третью строку
g.drawString(m_Str3, 10, yCurrent);

// Устанавливаем новую текущую позицию для
// вывода очередной строки
fm = g.getFontMetrics();
yCurrent += fm.getHeight();

// Выбираем шрифт в контекст отображения
if(m_Type2.equals("Bold"))
    g.setFont(new Font(m_Font4, Font.BOLD, 25));

```

```

else if(m_Type2.equals("Italic"))
    g.setFont(new Font(m_Font4, Font.ITALIC, 25));

else if(m_Type2.equals("Plain"))
    g.setFont(new Font(m_Font4, Font.PLAIN, 25));

// Рисуем четвертую строку
g.drawString(m_Str4, 10, yCurrent);

// Устанавливаем новую текущую позицию для
// вывода очередной строки
fm = g.getFontMetrics();
yCurrent += fm.getHeight();

// Выбираем шрифт в контекст отображения
if(m_Type3.equals("Bold"))
    g.setFont(new Font(m_Font5, Font.BOLD, 25));

else if(m_Type3.equals("Italic"))
    g.setFont(new Font(m_Font5, Font.ITALIC, 25));

else if(m_Type3.equals("Plain"))
    g.setFont(new Font(m_Font5, Font.PLAIN, 25));

// Рисуем пятую строку
g.drawString(m_Str5, 10, yCurrent);
}
}

```

Исходный текст документа HTML, в который встроен апплет TextOut, приведен в листинге 3.6.

Листинг 3.6. Файл TextOut\TextOut.html

```

<html>
<head>
<title>TextOut</title>
</head>
<body>
<hr>
<applet
    code=TextOut.class
    id=TextOut
    width=320
    height=240 >
    <param name=Str1 value="Hello 1">
    <param name=Str2 value="Hello 2">
    <param name=Str3 value="Hello 3">
    <param name=Str4 value="Hello 4">
    <param name=Str5 value="Hello 5">
    <param name=Font1 value="Dialog">
    <param name=Font2 value="Courier">
    <param name=Font3 value="TimesRoman">
    <param name=Font4 value="Helvetica">
    <param name=Font5 value="Undefined">
    <param name=Type1 value="Bold">
    <param name=Type2 value="Italic">
    <param name=Type3 value="Plain">
</applet>
<hr>
<a href="TextOut.java">The source.</a>
</body>
</html>

```

Описание исходных текстов

Если при создании шаблона апплета с помощью системы Java Applet Wizard вы указываете, что апплету передаются параметры и определяете их список, система Java Applet Wizard организует для вас прием и хранение параметров. Поэтому обработка параметров апплета не отнимет у вас много сил.

Какие строки добавляются системой Java Applet Wizard для обработки параметров?

Поля класса TextOut

Прежде всего, создаются поля класса для хранения значений параметров:

```

private String m_Str1 = "Hello 1";
. . .
private String m_Str5 = "Hello 5";
private String m_Font1 = "Arial";
. . .
private String m_Font5 = "Undefined";

```

```
private String m_Type1 = "Bold";
private String m_Type2 = "Italic";
private String m_Type3 = "Plain";
```

Поля инициализируются значениями по умолчанию, которые вы ввели при заполнении таблицы, показанной на рис. 3.13.

Далее в классе определяются поля с названиями параметров:

```
private final String PARAM_Str1 = "Str1";
. . .
private final String PARAM_Str5 = "Str5";
private final String PARAM_Font1 = "Font1";
. . .
private final String PARAM_Font5 = "Font5";
private final String PARAM_Type1 = "Type1";
private final String PARAM_Type2 = "Type2";
private final String PARAM_Type3 = "Type3";
```

Названия параметров будут нужны для извлечения значений параметров методом `getParameter` класса `Applet`.

Метод `getParameterInfo`

Система Java Applet Wizard переопределяет метод `getParameterInfo`, который возвращает ссылку на массив массивов с описаниями параметров:

```
public String[][] getParameterInfo()
{
    String[][] info =
    {
        { PARAM_Str1, "String", "Text string to write" },
        . . .
        { PARAM_Str5, "String", "Text string to write" },
        { PARAM_Font1, "String", "Text font" },
        . . .
        { PARAM_Font5, "String", "Text font" },
        { PARAM_Type1, "String", "Font type" },
        { PARAM_Type2, "String", "Font type" },
        { PARAM_Type3, "String", "Font type" },
    };
    return info;
}
```

Как мы уже говорили, эта информация может использоваться другими апплетами, размещенными в том же документе HTML и работающими одновременно с нашим апплетом.

Метод `init`

При инициализации апплета метод `init` читает все параметры и записывает их значения в соответствующие поля класса, как это показано ниже:

```
public void init()
{
    String param;

    param = getParameter(PARAM_Str1);
    if (param != null)
        m_Str1 = param;
    . . .
    param = getParameter(PARAM_Str5);
    if (param != null)
        m_Str5 = param;

    // Шрифты для отображения строк
    param = getParameter(PARAM_Font1);
    if (param != null)
        m_Font1 = param;
    . . .
    param = getParameter(PARAM_Font5);
    if (param != null)
        m_Font5 = param;

    // Начертание шрифтов
    param = getParameter(PARAM_Type1);
    if (param != null)
        m_Type1 = param;
    . . .
    param = getParameter(PARAM_Type3);
    if (param != null)
        m_Type3 = param;
}
```


Здесь все просто. Метод `init` по очереди получает значения параметров методом `getParameter`, которому в качестве параметра передается имя параметра апплета. Полученное значение сохраняется в рабочей переменной `param` и, если оно отлично от значения `null`, сохраняется в соответствующем поле класса.

Метод `paint`

После закрашивания фона желтым цветом и рисования вокруг окна апплета черной рамки метод `paint` анализирует значение параметра `m_Type1` и выбирает в контекст отображения шрифт для рисования первой строки:

```
if(m_Type1.equals("Bold"))
    g.setFont(new Font(m_Font1, Font.BOLD, 25));

else if(m_Type1.equals("Italic"))
    g.setFont(new Font(m_Font1, Font.ITALIC, 25));

else if(m_Type1.equals("Plain"))
    g.setFont(new Font(m_Font1, Font.PLAIN, 25));
```

Для сравнения строк класса `String` мы используем метод `equals`, который возвращает значение `true` при совпадении с заданной строкой и `false` в противном случае.

Методу выбора шрифта `setFont` мы передаем объект класса `Font`, созданный конструктором.

Конструктор получает в качестве первого параметра содержимое поля класса `m_Font1`, которое соответствует значению параметра апплета с именем `Font1`.

Значение второго параметра (стиль шрифта) выбирается исходя из значения параметра апплета с именем `m_Type1`. Здесь мы указываем константы, определенные в классе `Font`.

И, наконец, третий параметр конструктора класса `Font` задает размер символов шрифта, равный 25 пикселям.

После выбора шрифта мы выполняем отступ от верхней границы окна и рисуем первую строку в позиции (0, `yCurrent`):

```
yCurrent += yStart;
g.drawString(m_Str1, 10, yCurrent);
```

На следующем этапе метод `paint` получает метрику только что выбранного шрифта и увеличивает текущую позицию `yCurrent` на величину полной высоты символов шрифта, полученную с помощью метода `getHeight`:

```
FontMetrics fm = g.getFontMetrics();
yCurrent += fm.getHeight();
```

Далее эта же процедура повторяется для остальных четырех отображаемых в окне апплета строк.

Экспериментируя с апплетом, попробуйте изменить параметры, передаваемые апплету в документе HTML. Укажите, например, несуществующий шрифт и посмотрите, какой шрифт будет выбран навигатором для отображения.

4 ОБРАБОТКА СОБЫТИЙ

От апплетов Java было бы немного толку, если бы они не умели обрабатывать информацию, поступающую от мыши и клавиатуры. К счастью, такая обработка предусмотрена и она выполняется достаточно просто.

Когда пользователь выполняет операции с мышью или клавиатурой в окне апплета, возникают события, которые передаются соответствующим методам класса Applet. Переопределяя эти методы, вы можете организовать обработку событий, возникающих от мыши или клавиатуры.

Если вы создавали приложения для операционной системы Microsoft Windows, здесь для вас нет ничего нового - вспомните, как вы обрабатывали сообщение WM_LBUTTONDOWN или WM_CHAR. Когда пользователь выполнял действие с мышью или клавиатурой в окне приложения, функция этого окна получала соответствующее сообщение. Методы класса Applet, обрабатывающие события от мыши и клавиатуры, являются аналогами обработчиков указанных сообщений.

Заметим, что апплеты имеют дело только с левой клавишей мыши. В текущей версии Java вы не можете никаким образом задействовать в апплете правую или среднюю клавишу мыши.

Как обрабатываются события

Когда возникает событие, управление получает метод `handleEvent` из класса `Component`. Класс `Applet` является дочерним по отношению к классу `Component`.

Прототип метода `handleEvent` мы привели ниже:

```
public boolean handleEvent(Event evt);
```

В качестве параметра методу `handleEvent` передается объект класса `Event`, который содержит всю информацию о событии. По содержимому полей класса `Event` вы можете определить координаты курсора мыши в момент, когда пользователь нажал клавишу, отличить одинарный щелчок от двойного и так далее.

Ниже мы привели список полей класса `Event`, которые вы можете проанализировать:

Поле	Описание
<code>public Object arg;</code>	Произвольный аргумент события, значение которого зависит от типа события
<code>public int clickCount;</code>	Это поле имеет значение только для события с типом <code>MOUSE_DOWN</code> и содержит количество нажатий на клавишу мыши. Если пользователь сделал двойной щелчок мышью, в это поле будет записано значение 2
<code>public Event evt;</code>	Следующее событие в связанном списке
<code>public int id;</code>	Тип события. Ниже мы перечислим возможные значения для этого поля
<code>public int key;</code>	Код нажатой клавиши (только для события, созданного при выполнении пользователем операции с клавиатурой)
<code>public int modifiers;</code>	Состояние клавиш модификации <Alt>, <Ctrl>, <Shift>
<code>public Object target;</code>	Компонент, в котором произошло событие
<code>public long when;</code>	Время, когда произошло событие
<code>public int x;</code>	Координата по оси X
<code>public int y;</code>	Координата по оси Y

Поле `id` (тип события) может содержать следующие значения:

Значение	Тип события
<code>ACTION_EVENT</code>	Пользователь хочет, чтобы произошло некоторое событие
<code>GOT_FOCUS</code>	Компонент (в нашем случае окно апплета) получил фокус ввода. О фокусе ввода вы узнаете из раздела, посвященного работе с клавиатурой
<code>KEY_ACTION</code>	Пользователь нажал клавишу типа "Action"
<code>KEY_ACTION_RELEASE</code>	Пользователь отпустил клавишу типа "Action"
<code>KEY_PRESS</code>	Пользователь нажал обычную клавишу
<code>KEY_RELEASE</code>	Пользователь отпустил обычную клавишу
<code>LIST_DESELECT</code>	Отмена выделения элемента в списке
<code>LIST_SELECT</code>	Выделение элемента в списке
<code>LOAD_FILE</code>	Загрузка файла
<code>LOST_FOCUS</code>	Компонент потерял фокус ввода

MOUSE_DOWN	Пользователь нажал клавишу мыши
MOUSE_DRAG	Пользователь нажал клавишу мыши и начал выполнять перемещение курсора мыши
MOUSE_ENTER	Курсор мыши вошел в область окна апплета
MOUSE_EXIT	Курсор мыши покинул область окна апплета
MOUSE_MOVE	Пользователь начал выполнять перемещение курсора мыши, не нажимая клавишу мыши
MOUSE_UP	Пользователь отпустил клавишу мыши
SAVE_FILE	Сохранение файла
SCROLL_ABSOLUTE	Пользователь переместил движок полосы просмотра в новую позицию
SCROLL_LINE_DOWN	Пользователь выполнил над полосой просмотра операцию сдвига на одну строку вниз
SCROLL_LINE_UP	Пользователь выполнил над полосой просмотра операцию сдвига на одну строку вверх
SCROLL_PAGE_DOWN	Пользователь выполнил над полосой просмотра операцию сдвига на одну страницу вниз
SCROLL_PAGE_UP	Пользователь выполнил над полосой просмотра операцию сдвига на одну страницу вверх
WINDOW_DEICONIFY	Пользователь запросил операцию восстановления нормального размера окна после его минимизации
WINDOW_DESTROY	Пользователь собирается удалить окно
WINDOW_EXPOSE	Окно будет отображено
WINDOW_ICONIFY	Окно будет минимизировано
WINDOW_MOVED	Окно будет перемещено

Если событие связано с клавиатурой (тип события KEY_ACTION или KEY_ACTION_RELEASE), в поле key может находиться одно из следующих значений:

<i>Значение</i>	<i>Клавиша</i>
DOWN	Клавиша перемещения курсора вниз
END	<End>
F1	<F1>
F2	<F2>
F3	<F3>
F4	<F4>
F5	<F5>
F6	<F6>
F7	<F7>
F8	<F8>
F9	<F9>
F10	<F10>
F11	<F11>
F12	<F12>
HOME	<Home>
LEFT	Клавиша перемещения курсора влево
PGDN	<Page Down>
PGUP	<Page Up>
RIGHT	Клавиша перемещения курсора вправо
UP	Клавиша перемещения курсора вверх

Могут быть указаны следующие маски для поля модификаторов modifiers:

<i>Значение маски</i>	<i>Описание</i>
ALT_MASK	Была нажата клавиша <Alt>
META_MASK	Была нажата мета-клавиша (клавиша для ввода диактрических символов)
CTRL_MASK	Была нажата клавиша <Ctrl>
SHIFT_MASK	Была нажата клавиша <Shift>

Ваше приложение может переопределить метод `handleEvent` и обрабатывать события самостоятельно, однако есть более простой путь. Обработчик этого метода, который используется по умолчанию, вызывает несколько методов, которые более удобны в использовании, в частности, при обработке событий от мыши или клавиатуры.

События от мыши

В этом разделе мы рассмотрим события, которые возникают в результате того, что пользователь выполняет в окне апплета операции с мышью. Это такие операции, как нажатие и отпускание клавиши мыши, перемещение курсора мыши в окне апплета с нажатой или отпущенной клавишей, перемещение курсора мыши в окно апплета и удаление этого курсора из окна апплета.

Все перечисленные ниже методы должны вернуть значение true, если обработка события выполнена успешно и дальнейшая обработка не требуется. Если же методы вернут значение false, событие будет обработано методом из базового класса, то есть для него будет выполнена обработка, принятая по умолчанию.

Программисты, создававшие приложения для операционной системы Microsoft Windows, могут найти здесь аналогию с вызовом функции DefWindowProc, которая выполняет обработку сообщений, принятую по умолчанию.

Нажатие клавиши мыши

Переопределив метод mouseDown, вы сможете отслеживать нажатия клавиши мыши. Прототип этого метода приведен ниже:

```
public boolean mouseDown(Event evt, int x, int y);
```

Через параметр evt методу передается ссылка на объект Event, с помощью которой метод может получить полную информацию о событии.

Анализируя содержимое параметров x и y, приложение может определить координаты курсора на момент возникновения события.

Заметим, что для отслеживания двойного щелчка мыши не предусмотрено никакого отдельного метода. Однако анализируя содержимое поля clickCount переменной evt, вы можете определить кратность щелчка мыши:

```
if(evt.clickCount > 1)
    // Двойной щелчок
    showStatus("Mouse Double Click");
else
    // Одиночный щелчок
    showStatus("Mouse Down");
```

Отпускание клавиши мыши

При отпускании клавиши мыши управление получает метод mouseUp:

```
public boolean mouseUp(Event evt, int x, int y);
```

Анализируя параметры x и y, вы можете определить координаты точки, в которой пользователь отпустил клавишу мыши.

Перемещение курсора мыши

Когда пользователь перемещает курсор мыши над окном апплета, в процессе перемещения происходит вызов метода mouseMove:

```
public boolean mouseMove(Event evt, int x, int y);
```

Через переменные x и y передаются текущие координаты курсора мыши.

Выполнение операции Drag and Drop

Операция Drag and Drop выполняется следующим образом: пользователь нажимает клавишу мыши и, не отпуская ее, начинает перемещать курсор мыши. При этом происходит вызов метода mouseDrag:

```
public boolean mouseDrag(Event evt, int x, int y);
```

Через переменные x и y передаются текущие координаты курсора мыши. Метод mouseDrag вызывается даже в том случае, если в процессе перемещения курсор вышел за пределы окна апплета.

Вход курсора мыши в область окна апплета

Метод mouseEnter получает управление, когда курсор мыши в процессе перемещения по экрану попадает в область окна апплета:

```
public boolean mouseEnter(Event evt, int x, int y);
```

Вы можете использовать этот метод для активизации апплета, на который указывает курсор мыши.

Выход курсора мыши из области окна апплета

Метод mouseExit вызывается при покидании курсором окна апплета:

```
public boolean mouseExit(Event evt, int x, int y);
```

Если пользователь убрал курсор из окна апплета, активизированного методом mouseEnter, то метод mouseExit может переключить апплет в пассивное состояние.

Приложение mouseClicked

Апплет mouseClicked демонстрирует обработку событий, поступающих от мыши.

Когда мы создавали проект этого апплета, то в третьей диалоговой панели системы Java Applet Wizard включили три переключателя в поле Which mouse event handlers would you like added (рис. 4.1).

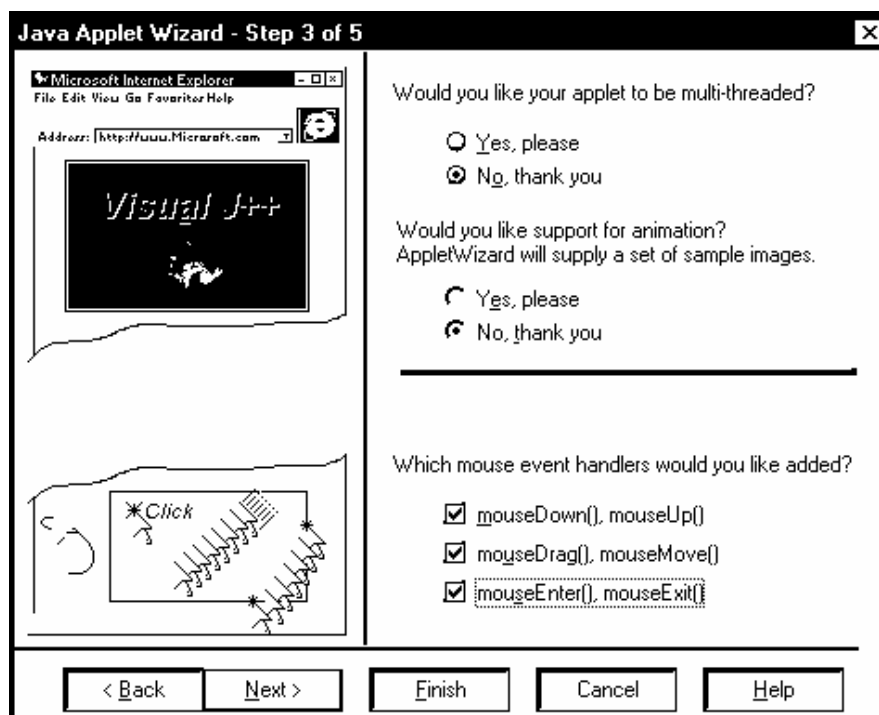


Рис. 4.1. Включение обработчиков событий от мыши

В результате в исходный текст приложения были добавлены перечисленные выше методы, обрабатывающие события, создаваемые мышью.

Мы изменили эти методы таким образом, чтобы в ответ на различные действия, выполняемые пользователем с помощью мыши, апплет реагировал соответствующим образом:

Действие пользователя	Реакция апплета
Перемещение курсора мыши при отжатой клавише	Игнорирование
Перемещение курсора мыши при нажатой клавише	В строку состояния записывается текстовая строка Mouse Drag
Нажатие клавиши мыши	В месте расположения курсора выводятся текущие координаты курсора мыши. Дополнительно в строку состояния записывается текстовая строка Mouse Down
Отжатие клавиши мыши	В строку состояния записывается текстовая строка Mouse Up
Курсор мыши входит в область окна апплета	В строку состояния записывается текстовая строка Mouse pointer enters applet's window
Курсор мыши выходит из области окна апплета	В строку состояния записывается текстовая строка Mouse pointer leaves applet's window

Внешний вид окна апплета, в котором отображаются координаты курсора, показан на рис. 4.2.

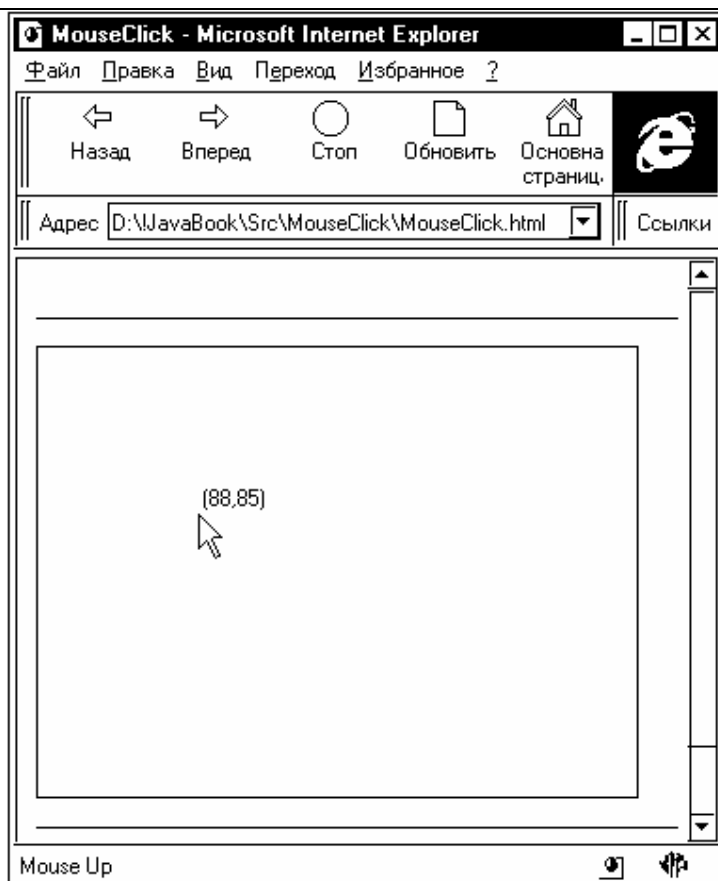


Рис. 4.2. Внешний вид окна апплета MouseClick, в котором отображаются координаты курсора

Исходные тексты приложения

Файл исходного текста приложения MouseClick представлен в листинге 4.1.

Листинг 4.1. Файл MouseClick\MouseClick.java

```
// =====
// Обработка событий от мыши
//
// (C) Фролов А.В., 1997
//
// E-mail: frolov@glas.apc.org
// WWW:    http://www.glasnet.ru/~frolov
//         или
//         http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

public class MouseClick extends Applet
{
    // Текущие координаты курсора при нажатии на
    // кнопку мыши
    Dimension dimMouseCursor;

    // Временная переменная для хранения события
    Event ev;

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: MouseClick\r\n" +
            "E-mail: frolov@glas.apc.org" +
            "WWW:    http://www.glasnet.ru/~frolov" +
            "Author: Alexandr Frolov\r\n" +
            "Created with Microsoft Visual J++ Version 1.0";
    }
}
```

```

// -----
// paint
// Метод paint, выполняющий рисование в окне апплета
// -----
public void paint(Graphics g)
{
    // Определяем текущие размеры окна апплета
    Dimension dimAppWndDimension = size();

    // Выбираем в контекст отображения желтый цвет
    g.setColor(Color.yellow);

    // Закрашиваем внутреннюю область окна апплета
    g.fillRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);

    // Выбираем в контекст отображения черный цвет
    g.setColor(Color.black);

    // Рисуем рамку вокруг окна апплета
    g.drawRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);

    // Отображаем текущие координаты курсора мыши
    // в точке, где находится этот курсор
    g.drawString("(" + ev.x + ", " + ev.y + ")", ev.x, ev.y);
}

// -----
// mouseDown
// Обработка щелчка кнопкой мыши
// -----
public boolean mouseDown(Event evt, int x, int y)
{
    // Определяем и сохраняем текущие координаты
    // курсора мыши
    dimMouseCursor = new Dimension(x, y);

    // Сохраняем событие во временной переменной
    ev = evt;

    // Если количество щелчков больше 1, считаем что
    // сделан двойной щелчок
    if(evt.clickCount > 1)

        // Выводим сообщение о двойном щелчке
        showStatus("Mouse Double Click");

    // Сделан одиночный щелчок
    else

        // Выводим сообщение о простом щелчке
        showStatus("Mouse Down");

    // Перерисовываем окно апплета
    repaint();

    // Возвращаем значение true при успешной
    // обработке события
    return true;
}

// -----
// mouseUp
// Отпускание клавиши мыши
// -----
public boolean mouseUp(Event evt, int x, int y)
{
    // Выводим сообщение в строке состояния
    showStatus("Mouse Up");
    return true;
}

// -----

```

```

// mouseDrag
// Перемещение курсора мыши при нажатой клавише
// -----
public boolean mouseDrag(Event evt, int x, int y)
{
    // Выводим сообщение в строке состояния
    showStatus("Mouse Drag");
    return true;
}

// -----
// mouseMove
// Перемещение курсора мыши при отжатой клавише
// -----
public boolean mouseMove(Event evt, int x, int y)
{
    return true;
}

// -----
// mouseEnter
// Курсор мыши вошел в область окна апплета
// -----
public boolean mouseEnter(Event evt, int x, int y)
{
    // Выводим сообщение в строке состояния
    showStatus("Mouse pointer enters applet's window");
    return true;
}

// -----
// mouseExit
// Курсор мыши покинул область окна апплета
// -----
public boolean mouseExit(Event evt, int x, int y)
{
    // Выводим сообщение в строке состояния
    showStatus("Mouse pointer leaves applet's window");
    return true;
}
}

```

Исходный текст документа HTML, созданного для нашего апплета системой Java Applet Wizard, приведен в листинге 4.2.

Листинг 4.2. Файл MouseClick\MouseClick.html

```

<html>
<head>
<title>MouseClick</title>
</head>
<body>
<hr>
<applet
    code=MouseClick.class
    id=MouseClick
    width=320
    height=240 >
</applet>
<hr>
<a href="MouseClick.java">The source.</a>
</body>
</html>

```

Описание исходного текста

В исходном тексте класса MouseClick мы определили поля класса с именами dimMouseCursor и ev: `Dimension dimMouseCursor;`
`Event ev;`

Первое из них предназначено для хранения координат курсора в момент возникновения события, а второе - хранит ссылку на это событие.

Метод getAppletInfo

Метод getAppletInfo ничем не отличается от аналогичных методов в предыдущих приложениях.

Метод paint

В начале своей работы метод paint определяет текущие размеры окна апплета, закрашивает это окно в желтый цвет и рисует вокруг него тонкую рамку черного цвета. Все это делается исключительно для того чтобы выделить апплет в документе HTML и обозначить его границы.

Далее метод paint отображает текущие координаты курсора мыши, взяв их из переменной ev:

```
g.drawString("(" + ev.x + "," + ev.y + ")", ev.x, ev.y);
```

Метод mouseDown

Когда пользователь делает щелчок левой клавишей мыши (напомним, что Java не работает с другими клавишами мыши), управление получает метод mouseDown.

Этот метод, переопределенный в нашем приложении, прежде всего сохраняет текущие координаты курсора мыши в переменной dimMouseCursor класса Dimension:

```
dimMouseCursor = new Dimension(x, y);
```

Событие, которое передается методу mouseDown через первый параметр, сохраняется в переменной ev:

```
ev = evt;
```

Далее метод mouseDown проверяет поле clickCount параметра evt:

```
if(evt.clickCount > 1)
    showStatus("Mouse Double Click");
else
    showStatus("Mouse Down");
```

В это поле записывается кратность щелчка мыши. Если пользователь сделал двойной щелчок, в строке состояния отображается текстовая строка Mouse Double Click, а если одинарный - строка Mouse Down.

Обратите внимание на метод showStatus. Этот метод позволяет апплету отобразить любую текстовую строку в строке состояния навигатора, поэтому он часто используется для отладки или выдачи текущей информации о состоянии апплета.

Заметим, однако, что в документе HTML может располагаться несколько разных апплетов, а строка состояния навигатора только одна. Поэтому сообщения от разных апплетов могут перекрывать друг друга, в результате чего в строке состояния появится только то сообщение, которое было записано туда последним.

После записи сообщения в строку состояния метод mouseDown перерисовывает окно апплета, вызывая для этого метод repaint:

```
repaint();
```

В результате вызова метода repaint происходит инициирование вызова метода paint, выполняющего перерисовку содержимого окна апплета. Однако не следует думать, будто метод repaint просто вызывает метод paint. Метод paint вызывается интерпретатором Java асинхронно по отношению к методу repaint в подходящий момент времени.

В последней строке метод mouseDown возвращает значение true:

```
return true;
```

Этим он сообщает, что обработка события завершена и это событие не нужно передавать обработчику из базового класса.

Методы mouseUp, mouseDrag, mouseEnter, mouseExit

Обработчики методов mouseUp, mouseDrag, mouseEnter и mouseExit выглядят одинаково:

```
public boolean mouseUp(Event evt, int x, int y)
{
    // Выводим сообщение в строке состояния
    showStatus("Mouse Up");
    return true;
}
```

Пользуясь методом showStatus, эти методы записывают соответствующее сообщение в строку состояния и возвращают значение true.

Метод mouseMove

Метод mouseMove выглядит следующим образом:

```
public boolean mouseMove(Event evt, int x, int y)
{
    return true;
}
```

Он ничего не делает, кроме того что возвращает значение true, блокируя обработку события, выполняемую в базовом классе.

Приложение LineDraw

Следующее приложение тоже работает с мышью. В его окне вы можете рисовать прямые линии черного цвета (рис. 4.3).

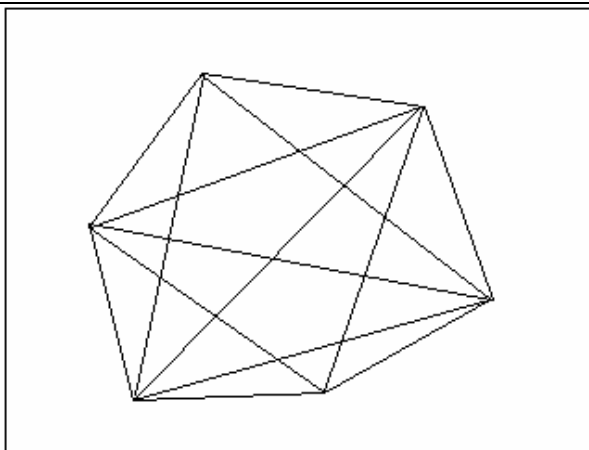


Рис. 4.3. Апплет, в окне которого можно рисовать прямые линии

Для того чтобы нарисовать линию в окне апплета LineDraw, вы должны установить курсор в начальную точку, нажать клавишу мыши и затем, не отпуская ее, переместить курсор в конечную точку. После отпускания клавиши мыши координаты линии будут сохранены апплетом в массиве, после чего произойдет перерисовка окна апплета.

По мере того как вы будете рисовать линии, апплет будет заполнять массив с координатами линий. Каждый раз, когда окно апплета будет перерисовываться, метод `paint` перерисует все линии заново, пользуясь координатами, сохраненными в массиве.

Для того чтобы стереть содержимое окна апплета, вам достаточно сделать двойной щелчок в его окне. При этом из массива координат линий будут удалены все элементы.

Исходные тексты приложения

Исходный текст апплета LineDraw приведен в листинге 4.3.

Листинг 4.3. Файл LineDraw\LineDraw.java

```
// =====
// Рисование прямых линий
//
// (C) Фролов А.В, 1997
//
// E-mail: frolov@glas.apc.org
// WWW:   http://www.glasnet.ru/~frolov
//        или
//        http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

// Добавляем для класса Vector
import java.util.*;

public class LineDraw extends Applet
{
    // Координаты курсора при нажатии кнопки мыши
    Dimension dmDown;

    // Координаты курсора при отжатии кнопки мыши
    Dimension dmUp;

    // Предыдущие координаты конца линии
    Dimension dmPrev;

    // Признак включения режима рисования
    boolean bDrawing;

    // Массив координат линий
    Vector lines;

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: LineDraw\r\n" +
            "E-mail: frolov@glas.apc.org" +

```

```

        "WWW:      http://www.glasnet.ru/~frolov" +
        "Author: Alexandr Frolov\r\n" +
        "Created with Microsoft Visual J++ Version 1.0";
    }

    // -----
    // init
    // Метод, получающий управление при инициализации апплета
    // -----
    public void init()
    {
        // Сброс признака рисования
        bDrawing = false;

        // Создание массива для хранения координат линий
        lines = new Vector();
    }

    // -----
    // paint
    // Метод paint, выполняющий рисование в окне апплета
    // -----
    public void paint(Graphics g)
    {
        // Определяем текущие размеры окна апплета
        Dimension dimAppWndDimension = size();

        // Выбираем в контекст отображения желтый цвет
        g.setColor(Color.yellow);

        // Закрашиваем внутреннюю область окна апплета
        g.fillRect(0, 0,
            dimAppWndDimension.width - 1,
            dimAppWndDimension.height - 1);

        // Выбираем в контекст отображения черный цвет
        g.setColor(Color.black);

        // Рисуем рамку вокруг окна апплета
        g.drawRect(0, 0,
            dimAppWndDimension.width - 1,
            dimAppWndDimension.height - 1);

        // Рисуем в цикле линии, координаты которых
        // хранятся в массиве lines
        for (int i=0; i < lines.size(); i++)
        {
            // Получаем координаты очередной линии
            Rectangle p = (Rectangle)lines.elementAt(i);

            // Рисуем линию
            g.drawLine( p.width, p.height, p.x, p.y);
        }

        // Сбрасываем режим рисования
        bDrawing = false;
    }

    // -----
    // mouseDown
    // Обработка щелчка кнопкой мыши
    // -----
    public boolean mouseDown(Event evt, int x, int y)
    {
        // Если количество щелчков больше 1, считаем что
        // сделан двойной щелчок
        if(evt.clickCount > 1)
        {
            // Удаляем все строки из массива lines
            lines.removeAllElements();

            // Перерисовываем окно апплета
            repaint();

            return true;
        }
    }

```

```

    // Сохраняем текущие координаты начала линии
    dmDown = new Dimension(x, y);

    // В начале процесса рисования линии устанавливаем
    // предыдущие координаты конца линии равными
    // координатам начала линии
    dmPrev = new Dimension(x, y);

    // Отключаем режим рисования
    bDrawing = false;
    return true;
}

// -----
// mouseUp
// Отпускание клавиши мыши
// -----
public boolean mouseUp(Event evt, int x, int y)
{
    // Проверяем, включен ли режим рисования
    if(bDrawing)
    {
        // Если режим рисования включен, добавляем
        // новый элемент в массив lines

        // Сохраняем координаты конца линии
        dmUp = new Dimension(x, y);

        // Добавляем линию в массив
        lines.addElement(
            new Rectangle(dmDown.width, dmDown.height, x, y));

        // Перерисовываем окно апплета
        repaint();

        // Отключаем режим рисования
        bDrawing = false;
    }
    return true;
}

// -----
// mouseDrag
// Перемещение курсора мыши при нажатой клавише
// -----
public boolean mouseDrag(Event evt, int x, int y)
{
    // Получаем контекст отображения для окна апплета
    Graphics g = getGraphics();

    // Включаем режим рисования
    bDrawing = true;

    // Закрашиваем предыдущую линию цветом фона
    // (то есть стираем ее)
    g.setColor(Color.yellow);

    g.drawLine(dmDown.width, dmDown.height,
        dmPrev.width, dmPrev.height);

    // Рисуем новую линию черным цветом
    g.setColor(Color.black);
    g.drawLine(dmDown.width, dmDown.height, x, y);

    // Сохраняем координаты предыдущей линии,
    // чтобы стереть ее в следующий раз
    dmPrev = new Dimension(x, y);
    return true;
}

// -----
// mouseMove
// Перемещение курсора мыши при отжатой клавише
// -----
public boolean mouseMove(Event evt, int x, int y)
{
    // Отключаем режим рисования

```

```

        bDrawing = false;
        return true;
    }
}

```

Исходный текст документа HTML, созданного для апплета LineDraw, вы найдете в листинге 4.4.

Листинг 4.4. Файл LineDraw\LineDraw.html

```

<html>
<head>
<title>LineDraw</title>
</head>
<body>
<hr>
<applet
    code=LineDraw.class
    id=LineDraw
    width=320
    height=240 >
</applet>
<hr>
<a href="LineDraw.java">The source.</a>
</body>
</html>

```

Описание исходного текста

В нашем апплете мы будем создавать объект класса Vector, который является массивом с динамически изменяемым размером. Этот класс имеет полное имя java.util.Vector, поэтому мы подключаем соответствующую библиотеку классов:

```
import java.util.*;
```

Поля класса LineDraw

В нашем классе мы определили несколько полей, предназначенных для хранения текущих координат рисуемых линий.

В переменную dmDown класса Dimension записываются координаты курсора на момент нажатия клавиши мыши. Если пользователь нажал клавишу мыши для того чтобы приступить к рисованию линии, это будет координатами начала линии.

Когда пользователь отпускает клавишу мыши, координаты записываются в переменную dmUp.

В процессе рисования линии метод mouseDrag стирает ранее нарисованную линию и рисует новую. Координаты конца старой линии хранятся в переменной dmPrev.

Переменная bDrawing типа boolean хранит текущее состояние апплета. Когда апплет находится в состоянии рисования линии, в эту переменную записывается значение true, а когда нет - значение false.

И, наконец, переменная lines типа Vector является динамическим массивом, в котором хранятся координаты нарисованных линий.

Метод getAppletInfo

Метод getAppletInfo возвращает информацию об апплете и не имеет никаких особенностей.

Метод init

Метод init сбрасывает признак рисования, записывая в поле bDrawing значение false, а также создает новый динамический массив в виде объекта класса Vector:

```

public void init()
{
    bDrawing = false;
    lines = new Vector();
}

```

Метод paint

После обычной для наших апплетов раскраски фона и рисования рамки метод paint перебирает в цикле все элементы массива lines, рисуя линии:

```

for (int i=0; i < lines.size(); i++)
{
    Rectangle p = (Rectangle)lines.elementAt(i);
    g.drawLine(p.width, p.height, p.x, p.y);
}

```

Для объектов класса Vector можно использовать метода size, возвращающий количество элементов в массиве, чем мы воспользовались для проверки условия выхода из цикла.

Чтобы извлечь элемент массива по его номеру, мы воспользовались методом elementAt, передав ему через единственный параметр номер извлекаемого элемента.

Так как в массиве хранятся объекты класса Rectangle, перед инициализацией ссылки p мы выполняем преобразование типов.

Перед завершением работы метод paint сбрасывает признак рисования, записывая в поле bDrawing значение false:

```
bDrawing = false;
```

Метод mouseDown

В начале своей работы метод mouseDown определяет, был ли сделан одинарный щелчок клавишей мыши, или двойной. Если был сделан двойной щелчок мышью, метод удаляет все элементы из массива list, а затем перерисовывает окно апплета, вызывая метод repaint:

```
lines.removeAllElements();
repaint();
```

После перерисовки окно апплета очищается от линий.

Если же был сделан одинарный щелчок клавишей мыши, метод mouseDown сохраняет текущие координаты курсора в переменных dmDown и dmPrev, а затем сбрасывает признак рисования:

```
dmDown = new Dimension(x, y);
dmPrev = new Dimension(x, y);
bDrawing = false;
```

Метод mouseUp

Когда пользователь отпускает клавишу мыши, вызывается метод mouseUp. В его задачу входит сохранение текущих координат курсора мыши в поле dmUp, а также добавление нового элемента в массив lines, как это показано ниже:

```
dmUp = new Dimension(x, y);
lines.addElement(
    new Rectangle(dmDown.width, dmDown.height, x, y));
repaint();
```

После добавления элемента в массив метод mouseUp инициирует перерисовку окна апплета, вызывая для этого метод repaint.

Заметим, что в качестве координат начала линии мы записываем в элемент массива координаты точки, где в последний раз пользователь нажимал курсор мыши. В качестве координат конца линии используются текущие координаты курсора на момент отпускания клавиши мыши.

Метод mouseDrag

До сих пор наши апплеты выполняли рисование только в методе paint, и так поступают большинство апплетов. Однако наш апплет должен рисовать линии во время перемещения курсора мыши, так как в противном случае пользователю не будет видно, как пройдет рисуемая линия.

Для того чтобы нарисовать что-либо в окне апплета, необходимо получить контекст отображения. Методу paint этот контекст передается через параметр как объект класса Graphics. Если же вы собираетесь рисовать в другом методе, отличном от paint, необходимо получить контекст отображения, например, так:

```
Graphics g = getGraphics();
```

После получения контекста отображения и включения режима рисования (записью в переменную bDrawing значения true) метод mouseDrag стирает линию, которая была нарисована ранее, в процессе предыдущего вызова этого же метода:

```
g.setColor(Color.yellow);
g.drawLine(dmDown.width, dmDown.height,
    dmPrev.width, dmPrev.height);
```

Для стирания линии мы рисуем ее на том же месте с использованием цвета, совпадающего с цветом фона.

Далее метод mouseDrag рисует новую линию черного цвета, соединяя точку, в которой была нажата клавиша мыши, с точкой текущего расположения курсора мыши:

```
g.setColor(Color.black);
g.drawLine(dmDown.width, dmDown.height, x, y);
```

После рисования линии координаты ее конца сохраняются в поле dmPrev для стирания этой линии при следующем вызове метода mouseDrag:

```
dmPrev = new Dimension(x, y);
return true;
```

Метод mouseMove

Метод mouseMove не делает ничего, за исключением того, что он отключает режим рисования. Таким образом, простое перемещение курсора мыши над окном апплета не приводит к рисованию линий.

События от клавиатуры

Апплет может обрабатывать события, создаваемые клавиатурой. Например, он может реагировать на функциональные клавиши или на клавиши ускоренного выбора функций.

Для того чтобы обработать события от клавиатуры, ваш апплет должен переопределить методы keyDown и keyUp:

```
public boolean keyDown(Event evt, int nKey)
{
    . . .
}
public boolean keyUp(Event evt, int nKey)
{
    . . .
}
```

```
} . . .
```

В качестве первого параметра этим методам передается объект типа Event, о полях которого мы рассказывали в разделе “Как обрабатываются события” этой главы.

Наибольший интерес представляют поля объекта evt с именами key и modifiers. Через них передается, соответственно, код нажатой клавиши и код модификации. Возможные значения для этих полей вы найдете в только что указанном разделе.

Второй параметр методов keyDown и keyUp с именем nKey дублирует поле key объекта evt.

Приложение KeyCode

Для демонстрации методов обработки клавиатурных событий мы подготовили апплет KeyCode. В его окне отображаются символы, соответствующие нажимаемым клавишам, код соответствующих клавиш и коды модификации (рис. 4.4).

Ъ -> key: 10	mod: 0
-> key: 32	mod: 1
-> key: 32	mod: 2
-> key: 32	mod: 0
Ъ -> key: 17	mod: 3
Ъ -> key: 17	mod: 2
Т -> key: 84	mod: 1
Р -> key: 82	mod: 1
Е -> key: 69	mod: 1
W -> key: 87	mod: 1
Q -> key: 81	mod: 1
у -> key: 121	mod: 0
т -> key: 116	mod: 0
г -> key: 114	mod: 0
е -> key: 101	mod: 0
w -> key: 119	mod: 0

Рис. 4.4. Окно апплета KeyCode

Апплет KeyCode интересен тем, что в процессе отображения новые строки появляются в верхней части окна, а старые сдвигаются вниз после отпускания клавиши. Таким образом, мы организовали свертку в окне апплета.

Прежде чем нажимать клавиши, вы должны передать фокус вводу окну апплета. Это можно сделать, щелкнув в окне левой клавишей мыши. Фокус ввода - это атрибут, который присваивается окну, обрабатывающему ввод от клавиатуры. Так как клавиатура одна, а апплетов и других активных окон на экране может быть много, необходим механизм, позволяющий определить, в какое окно направляются события, создаваемые клавиатурой. Такой механизм и обеспечивается атрибутом фокуса ввода.

Исходные тексты приложения KeyCode

Файл исходного текста приложения KeyCode приведен в листинге 4.5.

Листинг 4.5. Файл KeyCode\KeyCode.java

```
// =====
// Просмотр кодов клавиш
//
// (C) Фролов А.В., 1997
//
// E-mail: frolov@glas.apc.org
// WWW:   http://www.glasnet.ru/~frolov
//        или
//        http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

public class KeyCode extends Applet
{
    // Высота символов текущего шрифта
    int yHeight;

    // Текущие размеры окна апплета
    Dimension dimAppWndDimension;

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
```

```

public String getAppletInfo()
{
    return "Name: KeyCode\r\n" +
        "Author: Alexandr Frolov\r\n" +
        "WWW:      http://www.glasnet.ru/~frolov" +
        "Author: Alexandr Frolov\r\n" +
        "Created with Microsoft Visual J++ Version 1.0";
}

// -----
// init
// Метод, получающий управление при инициализации апплета
// -----
public void init()
{
    // Получаем контекст отображения
    Graphics g = getGraphics();

    // Определяем метрики текущего шрифта
    FontMetrics fm = g.getFontMetrics();

    // Сохраняем полную высоту символов шрифта
    yHeight = fm.getHeight();
}

// -----
// paint
// Метод paint, выполняющий рисование в окне апплета
// -----
public void paint(Graphics g)
{
    // Определяем текущие размеры окна апплета
    dimAppWndDimension = size();

    // Выбираем в контекст отображения желтый цвет
    g.setColor(Color.yellow);

    // Закрашиваем внутреннюю область окна апплета
    g.fillRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);

    // Выбираем в контекст отображения черный цвет
    g.setColor(Color.black);

    // Рисуем рамку вокруг окна апплета
    g.drawRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);
}

// -----
// keyDown
// Вызывается, когда пользователь нажимает на клавишу
// -----
public boolean keyDown(Event evt, int nKey)
{
    // Массив для преобразования кода символа в символ
    char[] chKey;

    // Временная строка
    String s;

    // Создаем массив из одного элемента
    chKey = new char[1];

    // Записываем в него код нажатой клавиши
    chKey[0] = (char)nKey;

    // Преобразуем в строку
    s = new String(chKey);

    // Получаем контекст отображения
    Graphics g = getGraphics();

    // Выбираем черный цвет для рисования
    g.setColor(Color.black);

```



```

// Рисуем символ, соответствующий нажатой клавише
g.drawString(s + " ", 10, 10);

// Рисуем код клавиши
g.drawString(
    " -> key: " + evt.key, 20, 10);

// Рисуем код модификации
g.drawString(" mod: " + evt.modifiers, 100, 10);

return true;
}

// -----
// keyUp
// Вызывается, когда пользователь отпускает клавишу
// -----
public boolean keyUp(Event evt, int nKey)
{
    // Получаем контекст отображения
    Graphics g = getGraphics();

    // Выполняем свертку нижней области окна
    g.copyArea(0, 1,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - yHeight - 5,
        0, yHeight + 1);

    // Закрашиваем область ввода желтым цветом
    g.setColor(Color.yellow);

    g.fillRect(1, 1,
        dimAppWndDimension.width - 2, yHeight + 1);

    return true;
}
}

```

В листинге 4.6 вы найдете исходный текст документа HTML, в который встроен наш апплет.

Листинг 4.6. Файл KeyCode\KeyCode.html

```

<html>
<head>
<title>KeyCode</title>
</head>
<body>
<hr>
<applet
    code=KeyCode.class
    id=KeyCode
    width=320
    height=240 >
</applet>
<hr>
<a href="KeyCode.java">The source.</a>
</body>
</html>

```

Описание исходного текста

Наш апплет определяет несколько полей в своем классе и переопределяет несколько методов базового класса.

Поля класса KeyCode

Поле yHeight используется для хранения полной высоты символов текущего шрифта, выбранного в контекст отображения окна апплета. Эта величина нужна для определения шага свертки окна.

В поле dimAppWndDimension типа Dimension хранятся текущие размеры окна апплета.

Метод getAppletInfo

Метод getAppletInfo возвращает информацию об апплете и не имеет никаких особенностей.

Метод init

Этот метод получает контекст отображения, однако не для рисования, а для определения метрик шрифта:

```
Graphics g = getGraphics();
```

```
FontMetrics fm = g.getFontMetrics();
```

В переменную `yHeight` заносится полная высота символов текущего шрифта:

```
yHeight = fm.getHeight();
```

Метод paint

Метод `paint` закрашивает окно апплета желтым цветом и обводит его рамкой. Никакой другой работы этот метод не выполняет.

Метод keyDown

Когда пользователь нажимает клавишу, управление передается методу `keyDown`. Обработчик метода `keyDown` преобразует код нажатой клавиши `nKey` в текстовую строку типа `String` и затем отображает эту строку и содержимое двух полей объекта `evt` в окне апплета.

Преобразование выполняется в два приема.

Вначале код символа, имеющий тип `int`, преобразуется к типу `char` и записывается в ячейку массива типа `char[]`, как это показано ниже:

```
char[] chKey;
String s;
chKey = new char[1];
chKey[0] = (char)nKey;
```

Затем этот массив, состоящий только из одного элемента, преобразуется в текстовую строку:

```
s = new String(chKey);
```

Далее метод `keyDown` получает контекст отображения, устанавливает в нем черный цвет и рисует в верхней части окна параметры клавиатурного события:

```
Graphics g = getGraphics();
g.setColor(Color.black);
g.drawString(s + " ", 10, 10);
g.drawString(" -> key: " + evt.key, 20, 10);
g.drawString(" mod: " + evt.modifiers, 100, 10);
```

Метод keyUp

Метод `keyUp` получает управление, когда пользователь отпускает ранее нажатую клавишу. Ему передаются те же параметры, что и только что рассмотренному методу `keyDown`.

Наш метод `keyUp` получает контекст отображения, а затем выполняет копирование верхней части окна, сдвигая ее вниз на размер символов текущего шрифта:

```
g.copyArea(0, 1,
    dimAppWndDimension.width - 1,
    dimAppWndDimension.height - yHeight - 5,
    0, yHeight + 1);
```

После сдвига метод закрашивает область ввода, расположенную в верхней части апплета, желтым цветом, расчищая таким образом место для ввода новой строки:

```
g.setColor(Color.yellow);
g.fillRect(1, 1,
    dimAppWndDimension.width - 2, yHeight + 1);
```

5 КОМПОНЕНТЫ В ОКНЕ АПЛЕТА

Практически каждое приложение Windows, за исключением самых простейших, имеет такие органы управления, как меню, кнопки, поля редактирования текстовой информации, переключатели с независимой и зависимой фиксацией и списки. Кроме того, приложение Windows может создавать диалоговые панели, содержащие перечисленные выше и другие органы управления.

В окне апплета вы также можете разместить некоторые из перечисленных выше органов управления, а именно:

- кнопки;
- переключатели с независимой фиксацией;
- переключатели с зависимой фиксацией;
- статические текстовые поля;
- однострочные и многострочные поля редактирования текста;
- списки;
- полосы просмотра

Самый большой и едва ли приятный сюрприз для вас это то, что при размещении перечисленных органов управления в окне апплета вы не можете задать для них точные координаты и размеры. Размещением занимается система управления внешним видом Layout Manager, которая располагает органы управления по-своему. Вы, однако, можете задавать несколько режимов размещения (последовательное, в ячейках таблицы и так далее), но не координаты или размеры. Это сделано для обеспечения независимости приложений Java от платформ, на которых они выполняются.

Органы управления создаются как объекты классов, порожденных от класса Component (рис. 5.1). Поэтому в дальнейшем мы будем называть органы управления компонентами.

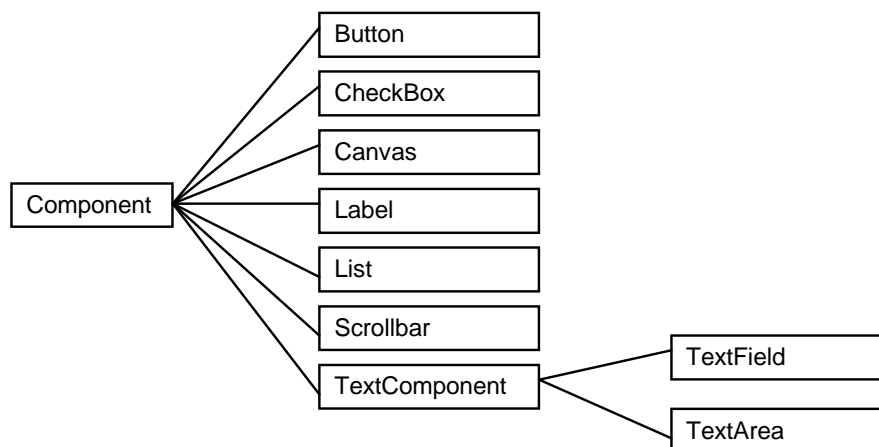


Рис. 5.1. Взаимосвязь классов органов управления в приложениях Java

Класс Button позволяет создавать стандартные кнопки. Если вам нужна нестандартная кнопка (например, графическая кнопка), вы можете создать ее на базе класса Canvas.

Для создания переключателей с независимой или зависимой фиксацией предназначен класс CheckBox.

С помощью класса Label вы можете создавать в окне апплета текстовые строки, например, надписи для других компонент. Эти строки не редактируются пользователем.

Класс List, как нетрудно догадаться из названия, предназначен для создания списков.

С помощью класса Scrollbar вы можете создавать полосы просмотра, которые используются, в частности, многострочными полями редактирования текста.

Класс TextComponent служит базовым для двух других классов - TextField и TextArea. Первый из них предназначен для создания однострочных редакторов текста, второй - для создания многострочных редакторов текста.

Для того чтобы понять, как компоненты размещаются на поверхности окна апплета системой Layout Manager, рассмотрим другую взаимосвязь классов Java, показанную на рис. 5.2.

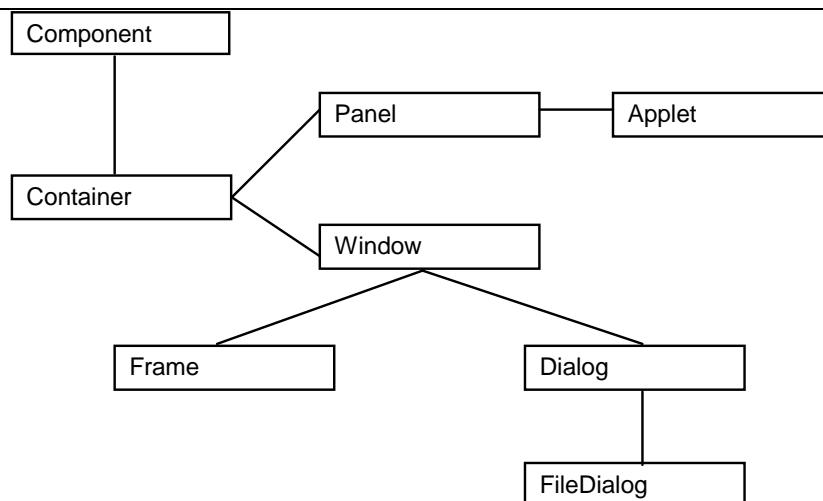


Рис. 5.2. Компоненты и контейнеры

Класс Component служит в качестве базового класса для класса Container. Объекты этого класса, которые мы будем называть контейнерами, могут содержать объекты классов Component и Container. Таким образом, внутри контейнеров могут находиться компоненты и другие контейнеры.

Класс Applet, так же как и другие классы, произведенные от класса Container, является контейнером. Это означает, что апплет может содержать в себе компоненты (такие как органы управления) и контейнеры.

Заметим, что класс Applet наследуется от класса Container через класс Panel, в котором определены методы системы Layout Manager. Настроивая соответствующим образом Layout Manager, мы можем менять стратегию размещения компонент внутри окна апплета.

В окне апплета вы можете создать несколько объектов класса Panel (панелей), разделяющих окно на части. Для каждой такой панели можно установить свою стратегию размещения компонент и контейнеров, что позволяет выполнять достаточно сложную компоновку в окне апплета.

Теперь после такого краткого введения в контейнеры и компоненты мы перейдем к описанию методов создания отдельных компонент.

Кнопки

Как мы уже говорили, стандартные кнопки создаются на базе класса Button. Этот класс очень простой, поэтому мы приведем его определение полностью:

```

public class java.awt.Button
    extends java.awt.Component
{
    // -----
    // Конструкторы
    // -----
    public Button();
    public Button(String label);

    // -----
    // Методы
    // -----

    // Вызов метода createButton
    public void addNotify();

    // Получение надписи на кнопке
    public String getLabel();

    // Получение строки параметров, отражающей
    // состояние кнопки
    protected String paramString();

    // Установка надписи на кнопке
    public void setLabel(String label);
}
  
```

В классе Button определены два конструктора, первый из которых позволяет создавать кнопку без надписи, а второй - кнопку с надписью. Обычно используется именно второй конструктор.

Из методов класса Button вы будете использовать чаще всего два - getLabel и setLabel. Первый из них позволяет получить строку надписи на кнопке, а второй - установить новую надпись.

Обычно апплет создает в своем окне кнопки в процессе своей инициализации при обработке метода init, например:

```

Button btn1;
. . .
  
```

```
public void init()
{
    btn1 = new Button("Button 1");
    add(btn1);
}
```

Здесь мы создали кнопку с надписью Button 1 и добавили ее в контейнер, которым является окно апплета, с помощью метода add.

Обработка событий от кнопки

Для обработки событий, создаваемых кнопками и другими компонентами, вы можете переопределить метод `handleEvent`, описанный нами ранее. Однако существует более простой способ.

Этот способ основан на переопределении метода `action`, который получает управление, когда пользователь совершает какое-либо действие с компонентом. Под действием подразумевается нажатие на кнопку, завершение ввода текстовой строки, выбор элемента из списка, изменение состояния переключателя и так далее.

Прототип метода `action` представлен ниже:

```
public boolean action(Event evt, Object obj)
{
    . . .
}
```

В качестве первого параметра методу передается ссылка на объект класса `Event`, содержащий всю информацию о событии. Второй параметр представляет собой ссылку на объект, вызвавший появление события.

Как обрабатывать событие в методе `action`?

Прежде всего необходимо проверить, объект какого типа создал событие. Это можно сделать, например, следующим образом:

```
if(evt.target instanceof Button)
{
    . . .
    return true;
}
return false;
```

Здесь мы с помощью оператора `instanceof` проверяем, является ли объект, вызвавший появление события, объектом класса `Button`.

Далее, если в окне апплета имеется несколько кнопок, необходимо выполнить ветвление по ссылкам на объекты кнопок, как это показано ниже:

```
if(evt.target.equals(btn1))
{
    . . .
}
else if(evt.target.equals(btn2))
{
    . . .
}

. . .

else
{
    return false;
}
return true;
```

Тем из вас, кто создавал приложения Windows на языке программирования C, этот фрагмент кода может напомнить длинный переключатель `switch` обработки сообщений Windows.

Приложение ButtonPress

В окне приложения `ButtonPress` мы создаем четыре кнопки с названиями от Button 1 до Button 4. Когда пользователь нажимает на одну из кнопок, название нажатой кнопки отображается в окне апплета и в строке состояния навигатора (рис. 5.3).

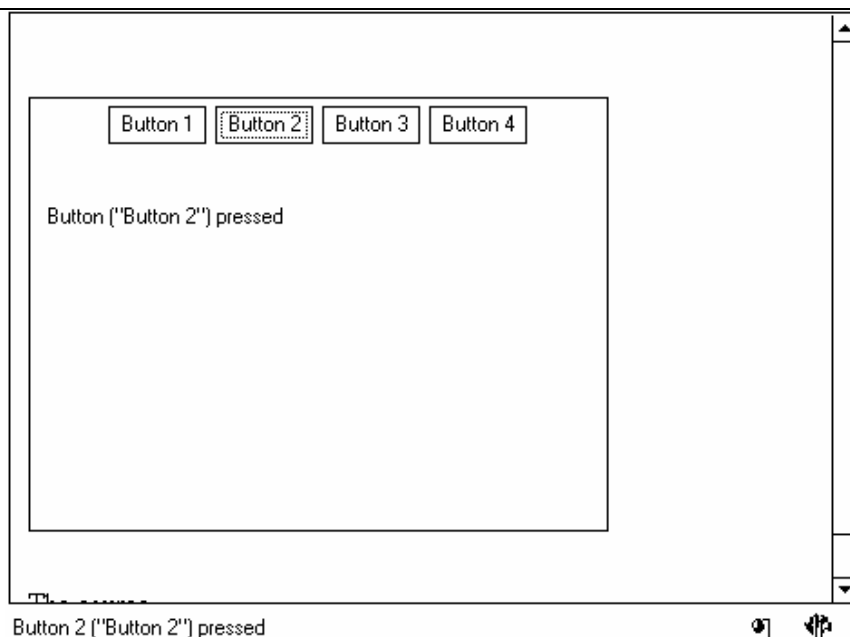


Рис. 5.3. Кнопки в окне апплета ButtonPress

Обратите внимание на расположение кнопок. По мере добавления, кнопки располагаются по горизонтали справа налево и центрируются в окне апплета. Если бы ширины окна апплета не хватило для размещения четырех кнопок, не поместившиеся кнопки были бы нарисованы ниже. Такую стратегию размещения выбирает по умолчанию система Layout Manager класса Panel, от которого, как вы знаете, произошел класс Applet.

Исходные тексты приложения ButtonPress

Исходный текст приложения ButtonPress приведен в листинге 5.1.

Листинг 5.1. Файл ButtonPress\ButtonPress.java

```
// =====
// Работа с кнопками
//
// (C) Фролов А.В., 1997
//
// E-mail: frolov@glas.apc.org
// WWW:   http://www.glasnet.ru/~frolov
//        или
//        http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

public class ButtonPress extends Applet
{
    // Создаем четыре ссылки на объекты типа Button
    Button btn1;
    Button btn2;
    Button btn3;
    Button btn4;

    // Строка для записи названия нажатой кнопки
    String sTextLabel;

    // -----
    // getAppletInfo
    // Метод, возвращающий строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: ButtonPress\r\n" +
            "E-mail: frolov@glas.apc.org" +
            "WWW:   http://www.glasnet.ru/~frolov" +
            "Author: Alexandr Frolov\r\n" +
            "Created with Microsoft Visual J++ Version 1.0";
    }
}

// -----
```

```

// init
// Метод, получающий управление при инициализации апплета
// -----
public void init()
{
    // Создаем четыре кнопки
    btn1 = new Button("Button 1");
    btn2 = new Button("Button 2");
    btn3 = new Button("Button 3");
    btn4 = new Button("Button 4");

    // Добавляем кнопки в контейнер, которым является
    // окно апплета
    add(btn1);
    add(btn2);
    add(btn3);
    add(btn4);

    // Название кнопки, нажатой в последний раз
    sTextLabel = new String("");
}

// -----
// action
// Метод вызывается, когда пользователь выполняет
// действие над компонентами
// -----
public boolean action(Event evt, Object obj)
{
    // Ссылка на кнопку, от которой пришло сообщение
    Button btn;

    // Проверяем, что событие вызвано кнопкой, а не
    // другим компонентом
    if(evt.target instanceof Button)
    {
        // Получаем ссылку на кнопку, вызвавшую событие
        btn = (Button)evt.target;

        // Получаем название кнопки
        sTextLabel = btn.getLabel();

        // Выполняем ветвление по кнопкам. Для каждой кнопки
        // записываем ее название
        // в строку состояния навигатора
        if(evt.target.equals(btn1))
        {
            showStatus(
                "Button 1 (\\"" + sTextLabel + "\\") pressed");
        }

        else if(evt.target.equals(btn2))
        {
            showStatus(
                "Button 2 (\\"" + sTextLabel + "\\") pressed");
        }

        else if(evt.target.equals(btn3))
        {
            showStatus(
                "Button 3 (\\"" + sTextLabel + "\\") pressed");
        }

        else if(evt.target.equals(btn4))
        {
            showStatus(
                "Button 4 (\\"" + sTextLabel + "\\") pressed");
        }

        // Если событие возникло от неизвестной кнопки,
        // мы его не обрабатываем
        else
        {
            return false;
        }

        // Перерисовываем окно апплета

```

```

        repaint();

        // возвращаем признак того, что мы обработали событие
        return true;
    }

    // Если событие вызвано не кнопкой,
    // мы его не обрабатываем
    return false;
}

// -----
// paint
// Метод paint, выполняющий рисование в окне апплета
// -----
public void paint(Graphics g)
{
    // Определяем текущие размеры окна апплета
    Dimension dimAppWndDimension = size();

    // Выбираем в контекст отображения желтый цвет
    g.setColor(Color.yellow);

    // Закрашиваем внутреннюю область окна апплета
    g.fillRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);

    // Выбираем в контекст отображения черный цвет
    g.setColor(Color.black);

    // Рисуем рамку вокруг окна апплета
    g.drawRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);

    // Рисуем название нажатой кнопки
    g.drawString("Button (" + sTextLabel + ") pressed",
        10, 70);
}
}

```

Исходный текст документа HTML, созданный для апплета ButtonPress, представлен в листинге 5.2.

Листинг 5.2. Файл ButtonPress\ButtonPress.html

```

<html>
<head>
<title>ButtonPress</title>
</head>
<body>
<hr>
<applet
    code=ButtonPress.class
    id=ButtonPress
    width=320
    height=240 >
</applet>
<hr>
<a href="ButtonPress.java">The source.</a>
</body>
</html>

```

Описание исходного текста

После того как исходный текст приложения был создан системой Java Applet Wizard, мы добавили поля для хранения ссылок на кнопки и текстовую строку для записи метки нажатой кнопки, а также добавили и изменили несколько методов.

Поля класса ButtonPress

Четыре поля класса Button с именами btn1, btn2, btn3 и btn4 предназначены для хранения ссылок на кнопки, размещенные в окне нашего апплета:

```

Button btn1;
Button btn2;
Button btn3;
Button btn4;

```


В поле sTextLabel класса String хранится строка, предназначенная для записи названия нажатой кнопки:

```
String sTextLabel;
```

Метод getAppletInfo

Метод getAppletInfo, возвращающий строку информации об апплете, не имеет никаких особенностей.

Метод init

При инициализации апплета метод init создает четыре кнопки, сохраняя ссылки на них в соответствующих полях нашего класса, а также добавляет эти кнопки в окно апплета, вызывая для этого метод add:

```
public void init()
{
    btn1 = new Button("Button 1");
    btn2 = new Button("Button 2");
    btn3 = new Button("Button 3");
    btn4 = new Button("Button 4");

    add(btn1);
    add(btn2);
    add(btn3);
    add(btn4);

    sTextLabel = new String("");
}
```

После добавления кнопок в строку sTextLabel записывается пустое значение, так как ни одна кнопка еще не была нажата.

Метод action

Метод action проверяет, является ли объект, создавший событие, кнопкой. Для этого он сравнивает ссылку на объект, передаваемую через поле evt.target, с объектом Button, пользуясь оператором instanceof. Так как поле evt.target может содержать ссылку на любой объект, способный создавать события, а не только на объект типа Button, эта проверка необходима для исключения ложных срабатываний на чужие события.

Если событие создано кнопкой, ссылка на эту кнопку сохраняется в переменной btn:

```
Button btn;
btn = (Button)evt.target;
```

При этом мы выполняем преобразование типов.

Далее метод action получает название кнопки (то есть строку, написанную на поверхности кнопки) и сохраняет его в переменной sTextLabel:

```
sTextLabel = btn.getLabel();
```

Для получения строки названия кнопки используется метод getLabel, определенный в классе Button.

Затем метод action проверяет, от какой конкретно кнопки пришло событие, выполняя ветвление с помощью оператора if - else if - else:

```
if(evt.target.equals(btn1))
{
    showStatus("Button 1 (\\"" + sTextLabel + "\\") pressed");
}

else if(evt.target.equals(btn2))
{
    showStatus("Button 2 (\\"" + sTextLabel + "\\") pressed");
}

. . .
else
{
    return false;
}
```

Название нажатой кнопки отображается в строке состояния навигатора. Если событие создано кнопкой, обработчик которой не предусмотрен в нашем методе action, метод просто возвращает значение false, отказываясь таким образом от обработки события.

Если ваша реализация метода action не обрабатывает событие, она может передать его методу action базового класса, как это показано ниже:

```
super.action(evt, obj);
```

В том случае, когда событие было обработано, метод action перерисовывает окно апплета, вызывая метод repaint, и затем возвращает значение true:

```
repaint();
return true;
```

Метод paint

Метод paint не содержит никакого кода для рисования кнопок, так как эта задача решается в рамках класса Button. После раскрашивания фона окна апплета и рисования рамки вокруг него, метод paint пишет название нажатой кнопки в окне апплета с помощью метода drawString:

```
g.drawString("Button (\\" + sTextLabel + "\\") pressed",
10, 70);
```

Переключатели

Апплеты Java могут создавать в своем окне переключатели двух типов: с независимой фиксацией и с зависимой фиксацией.

Переключатели с независимой фиксацией имеют прямоугольную форму и, исходя из названия, работают независимо друг от друга. Если такой переключатель находится во включенном состоянии, внутри изображения маленького квадрата появляется галочка, если в выключенном - галочка исчезает.

Обычно переключатели с независимой фиксацией используются для управления независимыми друг от друга режимами или параметрами.

Переключатели с зависимой фиксацией имеют круглую форму. В каждый момент времени может быть включен только один такой переключатель из группы переключателей с фиксацией. Апплет может создавать несколько групп переключателей с зависимой фиксацией.

Переключатели с зависимой фиксацией используются для выбора из нескольких взаимоисключающих возможностей, например, для установки одного из нескольких режимов.

Создание переключателей с независимой фиксацией

Переключатели с независимой и зависимой фиксацией создаются на базе класса Checkbox:

```
public class java.awt.Checkbox
extends java.awt.Component
{
// -----
// Конструкторы
// -----

// Создание переключателя с независимой фиксацией
// без названия
public Checkbox();

// Создание переключателя с независимой фиксацией
// и названием
public Checkbox(String label);

// Создание переключателя с зависимой фиксацией
// и названием
public Checkbox(String label, CheckboxGroup group,
boolean state);

// -----
// Методы
// -----

// Вызов метода createCheckbox
public void addNotify();

// Получение группы, к которой относится
// данный переключатель с зависимой фиксацией
public CheckboxGroup getCheckboxGroup();

// Получение названия переключателя
public String getLabel();

// Определение текущего состояния переключателя
public boolean getState();

// Получение строки параметров
protected String paramString();

// Установка группы, к которой относится
// данный переключатель с зависимой фиксацией
public void setCheckboxGroup(CheckboxGroup g);

// Установка названия переключателя
public void setLabel(String label);

// Установка нового состояния переключателя
public void setState(boolean state);
}
```

Создать переключатель с независимой фиксацией не сложнее, чем создать кнопку:

```
Checkbox rdbox1;
...
public void init()
{
    chbox1 = new Checkbox("Switch 1");
    add(chbox1);
}
```

В этом фрагменте кода мы создаем переключатель chbox1 с названием Switch 1, а затем с помощью метода add добавляем его в контейнер, которым является окно апплета.

Для определения текущего состояния переключателя вы можете использовать метод getState. Если переключатель включен, этот метод возвращает значение true, а если выключен - значение false.

Создание переключателей с зависимой фиксацией

Для каждой группы переключателей с зависимой фиксацией вы должны создать объект класса CheckboxGroup:

```
public class java.awt.CheckboxGroup
    extends java.lang.Object
{
    // -----
    // Конструктор
    // -----
    public CheckboxGroup();

    // -----
    // Методы
    // -----

    // Получение ссылки на переключатель, который
    // находится во включенном состоянии
    public Checkbox getCurrent();

    // Установка указанного переключателя в группе
    // во включенное состояние
    public void setCurrent(Checkbox box);

    // Получение строки, которая представляет группу
    public String toString();
}
```

Ссылка на этот объект указывается при создании отдельных переключателей с зависимой фиксацией, входящих в группу:

```
CheckboxGroup grModeGroup;
Checkbox rdbox1;
Checkbox rdbox2;
Checkbox rdbox3;
Checkbox rdbox4;
...
public void init()
{
    grModeGroup = new CheckboxGroup();

    rdbox1 = new Checkbox("Mode 1",grModeGroup, true);
    rdbox2 = new Checkbox("Mode 2",grModeGroup, false);
    rdbox3 = new Checkbox("Mode 3",grModeGroup, false);
    rdbox4 = new Checkbox("Mode 4",grModeGroup, false);

    add(rdbox1);
    add(rdbox2);
    add(rdbox3);
    add(rdbox4);
}
```

Через первый параметр конструктору Checkbox в этом примере передается название переключателя, через второй - ссылка на группу, а через третий - состояние, в которое должен быть установлен переключатель. Из всех переключателей группы только один может находиться во включенном состоянии.

Приложение CheckBoxes

Для демонстрации методов работы с различными переключателями мы подготовили приложение CheckBoxes. Окно соответствующего апплета показано на рис. 5.4.

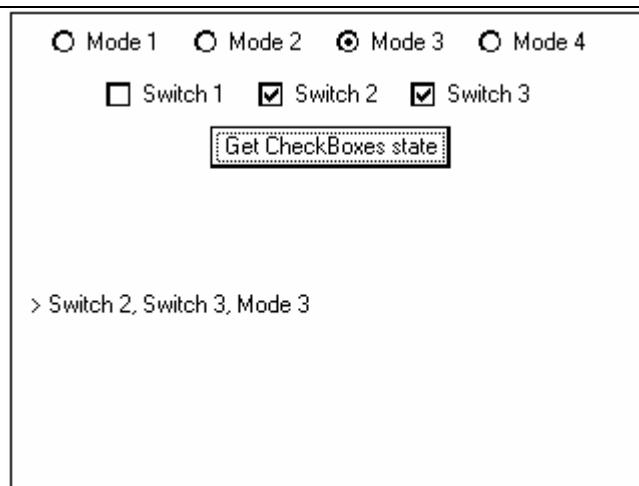


Рис. 5.4. Окно апплета CheckBoxes с переключателями и кнопкой

В верхней части окна располагаются четыре переключателя с зависимой фиксацией, принадлежащих к одной группе. Ниже находятся три переключателя с независимой фиксацией, а еще ниже - кнопка с надписью Get CheckBoxes state. Нажав на эту кнопку, вы можете увидеть в нижней части окна апплета список включенных переключателей.

Одновременно может быть включен только один из переключателей Mode 1 - Mode 4, так как эти переключатели составляют группу переключателей с зависимой фиксацией. Переключатели Switch 1, Switch 2 и Switch 3 могут находиться в произвольном состоянии независимо друг от друга.

Заметим, что переключатели и кнопка размещались в окне апплета автоматически по мере добавления. Если бы мы добавляли эти компоненты в другой последовательности или если бы окно апплета имело другие размеры, то переключатели могли бы не оказаться сгруппированными, как это показано на рис. 5.4. Позже в этой главе мы научим вас настраивать систему Layout Manager таким образом, чтобы вы смогли располагать компоненты в заданном вами порядке с предсказуемым результатом.

Исходные тексты приложения CheckBoxes

Файл исходного текста приложения CheckBoxes представлен в листинге 5.3.

Листинг 5.3. Файл CheckBoxes\CheckBoxes.java

```
// =====
// Работа с переключателями
//
// (C) Фролов А.В, 1997
//
// E-mail: frolov@glas.apc.org
// WWW:   http://www.glasnet.ru/~frolov
//        или
//        http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

public class CheckBoxes extends Applet
{
    // Создаем три ссылки на объекты типа Checkbox
    Checkbox chbox1;
    Checkbox chbox2;
    Checkbox chbox3;

    // Создаем ссылку на объект типа CheckboxGroup
    CheckboxGroup grModeGroup;

    // Создаем четыре ссылки на объекты типа Checkbox
    Checkbox rdbox1;
    Checkbox rdbox2;
    Checkbox rdbox3;
    Checkbox rdbox4;

    // Создаем ссылку на объект типа Button
    Button btnGet;

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
```

```

// -----
public String getAppletInfo()
{
    return "Name: CheckBoxes\r\n" +
        "WWW:      http://www.glasnet.ru/~frolov" +
        "E-mail: frolov@glas.apc.org" +
        "Author: Alexandr Frolov\r\n" +
        "Created with Microsoft Visual J++ Version 1.0";
}

// -----
// init
// Метод, получающий управление при инициализации апплета
// -----
public void init()
{
    // Устанавливаем желтый цвет фона
    setBackground(Color.yellow);

    // Создаем три переключателя с независимой фиксацией
    chbox1 = new Checkbox("Switch 1");
    chbox2 = new Checkbox("Switch 2");
    chbox3 = new Checkbox("Switch 3");

    // Создаем группу переключателей с зависимой фиксацией
    grModeGroup = new CheckboxGroup();

    // Создаем четыре переключателя с зависимой фиксацией,
    // принадлежащие группе grModeGroup
    rdbox1 = new Checkbox("Mode 1",grModeGroup, true);
    rdbox2 = new Checkbox("Mode 2",grModeGroup, false);
    rdbox3 = new Checkbox("Mode 3",grModeGroup, false);
    rdbox4 = new Checkbox("Mode 4",grModeGroup, false);

    // Создаем кнопку, предназначенную для определения
    // текущего состояния переключателей
    btnGet = new Button("Get CheckBoxes state");

    // Добавляем в окно апплета четыре переключателя
    // с зависимой фиксацией
    add(rdbox1);
    add(rdbox2);
    add(rdbox3);
    add(rdbox4);

    // Добавляем в окно апплета три переключателя
    // с независимой фиксацией
    add(chbox1);
    add(chbox2);
    add(chbox3);

    // Добавляем в окно апплета кнопку
    add(btnGet);
}

// -----
// action
// Метод вызывается, когда пользователь выполняет
// действие над компонентами
// -----
public boolean action(Event evt, Object obj)
{
    // Проверяем, что событие вызвано кнопкой, а не
    // другим компонентом
    if(evt.target instanceof Button)
    {
        // Выполняем ветвление по кнопкам.
        if(evt.target.equals(btnGet))
        {
            showStatus("Button 1 pressed");
        }

        // Если событие возникло от неизвестной кнопки,
        // мы его не обрабатываем
        else
        {
            return false;
        }
    }
}

```

```

    }

    // Перерисовываем окно апплета
    repaint();

    // возвращаем признак того, что мы обработали событие
    return true;
}

// Если событие вызвано не кнопкой,
// мы его не обрабатываем
return false;
}

// -----
// paint
// Метод paint, выполняющий рисование в окне апплета
// -----
public void paint(Graphics g)
{
    // Определяем текущие размеры окна апплета
    Dimension dimAppWndDimension = size();

    // Выбираем в контекст отображения черный цвет
    g.setColor(Color.black);

    // Рисуем рамку вокруг окна апплета
    g.drawRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);

    // Строка для записи списка
    // состояния переключателей
    String s = new String("> ");

    // Проверяем переключатели с независимой фиксацией
    if(chbox1.getState())
        s = s + chbox1.getLabel() + ", ";

    if(chbox2.getState())
        s = s + chbox2.getLabel() + ", ";

    if(chbox3.getState())
        s = s + chbox3.getLabel() + ", ";

    if(rdbbox1.getState())
        s = s + rdbbox1.getLabel();

    // Проверяем переключатели с зависимой фиксацией
    else if(rdbbox2.getState())
        s = s + rdbbox2.getLabel();

    else if(rdbbox3.getState())
        s = s + rdbbox3.getLabel();

    else if(rdbbox4.getState())
        s = s + rdbbox4.getLabel();

    // Рисуем строку состояния переключателей
    g.drawString(s, 10, 150);
}
}

```

В листинге 5.4 вы найдете исходный текст документа HTML, который был создан системой Java Applet Wizard для нашего апплета.

Листинг 5.4. Файл CheckBoxes\CheckBoxes.html

```

<html>
<head>
<title>CheckBoxes</title>
</head>
<body>
<hr>
<applet
    code=CheckBoxes.class
    id=CheckBoxes
    width=320

```

```

    height=240 >
</applet>
<hr>
<a href="CheckBoxes.java">The source.</a>
</body>
</html>

```

Описание исходного текста

Рассмотрим поля класса CheckBoxes и переопределенные нами методы.

Поля класса CheckBoxes

В нашем классе мы определили несколько полей, предназначенных для хранения ссылок на переключатели, группу переключателей и кнопку.

Ссылки на переключатели имеют тип Checkbox:

```

Checkbox chbox1;
Checkbox chbox2;
Checkbox chbox3;

```

```

Checkbox rdbbox1;
Checkbox rdbbox2;
Checkbox rdbbox3;
Checkbox rdbbox4;

```

Для того чтобы сгруппировать переключатели с зависимой фиксацией в группу, мы создали ссылку на объект класса CheckboxGroup:

```
CheckboxGroup grModeGroup;
```

Кроме того, нам потребуется ссылка на объект класса Button:

```
Button btnGet;
```

Метод getAppletInfo

Метод getAppletInfo возвращает информацию о нашем апплете.

Метод init

До сих пор для изменения цвета фона окна наших апплетов мы раскрашивали окно в желтый цвет явным образом в методе paint. Однако есть и другой способ, основанный на вызове метода setBackground:

```
setBackground(Color.yellow);
```

Дополнением к этому методу может послужить метод setForeground, с помощью которого можно установить цвет для рисования в окне.

Почему мы выбрали другой способ изменения фона окна?

Дело в том, что переключатели сами по себе являются окнами, обладающими такими атрибутами, как цвет фона и цвет изображения. Если просто нарисовать их в окне, закрашенным желтым цветом, то получится, что серые переключатели нарисованы на желтом фоне, что некрасиво. Метод setBackground, вызванный для окна апплета, позволяет задать цвет фона не только для контейнера, но и для всех компонент, расположенных в нем.

После установки цвета фона метод init создает три переключателя с независимой фиксацией, указывая их название:

```

chbox1 = new Checkbox("Switch 1");
chbox2 = new Checkbox("Switch 2");
chbox3 = new Checkbox("Switch 3");

```

Далее метод init создает группу переключателей с зависимой фиксацией в виде объекта класса CheckboxGroup:

```
grModeGroup = new CheckboxGroup();
```

Для создания переключателей с зависимой фиксацией необходимо использовать метод, допускающий указание группы и начального состояния переключателя:

```

rdbbox1 = new Checkbox("Mode 1",grModeGroup, true);
rdbbox2 = new Checkbox("Mode 2",grModeGroup, false);
rdbbox3 = new Checkbox("Mode 3",grModeGroup, false);
rdbbox4 = new Checkbox("Mode 4",grModeGroup, false);

```

Затем метод init создает кнопку с названием Get CheckBoxes state, предназначенную для определения текущего состояния переключателей:

```
btnGet = new Button("Get CheckBoxes state");
```

После создания компонент они добавляются в контейнер, которым является окно апплета. Для этого используется метод add.

Прежде всего мы добавляем четыре переключателя с зависимой фиксацией:

```

add(rdbbox1);
add(rdbbox2);
add(rdbbox3);
add(rdbbox4);

```

Размеры окна и размеры переключателей соотносятся между собой таким образом, что в верхней части окна апплета помещаются как раз четыре переключателя. Очевидно, если изменить размеры окна апплета, переключатели будут размещены по-другому.

Далее метод `init` добавляет в окно апплета переключатели с независимой фиксацией и кнопку:

```
add(chbox1);
add(chbox2);
add(chbox3);
add(btnGet);
```

Метод *action*

Метод *action* обрабатывает только те события, которые вызваны кнопкой `btnGet`:

```
if(evt.target instanceof Button)
{
    if(evt.target.equals(btnGet))
        showStatus("Button 1 pressed");
    else
        return false;
    repaint();
    return true;
}
```

Когда пользователь нажимает кнопку, метод *action* выводит сообщение об этом в строку состояния навигатора и перерисовывает окно апплета. Текущее состояние кнопок определяется методом `paint` во время перерисовки окна.

Метод *paint*

В методе `paint` мы не закрашиваем желтым цветом окно апплета, так как на этапе инициализации в обработчике метода `init` был установлен желтый цвет фона окна. Однако черную рамку вокруг границы окна апплета мы все же рисуем.

Основная задача метода `paint` заключается в отображении в нижней части окна апплета списка включенных переключателей. Для формирования этой строки мы создаем объект `String`:

```
String s = new String("> ");
```

Далее мы проверяем по очереди состояние всех переключателей с независимой фиксацией, дописывая к строке `s` название включенных переключателей:

```
if(chbox1.getState())
    s = s + chbox1.getLabel() + ", ";
if(chbox2.getState())
    s = s + chbox2.getLabel() + ", ";
if(chbox3.getState())
    s = s + chbox3.getLabel() + ", ";
```

Для определения текущего состояния переключателей мы вызываем метод `getState`. Этот метод возвращает значение `true` для включенного переключателя и `false` - для выключенного. Название переключателя легко определить с помощью метода `getLabel`.

Проверка состояний переключателей с зависимой фиксацией выполняется аналогично, но с учетом того, что одновременно может быть включен только один такой переключатель:

```
if(rdbbox1.getState())
    s = s + rdbbox1.getLabel();
else if(rdbbox2.getState())
    s = s + rdbbox2.getLabel();
else if(rdbbox3.getState())
    s = s + rdbbox3.getLabel();
else if(rdbbox4.getState())
    s = s + rdbbox4.getLabel();
```

После завершения формирования строки `s` она отображается в окне апплета методом `drawString`:

```
g.drawString(s, 10, 150);
```

Списки класса *Choice*

На базе класса *Choice* вы можете создать списки типа *Drop Down* или, как их еще называют, “выпадающие” списки. Такой список выглядит как текстовое поле высотой в одну строку, справа от которого располагается кнопка (рис. 5.5).

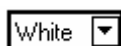


Рис. 5.5. Список типа *Drop Down*, созданный на базе класса *Choice*

Если нажать на эту кнопку, список раскроется и вы сможете сделать выбор из его элементов (рис. 5.6).



Рис. 5.6. Раскрытый список, созданный на базе класса Choice

В списке класса Choice одновременно можно выбрать только один элемент.

Рассмотрим класс Choice. Определение этого класса выглядит несложно:

```
public class java.awt.Choice
    extends java.awt.Component
{
    // -----
    // Конструктор
    // -----
    public Choice();

    // -----
    // Методы
    // -----

    // Добавление элемента в список
    public void addItem(String item);

    // Вызов метода createChoice
    public void addNotify();

    // Определение количества элементов в списке
    public int countItems();

    // Получение строки списка по номеру соответствующего
    // ему элемента списка
    public String getItem(int index);

    // Получение номера текущего выбранного элемента
    public int getSelectedIndex();

    // Получение строки, соответствующей текущему
    // выбранному элементу списка
    public String getSelectedItem();

    // Получение строки параметров
    protected String paramString();

    // Выбор в списке элемента по заданному номеру
    public void select(int pos);

    // Выбор в списке элемента по заданной строке
    public void select(String str);
}
```

Конструктор класса Choice не имеет параметров. Создание списка с его помощью не вызовет у вас никаких затруднений:

```
Choice chBackgroundColor;
chBackgroundColor = new Choice();
```

Для наполнения списка используйте метод addItem. В качестве параметра ему необходимо передать текстовую строку, которая будет связана с добавляемым элементом списка:

```
chBackgroundColor.addItem("Yellow");
```

Далее список можно добавить в окно апплета как компонент с помощью метода add:

```
add(chBackgroundColor);
```

Заметим, что список можно заполнять до или после добавления в окно апплета.

После наполнения списка по умолчанию выделяется элемент, который был добавлен в список первым. При помощи метода select вы можете выделить любой элемент списка по его номеру или строке, связанной с элементом.

Когда пользователь выбирает новую строку в списке, возникает событие. Обработчик этого события, реализованный, например, переопределением метода action, может получить номер выбранной строки при помощи метода getSelectedIndex. Пример обработки такого события вы найдете в разделе "Приложение ChoiceList".

Если вас интересует не номер выбранного элемента, а строка, связанная с выбранным элементом, воспользуйтесь методом getSelectedItem.

И, наконец, с помощью метода getItem вы можете получить текст строки, связанной с элементом, по номеру элемента.

Приложение ChoiceList

В приложении ChoiceList мы создали два списка, первый из которых управляет цветом фона окна апплета, а второй - цветом изображения, то есть цветом, которым рисуется изображение в этом окне (рис. 5.7).



Рис. 5.7. Окно апплета ChoiceList, в котором создано два списка класса Choice

Пользоваться этим апплетом очень просто - выбирайте из левого списка цвет фона, а из правого - цвет изображения, при этом следите за цветом, которым нарисована текстовая строка и рамка вокруг окна апплета.

Исходные тексты приложения ChoiceList

Исходный текст приложения ChoiceList вы найдете в листинге 5.5.

Листинг 5.5. Файл ChoiceList\ChoiceList.java

```
// =====
// Списки типа Drop Down
//
// (C) Фролов А.В, 1997
//
// E-mail: frolov@glas.apc.org
// WWW:    http://www.glasnet.ru/~frolov
//          или
//          http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

public class ChoiceList extends Applet
{
    // Создаем ссылки на объекты класса Choice
    Choice chBackgroundColor;
    Choice chForegroundColor;

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: ChoiceList\r\n" +
            "Author: Alexandr Frolov\r\n" +
            "E-mail: frolov@glas.apc.org" +
            "WWW:    http://www.glasnet.ru/~frolov" +
            "Created with Microsoft Visual J++ Version 1.0";
    }

    // -----
    // init
    // Метод, получающий управление при инициализации апплета
    // -----
    public void init()
    {
        // Создаем списки для выбора цвета фона и
        // цвета изображения
        chBackgroundColor = new Choice();
        chForegroundColor = new Choice();

        // Добавляем списки в окно апплета
        add(chBackgroundColor);
        add(chForegroundColor);

        // Заполняем список цвета фона
        chBackgroundColor.addItem("Yellow");
        chBackgroundColor.addItem("Green");
        chBackgroundColor.addItem("White");
    }
}
```

```

// Заполняем список цвета изображения
chForegroundColor.addItem("Black");
chForegroundColor.addItem("Red");
chForegroundColor.addItem("Blue");

// Устанавливаем цвет фона
setBackground(Color.yellow);

// Устанавливаем цвет изображения
setForeground(Color.black);
}

// -----
// action
// Метод вызывается, когда пользователь выполняет
// действие над компонентами
// -----
public boolean action(Event evt, Object obj)
{
    // Переменная для хранения ссылки на список,
    // вызвавший событие
    Choice ch;

    // Получаем ссылку на список
    ch = (Choice)evt.target;

    // Проверяем, что событие вызвано списком, а не
    // другим компонентом
    if(evt.target instanceof Choice)
    {
        // Выполняем ветвление по спискам

        // Список цвета фона
        if(evt.target.equals(chBackgroundColor))
        {
            // Получаем номер текущего элемента списка
            // и устанавливаем соответствующий
            // цвет фона
            if(ch.getSelectedIndex() == 0)
                setBackground(Color.yellow);

            else if(ch.getSelectedIndex() == 1)
                setBackground(Color.green);

            else if(ch.getSelectedIndex() == 2)
                setBackground(Color.white);
        }

        // Список цвета изображения
        else if(evt.target.equals(chForegroundColor))
        {
            // Получаем номер текущего элемента списка
            // и устанавливаем соответствующий
            // цвет изображения
            if(ch.getSelectedIndex() == 0)
                setForeground(Color.black);

            else if(ch.getSelectedIndex() == 1)
                setForeground(Color.red);

            else if(ch.getSelectedIndex() == 2)
                setForeground(Color.blue);
        }

        // Если событие возникло от неизвестного списка,
        // мы его не обрабатываем
        else
        {
            return false;
        }

        // Перерисовываем окно апплета
        repaint();

        // возвращаем признак того, что мы обработали событие
        return true;
    }
}

```

```

    }

    // Если событие вызвано не кнопкой,
    // мы его не обрабатываем
    return false;
}

// -----
// paint
// Метод paint, выполняющий рисование в окне апплета
// -----
public void paint(Graphics g)
{
    // Определяем текущие размеры окна апплета
    Dimension dimAppWndDimension = size();

    // Рисуем рамку вокруг окна апплета
    g.drawRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);

    // Рисуем строку
    g.drawString("Смотри на цвет фона и текста!", 10, 50);
}
}

```

Исходный текст документа HTML, созданного для нашего апплета, приведен в листинге 5.6.

Листинг 5.6. Файл ChoiceList\ChoiceList.html

```

<html>
<head>
<title>ChoiceList</title>
</head>
<body>
<hr>
<applet
    code=ChoiceList.class
    id=ChoiceList
    width=320
    height=240 >
</applet>
<hr>
<a href="ChoiceList.java">The source.</a>
</body>
</html>

```

Описание исходного текста

Рассмотрим поля класса ChoiceList и переопределенные нами методы.

Поля класса ChoiceList

В нашем классе мы определили два поля для хранения ссылок на списки цвета фона и цвета изображения:

```

Choice chBackgroundColor;
Choice chForegroundColor;

```

Метод getAppletInfo

Метод getAppletInfo возвращает информацию об апплете ChoiceList.

Метод init

В методе init мы создаем два списка как объекты класса Choice:

```

chBackgroundColor = new Choice();
chForegroundColor = new Choice();

```

Созданные списки пока пустые, но мы можем добавить их в окно апплета, вызвав метод add:

```

add(chBackgroundColor);
add(chForegroundColor);

```

Сразу после добавления списков мы их заполняем, вызывая для соответствующих объектов метод addItem:

```

chBackgroundColor.addItem("Yellow");
chBackgroundColor.addItem("Green");
chBackgroundColor.addItem("White");

chForegroundColor.addItem("Black");
chForegroundColor.addItem("Red");
chForegroundColor.addItem("Blue");

```

Элементы, добавленные в список первыми, будут выбраны в списке по умолчанию. В нашем случае будет выбран фон желтого цвета и изображение черного цвета.

Такие же цвета мы устанавливаем для окна апплета, вызывая методы setBackground и setForeground:

```
setBackground(Color.yellow);
setForeground(Color.black);
```

Метод action

Метод action обрабатывает событие, вызванное списками, - выбор элемента из списка.

Прежде всего, метод action сохраняет ссылку на список, в котором произошел выбор, в переменной ch:

```
Choice ch;
ch = (Choice)evt.target;
```

Далее выполняется проверка факта, что событие вызвано именно списком, после чего происходит анализ, в каком именно списке сделан выбор нового элемента:

```
if (evt.target.equals(chBackgroundColor))
{
    if(ch.getSelectedIndex() == 0)
        setBackground(Color.yellow);
    else if(ch.getSelectedIndex() == 1)
        setBackground(Color.green);
    else if(ch.getSelectedIndex() == 2)
        setBackground(Color.white);
}
else if (evt.target.equals(chForegroundColor))
{
    if(ch.getSelectedIndex() == 0)
        setForeground(Color.black);
    else if(ch.getSelectedIndex() == 1)
        setForeground(Color.red);
    else if(ch.getSelectedIndex() == 2)
        setForeground(Color.blue);
}
else
    return false;
```

Обратите внимание, что мы вначале выполняем преобразование типа evt.target к классу Choice, а только затем проверяем, действительно ли событие вызвано списком. Правильно ли это?

Вообще говоря, неправильно. Так как в поле evt.target могут находиться ссылки на объекты различных классов, в процессе выполнения преобразования типов может произойти исключение. Если предпринимается попытка выполнить преобразование для несовместимых типов. Но так как в нашем апплете события создаются только списками, исключение не возникает.

Корректнее было бы вначале проверить ссылку evt.target на принадлежность к классу Choice с помощью оператора instanceof, а только потом выполнять преобразование типов. Так мы и будем делать в следующих примерах апплетов, обрабатывающих события от различных источников.

С помощью метода getSelectedIndex метод action определяет номер выбранного элемента списка, устанавливая соответствующим образом цвет фона или изображения.

Метод paint

Обработчик метода paint рисует рамку вокруг окна апплета и текстовую строку в средней части этого окна.

```
Dimension dimAppWndDimension = size();
g.drawRect(0, 0,
    dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);
g.drawString("Смотри на цвет фона и текста!", 10, 50);
```

При этом мы не указали цвет фона, а также цвет изображения. При этом используются цвета, установленные методом action при выборе соответствующих строк из списков.

Списки класса List

На базе класса List вы можете сделать список другого типа, который допускает выбор не только одного, но и нескольких элементов. В отличие от списка, созданного на базе класса Choice, список класса List может занимать прямоугольную область, в которой помещаются сразу несколько элементов. Этот список всегда находится в раскрытом состоянии (рис. 5.8).



Рис. 5.8. Список класса List, все элементы которого помещаются в окне списка

Если размеры окна списка класса List недостаточны для того чтобы вместить в себя все элементы, в правой части окна списка автоматически создается полоса просмотра, с помощью которой можно пролистать весь список (рис. 5.9).

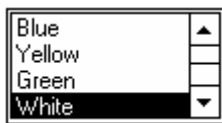


Рис. 5.9. Список класса List с полосой просмотра

Описание класса List

В классе List определено два конструктора и довольно много различных методов. Ниже мы привели краткое описание класса List:

```
public class java.awt.List
{
    extends java.awt.Component

    // -----
    // Конструкторы
    // -----

    // Конструктор без параметров
    public List();

    // Конструктор, позволяющий указать количество
    // отображаемых строк и флаг одновременного
    // выбора нескольких элементов
    public List(int rows, boolean multipleSelections);

    // -----
    // Методы
    // -----

    // Добавление элемента в список
    public void addItem(String item);

    // Добавление элемента в список с указанием номера позиции
    public void addItem(String item, int index);

    // Вызов метода createList
    public void addNotify();

    // Переключение списка в режим, при котором возможно
    // выбирать одновременно несколько элементов
    public boolean allowsMultipleSelections();

    // Удаление из списка всех элементов
    public void clear();

    // Определение количества элементов в списке
    public int countItems();

    // Удаление элемента из заданной позиции
    public void delItem(int position);

    // Удаление нескольких элементов
    public void delItems(int start, int end);

    // Отмена выделения элемента с заданной позицией
    public void deselect(int index);

    // Получение строки, связанной с элементом, по
    // позиции этого элемента
    public String getItem(int index);

    // Определение количества элементов, которые
    // видны в окне списка
    public int getRows();

    // Определение номера выделенного элемента
    public int getSelectedIndex();

    // Определение номеров выделенных элементов
```

```

public int[] getSelectedIndexes();

// Получение текстовой строки, связанной с
// выделенным элементом
public String getSelectedItem();

// Получение ссылки на массив строк, связанных
// с выделенными элементами
public String[] getSelectedItems();

// Определение номера элемента массива, который
// был сделан в последний раз выделенным
// с помощью метода makeVisible
public int getVisibleIndex();

// Проверка, является ли выделенной
// строка с заданным номером
public boolean isSelected(int index);

// Выполняется свертка элементов списка таким
// образом, чтобы элемент с заданным номером
// стал видимым
public void makeVisible(int index);

// Минимальные размеры области, необходимые
// для отображения списка
public Dimension minimumSize();

// Минимальные размеры области, необходимые
// для отображения списка с заданным
// количеством строк
public Dimension minimumSize(int rows);

// Получение строки параметров
protected String paramString();

// Предпочтительные размеры области, необходимые
// для отображения списка
public Dimension preferredSize();

// Предпочтительные размеры области, необходимые
// для отображения списка с заданным
// количеством строк
public Dimension preferredSize(int rows);

// Извещение об уничтожении узла
public void removeNotify();

// Замещение элемента списка с заданным номером
public void replaceItem(String newValue, int index);

// Выделение элемента с заданным номером
public void select(int index);

// Установка или сброс режима одновременного
// выделения нескольких строк
public void setMultipleSelections(boolean v);
}

```

Процесс создания списка класса List несложен:

```

List chBackgroundColor;
chBackgroundColor = new List(6, false);

```

При создании списка вы передаете конструктору количество одновременно отображаемых строк и флаг разрешения одновременного выбора нескольких строк. Если значение этого флага равно true, пользователь сможет выбирать из списка одновременно несколько строк, а если false - только одну строку.

Для наполнения списка вы можете использовать уже знакомый вам метод addItem:

```

chBackgroundColor.addItem("Yellow");
chBackgroundColor.addItem("Green");
chBackgroundColor.addItem("White");

```

Список класса List добавляется к окну апплета методом add:
`add(chBackgroundColor);`

Кратко остановимся на нескольких методах класса List.

Если вы разрешили пользователю выбирать из списка одновременно несколько элементов, то для получения ссылки на массив выбранных элементов вам пригодятся методы `getSelectedItems` и `getSelectedIndexes`:

```
public String[] getSelectedItems();
public int[] getSelectedIndexes();
```

С помощью метода `setMultipleSelections` вы можете динамически включать или выключать режим одновременного выбора нескольких элементов.

В некоторых случаях вам может пригодиться метод `clear`, удаляющий все элементы из списка:

```
public void clear();
```

Методика использования других методов очевидна из краткого описания класса `List`, приведенного в этом разделе.

Обработка событий от списка класса `List`

В отличие от списка класса `Choice`, для выбора строки (или нескольких строк) из списка класса `List`, пользователь должен сделать двойной щелчок левой клавишей мыши по выделенному элементу (или элементам, если выделено несколько элементов). При этом событие можно обработать переопределенным методом `action`, как мы это делали для списка класса `Choice`.

Однако список класса `List` создает события не только при двойном щелчке, но и при выделении или отмены выделения элементов, сделанном пользователем одинарным щелчком клавиши мыши. Апплет может перехватывать и обрабатывать такие события, переопределив метод `handleEvent`. Пример такой обработки вы найдете в исходных текстах приложения `ListBox`.

Приложение `ListBox`

В окне приложения `ListBox` мы создали два списка класса `List`, первый из которых предназначен для выбора цвета фона, а второй - для выбора цвета изображения. Размер первого списка достаточен для отображения всех добавленных в него элементов. Размер второго списка специально сделан меньше, поэтому справа от него появилась полоса просмотра (рис. 5.10).

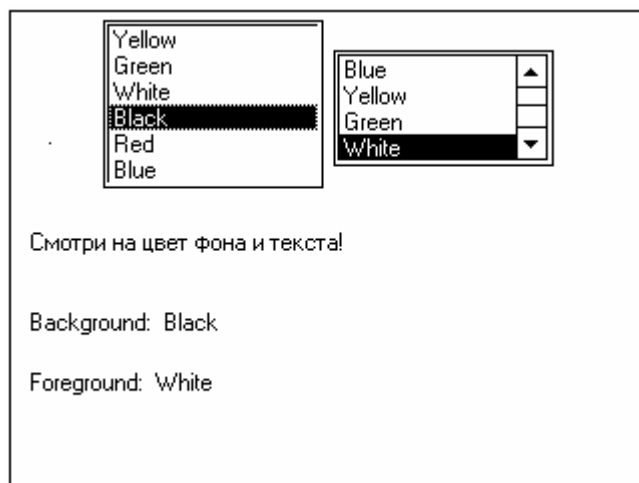


Рис. 5.10. Окно приложения `ListBox`

Если вы будете выделять различные строки списков одинарным щелчком клавиши мыши, в нижней части окна и строке состояния навигатора (на рисунке не показана) появятся названия выделенных цветов. Таким образом, апплет отслеживает выделение элементов этих двух списков, отображая связанные с ним текстовые строки.

В том случае, если вы сделаете двойной щелчок мышью внутри одного из списков, соответствующим образом изменится цвет фона или цвет изображения (текстовых строк и рамки вокруг окна апплета).

Исходные тексты приложения

Файл исходных текстов приложения `ListBox` приведен в листинге 5.7.

Листинг 5.7. Файл `ListBox\ListBox.java`

```
// =====
// Списки типа List Box
//
// (C) Фролов А.В., 1997
//
// E-mail: frolov@glas.apc.org
// WWW:   http://www.glasnet.ru/~frolov
//        или
//        http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
```

```

import java.awt.*;

public class ListBox extends Applet
{
    // Создаем ссылки на объекты класса List
    List chBackgroundColor;
    List chForegroundColor;

    // Строки для хранения названий выбираемого цвета
    String sSelBackground = new String("Yellow");
    String sSelForeground = new String("Black");

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: ListBox\r\n" +
            "Author: Alexandr Frolov\r\n" +
            "E-mail: frolov@glas.apc.org" +
            "WWW:      http://www.glasnet.ru/~frolov" +
            "Created with Microsoft Visual J++ Version 1.0";
    }

    // -----
    // init
    // Метод, получающий управление при инициализации апплета
    // -----
    public void init()
    {
        // Создаем списки для выбора цвета фона и
        // цвета изображения
        chBackgroundColor = new List(6, false);
        chForegroundColor = new List(4, false);

        // Добавляем списки в окно апплета
        add(chBackgroundColor);
        add(chForegroundColor);

        // Заполняем список цвета фона
        chBackgroundColor.addItem("Yellow");
        chBackgroundColor.addItem("Green");
        chBackgroundColor.addItem("White");
        chBackgroundColor.addItem("Black");
        chBackgroundColor.addItem("Red");
        chBackgroundColor.addItem("Blue");

        // Заполняем список цвета изображения
        chForegroundColor.addItem("Black");
        chForegroundColor.addItem("Red");
        chForegroundColor.addItem("Blue");
        chForegroundColor.addItem("Yellow");
        chForegroundColor.addItem("Green");
        chForegroundColor.addItem("White");

        // Устанавливаем цвет фона
        setBackground(Color.yellow);

        // Устанавливаем цвет изображения
        setForeground(Color.black);
    }

    // -----
    // action
    // Метод вызывается, когда пользователь выполняет
    // действие над компонентами
    // -----
    public boolean action(Event evt, Object obj)
    {
        // Переменная для хранения ссылки на список,
        // вызвавший событие
        List ch;

        // Получаем ссылку на список
        ch = (List)evt.target;
    }
}

```

```

// Проверяем, что событие вызвано списком, а не
// другим компонентом
if(evt.target instanceof List)
{
    // Выполняем ветвление по спискам

    // Список цвета фона
    if(evt.target.equals(chBackgroundColor))
    {
        // Получаем номер текущего элемента списка
        // и устанавливаем соответствующий
        // цвет фона
        if(ch.getSelectedIndex() == 0)
            setBackground(Color.yellow);

        else if(ch.getSelectedIndex() == 1)
            setBackground(Color.green);

        else if(ch.getSelectedIndex() == 2)
            setBackground(Color.white);

        else if(ch.getSelectedIndex() == 3)
            setBackground(Color.black);

        else if(ch.getSelectedIndex() == 4)
            setBackground(Color.red);

        else if(ch.getSelectedIndex() == 5)
            setBackground(Color.blue);
    }

    // Список цвета изображения
    else if(evt.target.equals(chForegroundColor))
    {
        // Получаем номер текущего элемента списка
        // и устанавливаем соответствующий
        // цвет изображения
        if(ch.getSelectedIndex() == 0)
            setForeground(Color.black);

        else if(ch.getSelectedIndex() == 1)
            setForeground(Color.red);

        else if(ch.getSelectedIndex() == 2)
            setForeground(Color.blue);

        else if(ch.getSelectedIndex() == 3)
            setForeground(Color.yellow);

        else if(ch.getSelectedIndex() == 4)
            setForeground(Color.green);

        else if(ch.getSelectedIndex() == 5)
            setForeground(Color.white);
    }

    // Если событие возникло от неизвестного списка,
    // мы его не обрабатываем
    else
    {
        return false;
    }

    // Перерисовываем окно апплета
    repaint();

    // возвращаем признак того, что мы обработали событие
    return true;
}

// Если событие вызвано не кнопкой,
// мы его не обрабатываем
return false;
}

// -----
// handleEvent

```

```

// Обработка событий
// -----
public boolean handleEvent(Event evt)
{
    // Переменная для хранения ссылки на список
    List ls;

    // Нас интересуют события, возникающие
    // только при выделении нового элемента списка
    if(evt.id == Event.LIST_SELECT)
    {
        // Получаем ссылку на список
        ls = (List)evt.target;

        // Получаем текущий выделенный цвет фона
        if(evt.target.equals(chBackgroundColor))
            sSelBackground = ls.getSelectedItemAt();

        // Получаем текущий выделенный цвет изображения
        else if(evt.target.equals(chForegroundColor))
            sSelForeground = ls.getSelectedItemAt();

        // Пишем цвет фона и изображения в строке
        // состояния навигатора
        showStatus("(" + sSelBackground
            + ", " + sSelForeground + ")");

        // Перерисовываем окно
        repaint();

        // Возвращаем признак того, что мы обработали
        // событие самостоятельно
        return true;
    }

    // Для тех событий, которые мы не обрабатываем,
    // вызываем метод handleEvent из базового класса
    else
        return super.handleEvent(evt);
}

// -----
// paint
// Метод paint, выполняющий рисование в окне апплета
// -----
public void paint(Graphics g)
{
    // Определяем текущие размеры окна апплета
    Dimension dimAppWndDimension = size();

    // Рисуем рамку вокруг окна апплета
    g.drawRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);

    // Рисуем строку
    g.drawString("Смотри на цвет фона и текста!", 10, 120);

    // Отображаем текущий выделенный цвет фона
    // и изображения
    g.drawString("Background: " + sSelBackground, 10, 160);
    g.drawString("Foreground: " + sSelForeground, 10, 190);
}
}

```

Исходный текст документа HTML, в который встроен апплет ListBox, представлен в листинге 5.8.

Листинг 5.8. Файл ListBox\ListBox.html

```

<html>
<head>
<title>ListBox</title>
</head>
<body>
<hr>
<applet
    code=ListBox.class
    id=ListBox

```

```

width=320
height=240 >
</applet>
<hr>
<a href="ListBox.java">The source.</a>
</body>
</html>

```

Описание исходного текста

В классе ListBox мы добавили четыре поля и переопределили несколько методов.

Поля класса ListBox

В нашем классе мы определили два поля для хранения ссылок на списки цвета фона и цвета изображения, а также две строки для хранения названий выбираемых цветов:

```

List chBackgroundColor;
List chForegroundColor;
String sSelBackground = new String("Yellow");
String sSelForeground = new String("Black");

```

Содержимое строк sSelBackground и sSelForeground изменяется в процессе выделения пользователем различных строк списков.

Метод getAppletInfo

Метод getAppletInfo возвращает информацию об апплете ListBox.

Метод init

В методе init мы создаем два списка как объекты класса List:

```

chBackgroundColor = new List(6, false);
chForegroundColor = new List(4, false);

```

Первый из этих списков способен одновременно отображать шесть строк, поэтому в нем поместились все шесть цветов для фона. Вертикальный размер второго списка меньше. В результате он снабжается полосой прокрутки. Оба списка не предназначены для одновременного выбора нескольких элементов, поэтому в качестве второго параметра мы передаем конструктору List значение false.

Созданные списки добавляются в окно апплета методом add:

```

add(chBackgroundColor);
add(chForegroundColor);

```

Сразу после добавления списков мы их заполняем, вызывая для соответствующих объектов метод addItem:

```

chBackgroundColor.addItem("Yellow");
chBackgroundColor.addItem("Green");
chBackgroundColor.addItem("White");
chBackgroundColor.addItem("Black");
chBackgroundColor.addItem("Red");
chBackgroundColor.addItem("Blue");

chForegroundColor.addItem("Black");
chForegroundColor.addItem("Red");
chForegroundColor.addItem("Blue");
chForegroundColor.addItem("Yellow");
chForegroundColor.addItem("Green");
chForegroundColor.addItem("White");

```

Затем метод выбирает для фона желтый цвет, а для изображения - черный:

```

setBackground(Color.yellow);
setForeground(Color.black);

```

Метод action

Метод action обрабатывает событие, вызванное списками, - выбор элемента из списка.

Прежде всего, метод action сохраняет ссылку на список, в котором произошел выбор, в переменной ch:

```

List ch;
ch = (List)evt.target;

```

Далее выполняется проверка факта, что событие вызвано именно списком класса List, а затем обрабатываются события, созданные списками:

```

if (evt.target.equals(chBackgroundColor))
{
    if (ch.getSelectedIndex() == 0)
        setBackground(Color.yellow);
    else if (ch.getSelectedIndex() == 1)
        setBackground(Color.green);
    else if (ch.getSelectedIndex() == 2)
        setBackground(Color.white);
    else if (ch.getSelectedIndex() == 3)

```

```

        setBackground(Color.black);
    else if(ch.getSelectedIndex() == 4)
        setBackground(Color.red);
    else if(ch.getSelectedIndex() == 5)
        setBackground(Color.blue);
}

else if(evt.target.equals(chForegroundColor))
{
    if(ch.getSelectedIndex() == 0)
        setForeground(Color.black);
    else if(ch.getSelectedIndex() == 1)
        setForeground(Color.red);
    else if(ch.getSelectedIndex() == 2)
        setForeground(Color.blue);
    else if(ch.getSelectedIndex() == 3)
        setForeground(Color.yellow);
    else if(ch.getSelectedIndex() == 4)
        setForeground(Color.green);
    else if(ch.getSelectedIndex() == 5)
        setForeground(Color.white);
}
else
    return false;

```

С помощью метода `getSelectedIndex` метод `action` определяет номер выбранного элемента списка, устанавливая соответствующим образом цвет фона или изображения. Затем метод перерисовывает окно апплета, вызывая метод `repaint`.

Метод *handleEvent*

Для того чтобы отследить выделение элементов списка, наш апплет переопределил метод `handleEvent`, обеспечив обработку события с идентификатором `Event.LIST_SELECT`.

Переопределение метода `handleEvent` нужно делать внимательно, так как этот метод вызывается при возникновении разных событий, например, при перемещении мыши в окне апплета. Если ваш метод `handleEvent` не обрабатывает какое-либо событие, он должен передать его одноименному методу из базового класса.

Наш метод `handleEvent` прежде всего проверяет код события, обрабатывая только события `Event.LIST_SELECT`, которые создаются при выделении пользователем элементов списка:

```

if(evt.id == Event.LIST_SELECT)
{
    . . .
}
else
    return super.handleEvent(evt);

```

Если событие подлежит обработке, наш метод `handleEvent` получает ссылку на объект, вызвавший событие, и сохраняет ее в переменной `ls` типа `List`:

```

List ls;
ls = (List)evt.target;

```

Затем метод определяет, какой список создал событие, проверяя поле `evt.target`, а затем получает и записывает выделенную строку в переменную `sSelBackground` (для списка цветов фона) или `sSelForeground` (для списка цветов изображения):

```

if(evt.target.equals(chBackgroundColor))
    sSelBackground = ls.getSelectedIndex();
else if(evt.target.equals(chForegroundColor))
    sSelForeground = ls.getSelectedIndex();

```

После этого цвет фона и изображения записывается в строку состояния навигатора в формате (<цвет фона>, <цвет изображения>):

```

showStatus("(" + sSelBackground + ", " + sSelForeground + ")");

```

После этого метод выполняет перерисовку окна и возвращает значение `true` - признак того, что он обработал событие:

```

repaint();
return true;

```

Метод *paint*

Обработчик метода `paint` рисует рамку вокруг окна апплета и текстовую строку в средней части этого окна.

В нижней части окна апплета метод `paint` отображает выделенные в списках цвета фона и изображения:

```

g.drawString("Background:  " + sSelBackground, 10, 160);
g.drawString("Foreground:  " + sSelForeground, 10, 190);

```

Текстовое поле класса Label

На базе класса Label вы можете создать в окне апплета однострочное текстовое поле, которое не поддается редактированию. Основное назначение таких полей - подпись других компонент, таких, например, как группы переключателей или списки.

Ниже мы привели краткое описание класса Label:

```
public class java.awt.Label
{
    extends java.awt.Component

    // -----
    // Поля
    // -----

    // Способ выравнивания текстового поля
    public final static int CENTER; // центрирование
    public final static int LEFT;   // по левой границе
    public final static int RIGHT;  // по правой границе

    // -----
    // Конструкторы
    // -----

    // Создание текстового поля без текста
    public Label();

    // Создание текстового поля с заданным текстом
    public Label(String label);

    // Создание текстового поля с заданным текстом
    // и заданным выравниванием
    public Label(String label, int alignment);

    // -----
    // Методы
    // -----

    // Вызов метода createLabel
    public void addNotify();

    // Определение текущего выравнивания текстового поля
    public int getAlignment();

    // Получение текста из поля
    public String getText();

    // Получение строки параметров
    protected String paramString();

    // Установка выравнивания текстового поля
    public void setAlignment(int alignment);

    // Запись текста в поле
    public void setText(String label);
}
```

Текстовое поле класса Label создается вызовом соответствующего конструктора. Например, ниже мы создали текстовое поле, указав строку, которую надо в него записать:

```
Label lbTextLabel;
lbTextLabel = new Label("Выберите выравнивание");
```

С помощью метода add вы можете добавить текстовое поле в окно апплета:

```
add(lbTextLabel);
```

Метод setAlignment позволяет при необходимости изменить выравнивание текста. Способ выравнивания необходимо указать через единственный параметр метода:

```
lbTextLabel.setAlignment(Label.LEFT);
```

При помощи метода setText вы сможете динамически изменять текст, расположенный в поле класса Label.

Приложение TextLabel

В окне приложения TextLabel, демонстрирующего способы работы с полями класса Label, мы разместили одно такое поле и три кнопки, позволяющие изменять выравнивание текста в поле (рис. 5.11).

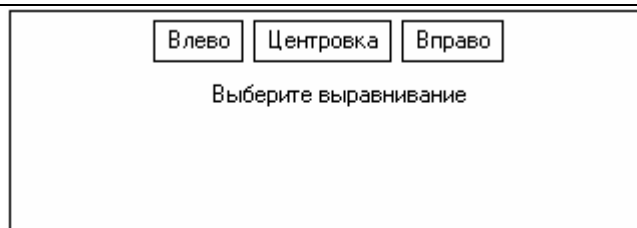


Рис. 5.11. Окно приложения TextLabel

Нажимая кнопки, вы можете заметить, как строка, расположенная под ними, немного сдвигается по горизонтали.

Исходные тексты приложения

Исходный текст приложения TextLabel приведен в листинге 5.9.

Листинг 5.9. Файл TextLabel\TextLabel.java

```
// =====
// Работа с компонентами Label
//
// (C) Фролов А.В, 1997
//
// E-mail: frolov@glas.apc.org
// WWW:    http://www.glasnet.ru/~frolov
//          или
//          http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

public class TextLabel extends Applet
{
    // Создаем ссылку на объекты типа Label
    Label lbTextLabel;

    // Создаем три ссылки на объекты типа Button
    Button btnLeft;
    Button btnCenter;
    Button btnRight;

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: TextLabel\r\n" +
            "Author: Alexandr Frolov\r\n" +
            "E-mail: frolov@glas.apc.org" +
            "WWW:    http://www.glasnet.ru/~frolov" +
            "Created with Microsoft Visual J++ Version 1.0";
    }

    // -----
    // init
    // Метод, получающий управление при инициализации апплета
    // -----
    public void init()
    {
        // Создаем компоненту Label
        lbTextLabel = new Label("Выберите выравнивание");

        // Создаем три кнопки
        btnLeft  = new Button("Влево");
        btnCenter = new Button("Центровка");
        btnRight  = new Button("Вправо");

        // Добавляем три кнопки
        add(btnLeft);
        add(btnCenter);
        add(btnRight);

        // Добавляем компоненту Label
        add(lbTextLabel);
    }
}
```

```

        // Устанавливаем цвет фона
        setBackground(Color.yellow);
    }

    // -----
    // action
    // Метод вызывается, когда пользователь выполняет
    // действие над компонентами
    // -----
    public boolean action(Event evt, Object obj)
    {
        // Ссылка на кнопку, от которой пришло сообщение
        Button btn;

        // Проверяем, что событие вызвано кнопкой, а не
        // другим компонентом
        if(evt.target instanceof Button)
        {
            // Получаем ссылку на кнопку, вызвавшую событие
            btn = (Button)evt.target;

            // Выполняем ветвление по кнопкам
            if(evt.target.equals(btnLeft))
            {
                // Выравниваем текст компоненты Label
                // по левой границе
                lbTextLabel.setAlignment(Label.LEFT);
            }

            else if(evt.target.equals(btnCenter))
            {
                // Центруем текст компоненты Label
                lbTextLabel.setAlignment(Label.CENTER);
            }

            else if(evt.target.equals(btnRight))
            {
                // Выравниваем текст компоненты Label
                // по правой границе
                lbTextLabel.setAlignment(Label.RIGHT);
            }

            // Если событие возникло от неизвестной кнопки,
            // мы его не обрабатываем
            else
            {
                return false;
            }

            // Возвращаем признак того, что мы обработали событие
            return true;
        }

        // Если событие вызвано не кнопкой,
        // мы его не обрабатываем
        return false;
    }

    // -----
    // paint
    // Метод paint, выполняющий рисование в окне апплета
    // -----
    public void paint(Graphics g)
    {
        // Определяем текущие размеры окна апплета
        Dimension dimAppWndDimension = size();

        // Выбираем в контекст отображения черный цвет
        g.setColor(Color.black);

        // Рисуем рамку вокруг окна апплета
        g.drawRect(0, 0,
            dimAppWndDimension.width - 1,
            dimAppWndDimension.height - 1);
    }
}

```


Исходный текст документа HTML, созданного для размещения апплета, представлен в листинге 5.10.

Листинг 5.10. Файл TextLabel\TextLabel.html

```
<html>
<head>
<title>TextLabel</title>
</head>
<body>
<hr>
<applet
    code=TextLabel.class
    id=TextLabel
    width=320
    height=240 >
</applet>
<hr>
<a href="TextLabel.java">The source.</a>
</body>
</html>
```

Описание исходного текста

В классе TextLabel мы определили четыре поля и несколько методов.

Поля класса TextLabel

Мы определили четыре поля - lbTextLabel, btnLeft, btnCenter и btnRight:

```
Label lbTextLabel;
Button btnLeft;
Button btnCenter;
Button btnRight;
```

Первое из них предназначено для хранения ссылки на объект класса Label (однострочное текстовое поле), остальные три - для хранения ссылок на кнопки, определяющие выравнивание.

Метод getAppletInfo

Метод getAppletInfo возвращает информацию о нашем апплете.

Метод init

Метод init создает одно текстовое поле, вызывая конструктор с одним параметром - текстовой строкой:

```
lbTextLabel = new Label("Выберите выравнивание");
```

Далее этот метод создает три кнопки, с помощью которых вы будете изменять выравнивание текста в поле класса Label:

```
btnLeft = new Button("Влево");
btnCenter = new Button("Центровка");
btnRight = new Button("Вправо");
```

Затем созданные кнопки и поле добавляются в окно апплета при помощи метода add:

```
add(btnLeft);
add(btnCenter);
add(btnRight);
add(lbTextLabel);
```

Последнее, что делает метод init перед возвращением управления, это изменение цвета фона:

```
setBackground(Color.yellow);
```

Метод action

Наш метод action обрабатывает только те события, которые вызваны кнопками. Проверка источника события выполняется так же, как и раньше, поэтому мы не будем на этом останавливаться.

Что же касается установки выравнивания, то она выполняется при помощи метода setAlignment:

```
if (evt.target.equals(btnLeft))
    lbTextLabel.setAlignment(Label.LEFT);
else if (evt.target.equals(btnCenter))
    lbTextLabel.setAlignment(Label.CENTER);
else if (evt.target.equals(btnRight))
    lbTextLabel.setAlignment(Label.RIGHT);
else
    return false;
```

Метод paint

Единственное, что делает метод paint, - это рисование рамки черного цвета вокруг окна апплета.

Текстовое поле класса TextField

Для редактирования одной строки текста вы можете создать текстовое поле на базе класса TextField, которое несложно в использовании. Класс TextField создан на базе другого класса с именем

TextComponent, поэтому при работе с текстовым полем класса TextField вы можете использовать и методы класса TextComponent.

Приведем краткое описание класса TextField:

```
public class java.awt.TextField
    extends java.awt.TextComponent
{
    // -----
    // Конструкторы
    // -----

    // Создание поля без текста
    public TextField();

    // Создание поля без текста с заданной шириной
    public TextField(int cols);

    // Создание поля и инициализация его текстом
    public TextField(String text);

    // Создание поля заданной ширины
    // и инициализация его текстом
    public TextField(String text, int cols);

    // -----
    // Методы
    // -----

    // Вызов метода createTextField
    public void addNotify();

    // Проверка, установлен ли для поля эхо-символ
    public boolean echoCharIsSet();

    // Определение размера поля
    public int getColumns();

    // Получение текущего эхо-символа
    public char getEchoChar();

    // Определение минимальных размеров области
    // для отображения поля
    public Dimension minimumSize();

    // Определение минимальных размеров области
    // для отображения поля заданной ширины
    public Dimension minimumSize(int cols);

    // Получение строки параметров
    protected String paramString();

    // Определение оптимальных размеров области
    // для отображения поля
    public Dimension preferredSize();

    // Определение оптимальных размеров области
    // для отображения поля заданной ширины
    public Dimension preferredSize(int cols);

    // Установка эхо-символа для отображения в поле
    public void setEchoCharacter(char c);
}
```

При создании текстового поля вы можете выбрать один из четырех конструкторов, соответственно, для создания поля без текста и без указания размера, без текста заданного размера, для создания поля с текстом и для создания поля с текстом указанного размера.

Вот фрагмент кода, в котором создается поле с текстом, имеющее ширину, достаточную для размещения 35 символов:

```
TextField txt;
txt = new TextField("Введите строку текста", 35);
```

Созданное поле добавляется в окно апплета методом add.

Большинство самых полезных методов, необходимых для работы с полем класса TextField, определено в классе TextComponent, краткое описание которого мы привели ниже:

```
public class java.awt.TextComponent
    extends java.awt.Component
{
    // -----
```

```
// Методы
// -----

// Получение текста, выделенного пользователем
// в окне поля
public String getSelectedText();

// Получение позиции конца выделенной области
public int getSelectionEnd();

// Получение позиции начала выделенной области
public int getSelectionStart();

// Получение полного текста из поля
public String getText();

// Проверка, возможно ли редактирование
// текста в поле
public boolean isEditable();

// Получение строки параметров
protected String paramString();

// Удаление извещения
public void removeNotify();

// Выделение заданной области текста
public void select(int selStart, int selEnd);

// Выделение всего текста
public void selectAll();

// Включение или выключение возможности
// редактирования текста
public void setEditable(boolean t);

// Установка текста в поле
public void setText(String t);
}
```

С помощью метода `getText` вы можете получить весь текст, который имеется в поле. Метод `getSelectedText` позволяет получить только ту часть текста, которая предварительно была выделена пользователем.

Приложение может выделить любой фрагмент текста или весь текст при помощи методов `select` и `selectAll`, соответственно.

Для записи текста в поле приложение может воспользоваться методом `setText`.

Возможно, для вас будет интересен метод `setEditable`, позволяющий переключать текстовое поле из режима, при котором редактирование заблокировано, в режим с разрешенным редактированием и обратно.

Приложение TxtField

В приложении `TxtField` мы создали однострочное поле редактирования на базе класса `TextField` и кнопку, с помощью которой можно извлечь текст из поля для отображения (рис. 5.12).

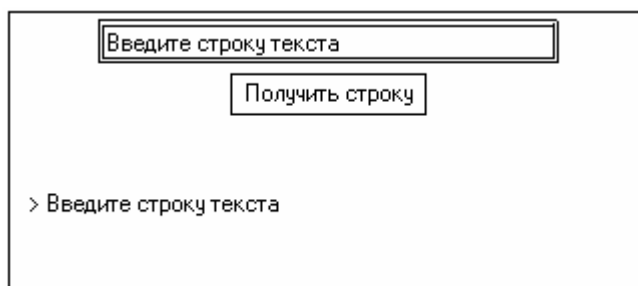


Рис. 5.12. Окно апплета `TxtField`

Изменив текст в поле редактирования, нажмите кнопку “Получить строку”. В нижней части окна апплета вы увидите измененный вами текст.

Исходные тексты приложения

Исходные тексты приложения `TxtField` представлены в листинге 5.11.

Листинг 5.11. Файл TxtField\TxtField.java

```
// =====
// Однострочное текстовое поле класса TextField
//
// (C) Фролов А.В, 1997
//
// E-mail: frolov@glas.apc.org
// WWW:    http://www.glasnet.ru/~frolov
//         или
//         http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

public class TxtField extends Applet
{
    // Создаем ссылку на объекты типа TextField
    TextField txt;

    // Создаем ссылку на объекты типа Button
    Button btnGetText;

    // Строка для хранения введенных данных
    String str;

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: TxtField\r\n" +
            "Author: Alexandr Frolov\r\n" +
            "E-mail: frolov@glas.apc.org" +
            "WWW:    http://www.glasnet.ru/~frolov" +
            "Created with Microsoft Visual J++ Version 1.0";
    }

    // -----
    // init
    // Метод, получающий управление при инициализации апплета
    // -----
    public void init()
    {
        // Создаем редактируемое однострочное текстовое поле
        txt = new TextField("Введите строку текста", 35);

        // Создаем кнопку, с помощью которой можно получить
        // содержимое текстового поля
        btnGetText = new Button("Получить строку");

        // Добавляем текстовое поле в окно апплете
        add(txt);

        // Добавляем кнопку в окно апплете
        add(btnGetText);

        // Получаем и сохраняем текущий текст,
        // установленный в поле
        str = txt.getText();

        // Устанавливаем цвет фона
        setBackground(Color.yellow);
    }

    // -----
    // action
    // Метод вызывается, когда пользователь выполняет
    // действие над компонентами
    // -----
    public boolean action(Event evt, Object obj)
    {
        // Ссылка на кнопку, от которой пришло сообщение
        Button btn;

        // Проверяем, что событие вызвано кнопкой, а не
```

```

// другим компонентом
if(evt.target instanceof Button)
{
    // Получаем ссылку на кнопку, вызвавшую событие
    btn = (Button)evt.target;

    // Проверяем ссылку на кнопку
    if(evt.target.equals(btnGetText))
    {
        // Получаем и сохраняем текущий текст,
        // установленный в поле
        str = txt.getText();

        // Перерисовываем окно апплета
        repaint();
    }

    // Если событие возникло от неизвестной кнопки,
    // мы его не обрабатываем
    else
    {
        return false;
    }

    // Возвращаем признак того, что мы обработали событие
    return true;
}

// Если событие вызвано не кнопкой,
// мы его не обрабатываем
return false;
}

// -----
// paint
// Метод paint, выполняющий рисование в окне апплета
// -----
public void paint(Graphics g)
{
    // Определяем текущие размеры окна апплета
    Dimension dimAppWndDimension = size();

    // Выбираем в контекст отображения черный цвет
    g.setColor(Color.black);

    // Рисуем рамку вокруг окна апплета
    g.drawRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);

    // Рисуем строку, полученную из текстового поля
    g.drawString("> " + str, 10, 100);
}
}

```

Документ HTML, созданный для нашего апплета, представлен в листинге 5.12.

Листинг 5.12. Файл TxtField\TxtField.html

```

<html>
<head>
<title>TxtField</title>
</head>
<body>
<hr>
<applet
    code=TxtField.class
    id=TxtField
    width=320
    height=240 >
</applet>
<hr>
<a href="TxtField.java">The source.</a>
</body>
</html>

```

Описание исходного текста

В классе TxtField мы определили три поля и несколько методов.

Поля класса TxtField

В поле txt хранится ссылка на объект класса TextField - наше однострочное поле редактирования:

```
TextField txt;
```

В полях btnGetText и str хранятся, соответственно, ссылки на кнопку и текстовую строку, в которую записывается текущее содержимое поля редактирования:

```
Button btnGetText;
```

```
String str;
```

Метод getAppletInfo

Метод getAppletInfo возвращает информацию о нашем апплете.

Метод init

Метод init создает одно текстовое поле редактирования, вызывая конструктор с параметром в виде текстовой строки:

```
txt = new TextField("Введите строку текста", 35);
```

Далее этот метод создает кнопку, с помощью которой можно получить текущее содержимое поля редактирования:

```
btnGetText = new Button("Получить строку");
```

Затем созданные поле и кнопка добавляются в окно апплета при помощи метода add:

```
add(txt);
```

```
add(btnGetText);
```

После этого метод init получает текущее содержимое поля редактирования и записывает его в строку str:

```
str = txt.getText();
```

В завершении метод init изменяет цвет фона:

```
setBackground(Color.yellow);
```

Метод action

Наш метод action обрабатывает только те события, которые вызваны кнопкой.

Обработка заключается в извлечении текста из поля редактирования и записи его в строку str:

```
str = txt.getText();
```

```
repaint();
```

Метод paint

После рисования рамки черного цвета вокруг окна апплета метод paint отображает текущее содержимое строки str в нижней части окна:

```
g.drawString("> " + str, 10, 100);
```

Многострочное текстовое поле класса TextArea

Если вам нужно поле для ввода многострочной информации, обратите внимание на класс TextArea. С его помощью вы можете создать многострочное поле заданной ширины и высоты, снабженное полосами просмотра.

Класс TextArea создан на базе класса TextComponent, рассмотренном нами ранее, поэтому для работы с многострочными полями вы можете использовать методы этого класса. В частности, вам доступен метод, с помощью которого можно получать из окна редактирования не весь текст, а только выделенную пользователем область.

Краткое описание класса TextArea мы привели ниже:

```
public class java.awt.TextArea
    extends java.awt.TextComponent
{
    // -----
    // Конструкторы
    // -----

    // Создание поля без текста и без указания размеров
    public TextArea();

    // Создание поля без текста с указанием размеров
    public TextArea(int rows, int cols);

    // Создание поля с текстом без указания размеров
    public TextArea(String text);

    // Создание поля с текстом и с указанием размеров
    public TextArea(String text, int rows, int cols);

    // -----
}
```

```

// Методы
// -----

// Вызов метода createTextArea
public void addNotify();

// Добавление текста в поле редактирования
public void appendText(String str);

// Определение количества столбцов поля
public int getColumns();

// Определение количества строк поля
public int getRows();

// Добавление текста в поле редактирования
// начиная с заданной позиции
public void insertText(String str, int pos);

// Определение минимальных размеров области
// для размещения многострочного текстового поля
public Dimension minimumSize();

// Определение минимальных размеров области
// для размещения многострочного текстового поля
// с заданным количеством строк и столбцов
public Dimension minimumSize(int rows, int cols);

// Получение строки параметров
protected String paramString();

// Определение предпочтительных размеров области
// для размещения многострочного текстового поля
public Dimension preferredSize();

// Определение предпочтительных размеров области
// для размещения многострочного текстового поля
// с заданным количеством строк и столбцов
public Dimension preferredSize(int rows, int cols);

// Замещение блока текста, начиная с первой позиции
// и до второй позиции
public void replaceText(String str, int start, int end);
}

```

Когда вы создаете многострочное текстовое поле редактирования, то можете использовать конструктор, допускающий указание размеров поля в строках и столбцах:

```

TextArea txt;
txt = new TextArea("Введите строку текста", 5, 35);

```

Созданное поле добавляется в окно апплета методом add.

Отметим, что в классе TextArea есть методы для работы с блоками текста (вставка и замена), а также методы, с помощью которых можно определить количество строк и столбцов в поле редактирования.

Приложение TextEdit

Приложение TextEdit (рис. 5.13) демонстрирует некоторые приемы работы с многострочным полем редактирования текста, созданным на базе класса TextArea.



Рис. 5.13. Окно приложения TextEdit

В окне редактирования вы можете вводить строки текста. Если нажать на кнопку “Получить все”, в нижней части окна отобразится полное содержимое окна редактирования. Каждая строка будет отделена символом перехода на новую строку.

Если же нажать кнопку “Получить выделенное”, в нижней части появится только выделенный фрагмент текста (как это показано на рис. 5.13).

Исходные тексты приложения

Исходный текст приложения приведен в листинге 5.13.

Листинг 5.13. Файл TextEdit\TextEdit.java

```
// =====
// Многострочное текстовое поле класса TextArea
//
// (C) Фролов А.В., 1997
//
// E-mail: frolov@glas.apc.org
// WWW:   http://www.glasnet.ru/~frolov
//        или
//        http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

public class TextEdit extends Applet
{
    // Создаем ссылку на объект типа TextArea
    TextArea txt;

    // Создаем ссылку на объекты типа Button
    Button btnGetText;
    Button btnGetSelectedText;

    String str;

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: TextEdit\r\n" +
            "Author: Alexandr Frolov\r\n" +
            "E-mail: frolov@glas.apc.org" +
            "WWW:   http://www.glasnet.ru/~frolov" +
            "Created with Microsoft Visual J++ Version 1.0";
    }

    // -----
    // init
    // Метод, получающий управление при инициализации апплета
    // -----
    public void init()
    {
        // Создаем редактируемое многострочное текстовое поле
```

```

txt = new TextArea("Введите строку текста", 5, 35);

// Создаем кнопку, с помощью которой можно получить
// полное содержимое текстового поля
btnGetText = new Button("Получить все");

// Создаем кнопку, с помощью которой можно получить
// содержимое выделенной области текстового поля
btnGetSelectedText = new Button("Получить выделенное");

// Добавляем текстовое поле в окно апплета
add(txt);

// Добавляем кнопки в окно апплета
add(btnGetText);
add(btnGetSelectedText);

// Получаем и сохраняем текущий текст,
// установленный в поле
str = txt.getText();

// Устанавливаем цвет фона
setBackground(Color.yellow);
}

// -----
// action
// Метод вызывается, когда пользователь выполняет
// действие над компонентами
// -----
public boolean action(Event evt, Object obj)
{
    // Ссылка на кнопку, от которой пришло сообщение
    Button btn;

    // Проверяем, что событие вызвано кнопкой, а не
    // другим компонентом
    if(evt.target instanceof Button)
    {
        // Получаем ссылку на кнопку, вызвавшую событие
        btn = (Button)evt.target;

        // Проверяем ссылку на кнопку
        if(evt.target.equals(btnGetText))
        {
            // Получаем и сохраняем текущий текст,
            // установленный в поле
            str = txt.getText();

            // Перерисовываем окно апплета
            repaint();
        }

        else if(evt.target.equals(btnGetSelectedText))
        {
            // Получаем и сохраняем выделенную область
            str = txt.getSelectedText();

            // Перерисовываем окно апплета
            repaint();
        }

        // Если событие возникло от неизвестной кнопки,
        // мы его не обрабатываем
        else
        {
            return false;
        }

        // Возвращаем признак того, что мы обработали событие
        return true;
    }

    // Если событие вызвано не кнопкой,
    // мы его не обрабатываем
    return false;
}

```

```
// -----
// paint
// Метод paint, выполняющий рисование в окне апплета
// -----
public void paint(Graphics g)
{
    // Определяем текущие размеры окна апплета
    Dimension dimAppWndDimension = size();

    // Выбираем в контекст отображения черный цвет
    g.setColor(Color.black);

    // Рисуем рамку вокруг окна апплета
    g.drawRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);

    // Рисуем строку, полученную из текстового поля
    g.drawString("> " + str, 10, 150);
}
}
```

Исходный текст документа HTML, созданного для апплета, приведен в листинге 5.14.

Листинг 5.14. Файл TextEdit\TextEdit.html

```
<html>
<head>
<title>TextEdit</title>
</head>
<body>
<hr>
<applet
    code=TextEdit.class
    id=TextEdit
    width=320
    height=240 >
</applet>
<hr>
<a href="TextEdit.java">The source.</a>
</body>
</html>
```

Описание исходного текста

В классе TextEdit мы определили четыре поля и несколько методов.

Поля класса TxtField

В поле txt хранится ссылка на объект класса TextArea - многострочное поле редактирования:

```
TextArea txt;
```

В полях btnGetText, btnGetSelectedText и str хранятся, соответственно, ссылки на кнопки и текстовую строку, в которую записывается текущее содержимое поля редактирования:

```
Button btnGetText;
Button btnGetSelectedText;
String str;
```

Метод getAppletInfo

Метод getAppletInfo возвращает информацию о нашем апплете.

Метод init

Метод init создает одно текстовое поле редактирования, вызывая конструктор следующего вида:

```
txt = new TextArea("Введите строку текста", 5, 35);
```

Здесь создается поле из 5 строк и 35 столбцов.

Далее этот метод создает кнопки, с помощью которых можно получить текущее содержимое всего поля редактирования и области, выделенной пользователем:

```
btnGetText = new Button("Получить все");
btnGetSelectedText = new Button("Получить выделенное");
```

Затем созданные поле и кнопки добавляются в окно апплета при помощи метода add:

```
add(txt);
add(btnGetText);
add(btnGetSelectedText);
```

После этого метод init получает текущее содержимое поля редактирования и записывает его в строку str, а затем изменяет цвет фона:

```
str = txt.getText();
setBackground(Color.yellow);
```

Метод action

Наш метод action обрабатывает только те события, которые вызваны кнопками.

Обработка заключается в извлечении текста из поля редактирования и записи его в строку str. В зависимости от того, какая кнопка была нажата, извлекается либо весь текст, либо только выделенный фрагмент:

```
if (evt.target.equals(btnGetText))
{
    str = txt.getText();
    repaint();
}
else if (evt.target.equals(btnGetSelectedText))
{
    str = txt.getSelectedText();
    repaint();
}
else
    return false;
```

Для извлечения всего текста мы вызываем метод getText, а для извлечения выделенного фрагмента - метод getSelectedText.

После записи извлеченного текста метод action перерисовывает окно апплета, вызывая метод repaint.

Метод paint

После рисования рамки черного цвета вокруг окна апплета метод paint отображает текущее содержимое строки str в нижней части окна:

```
g.drawString("> " + str, 10, 100);
```

6 НАСТРОЙКА СИСТЕМЫ LAYOUT MANAGER

В предыдущей главе мы рассказали вам о том, как создавать компоненты и размещать их в контейнере. Однако предложенный способ размещения компонент в окне контейнера едва ли можно назвать удобным, так как заранее трудно предугадать, на каком месте окажется тот или иной орган управления.

К счастью, имеются способы, позволяющие контролировать размещение отдельных компонент в окне контейнера. И хотя эти способы не позволяют задавать конкретные координаты и размеры органов управления, использованные схемы размещения компонент будут правильно работать на любой аппаратной платформе (не забывайте, что Java создавалась как средство разработки приложений, способных выполняться на любой платформе).

В чем трудность создания пользовательского интерфейса для мультиплатформных систем?

В том, что разработчик приложения никогда не знает характеристики устройства отображения, установленные у пользователя. Он, в частности, не может заранее знать разрешение монитора, размер системного шрифта и другие характеристики, необходимые для компоновки диалоговых панелей в терминах абсолютных координат.

Средства пользовательского интерфейса AWT способны динамически изменять размеры компонент, подгоняя их “по месту” в системе пользователя. В результате значительно повышается вероятность того, что внешний вид диалоговой панели, в каком она предстанет перед пользователем, будет похож на то, что ожидал разработчик.

Как мы уже говорили в начале предыдущей главы, расположением компонент внутри окна контейнера (например, внутри окна апплета) управляет система Layout Manager. Способ, которым она это делает, весьма непривычен для тех, кто создавал приложения Windows. Выбор этого способа обоснован необходимостью обеспечения совместимости с различными компьютерными платформами.

Режимы системы Layout Manager

Прежде чем мы рассмотрим различные режимы компоновки системы Layout Manager, вспомним, как происходит наследование класса Applet (рис. 6.1).

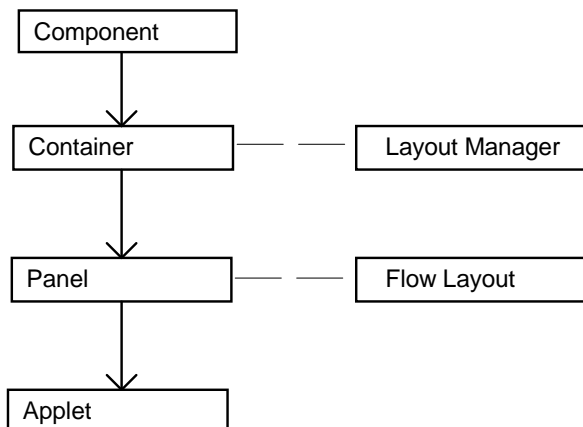


Рис. 6.1. Наследование класса Applet

Класс Applet наследуется от класса Panel, который, в свою очередь, наследуется от класса Container и Component. Класс Container пользуется интерфейсом LayoutManager, что позволяет выбирать для контейнеров один из нескольких режимов размещения компонент в окне контейнера.

Что же касается класса Panel, то для него по умолчанию выбирается режим размещения компонент с названием Flow Layout. Разумеется, вы можете выбрать другой режим размещения, указав его явным образом.

Ниже мы перечислили все возможные режимы системы Layout Manager:

Режим размещения компонент	Описание
FlowLayout	Компоненты заполняют окно контейнера “поток” по мере их добавления методом add. Они размещаются слева направо и сверху вниз
GridLayout	Компоненты размещаются в виде таблицы по мере добавления слева направо и сверху вниз. Для этой таблицы можно указать количество столбцов и строк
GridBagLayout	Аналогично предыдущему, однако при добавлении компонент в таблицу можно указать координаты ячейки, в которую помещается компонента

BorderLayout	При размещении компоненты указывается одно из нескольких направлений: юг, север, запад, восток, центр. Направление определяется относительно центра окна контейнера
CardLayout	Размещение компонент друг над другом в одном окне. Этот режим позволяет организовать набор диалоговых панелей в виде блокнота

Каждому режиму соответствует одноименный класс, методы и конструкторы которого позволяют выбирать различные компоновки.

Далее на примере конкретных приложений мы рассмотрим использование перечисленных выше режимов системы Layout Manager.

Режим FlowLayout

В этом режиме мы добавляли компоненты во всех примерах апплетов, приведенных в предыдущей главе, так как по умолчанию для апплетов используется именно режим FlowLayout.

Ниже мы приведем краткое описание класса FlowLayout:

```
public class java.awt.FlowLayout
    extends java.lang.Object
    implements java.awt.LayoutManager
{
    // -----
    // Поля
    // -----

    // Способы выравнивания
    public final static int CENTER; // центрирование
    public final static int LEFT; // по левой границе
    public final static int RIGHT; // по правой границе

    // -----
    // Конструкторы
    // -----

    // Без указания выравнивания и зазора между компонентами
    public FlowLayout();

    // С указанием выравнивания
    public FlowLayout(int align);

    // С указанием выравнивания и зазора между компонентами
    // по вертикали и горизонтали
    public FlowLayout(int align, int hgap, int vgap);

    // -----
    // Методы
    // -----

    // Не используется
    public void addLayoutComponent(String name,
        Component comp);

    // Предназначен для того чтобы компоненты могли
    // установить для себя предпочтительный размер
    public void layoutContainer(Container target);

    // Определение минимального размера окна контейнера,
    // необходимого для размещения всех компонент
    public Dimension minimumLayoutSize(Container target);

    // Определение предпочтительного размера окна контейнера,
    // необходимого для размещения всех компонент
    public Dimension preferredLayoutSize(Container target);

    // Удаление компоненты из контейнера
    public void removeLayoutComponent(Component comp);

    // Получение строки названия метода компоновки
    public String toString();
}
```

Обычно приложения не вызывают методы класса FlowLayout, устанавливая варианты компоновки при помощи конструкторов.

Первый конструктор класса FlowLayout, не имеющий параметров, устанавливает по умолчанию режим центрирования компонент и зазор между компонентами по вертикали и горизонтали, равный 5 пикселям.

Именно этот режим и использовался во всех наших апплетах из предыдущей главы, так как именно он применяется по умолчанию объектами класса `Panel`, от которого наследуется класс `Applet`.

С помощью второго конструктора вы можете выбрать режим размещения с заданным выравниванием компонент в окне контейнера по горизонтали. В качестве параметров этому конструктору необходимо передавать значения `FlowLayout.LEFT`, `FlowLayout.RIGHT`, или `FlowLayout.CENTER`. Зазор между компонентами будет при этом равен по умолчанию 5 пикселям.

И, наконец, третий конструктор допускает отдельное указание режима выравнивания, а также зазоров между компонентами по вертикали и горизонтали в пикселях.

Режим `GridLayout`

В режиме `GridLayout` компоненты размещаются в ячейках таблицы, параметры которой можно задать с помощью конструкторов класса `GridLayout`:

```
public class java.awt.GridLayout
    extends java.lang.Object
    implements java.awt.LayoutManager
{
    // -----
    // Конструкторы
    // -----

    // Создание таблицы с заданным
    // количеством строк и столбцов
    public GridLayout(int rows, int cols);

    // Создание таблицы с заданным количеством строк и
    // столбцов и с заданным зазором между компонентами
    public GridLayout(int rows, int cols, int hgap, int vgap);

    // -----
    // Методы
    // -----
    public void addLayoutComponent(String name,
        Component comp);
    public void layoutContainer(Container target);
    public Dimension minimumLayoutSize(Container target);
    public Dimension preferredLayoutSize(Container target);
    public void removeLayoutComponent(Component comp);
    public String toString();
}
```

При размещении компонент внутри ячеек таблицы все они получают одинаковые размеры. Если один из параметров, задающих размерность таблицы, равен нулю, это означает, что соответствующий столбец или строка может содержать любое количество элементов.

Заметим, что оба параметра `rows` и `cols` не могут быть равны нулю одновременно.

Приложение `Grid`

Приложение `Grid` демонстрирует использование режима размещения компонент `GridLayout`. В окне апплета мы создали таблицу размером 3x3, разместив в ее ячейках восемь кнопок (рис. 6.2).

Button 1	Button 2	Button 3
Button 4	Button 5	Button 6
Button 7	Button 8	

Рис. 6.2. Окно приложения `Grid`

Обратите внимание, что все кнопки в окне апплета имеют одинаковый размер. Если вы станете нажимать на них, в строке состояния навигатора будет появляться название нажатой кнопки (на рис. 6.2 не показано).

Исходные тексты приложения

Исходный текст апплета Grid приведен в листинге 6.1.

Листинг 6.1. Файл Grid\Grid.java

```
// =====
// Режим компоновки GridLayout
//
// (C) Фролов А.В., 1997
//
// E-mail: frolov@glas.apc.org
// WWW:    http://www.glasnet.ru/~frolov
//         или
//         http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

public class Grid extends Applet
{
    // Создаем ссылки на объекты типа Button
    Button btn1;
    Button btn2;
    Button btn3;
    Button btn4;
    Button btn5;
    Button btn6;
    Button btn7;
    Button btn8;

    // Строка для записи названия нажатой кнопки
    String sTextLabel;

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: Grid\r\n" +
            "Author: Alexandr Frolov\r\n" +
            "E-mail: frolov@glas.apc.org" +
            "WWW:    http://www.glasnet.ru/~frolov" +
            "Created with Microsoft Visual J++ Version 1.1";
    }

    // -----
    // init
    // Метод, получающий управление при инициализации апплета
    // -----
    public void init()
    {
        // Устанавливаем желтый цвет фона
        setBackground(Color.yellow);

        // Создаем кнопки
        btn1 = new Button("Button 1");
        btn2 = new Button("Button 2");
        btn3 = new Button("Button 3");
        btn4 = new Button("Button 4");
        btn5 = new Button("Button 5");
        btn6 = new Button("Button 6");
        btn7 = new Button("Button 7");
        btn8 = new Button("Button 8");

        // Устанавливаем режим GridLayout
        setLayout(new GridLayout(3, 3));

        // Добавляем кнопки в контейнер
        add(btn1);
        add(btn2);
        add(btn3);
        add(btn4);
```

```

        add(btn5);
        add(btn6);
        add(btn7);
        add(btn8);
    }

    // -----
    // action
    // Метод вызывается, когда пользователь выполняет
    // действие над компонентами
    // -----
    public boolean action(Event evt, Object obj)
    {
        // Ссылка на кнопку, от которой пришло сообщение
        Button btn;

        // Проверяем, что событие вызвано кнопкой, а не
        // другим компонентом
        if(evt.target instanceof Button)
        {
            // Получаем ссылку на кнопку, вызвавшую событие
            btn = (Button)evt.target;

            // Получаем название кнопки
            sTextLabel = btn.getLabel();

            // Записываем название кнопки
            // в строку состояния навигатора
            showStatus("Button (\\"" + sTextLabel + "\\") pressed");

            // возвращаем признак того, что мы обработали событие
            return true;
        }

        // Если событие вызвано не кнопкой,
        // мы его не обрабатываем
        return false;
    }

    // -----
    // paint
    // Метод paint, выполняющий рисование в окне апплета
    // -----
    public void paint(Graphics g)
    {
        // Определяем текущие размеры окна апплета
        Dimension dimAppWndDimension = size();

        // Выбираем в контекст отображения черный цвет
        g.setColor(Color.black);

        // Рисуем рамку вокруг окна апплета
        g.drawRect(0, 0,
            dimAppWndDimension.width - 1,
            dimAppWndDimension.height - 1);
    }
}

```

Листинг 6.2 содержит исходный текст документа HTML, в который встроен апплет.

Листинг 6.2. Файл Grid\Grid.html

```

<html>
<head>
<title>Grid</title>
</head>
<body>
<hr>
<applet
    code=Grid.class
    id=Grid
    width=320
    height=240 >
</applet>
<hr>
<a href="Grid.java">The source.</a>
</body>
</html>

```


Описание исходного текста

В исходном тексте приложения Grid вы сможете разобраться самостоятельно. Заметим только, что в методе `init` мы вызываем метод `setLayout`, передавая ему объект класса `GridLayout`:

```
setLayout(new GridLayout(3, 3));
```

В свою очередь, параметры конструктора объекта `GridLayout` определяют, что для размещения компонент будет использована таблица, состоящая из трех строк и трех столбцов.

Компоненты (кнопки) добавляются в окно апплета хорошо известным вам способом - с помощью метода `add`:

```
add(btn1);
add(btn2);
add(btn3);
add(btn4);
add(btn5);
add(btn6);
add(btn7);
add(btn8);
```

Как видно из рис. 6.2, таблица заполняется кнопками слева направо и сверху вниз, как это и предполагает режим компоновки `GridLayout`.

Режим BorderLayout

При использовании режима `BorderLayout` окно контейнера разделяется на рамку и центральную часть. При размещении компонент указывается направление от центра окна, в котором слудует размещать компоненты.

Ниже приведено краткое описание класса `BorderLayout`:

```
public class java.awt.BorderLayout
    extends java.lang.Object
    implements java.awt.LayoutManager
{
    // -----
    // Конструктор
    // -----
    public BorderLayout();
    public BorderLayout(int hgap, int vgap);

    // -----
    // Методы
    // -----
    public void addLayoutComponent(String name,
        Component comp);
    public void layoutContainer(Container target);
    public Dimension minimumLayoutSize(Container target);
    public Dimension preferredLayoutSize(Container target);
    public void removeLayoutComponent(Component comp);
    public String toString();
}
```

Два конструктора предназначены для создания схемы размещения, соответственно, без зазора между компонентами и с зазором заданной величины.

Добавляя компоненты к контейнеру, вы должны использовать метод `add` с двумя параметрами, первый из которых указывает направление размещения, а второй - ссылку на добавляемый объект:

```
add("North", btn1);
add("East", btn2);
add("West", btn3);
add("South", btn4);
add("Center", btn5);
```

Приложение Border

В приложении `Border` создается пять кнопок, которые размещаются в режиме `BorderLayout` (рис. 6.3).

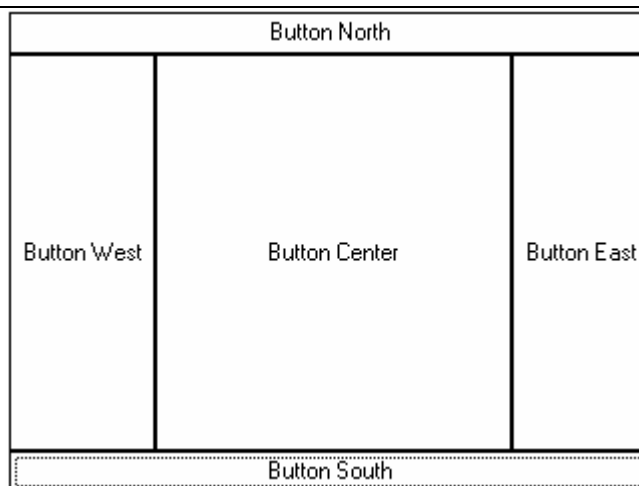


Рис. 6.3. Окно апплета Border

Заметьте, что в процессе размещения северная и южная кнопка (верхняя и нижняя на нашем рисунке) заняли по ширине все окно апплета. Высота же этих кнопок была установлена достаточной для размещения одной строки текста.

Восточная и западная кнопка (левая и правая) имеют ширину, достаточную для размещения текста, и высоту, равную высоте области, которая осталась от северной и южной кнопок.

Для центральной кнопки было выделено все оставшееся пространство в центре окна апплета.

Исходные тексты приложения

Исходный текст приложения Border приведен в листинге 6.3.

Листинг 6.3. Файл Border\Border.java

```
// =====
// Режим компоновки BorderLayout
//
// (C) Фролов А.В, 1997
//
// E-mail: frolov@glas.apc.org
// WWW:   http://www.glasnet.ru/~frolov
//        или
//        http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

public class Border extends Applet
{
    // Создаем ссылки на объекты типа Button
    Button btn1;
    Button btn2;
    Button btn3;
    Button btn4;
    Button btn5;

    // Строка для записи названия нажатой кнопки
    String sTextLabel;

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: Grid\r\n" +
            "Author: Alexandr Frolov\r\n" +
            "E-mail: frolov@glas.apc.org" +
            "WWW:   http://www.glasnet.ru/~frolov" +
            "Created with Microsoft Visual J++ Version 1.1";
    }

    // -----
    // init
    // Метод, получающий управление при инициализации апплета
    // -----
    public void init()
```

```

{
    // Устанавливаем желтый цвет фона
    setBackground(Color.yellow);

    // Создаем кнопки
    btn1 = new Button("Button North");
    btn2 = new Button("Button East");
    btn3 = new Button("Button West");
    btn4 = new Button("Button South");
    btn5 = new Button("Button Center");

    // Устанавливаем режим GridLayout
    setLayout(new BorderLayout());

    // Добавляем кнопки в контейнер
    add("North", btn1);
    add("East", btn2);
    add("West", btn3);
    add("South", btn4);
    add("Center", btn5);
}

// -----
// action
// Метод вызывается, когда пользователь выполняет
// действие над компонентами
// -----
public boolean action(Event evt, Object obj)
{
    // Ссылка на кнопку, от которой пришло сообщение
    Button btn;

    // Проверяем, что событие вызвано кнопкой, а не
    // другим компонентом
    if(evt.target instanceof Button)
    {
        // Получаем ссылку на кнопку, вызвавшую событие
        btn = (Button)evt.target;

        // Получаем название кнопки
        sTextLabel = btn.getLabel();

        // Записываем название кнопки
        // в строку состояния навигатора
        showStatus("Button (\\"" + sTextLabel + "\") pressed");

        // возвращаем признак того, что мы обработали событие
        return true;
    }

    // Если событие вызвано не кнопкой,
    // мы его не обрабатываем
    return false;
}

// -----
// paint
// Метод paint, выполняющий рисование в окне апплета
// -----
public void paint(Graphics g)
{
    // Определяем текущие размеры окна апплета
    Dimension dimAppWndDimension = size();

    // Выбираем в контекст отображения черный цвет
    g.setColor(Color.black);

    // Рисуем рамку вокруг окна апплета
    g.drawRect(0, 0,
        dimAppWndDimension.width - 1,
        dimAppWndDimension.height - 1);
}
}

```

Документ HTML, который был создан для нашего апплета, вы найдете в листинге 6.4.

Листинг 6.4. Файл Border\Border.html

```

<html>
<head>
<title>Border</title>
</head>
<body>
<hr>
<applet
    code=Border.class
    id=Border
    width=320
    height=240 >
</applet>
<hr>
<a href="Border.java">The source.</a>
</body>
</html>

```

Описание исходного текста

Все самое интересное в приложении Border происходит в методе init, поэтому мы ограничимся описанием только этого метода.

Прежде всего метод init устанавливает желтый цвет фона:

```
setBackground(Color.yellow);
```

И хотя весь фон в нашем апплете закрыт кнопками, вы можете изменить это, выбрав конструктор класса BorderLayout, допускающий задание зазора между компонентами по вертикали и горизонтали.

Далее метод init создает пять кнопок для размещения в окне апплета. Здесь тоже для вас нет ничего нового:

```

btn1 = new Button("Button North");
btn2 = new Button("Button East");
btn3 = new Button("Button West");
btn4 = new Button("Button South");
btn5 = new Button("Button Center");

```

Далее мы устанавливаем режим размещения компонент в окне контейнера, вызывая для этого метод setLayout:

```
setLayout(new BorderLayout());
```

В качестве параметра методу setLayout передается ссылка на только что созданный объект класса BorderLayout. Так как выбран конструктор класса BorderLayout без параметров, зазор между компонентами будет отсутствовать.

Добавление компонент выполняется методом add с указанием направления расположения компоненты:

```

add("North", btn1);
add("East", btn2);
add("West", btn3);
add("South", btn4);
add("Center", btn5);

```

Заметим, что нельзя размещать несколько компонент в одном и том же направлении.

Режим CardLayout

Режим CardLayout предназначен для создания набора диалоговых панелей, которые можно показывать по очереди в одном окне прямоугольной формы. Обычно для управления процессом перебора диалоговых панелей в режиме CardLayout используются отдельные органы управления, расположенные в другой панели или даже в другом апплете на той же самой странице сервера WWW. Пример такого апплета мы приведем после того как рассмотрим использование класса Panel.

Класс CardLayout содержит два конструктора и несколько методов:

```

public class java.awt.CardLayout
    extends java.lang.Object
    implements java.awt.LayoutManager
{
    // -----
    // Конструкторы
    // -----

    // Режим без зазоров
    public CardLayout();

    // Режим с зазорами по вертикали и горизонтали
    // между компонентами и окном контейнера
    public CardLayout(int hgap, int vgap);

    // -----
    // Методы
    // -----
}

```

```

// Добавление компоненты с указанием имени
public void addLayoutComponent(String name,
    Component comp);

// Отображение первой страницы блокнота
public void first(Container target);

// Отображение последней страницы блокнота
public void last(Container target);

// Отображение следующей страницы блокнота
public void next(Container target);

// Отображение предыдущей страницы блокнота
public void previous(Container target);

// Выполнение размещения компонент
public void layoutContainer(Container target);

// Определение минимальных размеров окна,
// необходимых для размещения компонент
public Dimension minimumLayoutSize(Container target);

// Определение предпочтительных размеров окна,
// необходимых для размещения компонент
public Dimension preferredLayoutSize(Container target);

// Удаление заданной компоненты
public void removeLayoutComponent(Component comp);

// Отображение произвольной страницы блокнота
// по ее имени
public void show(Container target, String name);

// Получение текстовой строки названия режима размещения
public String toString();
}

```

Как пользоваться режимом размещения CardLayout?

Обычно в окне апплета создается две панели, одна из которых предназначена для отображения страниц блокнота в режиме размещения CardLayout, а вторая содержит органы управления перелистыванием страниц, например, кнопки.

Такие методы, как first, last, next и previous позволяют отображать, соответственно, первую, последнюю, следующую и предыдущую страницу блокнота. Если вызвать метод next при отображении последней страницы, в окне появится первая страница. Аналогично, при вызове метода previous для первой страницы блокнота вы увидите последнюю страницу.

А как отобразить произвольную страницу, не перебирая их по одной методами next и previous?

Для этого существует метод show. Учтите, что этот метод позволяет отображать только такие страницы, при добавлении которых методом add было указано имя, например:

```

pCardPanel.add("BackgroundColor", pBackgroundColor);
pCardPanel.add("ForegroundColor", pForegroundColor);
pCardPanel.add("Font", pFont);

```

Здесь в панель pCardPanel добавляются панели pBackgroundColor, pForegroundColor и pFont, имеющие имена, соответственно, "BackgroundColor", "ForegroundColor" и "Font".

Режим GridBagLayout

Режим GridBagLayout намного сложнее только что описанного режима GridLayout. Он позволяет размещать компоненты разного размера в таблице, задавая при этом для отдельных компонент размеры отступов и количество занимаемых ячеек.

В нашей книге мы не будем рассматривать этот режим, так как сходные результаты могут быть достигнуты другими, менее сложными способами. Например, вы можете создать в контейнере несколько панелей, используя внутри каждой свой метод размещения компонент.

Если вы создаете апплеты для размещения в документах HTML, никто не заставляет вас ограничиваться только одним апплетом для одного документа HTML - вы можете разместить там произвольное количество апплетов, организовав взаимодействие с одной стороны, между отдельными апплетами, а с другой - между апплетами и расширениями сервера WWW.

Тех, кого интересует режим GridBagLayout, мы адресуем к документации, которая входит в комплект Microsoft Visual J++, а также к книге Д. Родли "Создание Java-апплетов", которая издана на русском языке.

В интегрированной системе разработки приложений Java Microsoft Visual J++ версии 1.1 имеется система автоматизированного проектирования пользовательского интерфейса, в результате работы которой создаются исходные тексты классов. Размещение органов управления при этом выполняется интерактивными средствами, аналогичными средствам разработки диалоговых панелей для приложений

Microsoft Windows. В следующем томе “Библиотеки системного программиста”, посвященном Java, мы научим вас пользоваться этой системой.

7 РАБОТА С ПАНЕЛЯМИ

Панели, создаваемые на базе класса `Panel`, являются мощным средством организации диалогового интерфейса. Так как класс `Panel` произошел от класса `Container`, панель может содержать компоненты и другие панели. Для каждой панели можно определить режим размещения компонент, что позволяет создавать достаточно сложный пользовательский интерфейс.

В окне апплета вы можете создать несколько панелей, разделяющих его на части. В свою очередь, пространство, занимаемое панелями, также может быть разделено с использованием одного из описанных выше режимов размещения (рис. 7.1).



Рис. 7.1. Размещение нескольких панелей в окне апплета

Отдельные панели могут содержать в себе такие компоненты, как кнопки, переключатели, списки, текстовые поля и так далее.

Создание панелей

Панель создается очень просто. Прежде всего необходимо выбрать для окна апплета схему размещения компонент, соответствующую требуемому расположению панелей. Например, для создания в окне апплета двух панелей, разделяющих его по горизонтали, следует выбрать режим `GridLayout`:

```
setLayout(new GridLayout(2, 1));
```

Панели будут размещаться в ячейках таблицы, состоящей из одного столбца и двух строк.

Далее нужно создать объекты класса `Panel`:

```
Panel pTopPanel;  
pTopPanel = new Panel();  
Panel pBottomPanel;  
pBottomPanel = new Panel();
```

Ссылка на панель, которая будет располагаться сверху, записывается в переменную `pTopPanel`, а на ту, что будет располагаться снизу - в переменную `pBottomPanel`.

Добавление панелей

Создав панели, вы можете добавить их в окно апплета, вызвав метод `add`, как это показано ниже:

```
add(pTopPanel);  
add(pBottomPanel);
```

Заметим, что вы можете добавлять панели в панели, указывая, для какой панели нужно вызывать метод `add`:

```
Panel pLeft;  
Panel pRight;  
pLeft = new Panel();  
pRight = new Panel();  
pTopPanel.setLayout(new GridLayout(1, 2));  
pTopPanel.add(pLeft);  
pTopPanel.add(pRight);
```

Здесь мы создали две панели `pLeft` и `pRight`, которые по нашему замыслу должны разделить пространство панели `pTopPanel` на две части по вертикали. Для обеспечения вертикального размещения панелей `pLeft` и `pRight` в панели `pTopPanel` мы вызвали для панели `pTopPanel` метод `setLayout`. При этом мы указали, что компоненты, добавляемые в эту панель, должны размещаться в таблице, состоящей из одной строки и двух столбцов.

Затем панели `pLeft` и `pRight` были добавлены в панель `pTopPanel` методом `add`.

Добавление компонент в панели

Для добавления компонент в панель вы должны указать, для какой панели вызывается метод `add`, например:

```
Button btn1;
Button btn2;
btn1 = new Button();
btn2 = new Button();
pBottomPanel.add(btn1);
pBottomPanel.add(btn2);
```

Рисование в окне панели

Как вы знаете, для того чтобы что-нибудь нарисовать, необходимо вначале получить контекст отображения. Методу `paint` передается контекст отображения, связанный с окном апплета. Если в окне имеются панели, то для рисования внутри них необходимо получить контекст отображения окон панелей.

Проще всего это сделать с помощью метода `getGraphics`, вызвав его для объекта класса `Panel`:

```
Graphics gpDraw;
gpDraw = pDraw.getGraphics();
```

Здесь в переменную `gpDraw` мы записали ссылку на контекст отображения для панели `pDraw`.

Получив контекст отображения, можно приступить к рисованию. Вот, например, как можно нарисовать вокруг панели тонкую рамку:

```
Dimension dimAppWndDimension = pDraw.size();
gpDraw.drawRect(0, 0, dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);
```

В этом фрагменте кода мы вначале определили размеры панели, вызвав для нее метод `size`, а затем при помощи метода `drawRect`, вызванного для контекста отображения `gpDraw`, нарисовали рамку.

Для установки шрифта и рисования текста в окне панели вы также должны указывать ссылку на контекст отображения вашей панели:

```
gpDraw.setFont(new Font("Courier", Font.PLAIN, 12));
gpDraw.drawString("Текст внутри окна панели", 10, 50);
```

Другой способ основан на создании собственного класса на базе класса `Panel` и переопределения в этом классе метода `paint`. Мы рассмотрим его позже в разделе "Переопределение класса `Panel`".

Приложение PanelDemo

В приложении `PanelDemo` мы создаем две панели, расположенные горизонтально. Первая из них используется как блокнот, на каждой странице которого находится кнопка, вторая содержит две управляющие кнопки, позволяющие перебирать страницы блокнота по очереди (рис. 7.2).

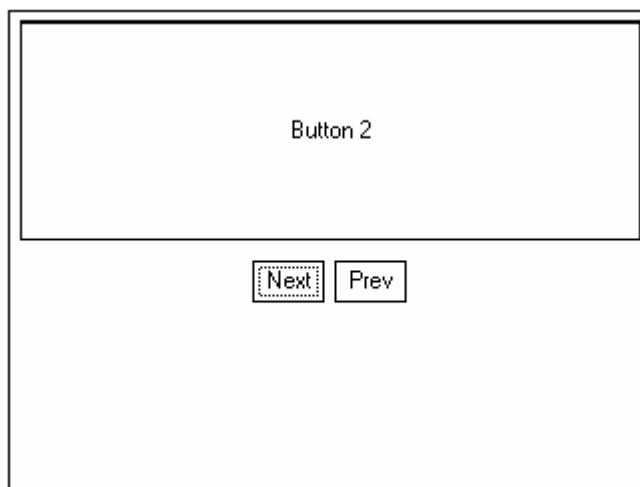


Рис. 7.2. Окно апплета `PanelDemo`

Объемное изображение схемы расположения панелей и кнопок относительно окна апплета показано на рис. 7.3.

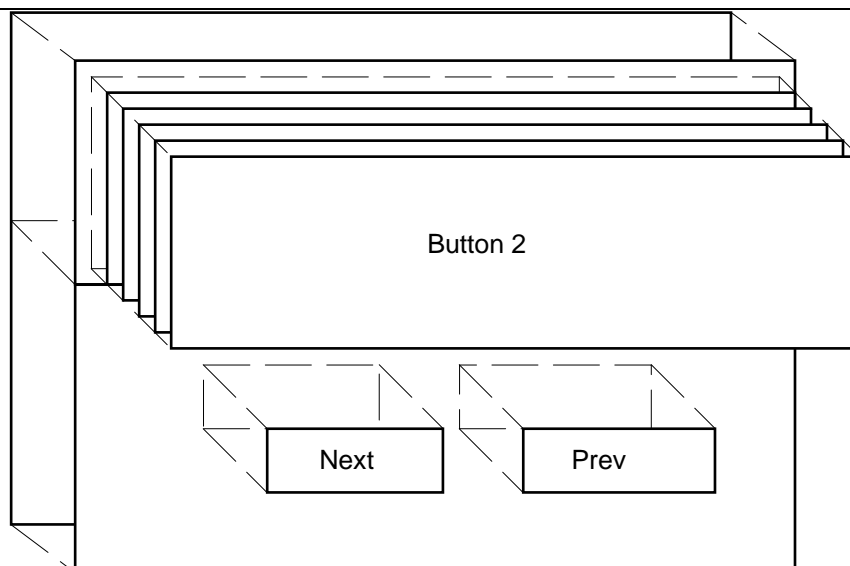


Рис. 7.3. Объемное изображение схемы расположения панелей и кнопок

В верхней панели друг над другом располагаются пять кнопок (как колода карт), причем видна только одна из них. В нижней панели только две кнопки, с помощью которых можно выдвигать на передний план по очереди все кнопки из верхней панели.

Нажимая на кнопки Next и Prev, попробуйте понажимать на кнопки в верхней панели. В строке состояния навигатора при этом будет отображаться название нажатой кнопки (на рис. 6.5 это не показано).

Исходные тексты приложения

Файл исходного текста приложения PanelDemo представлен в листинге 7.1.

Листинг 7.1. Файл PanelDemo\PanelDemo.java

```
// =====
// Работа с панелями класса Panel
//
// (C) Фролов А.В., 1997
//
// E-mail: frolov@glas.apc.org
// WWW:   http://www.glasnet.ru/~frolov
//        или
//        http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

public class PanelDemo extends Applet
{
    // Панель для размещения блокнота
    Panel pCardPanel;

    // Панель для размещения кнопок управления блокнотом
    Panel pButtonPanel;

    // Создаем ссылки на объекты типа Button
    Button btn1;
    Button btn2;
    Button btn3;
    Button btn4;
    Button btn5;
    Button btnNext;
    Button btnPrev;

    // Строка для записи названия нажатой кнопки
    String sTextLabel;

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: Grid\r\n" +
```

```

        "Author: Alexandr Frolov\r\n" +
        "E-mail: frolov@glas.apc.org" +
        "WWW:      http://www.glasnet.ru/~frolov" +
        "Created with Microsoft Visual J++ Version 1.1";
    }

    // -----
    // init
    // Метод, получающий управление при инициализации апплета
    // -----
    public void init()
    {
        // Устанавливаем желтый цвет фона
        setBackground(Color.yellow);

        // Создаем в окне апплета две панели, разделяющие
        // окно по горизонтали. В верхней панели будет
        // находиться блокнот,
        // в нижней - кнопки управления блокнотом
        setLayout(new GridLayout(2, 1));

        // Создаем кнопки блокнота
        btn1 = new Button("Button 1");
        btn2 = new Button("Button 2");
        btn3 = new Button("Button 3");
        btn4 = new Button("Button 4");
        btn5 = new Button("Button 5");

        // Создаем панель блокнота
        pCardPanel = new Panel();

        // Устанавливаем режим размещения для блокнота
        pCardPanel.setLayout(new CardLayout(5, 5));

        // Добавляем кнопки в блокнот
        pCardPanel.add(btn1);
        pCardPanel.add(btn2);
        pCardPanel.add(btn3);
        pCardPanel.add(btn4);
        pCardPanel.add(btn5);

        // Добавляем панель блокнота в окно апплета
        add(pCardPanel);

        // Создаем кнопки управления блокнотом

        // Кнопка просмотра следующей страницы блокнота
        btnNext = new Button("Next");

        // Кнопка просмотра предыдущей страницы блокнота
        btnPrev = new Button("Prev");

        // Создаем панель кнопок управления блокнотом
        pButtonPanel = new Panel();

        // Устанавливаем режим размещения для панели кнопок
        pButtonPanel.setLayout(new FlowLayout());

        // Добавляем кнопки в панель кнопок
        pButtonPanel.add(btnNext);
        pButtonPanel.add(btnPrev);

        // Добавляем панель кнопок
        add(pButtonPanel);
    }

    // -----
    // action
    // Метод вызывается, когда пользователь выполняет
    // действие над компонентами
    // -----
    public boolean action(Event evt, Object obj)
    {
        // Ссылка на кнопку, от которой пришло сообщение
        Button btn;

        // Проверяем, что событие вызвано кнопкой, а не

```

```

// другим компонентом
if(evt.target instanceof Button)
{
    // Получаем ссылку на кнопку, вызвавшую событие
    btn = (Button)evt.target;

    // Получаем название кнопки
    sTextLabel = btn.getLabel();

    // Записываем название кнопки
    // в строку состояния навигатора
    showStatus("Button (\\"" + sTextLabel + "\\") pressed");

    // Выполняем ветвление по кнопкам. Для каждой кнопки
    // записываем ее название
    // в строку состояния навигатора
    if(evt.target.equals(btnNext))
    {
        // Выбираем следующую страницу в блокноте
        ((CardLayout)pCardPanel.getLayout()).next(pCardPanel);
    }

    else if(evt.target.equals(btnPrev))
    {
        // Выбираем предыдущую страницу в блокноте
        ((CardLayout)pCardPanel.getLayout()).previous(pCardPanel);
    }

    else if(evt.target.equals(btn1))
    {
        showStatus("Button 1 (\\"" + sTextLabel
            + "\\") pressed");
    }

    else if(evt.target.equals(btn2))
    {
        showStatus("Button 2 (\\"" + sTextLabel
            + "\\") pressed");
    }

    else if(evt.target.equals(btn3))
    {
        showStatus("Button 3 (\\"" + sTextLabel
            + "\\") pressed");
    }

    else if(evt.target.equals(btn4))
    {
        showStatus("Button 4 (\\"" + sTextLabel
            + "\\") pressed");
    }

    // Если событие возникло от неизвестной кнопки,
    // мы его не обрабатываем
    else
    {
        return false;
    }

    // Возвращаем признак того, что мы обработали событие
    return true;
}

// Если событие вызвано не кнопкой,
// мы его не обрабатываем
return false;
}
}

```

В листинге 7.2 вы найдете исходный текст документа HTML, созданного для размещения нашего апплета.

Листинг 7.2. Файл PanelDemo\PanelDemo.html

```

<html>
<head>
<title>PanelDemo</title>
</head>

```

```

<body>
<hr>
<applet
    code=PanelDemo.class
    id=PanelDemo
    width=320
    height=240 >
</applet>
<hr>
<a href="PanelDemo.java">The source.</a>
</body>
</html>

```

Описание исходного текста

Приведем описание полей и методов, определенных в нашем апплете.

Поля класса *PanelDemo*

В поле `pCardPanel` хранится ссылка на панель блокнота, страницы которого содержать кнопки. Эта панель располагается в верхней части окна апплета.

Поле `pButtonPanel` предназначено для хранения ссылки на панель кнопок, предназначенных для перелистывания страниц блокнота `pCardPanel`.

Ссылки на кнопки, расположенные на страницах блокнота, хранятся в полях `btn1`, `btn2`, `btn3`, `btn4` и `btn5`.

Ссылки на кнопки, предназначенные для перелистывания страниц блокнота, записываются в поля `btnNext` и `btnPrev` (соответственно, кнопка пролистывания блокнота в прямом и обратном направлении).

В поле `sTextLabel` хранится название нажатой кнопки. Это название отображается в строке состояния навигатора, когда пользователь нажимает какую-либо кнопку в верхней или нижней панели.

Метод *getAppletInfo*

Метод `getAppletInfo` возвращает информацию об апплете и не имеет никаких особенностей.

Метод *init*

Метод `init` создает все необходимые панели и добавляет в них компоненты.

В самом начале своей работы метод `init` устанавливает желтый цвет фона для окна апплета:

```
setBackground(Color.yellow);
```

После этого выбирается такой режим размещения компонентов, при котором они добавляются в таблицу, состоящую из двух строк и одного столбца:

```
setLayout(new GridLayout(2, 1));
```

Мы будем добавлять панели, поэтому первая из добавленных панелей окажется в верхней строке этой таблицы и займет верхнюю половину окна апплета, а вторая - нижнюю.

Далее метод `init` создает пять кнопок, которые будут добавлены на страницы блокнота, расположенного в верхней панели:

```

btn1 = new Button("Button 1");
btn2 = new Button("Button 2");
btn3 = new Button("Button 3");
btn4 = new Button("Button 4");
btn5 = new Button("Button 5");

```

После создания кнопок метод `init` приступает к созданию и заполнению панели блокнота. Панель создается при помощи конструктора класса `Panel`:

```
pCardPanel = new Panel();
```

Для этой панели устанавливается режим размещения компонент типа `CardLayout`, причем между границами окна панели и границами окна добавляемых компонент по вертикали и горизонтали оставлен зазор 5 пикселей:

```
pCardPanel.setLayout(new CardLayout(5, 5));
```

После установки режима добавления можно заполнять блокнот кнопками:

```

pCardPanel.add(btn1);
pCardPanel.add(btn2);
pCardPanel.add(btn3);
pCardPanel.add(btn4);
pCardPanel.add(btn5);

```

Здесь мы воспользовались известным вам методом `add`, вызвав его для объекта `pCardPanel`.

Заполненная панель блокнота при помощи все того же метода `add` добавляется в окно апплета:

```
add(pCardPanel);
```

Так как эта панель добавляется в окно апплета первой, она займет верхнюю половину этого окна.

Завершив с панелью блокнота, метод `init` приступает к формированию панели управляющих кнопок.

Вначале метод создает сами управляющие кнопки:

```

btnNext = new Button("Next");
btnPrev = new Button("Prev");

```

Первая из них перелистывает страницы блокнота в прямом направлении, а вторая - в обратном.

Затем создается панель кнопок:

```
pButtonPanel = new Panel();
```

Для панели кнопок мы выбираем режим выравнивания FlowLayout, при котором компоненты добавляются слева направо и сверху вниз:

```
pButtonPanel.setLayout(new FlowLayout());
```

После этого в панель добавляются две управляющие кнопки, предназначенные для перелистывания страниц блокнота:

```
pButtonPanel.add(btnNext);
pButtonPanel.add(btnPrev);
```

На завершающем этапе своей работы метод init добавляет панель управляющих кнопок в окно апплета:

```
add(pButtonPanel);
```

Данная панель добавляется в окно апплета второй по счету, поэтому она будет расположена в нижней половине этого окна.

Метод action

Метод action обрабатывает события, связанные с кнопками, расположенными в обеих панелях.

Вначале метод проверяет, что событие создано кнопкой. Далее идентификатор кнопки, вызвавшей событие, записывается в переменную btn:

```
btn = (Button)evt.target;
```

После этого метод action получает название кнопки, сохраняет его в строке sTextLabel, а затем отображает в строке состояния навигатора:

```
sTextLabel = btn.getLabel();
showStatus("Button (\\" + sTextLabel + "\\") pressed");
```

Далее анализируется ссылка на кнопку. Если была нажата одна из управляющих кнопок, происходит перелистывание страниц блокнота, в прямом или обратном направлении:

```
if(evt.target.equals(btnNext))
{
    ((CardLayout)pCardPanel.getLayout()).next(pCardPanel);
}
else if(evt.target.equals(btnPrev))
{
    ((CardLayout)pCardPanel.getLayout()).previous(pCardPanel);
}
```

Здесь мы с помощью метода getLayout получаем ссылку на интерфейс системы Layout Manager, установленной для панели pCardPanel, а затем, пользуясь полученной ссылкой, вызываем методы next или previous. Обратите также внимание на необходимость явного приведения типа к классу CardLayout, в котором определены указанные методы.

Обработка событий, создаваемых кнопками, которые расположены на страницах блокнота, не имеет никаких особенностей:

```
else if(evt.target.equals(btn1))
{
    showStatus("Button 1 (\\" + sTextLabel
        + "\\") pressed");
}
. . .
else if(evt.target.equals(btn4))
{
    showStatus("Button 4 (\\" + sTextLabel
        + "\\") pressed");
}
```

Название кнопки просто отображается в строке состояния навигатора.

Приложение Notebook

Приложение Notebook служит в качестве более сложного примера техники работы с панелями.

В окне апплета Notebook создаются три панели, расположенные в одном столбце. В верхней панели, имеющей рамку по периметру, рисуется строка текста "Смотри на шрифт, цвет фона и текста!". Средняя панель представляет собой блокнот, предназначенный для выбора цвета фона, цвета изображения и шрифта для верхней панели. И, наконец, нижняя панель содержит кнопки, позволяющие перелистывать страницы блокнота.

На рис. 7.4 показана страница, предназначенная для выбора цвета фона:

Смотри на шрифт, цвет фона и текста!		
Chose Background Color: White		
Background Color	Foreground Color	Set Text Font
Next		Prev

Рис. 7.4. Страница, предназначенная для выбора цвета фона

Нажимая кнопки Background Color, Foreground Color и Set Text Font, вы сможете выдвигать на передний план страницы блокнота, предназначенные, соответственно, для выбора цвета фона, изображения и шрифта, которым отображается текст в верхней панели.

Кнопки Next и Prev работают таким же образом, что и в предыдущем приложении, а именно: если нажать на кнопку Next, произойдет пролистывание страниц в прямом направлении, а если на кнопку Prev - в обратном направлении.

На рис. 7.5 изображена страница блокнота, предназначенная для выбора цвета изображения.

Смотри на шрифт, цвет фона и текста!		
Chose Foreground Color: Black		
Background Color	Foreground Color	Set Text Font
Next		Prev

Рис. 7.5. Страница, предназначенная для выбора цвета изображения

На рис. 7.6 представлена страница, с помощью которой можно выбрать один из нескольких шрифтов для рисования текста в верхней панели.

Смотри на шрифт, цвет фона и текста!		
Chose Font: Courier		
Background Color	Foreground Color	Set Text Font
Next		Prev

Рис. 7.6. Страница, предназначенная для выбора шрифта

Рассмотрим исходные тексты приложения Notebook/

Исходные тексты приложения

Исходный текст приложения Notebook приведен в листинге 7.3.

Листинг 7.3. Файл Notebook\Notebook.java

```
// =====
// Набор диалоговых панелей
//
// (C) Фролов А.В, 1997
//
// E-mail: frolov@glas.apc.org
// WWW:    http://www.glasnet.ru/~frolov
//         или
//         http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

public class Notebook extends Applet
{
    // Панель для размещения блокнота
    Panel pCardPanel;

    // Панель для размещения кнопок управления блокнотом
    Panel pButtonPanel;

    // Панель для рисования
    Panel pDraw;

    // Панели отдельных страниц
    Panel pBackgroundColor; // страница выбора цвета фона
    Panel pForegroundColor; // страница выбора цвета
                           // изображения
    Panel pFont;            // страница выбора шрифта

    // Кнопки выбора страниц блокнота
    Button btnNext;         // следующая
    Button btnPrev;        // предыдущая
    Button btnBackgroundColor; // фон
    Button btnForegroundColor; // изображение
    Button btnFont;         // шрифт

    // Создаем ссылки на объекты класса Choice
    Choice chBackgroundColor; // список цветов фона
    Choice chForegroundColor; // список цветов изображения
    Choice chFont;            // список шрифтов

    // Текстовые метки списков
    Label tBackgroundColor; // метка списка цветов фона
    Label tForegroundColor; // метка списка цветов
                           // изображения
    Label tFont;            // метка списка шрифтов

    // Строка для хранения название выбранного шрифта
    String sFontName;

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: Grid\r\n" +
            "Author: Alexandr Frolov\r\n" +
            "E-mail: frolov@glas.apc.org" +
            "WWW:    http://www.glasnet.ru/~frolov" +
            "Created with Microsoft Visual J++ Version 1.1";
    }

    // -----
    // init
    // Метод, получающий управление при инициализации апплета
    // -----
    public void init()
    {
        // Устанавливаем желтый цвет фона
        setBackground(Color.yellow);
    }
}
```

```

// -----
// Создаем в окне апплета две панели, разделяющие
// окно по горизонтали. В верхней панели будет
// находиться блокнот, в нижней - кнопки управления
// блокнотом
// -----
setLayout(new GridLayout(3, 1));

// Создаем панель блокнота
pCardPanel = new Panel();

// -----
// Создаем панели страниц блокнота
// -----

// Панель для выбора цвета фона
pBackgroundColor = new Panel();

// Панель для выбора цвета изображения
pForegroundColor = new Panel();

// Панель для выбора шрифта
pFont = new Panel();

// -----
// Создаем списки для выбора цвета фона и
// цвета изображения
// -----

// Список для выбора цвета фона
chBackgroundColor = new Choice();

// Список для выбора цвета изображения
chForegroundColor = new Choice();

// Список для выбора шрифта
chFont = new Choice();

// -----
// Создаем метки для списков
// -----

// Метка для списка цвета фона
tBackgroundColor = new Label("Chose Background Color:");

// Метка для списка цвета изображения
tForegroundColor = new Label("Chose Foreground Color:");

// Метка для списка шрифтов
tFont = new Label("Chose Font:");

// -----
// Добавляем списки и метки в панели
// -----

// Список цвета фона и метка
pBackgroundColor.add(tBackgroundColor);
pBackgroundColor.add(chBackgroundColor);

// Список цвета изображения и метка
pForegroundColor.add(tForegroundColor);
pForegroundColor.add(chForegroundColor);

// Список шрифтов и метка
pFont.add(tFont);
pFont.add(chFont);

// -----
// Заполняем списки
// -----

// Заполняем список цвета фона
chBackgroundColor.addItem("Yellow");
chBackgroundColor.addItem("Green");
chBackgroundColor.addItem("White");

```

```

// Заполняем список цвета изображения
chForegroundColor.addItem("Black");
chForegroundColor.addItem("Red");
chForegroundColor.addItem("Blue");

// Заполняем список шрифтов
chFont.addItem("Helvetica");
chFont.addItem("Courier");
chFont.addItem("TimesRoman");

// Создаем панель для рисования
pDraw = new Panel();

// Устанавливаем режим размещения для блокнота
pCardPanel.setLayout(new CardLayout(5, 5));

// -----
// Добавляем панели страниц в блокнот
// -----

// Панель выбора цвета фона
pCardPanel.add("BackgroundColor", pBackgroundColor);

// Панель выбора цвета изображения
pCardPanel.add("ForegroundColor", pForegroundColor);

// Панель выбора шрифта
pCardPanel.add("Font", pFont);

// Добавляем панель для рисования в окно апплета
add(pDraw);

// Добавляем панель блокнота в окно апплета
add(pCardPanel);

// -----
// Создаем кнопки управления блокнотом
// -----

// Кнопка просмотра следующей страницы блокнота
btnNext = new Button("Next");

// Кнопка просмотра предыдущей страницы блокнота
btnPrev = new Button("Prev");

// Выбор панели цвета фона
btnBackgroundColor = new Button("Background Color");

// Выбор панели цвета изображения
btnForegroundColor = new Button("Foreground Color");

// Выбор панели шрифтов
btnFont = new Button("Set Text Font");

// Создаем панель кнопок управления блокнотом
pButtonPanel = new Panel();

// Устанавливаем режим размещения для панели кнопок
pButtonPanel.setLayout(new FlowLayout());

// Добавляем кнопки в панель кнопок
pButtonPanel.add(btnBackgroundColor);
pButtonPanel.add(btnForegroundColor);
pButtonPanel.add(btnFont);
pButtonPanel.add(btnNext);
pButtonPanel.add(btnPrev);

// Добавляем панель кнопок
add(pButtonPanel);

// Выбираем шрифт по умолчанию
sFontName = new String("Helvetica");

// Отображаем окно апплета
show();
}

```

```

// -----
// action
// Метод вызывается, когда пользователь выполняет
// действие над компонентами
// -----
public boolean action(Event evt, Object obj)
{
    // Проверяем, что событие вызвано кнопкой, а не
    // другим компонентом
    if(evt.target instanceof Button)
    {
        // Ссылка на кнопку, от которой пришло сообщение
        Button btn;

        // Получаем ссылку на кнопку, вызвавшую событие
        btn = (Button)evt.target;

        // Выполняем ветвление по кнопкам. Для каждой кнопки
        // записываем ее название в строку состояния
        // навигатора
        if(evt.target.equals(btnNext))

            // Выбираем следующую страницу в блокноте
            ((CardLayout)pCardPanel.getLayout()).next(pCardPanel);

        else if(evt.target.equals(btnPrev))

            // Выбираем предыдущую страницу в блокноте
            ((CardLayout)pCardPanel.getLayout()).previous(pCardPanel);

        else if(evt.target.equals(btnBackgroundColor))

            // Выбираем страницу цвета фона
            ((CardLayout)pCardPanel.getLayout()).show(
                pCardPanel, "BackgroundColor");

        else if(evt.target.equals(btnForegroundColor))

            // Выбираем страницу цвета изображения
            ((CardLayout)pCardPanel.getLayout()).show(
                pCardPanel, "ForegroundColor");

        else if(evt.target.equals(btnFont))

            // Выбираем страницу шрифтов
            ((CardLayout)pCardPanel.getLayout()).show(
                pCardPanel, "Font");

        // Если событие возникло от неизвестной кнопки,
        // мы его не обрабатываем, передавая методу action
        // родительского класса
        else
            return super.action(evt, obj);

        // Перерисовываем окно панели pDraw и апплета
        pDraw.repaint();
        repaint();

        // Возвращаем признак того, что мы обработали событие
        return true;
    }

    // Обработка событий от списков
    else if(evt.target instanceof Choice)
    {
        // Переменная для хранения ссылки на список,
        // вызвавший событие
        Choice ch;

        // Получаем ссылку на список
        ch = (Choice)evt.target;

        // Выполняем ветвление по спискам

        // Список цвета фона
        if(evt.target.equals(chBackgroundColor))
        {

```

```

        // Получаем номер текущего элемента списка
        // и устанавливаем соответствующий
        // цвет фона
        if(ch.getSelectedIndex() == 0)
            pDraw.setBackground(Color.yellow);

        else if(ch.getSelectedIndex() == 1)
            pDraw.setBackground(Color.green);

        else if(ch.getSelectedIndex() == 2)
            pDraw.setBackground(Color.white);
    }

    // Список цвета изображения
    else if(evt.target.equals(chForegroundColor))
    {
        // Получаем номер текущего элемента списка
        // и устанавливаем соответствующий
        // цвет изображения
        if(ch.getSelectedIndex() == 0)
            pDraw.setForeground(Color.black);

        else if(ch.getSelectedIndex() == 1)
            pDraw.setForeground(Color.red);

        else if(ch.getSelectedIndex() == 2)
            pDraw.setForeground(Color.blue);
    }

    // Список шрифтов
    else if(evt.target.equals(chFont))
    {
        // Получаем номер текущего элемента списка
        // и записываем имя соответствующего шрифта
        // в строку sFontName
        if(ch.getSelectedIndex() == 0)
            sFontName = "Helvetica";

        else if(ch.getSelectedIndex() == 1)
            sFontName = "Courier";

        else if(ch.getSelectedIndex() == 2)
            sFontName = "TimesRoman";
    }

    // Если событие возникло от неизвестного списка,
    // мы его не обрабатываем, передавая методу action
    // родительского класса
    else
        return super.action(evt, obj);

    // Перерисовываем панель pDraw
    pDraw.repaint();

    // Перерисовываем окно апплета
    repaint();

    // Возвращаем признак того, что мы обработали событие
    return true;
}

// Вызываем метод action родительского класса
return super.action(evt, obj);
}

// -----
// paint
// Метод paint, выполняющий рисование в окне апплета
// -----
public void paint(Graphics g)
{
    Graphics gpDraw;

    // Получаем контекст отображения для панели рисования
    gpDraw = pDraw.getGraphics();

    // Определяем текущие размеры

```

```

Dimension dimAppWndDimension = pDraw.size();

// Рисуем рамку вокруг окна апплета
gpDraw.drawRect(0, 0,
    dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);

// Устанавливаем шрифт
gpDraw.setFont(new Font(sFontName, Font.PLAIN, 12));

// Рисуем строку
gpDraw.drawString(
    "Смотри на шрифт, цвет фона и текста!",
    10, 50);

// Получаем контекст отображения для панели блокнота
gpDraw = pCardPanel.getGraphics();

// Определяем размеры панели блокнота
dimAppWndDimension = pCardPanel.size();

// Обводим блокнот рамкой
gpDraw.drawRect(0, 0,
    dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);
}
}

```

В листинге 7.4 вы найдете исходный текст документа HTML, созданного для размещения апплета.

Листинг 7.4. Файл Notebook\Notebook.html

```

<html>
<head>
<title>Notebook</title>
</head>
<body>
<hr>
<applet
    code=Notebook.class
    id=Notebook
    width=320
    height=240 >
</applet>
<hr>
<a href="Notebook.java">The source.</a>
</body>
</html>

```

Описание исходного текста

В классе Notebook определено довольно много полей и переопределено несколько методов.

Поля класса Notebook

В полях pDraw, pCardPanel и pButtonPanel находятся ссылки, соответственно, на верхнюю, среднюю и нижнюю панели, предназначенные для рисования, размещения блокнота диалоговых панелей настроек и кнопок управления блокнотом.

В предыдущем приложении на страницах блокнота размещались кнопки. Теперь мы решили более сложную задачу - поместили на страницы блокнота три панели, по одной на каждую страницу. Первая из этих панелей содержит список для выбора цвета фона, вторая - для выбора цвета изображения и, наконец, третья, для выбора шрифта. Поля pBackgroundColor, pForegroundColor и pFont хранят ссылки на соответствующие панели настроек.

Нижняя панель содержит кнопки управления страницами блокнота. С помощью кнопок, ссылки на которые хранятся в полях btnBackgroundColor, btnForegroundColor и btnFont вы можете выбирать для отображения страницы блокнота, содержащие панели настройки цвета фона, изображения и шрифта. Таким образом, нет необходимости перебирать страницы блокнота по очереди до тех пор, пока в окне не появится нужная страница. Тем не менее, мы предусмотрели кнопки и для циклического перебора страниц блокнота. Ссылки на эти кнопки хранятся в полях btnNext и btnPrev.

На каждой панели в блокноте размещается один список и одна надпись, объясняющая назначение списка. Списки создаются как объекты класса Choice, а надписи - как объекты класса Label.

Поля chBackgroundColor, chForegroundColor и chFont хранят ссылки на списки, соответственно, цвета фона, цвета изображения и шрифтов. В полях tBackgroundColor, tForegroundColor и tFont хранятся ссылки надписей.

Поле sFontName класса String предназначено для хранения названия текущего шрифта, с использованием которого отображается текст в верхней панели.

Метод `getAppletInfo`

Метод `getAppletInfo` возвращает информацию об апплете.

Метод `init`

Метод `init` выполняет достаточно громоздкую работу по созданию и добавлению различных панелей и других компонентов. К сожалению, приложениям Java не доступны ресурсы, аналогичные ресурсам операционной системы Microsoft Windows, поэтому формирование диалоговых панелей и других элементов пользовательского интерфейса приходится выполнять чисто программными методами на этапе выполнения приложения. Средства среды разработки приложений Java Microsoft Visual J++ версии 1.1, о которых мы уже упоминали, позволяют несколько упростить этот процесс.

Свою работу метод `init` начинает с установки желтого цвета фона для окна апплета.

Далее устанавливается режим добавления `GridLayout`, разделяющий окно апплета на три части по горизонтали:

```
setLayout(new GridLayout(3, 1));
```

Соответствующая таблица, в которую будут добавляться компоненты, имеет три строки и один столбец.

Панель блокнота создается следующим образом:

```
pCardPanel = new Panel();
```

Затем создаются три панели, которые будут добавляться в панель `pCardPanel`:

```
pBackgroundColor = new Panel();
```

```
pForegroundColor = new Panel();
```

```
pFont = new Panel();
```

Эти панели предназначены для размещения компонент, с помощью которых можно будет выбирать цвет фона и изображения, а также шрифт.

На следующем этапе создаются три списка, которые будут размещаться по одному на указанных панелях:

```
chBackgroundColor = new Choice();
```

```
chForegroundColor = new Choice();
```

```
chFont = new Choice();
```

Каждый такой список снабжается надписью, поясняющей его назначение. Надписи создаются следующим образом:

```
tBackgroundColor = new Label("Chose Background Color:");
```

```
tForegroundColor = new Label("Chose Foreground Color:");
```

```
tFont = new Label("Chose Font:");
```

Созданные метки и списки добавляются в панели, расположенные на страницах блокнота:

```
pBackgroundColor.add(tBackgroundColor);
```

```
pBackgroundColor.add(chBackgroundColor);
```

```
pForegroundColor.add(tForegroundColor);
```

```
pForegroundColor.add(chForegroundColor);
```

```
pFont.add(tFont);
```

```
pFont.add(chFont);
```

Вначале мы добавляем надпись, затем список. Поэтому надпись будет расположена слева от списка.

Заполнение списков выполняется с помощью метода `addItem`, который вызывается по очереди для каждого списка и для каждого добавляемого элемента списка:

```
chBackgroundColor.addItem("Yellow");
```

```
chBackgroundColor.addItem("Green");
```

```
chBackgroundColor.addItem("White");
```

```
chForegroundColor.addItem("Black");
```

```
chForegroundColor.addItem("Red");
```

```
chForegroundColor.addItem("Blue");
```

```
chFont.addItem("Helvetica");
```

```
chFont.addItem("Courier");
```

```
chFont.addItem("TimesRoman");
```

Далее метод `init` приступает к формированию верхней панели, предназначенной для отображения текстовой строки.

Панель создается следующим образом:

```
pDraw = new Panel();
```

Затем метод `init` начинает наполнение страниц блокнота.

Прежде всего он устанавливает режим добавления `CardLayout`, оставляя зазор по вертикали и горизонтали в 5 пикселей:

```
pCardPanel.setLayout(new CardLayout(5, 5));
```

Панели добавляются методом `add`, причем мы выбрали вариант этого метода, допускающий присваивание имен добавляемым компонентам:

```
pCardPanel.add("BackgroundColor", pBackgroundColor);
```

```
pCardPanel.add("ForegroundColor", pForegroundColor);
```

```
pCardPanel.add("Font", pFont);
```

Имена, которые передаются через первый параметр, необходимы для отображения страниц блокнота с помощью кнопок `btnBackgroundColor`, `btnForegroundColor` и `btnFont`.

После заполнения страниц блокнота метод `init` добавляет верхнюю и среднюю панели в окно апплета:

```
add(pDraw);
add(pCardPanel);
```

В процессе формирования нижней панели метод `init` создает кнопки, предназначенные для управления блокнотом:

```
btnNext = new Button("Next");
btnPrev = new Button("Prev");
btnBackgroundColor = new Button("Background Color");
btnForegroundColor = new Button("Foreground Color");
btnFont = new Button("Set Text Font");
pButtonPanel = new Panel();
```

Перед добавлением кнопок в нижней панели устанавливается режим размещения `FlowLayout`:

```
pButtonPanel.setLayout(new FlowLayout());
```

Затем кнопки добавляются в панель вызовом метода `add`:

```
pButtonPanel.add(btnBackgroundColor);
pButtonPanel.add(btnForegroundColor);
pButtonPanel.add(btnFont);
pButtonPanel.add(btnNext);
pButtonPanel.add(btnPrev);
```

После формирования панели кнопок эта панель добавляется в окно апплета, располагаясь в его нижней части:

```
add(pButtonPanel);
```

В поле `sFontName` записывается имя шрифта, выбранного по умолчанию:

```
sFontName = new String("Helvetica");
```

На завершающем этапе метод `init` выполняет принудительную перерисовку и отображение панелей, вызывая специально предназначенный для этого метод `show`:

```
show();
```

Метод *action*

Метод `action` выполняет отдельную обработку событий, вызванных кнопками и списками.

Если событие было вызвано кнопками, выполняется переключение страниц блокнота:

```
if (evt.target.equals(btnNext))
    ((CardLayout)pCardPanel.getLayout()).next(pCardPanel);
else if (evt.target.equals(btnPrev))
    ((CardLayout)pCardPanel.getLayout()).previous(pCardPanel);
else if (evt.target.equals(btnBackgroundColor))
    ((CardLayout)pCardPanel.getLayout()).show(
        pCardPanel, "BackgroundColor");
else if (evt.target.equals(btnForegroundColor))
    ((CardLayout)pCardPanel.getLayout()).show(
        pCardPanel, "ForegroundColor");
else if (evt.target.equals(btnFont))
    ((CardLayout)pCardPanel.getLayout()).show(
        pCardPanel, "Font");
```

Для выбора следующей и предыдущей страницы здесь использованы методы `next` и `previous`.

Выбор конкретной страницы по ее имени осуществляется с помощью метода `show`. В качестве параметров этому методу передается ссылка на панель блокнота и имя страницы.

Обратите также внимание на способ обработки событий, не имеющих отношения к нашим компонентам:

```
return super.action(evt, obj);
```

Здесь мы вызываем метод `action` из базового класса, который после соответствующей обработки события вернет значение `true` или `false`.

Если событие вызвано кнопками управления блокнотом, мы перерисовываем окно верхней панели, окно всего апплета и затем возвращаем признак успешной обработки события:

```
pDraw.repaint();
repaint();
return true;
```

Как вы увидите дальше, в процессе перерисовки окна всего апплета метод `paint` выполнит рисование в окне верхней панели.

События, связанные с выбором нового цвета фона и изображения обрабатываются следующим образом:

```
if (evt.target.equals(chBackgroundColor))
{
    if (ch.getSelectedIndex() == 0)
        pDraw.setBackground(Color.yellow);
    else if (ch.getSelectedIndex() == 1)
        pDraw.setBackground(Color.green);
    else if (ch.getSelectedIndex() == 2)
        pDraw.setBackground(Color.white);
}
else if (evt.target.equals(chForegroundColor))
```

```
{
    if(ch.getSelectedIndex() == 0)
        pDraw.setForeground(Color.black);
    else if(ch.getSelectedIndex() == 1)
        pDraw.setForeground(Color.red);
    else if(ch.getSelectedIndex() == 2)
        pDraw.setForeground(Color.blue);
}
```

Здесь методы setBackground и setForeground устанавливают цвет фона и изображения для панели pDraw.

Если событие вызвано списком шрифтов, то мы получаем номер элемента списка и записываем название выбранного шрифта в текстовую строку sFontName:

```
else if(evt.target.equals(chFont))
{
    if(ch.getSelectedIndex() == 0)
        sFontName = "Helvetica";
    else if(ch.getSelectedIndex() == 1)
        sFontName = "Courier";
    else if(ch.getSelectedIndex() == 2)
        sFontName = "TimesRoman";
}
```

После обработки события, вызванного списками, мы перерисовываем окно панели рисования pDraw и окно апплета:

```
pDraw.repaint();
repaint();
```

Метод paint

Метод paint рисует в окне панели pDraw, а не в главном окне апплета. Для этого метод paint получает контекст отображения этой панели, вызывая для этого метод getGraphics:

```
Graphics gpDraw;
gpDraw = pDraw.getGraphics();
```

Далее метод paint определяет размеры панели pDraw и рисует рамку вокруг окна этой панели:

```
Dimension dimAppWndDimension = pDraw.size();
gpDraw.drawRect(0, 0, dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);
```

После того как рамка будет нарисована, метод paint устанавливает в панели pDraw шрифт, название которого хранится в строке sFontName. Для этого используется контекст отображения gpDraw, полученный нами ранее для панели pDraw:

```
gpDraw.setFont(new Font(sFontName, Font.PLAIN, 12));
```

Текстовая строка отображается с использованием текущего цвета изображения и текущего шрифта, установленного в контексте отображения gpDraw, при помощи метода drawString:

```
gpDraw.drawString("Смотри на шрифт, цвет фона и текста!",
    10, 50);
```

Затем метод paint обводит панель блокнота рамкой. Вначале он получает контекст отображения для панели блокнота, как это показано ниже:

```
gpDraw = pCardPanel.getGraphics();
```

После получения контекста отображения метод paint определяет размеры панели блокнота и рисует рамку вокруг его окна:

```
dimAppWndDimension = pCardPanel.size();
gpDraw.drawRect(0, 0, dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);
```

Создание нового класса на базе класса Panel

Если ваш апплет создает много панелей, техника рисования в окнах этих панелей, использованная в только что рассмотренном приложении Notebook, может привести к усложнению исходного текста приложения. Так как рисование в окнах панелей выполняется в методе paint класса апплета, вам придется получать контекст отображения для каждой панели.

Намного проще создать несколько дочерних классов от класса Panel, переопределив в каждом из них метод paint. В этом случае для каждой панели вы можете создать свой метод paint, которому будет автоматически передаваться контекст отображения, связанный с окном соответствующей панели.

Эта техника использована в приложении Panel2, которое мы рассмотрим в следующем разделе.

Приложение Panel2

В окне апплета Panel2 мы создали две панели, одна из которых занимает верхнюю половину окна, а другая - нижнюю (рис. 7.7).

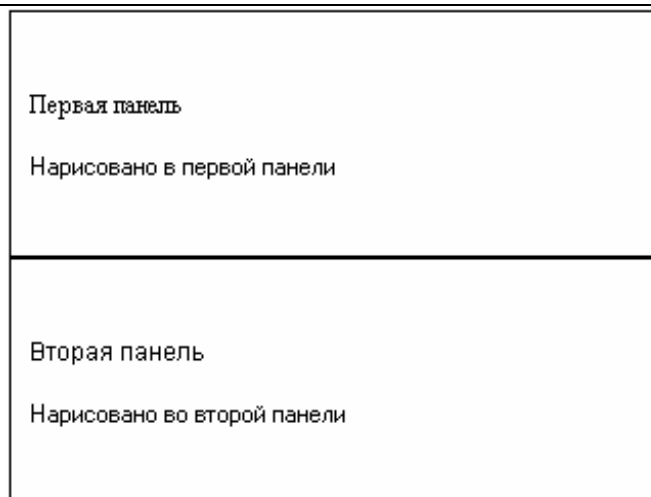


Рис. 7.7. Окно апплета Panel2 с двумя панелями

Для каждой панели мы создали два отдельных класса на базе класса Panel. В созданных нами классах переопределен метод paint, который рисует одну текстовую строку в окне своей панели. В верхней панели рисуется строка "Первая панель", в нижней - "Вторая панель".

Кроме того, метод paint класса нашего апплета рисует две строки в окнах обеих панелей, получая контекст отображения для панелей и указывая ссылку на этот контекст явным образом.

В процессе инициализации метод init класса нашего апплета устанавливает различный цвет фона и изображения для панелей, поэтому рамки вокруг окон панелей и текст получаются нарисованными различным цветом независимо от способа их рисования.

Исходные тексты приложения

Файл исходных текстов приложения приведен в листинге 7.5.

Листинг 7.5. Файл Panel2\Panel2.java

```
// =====
// Работа с панелями класса Panel
// Наследование от класса Panel
//
// (C) Фролов А.В., 1997
//
// E-mail: frolov@glas.apc.org
// WWW:    http://www.glasnet.ru/~frolov
//         или
//         http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

// =====
// Класс Panel2
// Это наш апплет
// =====
public class Panel2 extends Applet
{
    // Первая панель
    FirstPanel pPanel1;

    // Вторая панель
    SecondPanel pPanel2;

    // -----
    // getAppletInfo
    // Метод, возвращающей строку информации об апплете
    // -----
    public String getAppletInfo()
    {
        return "Name: Panel2\r\n" +
            "Author: Alexandr Frolov\r\n" +
            "E-mail: frolov@glas.apc.org" +
            "WWW:    http://www.glasnet.ru/~frolov" +
            "Created with Microsoft Visual J++ Version 1.0";
    }
}

// -----
```

```

// init
// Метод, получающий управление при инициализации апплета
// -----
public void init()
{
    // Создаем в окне апплета две панели, разделяющие
    // окно по горизонтали
    setLayout(new GridLayout(2, 1));

    // Создаем первую панель
    pPanel1 = new FirstPanel();

    // Добавляем первую панель в окно апплета
    add(pPanel1);

    // Создаем вторую панель
    pPanel2 = new SecondPanel();

    // Добавляем вторую панель
    add(pPanel2);

    // Устанавливаем желтый цвет фона для первой панели
    pPanel1.setBackground(Color.yellow);

    // Устанавливаем черный цвет изображения
    // для первой панели
    pPanel1.setForeground(Color.black);

    // Устанавливаем белый цвет фона для второй панели
    pPanel2.setBackground(Color.white);

    // Устанавливаем красный цвет изображения
    // для второй панели
    pPanel2.setForeground(Color.red);

    // Иницилируем вызов метода paint
    repaint();
}

// -----
// paint
// Метод paint, выполняющий рисование в окне апплета
// -----
public void paint(Graphics g)
{
    Graphics gPanel1;
    Graphics gPanel2;

    // Получаем контекст отображения для первой панели
    gPanel1 = pPanel1.getGraphics();

    // Рисуем строку в окне первой панели
    gPanel1.drawString("Нарисовано в первой панели",
        10, 80);

    // Получаем контекст отображения для второй панели
    gPanel2 = pPanel2.getGraphics();

    // Рисуем строку в окне второй панели
    gPanel2.drawString("Нарисовано во второй панели",
        10, 80);
}
}

// =====
// Класс FirstPanel
// Первая панель
// =====
class FirstPanel extends Panel
{
    // -----
    // paint
    // Метод paint, выполняющий рисование в окне апплета
    // -----
    public void paint(Graphics g)
    {
        // Определяем текущие размеры

```

```

Dimension dimAppWndDimension = size();

// Рисуем рамку вокруг окна апплета
g.drawRect(0, 0,
    dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);

// Устанавливаем шрифт
g.setFont(new Font("TimesRoman", Font.PLAIN, 12));

// Рисуем строку
g.drawString("Первая панель", 10, 50);

// Вызываем метод paint родительского класса
super.paint(g);
}
}

// =====
// Класс SecondPanel
// Вторая панель
// =====
class SecondPanel extends Panel
{
    // -----
    // paint
    // Метод paint, выполняющий рисование в окне апплета
    // -----
    public void paint(Graphics g)
    {
        // Определяем текущие размеры
        Dimension dimAppWndDimension = size();

        // Рисуем рамку вокруг окна апплета
        g.drawRect(0, 0,
            dimAppWndDimension.width - 1,
            dimAppWndDimension.height - 1);

        // Устанавливаем шрифт
        g.setFont(new Font("Helvetica", Font.PLAIN, 12));

        // Рисуем строку
        g.drawString("Вторая панель", 10, 50);

        // Вызываем метод paint родительского класса
        super.paint(g);
    }
}

```

Исходный текст документа HTML, предназначенного для размещения нашего апплета, представлен в листинге 7.6.

Листинг 7.6. Файл Panel2\Panel2.html

```

<html>
<head>
<title>Panel2</title>
</head>
<body>
<hr>
<applet
    code=Panel2.class
    id=Panel2
    width=320
    height=240 >
</applet>
<hr>
<a href="Panel2.java">The source.</a>
</body>
</html>

```

Описание исходного текста

Как мы уже говорили, в приложении Panel2 мы создали два класса, взяв для них в качестве базового класс Panel. Имена этих классов - FirstPanel и SecondPanel. После трансляции проекта системой Microsoft Visual J++ получаются три двоичных файла с именами Panel2.class, FirstPanel.class и SecondPanel.class - по одному для каждого класса.

Поля класса Panel2

В классе Panel2 определено два поля с именами pPanel1 и pPanel2 класса Panel. Первое из них предназначено для хранения ссылки на верхнюю панель, второе - на нижнюю (в соответствии с их расположением в окне апплета).

Метод getAppletInfo класса Panel2

Метод getAppletInfo возвращает информацию об апплете.

Метод init класса Panel2

Прежде всего метод init устанавливает для окна панели режим добавления компонент GridLayout, определяя таблицу из двух строк и одного столбца.

Первая панель создается на базе класса FirstPanel, определенного в нашем приложении:

```
pPanel1 = new FirstPanel();
```

Этот класс мы рассмотрим позже.

Созданная панель добавляется в окно апплета методом add:

```
add(pPanel1);
```

Аналогично мы создаем и вторую панель, на этот раз как объект класса SecondPanel:

```
pPanel2 = new SecondPanel();
```

Вторая панель добавляется в окно апплета точно также, как и первая:

```
add(pPanel2);
```

Для того чтобы выделить панели на фоне окна апплета, мы устанавливаем для них разные цвета фона и изображения. Для первой панели устанавливается желтый цвет фона и черный цвет изображения:

```
pPanel1.setBackground(Color.yellow);
```

```
pPanel1.setForeground(Color.black);
```

Для второй панели мы устанавливаем белый цвет фона и красный цвет изображения:

```
pPanel2.setBackground(Color.white);
```

```
pPanel2.setForeground(Color.red);
```

В результате цвета рамки окна и текста первой и второй панели будут разными.

На последнем этапе инициализации мы инициируем вызов метода paint класса Panel2, вызывая для этого метод repaint:

```
repaint();
```

Это нужно для того, чтобы сразу после отображения окна апплета выполнить рисование текстовых строк внутри окон панелей.

Метод paint класса Panel2

Метод paint класса Panel2 рисует две строки в окнах панелей, расположенных в окне апплета. Для этого он получает контекст отображения каждой панели и вызывает для этого контекста метод drawGraphics:

```
Graphics gPanel1;
```

```
Graphics gPanel2;
```

```
gPanel1 = pPanel1.getGraphics();
```

```
gPanel1.drawString("Нарисовано в первой панели", 10, 80);
```

```
gPanel2 = pPanel2.getGraphics();
```

```
gPanel2.drawString("Нарисовано во второй панели", 10, 80);
```

Хотя при рисовании строк мы указали одинаковые координаты начала строки (10, 80), наложения строк не произойдет. Это потому, что эти строки рисуются в разных графических контекстах, которые относятся к окнам разных панелей.

Цвет рисуемого текста и фона, на котором он будет нарисован, также получится разным, так как для наших панелей установлен разный цвет изображения и фона.

Класс FirstPanel

Класс FirstPanel создан на базе класса Panel:

```
class FirstPanel extends Panel
{
    public void paint(Graphics g)
    {
        . . .
        super.paint(g);
    }
}
```

В нашем приложении мы создаем верхнюю панель как объект этого класса. Единственный метод, переопределенный в классе FirstPanel - это метод paint.

Метод paint класса FirstPanel

Задачей метода paint класса FirstPanel является рисование рамки вокруг первой панели и текстовой строки в окне этой панели. В качестве параметра метод paint класса FirstPanel получает ссылку на контекст отображения для окна первой панели. Мы можем использовать этот контекст для того чтобы нарисовать что-нибудь внутри первой панели.

Процедура рисования не имеет никаких особенностей:

```
Dimension dimAppWndDimension = size();
g.drawRect(0, 0, dimAppWndDimension.width - 1,
    dimAppWndDimension.height - 1);
g.setFont(new Font("TimesRoman", Font.PLAIN, 12));
g.drawString("Первая панель", 10, 50);
```

Заметим, однако, что после завершения рисования мы вызываем метод `paint` из базового класса, позволяя этому классу выполнить свою обработку:

```
super.paint(g);
```

Класс SecondPanel

Класс `SecondPanel` создан также на базе класса `Panel`:

```
class SecondPanel extends Panel
{
    public void paint(Graphics g)
    {
        . . .
        super.paint(g);
    }
}
```

С использованием этого класса создается нижняя панель. В классе `SecondPanel`, как и в классе `FirstPanel`, переопределен метод `paint`.

Метод paint класса SecondPanel

Метода `paint` класса `SecondPanel` выполняет те же самые операции, что и метод `paint` класса `FirstPanel`, однако ему передается контекст отображения второй панели. Поэтому он нарисует рамку и текстовую строку во второй панели, а не в первой или где-нибудь еще.

8 ОКНА И ДИАЛГОВЫЕ ПАНЕЛИ

До сих пор мы рисовали только в окне апплета или в окнах панелей, расположенных внутри окна апплета. Однако есть и другая возможность - приложения Java, полноценные и апплеты, могут создавать обычные перекрывающиеся окна, такие, например, как окно навигатора. Эти окна могут иметь меню (в отличие от окон апплетов). Пользователь может изменять размер таких окон при помощи мыши, перемещая рамку окна.

В составе библиотеки классов AWT имеется несколько классов, предназначенных для работы с окнами. Это класс Window, который произошел от класса Container, и его дочерние классы - Frame, Dialog и FileDialog (рис. 8.1).

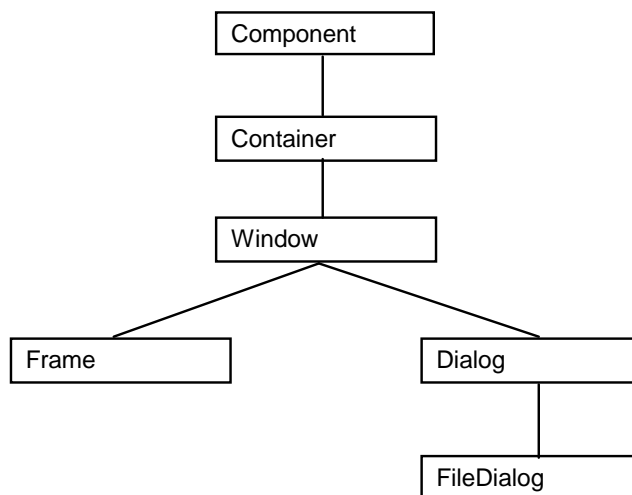


Рис. 8.1. Иерархия классов, предназначенных для создания окон

Окно, созданное на базе класса Frame, больше всего похоже на главное окно обычного приложения Windows. Оно может иметь главное меню, для него можно устанавливать форму курсора. Внутри такого окна можно рисовать. Так как окно класса Frame (так же как и другие окна AWT) произошли от класса Container, вы можете добавлять в них различные компоненты и панели, как мы это делали с окнами апплетов и панелей.

На базе класса Dialog создаются окна диалоговых панелей, очень похожих на обычные диалоговые панели Windows. Такие панели не могут иметь меню и обычно предназначены для запроса какой-либо информации у пользователя.

Класс FileDialog предназначен для создания диалоговых панелей, с помощью которых можно выбирать файлы на локальных дисках компьютера. Так как апплеты не могут работать с файлами, в этой книге мы не будем рассматривать класс FileDialog. Информацию о том, как работать с файлами в приложениях Java, а также сведения об этом классе мы планируем включить в следующий том "Библиотеки системного программиста", посвященный системе разработки Microsoft Visual J++.

Что же касается класса Window, то непосредственно этот класс редко применяется для создания окон, так как классы Frame, Dialog и FileDialog более удобны и обеспечивают все необходимые возможности.

Окна класса Frame

Ниже мы привели краткое описание класса Frame. Так как этот класс реализует интерфейс java.awt.MenuContainer, окно класса Frame может содержать меню.

```

public class java.awt.Frame
    extends java.awt.Window
    implements java.awt.MenuContainer
{
    // -----
    // Поля
    // -----

    // Различные типы курсоров
    public final static int CROSSHAIR_CURSOR;
    public final static int DEFAULT_CURSOR;
    public final static int E_RESIZE_CURSOR;
    public final static int HAND_CURSOR;
    public final static int MOVE_CURSOR;
    public final static int N_RESIZE_CURSOR;
    public final static int NE_RESIZE_CURSOR;
    public final static int NW_RESIZE_CURSOR;
  }
  
```

```

public final static int S_RESIZE_CURSOR;
public final static int SE_RESIZE_CURSOR;
public final static int SW_RESIZE_CURSOR;
public final static int TEXT_CURSOR;
public final static int W_RESIZE_CURSOR;
public final static int WAIT_CURSOR;

// -----
// Конструкторы
// -----

// Создание окна без заголовка
public Frame();

// Создание окна с заголовком
public Frame(String title);

// -----
// Методы
// -----

// Вызов метода createFrame
public void addNotify();

// Удаление окна и освобождение связанных с ним ресурсов
public void dispose();

// Определение типа курсора
public int getCursorType();

// Получение пиктограммы, установленной для окна
public Image getIconImage();

// Получение ссылки на главное меню
public MenuBar getMenuBar();

// Получение заголовка окна
public String getTitle();

// Определение возможности изменения
// размеров окна пользователем
public boolean isResizable();

// Получение строки параметров
protected String paramString();

// Удаление компоненты меню
public void remove(MenuComponent m);

// Установка типа курсора
public void setCursor(int cursorType);

// Установка пиктограммы
public void setIconImage(Image image);

// Установка главного меню
public void setMenuBar(MenuBar mb);

// Включение или выключение возможности
// изменения размеров окна
public void setResizable(boolean resizable);

// Установка заголовка окна
public void setTitle(String title);
}

```

Для того чтобы создать свое окно на базе класса Frame, вы должны определить свой класс, унаследовав его от класса Frame следующим образом:

```

class MainFrameWnd extends Frame
{
    . . .
    public MainFrameWnd(String sTitle)
    {
        super(sTitle);
        . . .
        resize(400, 200);
    }
}

```

```
} . . .
```

Если мы будем создавать окно с заголовком, нам необходимо соответствующим образом определить конструктор класса этого окна. В частности, наш конструктор должен вызывать конструктор базового класса, передавая ему в качестве параметра строку заголовка окна. Напомним, что конструктор базового класса должен вызываться в конструкторе дочернего класса перед выполнением каких-либо других действий.

Обратите также внимание на вызов метода `resize`. Этот вызов необходим для задания размеров окна.

В конструкторе вы можете определить различные параметры создаваемого вами окна, например, указать форму курсора, пиктограмму, представляющую окно, задать меню, определить возможность изменения размеров окна и так далее. Мы остановимся подробнее на процедуре добавления меню к окну класса `Frame`, так как она требует пояснений. С изменением других характеристик окна вы справитесь самостоятельно.

При создании окна классов `Frame` и `Dialog` для них устанавливается режим размещения `BorderLayout`. Если вам нужен другой режим размещения, необходимо установить его явным образом.

Кроме того, созданное окно появится на экране только после вызова для него метода `show`.

Убрать окно с экрана вы можете методом `hide`. Этот метод прячет окно, но оставляет в памяти все связанные с ним ресурсы, поэтому вы сможете вновь отобразить спрятанное окно, вызвав метод `show`.

В отличие от метода `hide`, метод `dispose` удаляет окно и освобождает все связанные с ним ресурсы. Этот метод применяется для окончательного удаления окна с экрана и из памяти.

Еще одно замечание касается обработки операции уничтожения окна при помощи двойного щелчка левой клавиши мыши по системному меню окна или при помощи кнопки уничтожения окна, расположенной в правой части заголовка.

Когда пользователь пытается уничтожить окно класса `Frame` или `Dialog` подобным образом, возникает событие `Event.WINDOW_DESTROY`. Вы должны предусмотреть обработку этого события, выполняя действия, соответствующие логике работы вашего окна. Обычно окно уничтожается вызовом метода `dispose`, как это показано ниже:

```
public boolean handleEvent(Event evt)
{
    if(evt.id == Event.WINDOW_DESTROY)
    {
        dispose();
        return true;
    }
    else
        return super.handleEvent(evt);
}
```

Меню в окне класса `Frame`

Как мы уже говорили, окно класса `Frame` может иметь главное меню (`Menu Bar`) или, как еще говорят, строку меню. Главное меню создается на базе класса `MenuBar`, краткое описание которого приведено ниже:

```
public class java.awt.MenuBar
    extends java.awt.MenuComponent
    implements java.awt.MenuContainer
{
    // -----
    // Конструктор
    // -----
    public MenuBar();

    // -----
    // Методы
    // -----

    // Добавление меню в главное меню окна
    public Menu add(Menu m);

    // Вызов метода createMenuBar
    public void addNotify();

    // Определение количества меню, добавленных
    // в главное меню
    public int countMenus();

    // Получение ссылки на меню Help
    public Menu getHelpMenu();

    // Получение ссылки на меню с заданным номером
    public Menu getMenu(int i);

    // Удаление меню с заданным номером из главного меню
    public void remove(int index);
```

```
// Удаление компоненты меню
public void remove(MenuComponent m);

// Извещение об удалении меню
public void removeNotify();

// Установка меню Help
public void setHelpMenu(Menu m);
}
```

Для формирования главного меню окна вы должны создать объект класса `MenuBar` с помощью конструктора, а затем добавить в него отдельные меню.

Объект главного меню создается следующим образом:

```
MenuBar mbMainMenuBar;
mbMainMenuBar = new MenuBar();
```

Отдельные меню создаются на базе класса `Menu`, например:

```
Menu mnFile;
Menu mnHelp;
mnFile = new Menu("File");
mnHelp = new Menu("Help");
```

Создав меню, вы должны добавить в них строки. Для этого нужно вызвать метод `add`, передав ему в качестве параметра текст строки меню, например:

```
mnFile.add("New");
mnFile.add("-");
mnFile.add("Exit");

mnHelp.add("Content");
mnHelp.add("-");
mnHelp.add("About");
```

Далее сформированные меню добавляются в главное меню:

```
mbMainMenuBar.add(mnFile);
mbMainMenuBar.add(mnHelp);
```

И, наконец, теперь можно устанавливать главное меню в окне класса, созданного на базе класса `Frame`:

```
setMenuBar(mbMainMenuBar);
```

Классы `Menu` и `MenuItem`

Для того чтобы дать вам представление о том, что можно делать с меню, приведем краткое описание класса `Menu`:

```
public class java.awt.Menu
    extends java.awt.MenuItem
    implements java.awt.MenuContainer
{
    // -----
    // Конструкторы
    // -----

    // Создание меню с заданным названием
    public Menu(String label);

    // Создание меню с заданным названием,
    // которое может оставаться на экране после того как
    // пользователь отпустил клавишу мыши
    public Menu(String label, boolean tearOff);

    // -----
    // Методы
    // -----

    // Добавление элемента меню
    public MenuItem add(MenuItem mi);

    // Добавление строки в меню
    public void add(String label);

    // Вызов метода createMenu
    public void addNotify();

    // Добавление разделителя в меню
    public void addSeparator();

    // Определение количества строк в меню
    public int countItems();
}
```



```
// Получение ссылки на элемент меню с заданным номером
public MenuItem getItem(int index);

// Проверка, остается ли меню на экране после того как
// пользователь отпустил клавишу мыши
public boolean isTearOff();

// Удаление заданного элемента меню
public void remove(int index);

// Удаление заданной компоненты меню
public void remove(MenuComponent item);

// Извещение об удалении меню
public void removeNotify();
}
```

Метод `addSeparator` используется для добавления в меню разделительной строки. Аналогичный результат достигается и при добавлении в меню строки "-":
`mnHelp.add("-");`

Заметим, что вы можете просто добавлять в меню строки по их названию, пользуясь методом `add(String label)`, либо добавлять в меню элементы класса `MenuItem`, вызывая метод `add(MenuItem mi)`.

Класс `MenuItem` определяет поведение отдельных элементов меню:

```
public class java.awt.MenuItem
    extends java.awt.MenuComponent
{
    // -----
    // Конструктор
    // -----
    public MenuItem(String label);

    // -----
    // Методы
    // -----

    // Вызов метода createMenuItem
    public void addNotify();

    // Блокирование элемента меню
    public void disable();

    // Разблокирование элемента меню
    public void enable();

    // Блокирование или разблокирование элемента меню
    public void enable(boolean cond);

    // Получение текстовой строки меню
    public String getLabel();

    // Проверка, является ли элемент меню заблокированным
    public boolean isEnabled();

    // Получение строки параметров
    public String paramString();

    // Установка текстовой строки для элемента меню
    public void setLabel(String label);
}
```

Пользуясь методами класса `MenuItem` вы можете блокировать или разблокировать отдельные строки меню, что нужно делать, например, если в данный момент функция, соответствующая строке меню, недоступна или не определена. Вы также можете изменять текстовые строки, соответствующие элементам меню, что может пригодиться для переопределения их назначения.

Создание диалоговых панелей

Диалоговые панели создаются на базе класса `Dialog`, краткое описание которого приведено ниже:

```
public class java.awt.Dialog
    extends java.awt.Window
{
    // -----
    // Конструкторы
    // -----

    // Создание диалоговой панели без заголовка
```

```

public Dialog(Frame parent, boolean modal);

// Создание диалоговой панели с заголовком
public Dialog(Frame parent, String title, boolean modal);

// -----
// Методы
// -----

// Вызов метода createDialog
public void addNotify();

// Получение строки заголовка диалоговой панели
public String getTitle();

// Определение, является ли диалоговая панель модальной
public boolean isModal();

// Определение возможности изменения размеров окна
// диалоговой панели
public boolean isResizable();

// Получение строки параметров
protected String paramString();

// Включение или выключение возможности изменения
// размеров окна диалоговой панели
public void setResizable(boolean resizable);

// Установка заголовка диалоговой панели
public void setTitle(String title);
}

```

Для того чтобы создать свою диалоговую панель, вы должны определить новый класс, унаследовав его от класса Dialog, как это показано ниже:

```

class MessageBox extends Dialog
{
    . . .
    public MessageBox(String sMsg,
        Frame parent, String sTitle, boolean modal)
    {
        super(parent, sTitle, modal);
        . . .
        resize(200, 100);
        . . .
    }
}

```

В этом классе нужно определить конструктор, который вызывает конструктор базового метода класса Dialog и определяет размеры окна диалоговой панели. Кроме того, в конструкторе вы должны создать все необходимые компоненты для размещения внутри диалоговой панели (кнопки, списки, текстовые поля, переключатели и так далее), а также выполнить размещение этих компонент, установив нужный режим размещения.

Для окон класса Dialog устанавливается режим размещения BorderLayout. Если нужен другой режим размещения, необходимо установить его явным образом методом setLayout.

Для отображения окна диалоговой панели необходимо вызвать метод show. Чтобы спрятать диалоговое окно, применяйте метод hide. Метод dispose удаляет окно диалоговой панели окончательно и освобождает все связанные с ним ресурсы.

Когда пользователь пытается уничтожить окно диалоговой панели при помощи органов управления, расположенных в заголовке такого окна, возникает событие Event.WINDOW_DESTROY. Вы должны обработать его, обеспечив удаление окна диалоговой панели вызовом метода dispose, если, конечно, это соответствует логике работы вашей панели.

Приложение FrameWnd

В приложении FrameWnd мы демонстрируем создание окон, меню и диалоговых панелей на базе классов, описанных в этой главе.

В окне апплета FrameWnd расположены две кнопки с названиями Show Frame Window и Hide Frame Window. Первая из них предназначена для отображения окна Main Frame Window, а вторая - для его временного удаления (скрытия).

В окне Main Frame Window мы создали главное меню, содержащее меню File и Help. При выборе любой строки из этого меню, кроме строки Exit меню File, на экране появляется окно диалоговой панели Dialog from Frame с названием выбранной строки меню (рис. 8.2).

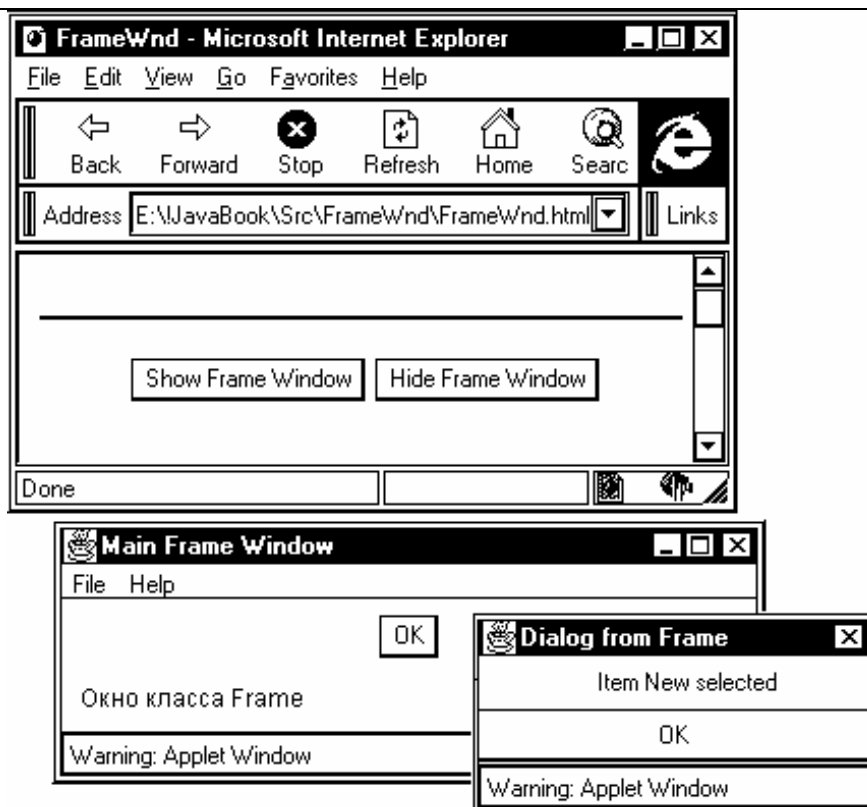


Рис. 8.2. Окно и диалоговая панель, создаваемая апплетом FrameWnd

Помимо меню, в окне Main Frame Window находится кнопка OK, нажатие на которую вызывает удаление окна. Кроме того, в нижней части окна отображается строка “Окно класса Frame”.

В окне диалоговой панели, разделенном по вертикали на две части, находится текстовое поле для отображения сообщения и кнопка для завершения работы диалоговой панели.

Обратите также внимание на то, что в самой нижней части окна Main Frame Window и Dialog from Frame находится предупреждающее сообщение “Warning: Applet Window”. Это предупреждение пользователю, что данное окно или диалоговая панель выведена не локальным приложением, запущенным на компьютере пользователя, а апплетом, загруженным из сети. Пользователь должен понимать, что данные, введенные им в окнах, созданных апплетами, передаются по сети и могут оказаться доступной кому угодно. Поэтому он не должен вводить конфиденциальную информацию, например, номера своих кредитных карточек.

Исходные тексты приложения

Исходный текст приложения FrameWnd приведен в листинге 8.1.

Листинг 8.1. Файл FrameWnd\FramWnd.java

```
// =====
// Работа с окнами и диалоговыми панелями
//
// (C) Фролов А.В., 1997
//
// E-mail: frolov@glas.apc.org
// WWW:   http://www.glasnet.ru/~frolov
//        или
//        http://www.dials.ccas.ru/frolov
// =====
import java.applet.*;
import java.awt.*;

// =====
// Класс FrameWnd
// Это наш апплет
// =====
public class FrameWnd extends Applet
{
    // Окно, которое будет возникать из апплета
    MainFrameWnd fMainFrame;

    // Кнопка для отображения окна fMainFrame
    Button btnShowFrame;
```

```

// Кнопка для удаления окна fMainFrame
Button btnHideFrame;

// -----
// getAppletInfo
// Метод, возвращающей строку информации об апплете
// -----
public String getAppletInfo()
{
    return "Name: FrameWnd\r\n" +
        "Author: Alexandr Frolov\r\n" +
        "E-mail: frolov@glas.apc.org" +
        "WWW: http://www.glasnet.ru/~frolov" +
        "Created with Microsoft Visual J++ Version 1.0";
}

// -----
// init
// Метод, получающий управление при инициализации апплета
// -----
public void init()
{
    // Создаем новое окно на базе класса MainFrameWnd
    fMainFrame = new MainFrameWnd("Main Frame Window");

    // Создаем кнопку для отображения этого окна
    btnShowFrame = new Button("Show Frame Window");

    // Добавляем кнопку в окно апплета
    add(btnShowFrame);

    // Создаем кнопку для удаления окна fMainFrame
    btnHideFrame = new Button("Hide Frame Window");

    // Добавляем кнопку в окно апплета
    add(btnHideFrame);
}

// -----
// destroy
// Метод, получающий управление при завершении
// работы апплета
// -----
public void destroy()
{
    // Удаляем окно fMainFrame и освобождаем все связанные
    // с ним ресурсы
    fMainFrame.dispose();
}

// -----
// action
// Метод вызывается, когда пользователь выполняет
// действие над компонентами
// -----
public boolean action(Event evt, Object obj)
{
    // Ссылка на кнопку, от которой пришло сообщение
    Button btn;

    // Проверяем, что событие вызвано кнопкой, а не
    // другим компонентом
    if(evt.target instanceof Button)
    {
        // Получаем ссылку на кнопку, вызвавшую событие
        btn = (Button)evt.target;

        // Если нажата кнопка отображения окна fMainFrame,
        // показываем его с помощью метода show
        if(evt.target.equals(btnShowFrame))
        {
            fMainFrame.show();
        }

        // Если нажата кнопка удаления окна fMainFrame,
        // удаляем его с помощью метода hide
        else if(evt.target.equals(btnHideFrame))
    }
}

```

```

        {
            fMainFrame.hide();
        }
        // Если событие возникло от неизвестной кнопки,
        // мы его не обрабатываем
        else
            return false;

        return true;
    }

    // Если событие вызвано не кнопкой,
    // мы его не обрабатываем
    return false;
}
}

// =====
// Класс MainFrameWnd
// На базе этого класса создается окно с меню
// =====
class MainFrameWnd extends Frame
{
    // Кнопка, с помощью которой можно закрыть окно
    Button btnOK;

    // Главное меню окна
    MenuBar mbMainMenuBar;

    // Меню File
    Menu mnFile;

    // Меню Help
    Menu mnHelp;

    // -----
    // MainFrameWnd
    // Конструктор класса
    // -----
    public MainFrameWnd(String sTitle)
    {
        // Создаем окно, вызывая конструктор из базового класса
        super(sTitle);

        // Устанавливаем размеры окна
        resize(400, 200);

        // Устанавливаем цвет фона и изображения для окна
        setBackground(Color.yellow);
        setForeground(Color.black);

        // Устанавливаем режим добавления компонент FlowLayout
        setLayout(new FlowLayout());

        // Создаем и добавляем в окно кнопку OK
        btnOK = new Button("OK");
        add(btnOK);

        // Создаем главное меню
        mbMainMenuBar = new MenuBar();

        // Создаем меню File
        mnFile = new Menu("File");

        // Заполняем меню File
        mnFile.add("New");           // строка New
        mnFile.add("-");             // разделитель
        mnFile.add("Exit");          // строка Exit

        // Создаем меню Help
        mnHelp = new Menu("Help");

        // Заполняем меню Help
        mnHelp.add("Content");       // строка Content
        mnHelp.add("-");             // разделитель
        mnHelp.add("About");         // строка About
    }
}

```

```

    // Добавляем меню File и Help в главное
    // меню нашего окна
    mbMainMenuBar.add(mnFile);
    mbMainMenuBar.add(mnHelp);

    // Устанавливаем для окна главное меню
    setMenuBar(mbMainMenuBar);
}

// -----
// paint
// Метод paint, выполняющий рисование
// в созданном нами окне
// -----
public void paint(Graphics g)
{
    // Устанавливаем шрифт
    g.setFont(new Font("Helvetica", Font.PLAIN, 12));

    // Рисуем строку
    g.drawString("Окно класса Frame", 10, 50);

    // Вызываем метод paint родительского класса
    super.paint(g);
}

// -----
// handleEvent
// Обработка событий для окна
// -----
public boolean handleEvent(Event evt)
{
    // Если пользователь закрывает окно,
    // скрываем его с помощью метода hide
    if(evt.id == Event.WINDOW_DESTROY)
    {
        hide();
        return true;
    }

    else
        return super.handleEvent(evt);
}

// -----
// action
// Метод вызывается, когда пользователь выполняет
// действие над компонентами в нашем окне
// -----
public boolean action(Event evt, Object obj)
{
    // Ссылка на кнопку, от которой пришло сообщение
    Button btn;

    // Ссылка на элемент меню
    MenuItem mnItem;

    // Проверяем, что событие вызвано кнопкой, а не
    // другим компонентом
    if(evt.target instanceof Button)
    {
        // Получаем ссылку на кнопку, вызвавшую событие
        btn = (Button)evt.target;

        // Если пользователь нажал кнопку ОК, скрываем окно
        if(evt.target.equals(btnOK))
        {
            hide();
        }
        // Если событие возникло от неизвестной кнопки,
        // мы его не обрабатываем
        else
            return false;

        return true;
    }
}

```

```

// Обработка событий от меню
else if(evt.target instanceof MenuItem)
{
    // Получаем ссылку на элемент меню
    mnItem = (MenuItem)evt.target;

    // Если из меню File выбрана строка Exit,
    // завершаем работу виртуальной машины Java
    if(obj.equals("Exit"))
    {
        System.exit(0);
    }

    // Если из меню File выбрана строка New,
    // отображаем название строки в диалоговой панели
    else if(obj.equals("New"))
    {
        // Ссылка на диалоговую панель
        MessageBox mbox;

        // Создаем новую диалоговую панель
        mbox = new MessageBox("Item New selected",
            this, "Dialog from Frame", true);

        // Отображаем диалоговую панель
        mbox.show();
    }

    // Если из меню File выбрана строка Content,
    // отображаем название строки в диалоговой панели
    else if(obj.equals("Content"))
    {
        MessageBox mbox;

        mbox = new MessageBox("Item Content selected",
            this, "Dialog from Frame", true);

        mbox.show();
    }

    // Если из меню File выбрана строка About,
    // отображаем название строки в диалоговой панели
    else if(obj.equals("About"))
    {
        MessageBox mbox;
        mbox = new MessageBox("Item About selected",
            this, "Dialog from Frame", true);

        mbox.show();
    }
    else
        return false;

    return true;
}
return false;
}
}

// =====
// Класс MessageBox
// На базе этого класса мы создаем диалоговые панели
// =====
class MessageBox extends Dialog
{
    // Поле для отображения текста сообщения
    Label lbMsg;

    // Кнопка для удаления диалоговой панели
    Button btnOK;

    // -----
    // MessageBox
    // Конструктор класса
    // -----
    public MessageBox(String sMsg,

```

```

Frame parent, String sTitle, boolean modal)
{
    // Создаем диалоговую панель, вызывая конструктор
    // базового класса Dialog
    super(parent, sTitle, modal);

    // Устанавливаем размеры окна диалоговой панели
    resize(200, 100);

    // Устанавливаем режим размещения компонент GridLayout
    setLayout(new GridLayout(2, 1));

    // Создаем и добавляем поле для отображения сообщения
    lblMsg = new Label(sMsg, Label.CENTER);
    add(lblMsg);

    // Создаем и добавляем кнопку для завершения работы
    // диалоговой панели
    btnOK = new Button("OK");
    add(btnOK);
}

// -----
// handleEvent
// Обработка событий для диалоговой панели
// -----
public boolean handleEvent(Event evt)
{
    // Если пользователь закрывает окно диалоговой панели,
    // скрываем его с помощью метода hide
    if(evt.id == Event.WINDOW_DESTROY)
    {
        dispose();
        return true;
    }

    else
        return super.handleEvent(evt);
}

// -----
// action
// Метод вызывается, когда пользователь выполняет
// действие над компонентами в диалоговой панели
// -----
public boolean action(Event evt, Object obj)
{
    // Ссылка на кнопку, от которой пришло сообщение
    Button btn;

    // Проверяем, что событие вызвано кнопкой, а не
    // другим компонентом
    if(evt.target instanceof Button)
    {
        // Получаем ссылку на кнопку, вызвавшую событие
        btn = (Button)evt.target;

        // Если нажата кнопка OK, удаляем диалоговую панель
        // и освобождаем все связанные с ней ресурсы
        if(evt.target.equals(btnOK))
        {
            dispose();
        }

        // Если событие возникло от неизвестной кнопки,
        // мы его не обрабатываем
        else
            return false;

        return true;
    }
    return false;
}
}

```

Исходный текст документа HTML, созданного для размещения апплета, приведен в листинге 8.2.

Листинг 8.2. Файл FrameWnd\FrameWnd.html

```

<html>
<head>
<title>FrameWnd</title>
</head>
<body>
<hr>
<applet
    code=FrameWnd.class
    id=FrameWnd
    width=320
    height=240 >
</applet>
<hr>
<a href="FrameWnd.java">The source.</a>
</body>
</html>

```

Описание исходного текста

Рассмотрим по очереди поля и методы классов, определенных в нашем приложении.

Поля класса FrameWnd

В поле fMainFrame класса MainFrameWnd хранится ссылка на окно, которое будет создано, если пользователь нажмет кнопку "Show Frame Window", расположенную в окне апплета. Класс MainFrameWnd создан нами на базе класса Frame.

Поля с именами btnShowFrame и btnHideFrame предназначены, соответственно, для хранения ссылок на только что указанную кнопку и кнопку "Hide Frame Window", с помощью которой можно скрыть окно.

Метод getAppletInfo класса FrameWnd

Этот метод возвращает информацию об апплете FrameWnd.

Метод init класса FrameWnd

В процессе инициализации апплета метод init создает объект класса MainFrameWnd - перекрывающееся окно с заголовком "Main Frame Window":

```
fMainFrame = new MainFrameWnd("Main Frame Window");
```

Для этого вызывается конструктор из класса MainFrameWnd, созданного нами на базе класса Frame.

После этого метод init создает две кнопки и добавляет их в окно апплета:

```

btnShowFrame = new Button("Show Frame Window");
add(btnShowFrame);
btnHideFrame = new Button("Hide Frame Window");
add(btnHideFrame);

```

Первая из этих кнопок предназначена для отображения перекрывающегося окна, а вторая - для временного удаления или скрытия этого окна.

Метод destroy класса FrameWnd

При завершении работы апплета мы удаляем созданное нами окно и освобождаем все связанные с ним ресурсы, вызывая для окна метод dispose:

```
fMainFrame.dispose();
```

Метод action класса FrameWnd

Назначение метода action класса FrameWnd - обработка событий, вызванных кнопками кнопку "Show Frame Window" и "Hide Frame Window", созданных в окне апплета:

```

if (evt.target.equals(btnShowFrame))
    fMainFrame.show();
else if (evt.target.equals(btnHideFrame))
    fMainFrame.hide();

```

Если нажата кнопка "Show Frame Window", для окна fMainFrame вызывается метод show, что приводит к появлению окна на экране. Если нажата кнопка "Hide Frame Window", для этого окна вызывается метод hide, после чего окно исчезает с экрана. Исчезнувшее окно не уничтожается и вы можете снова отобразить его на экране, нажав кнопку "Show Frame Window".

Класс MainFrameWnd

Класс MainFrameWnd предназначен для создания автономного перекрывающегося окна, которое существует вне окна навигатора. Этот класс был нами создан на базе класса Frame:

```

class MainFrameWnd extends Frame
{
    . . .
}

```

В классе мы определили несколько полей, конструктор для создания окна, метод paint для рисования в окне, метод handleEvent для обработки запроса на уничтожение окна, метод action для обработки события, вызванного кнопкой, расположенной в окне, а также выбором строк меню, созданного для окна.

Поля класса *MainFrameWnd*

В поле btnOK хранится ссылка на кнопку, при нажатии которой окно удаляется.

Поле mbMainMenuBar класса MenuBar предназначено для хранения ссылки на главное меню окна. В него мы будем добавлять меню "File" и "Help", идентификаторы которых хранятся в полях mnFile и mnHelp, соответственно.

Конструктор класса *MainFrameWnd*

В качестве единственного параметра нашему конструктору передается заголовок создаваемого окна. В первой исполняемой строке наш конструктор вызывает конструктор из базового класса, передавая ему строку заголовка через параметр:

```
super(sTitle);
```

Далее конструктор определяет размеры окна, вызывая для него метод resize:

```
resize(400, 200);
```

Затем мы устанавливаем для нашего окна желтый цвет фона и черный цвет изображения:

```
setBackground(Color.yellow);
setForeground(Color.black);
```

По умолчанию для окон класса Frame устанавливается режим добавления компонент BorderLayout. Мы изменяем этот режим на FlowLayout, вызывая метод setLayout:

```
setLayout(new FlowLayout());
```

Установив новый режим добавления компонент, мы располагаем в нашем окне кнопку, предварительно создав ее с помощью конструктора класса Button:

```
btnOK = new Button("OK");
add(btnOK);
```

Далее метод init приступает к формированию главного меню окна. Это меню создается как объект класса MenuBar:

```
mbMainMenuBar = new MenuBar();
```

Затем мы создаем и наполняем меню "File":

```
mnFile = new Menu("File");
mnFile.add("New");           // строка New
mnFile.add("-");             // разделитель
mnFile.add("Exit");          // строка Exit
```

Это меню создается на базе класса Menu. Обратите внимание, что между строками New и File расположен разделитель.

Аналогичным образом мы добавляем в главное меню другое меню - "Help":

```
mnHelp = new Menu("Help");
mnHelp.add("Content");       // строка Content
mnHelp.add("-");             // разделитель
mnHelp.add("About");         // строка About
```

После своего окончательного формирования меню "File" и "Help" добавляются в главное меню окна mbMainMenuBar:

```
mbMainMenuBar.add(mnFile);
mbMainMenuBar.add(mnHelp);
```

И, наконец, когда главное меню будет сформировано, оно подключается к окну вызовом метода setMenuBar, как это показано ниже:

```
setMenuBar(mbMainMenuBar);
```

Метод *paint* класса *MainFrameWnd*

Метод paint получает в качестве параметра ссылку на контекст отображения, пригодный для рисования в нашем окне. Пользуясь этим контекстом, мы устанавливаем шрифт текста и рисуем текстовую строку. Затем мы вызываем метод paint из базового класса Frame, на основе которого создан наш класс MainFrameWnd:

```
g.setFont(new Font("Helvetica", Font.PLAIN, 12));
g.drawString("Окно класса Frame", 10, 50);
super.paint(g);
```

Метод *handleEvent* класса *MainFrameWnd*

Для того чтобы определить реакцию окна на попытку пользователя закрыть окно с помощью органов управления, расположенных в заголовке окна, или другим способом, мы переопределили метод handleEvent.

При получении кода события Event.WINDOW_DESTROY (удаление окна) мы просто скрываем окно, вызывая метод hide:

```
if(evt.id == Event.WINDOW_DESTROY)
{
    hide();
    return true;
}
else
    return super.handleEvent(evt);
```

Все другие события передаются для обработки методу handleEvent из базового класса.

Метод action класса MainFrameWnd

Этот метод обрабатывает события, связанные с кнопкой и меню.

Если пользователь нажимает на кнопку ОК, расположенную в окне, окно скрывается методом hide:

```
if (evt.target.equals(btnOK))
{
    hide();
}
```

Здесь для вас нет ничего нового.

Рассмотрим более подробно процедуру обработки событий от меню.

Вначале метод action проверяет, вызвано ли событие выбором строки из меню, сравнивая объект события с классом MenuItem:

```
else if (evt.target instanceof MenuItem)
{
    . . .
}
else
    return false;
```

Если это так, в поле menuItem сохраняется ссылка на элемент меню, вызвавший событие:

```
menuItem = (MenuItem)evt.target;
```

Однако мы не используем этот элемент, так как для определения строки, выбранной пользователем, нам достаточно проанализировать второй параметр метода action:

```
if (obj.equals("Exit"))
    System.exit(0);
else if (obj.equals("New"))
{
    MessageBox mbox;
    mbox = new MessageBox("Item New selected",
        this, "Dialog from Frame", true);
    mbox.show();
}
else if (obj.equals("Content"))
{
    . . .
}
else if (obj.equals("About"))
{
    . . .
}
```

В данном случае второй параметр метода action будет представлять собой ссылку на строку, выбранную из меню, поэтому для определения выбранной строки мы можем выполнить простое сравнение методом equals.

Если пользователь выбрал из меню File строку Exit, мы вызываем метод System.exit, предназначенный для завершения работы виртуальной машины Java. Таким способом вы можете завершить работу апплета, когда он выполняется в среде Microsoft Visual J++ в процессе отладки. Если же апплет запущен автономно в навигаторе, то завершения работы навигатора не произойдет.

В том случае когда пользователь выбирает любую другую строку из меню, метод action создает диалоговую панель на базе определенного нами класса MessageBox. В этой диалоговой панели отображается название выбранной строки меню.

Заметим, что сразу после создания конструктором диалоговая панель не появляется на экране. Мы отображаем ее, вызывая метод show.

Класс MessageBox

Для отображения названий выбранных строк меню мы создаем диалоговую панель, определив свой класс MessageBox на базе класса Dialog, как это показано ниже:

```
class MessageBox extends Dialog
{
    . . .
}
```

В классе MessageBox есть два поля, конструктор, методы handleEvent и action.

Поля класса MessageBox

Внутри диалоговой панели мы расположили текстовое поле класса Label, предназначенное для отображения сообщения, и кнопку с надписью ОК, с помощью которой можно завершить работу диалоговой панели.

Ссылка на текстовое поле хранится в поле lblMsg, на кнопку - в поле btnOK.

Конструктор класса MessageBox

Наш конструктор создает диалоговую панель с заданным сообщением внутри нее. Ссылка на строку сообщения передается конструктору через первый параметр. Остальные параметры используются конструктором базового класса Dialog для создания диалоговой панели:

```
super(parent, sTitle, modal);
```

После вызова конструктора из базового класса наш конструктор устанавливает размеры окна созданной диалоговой панели, вызывая метод `resize`:

```
resize(200, 100);
```

Отменяя установленный по умолчанию режим размещения компонент `BorderLayout`, конструктор устанавливает режим `GridLayout`:

```
setLayout(new GridLayout(2, 1));
```

Окно диалоговой панели при этом разделяется на две части по горизонтали. В верхнюю часть добавляется текстовое поле для отображения сообщения, в нижнюю - кнопка ОК:

```
lbMsg = new Label(sMsg, Label.CENTER);
```

```
add(lbMsg);
```

```
btnOK = new Button("OK");
```

```
add(btnOK);
```

Метод `handleEvent` класса `MessageBox`

Когда пользователь пытается закрыть окно диалоговой панели, например, сделав двойной щелчок левой клавишей мыши по системному меню или одиночный щелчок по кнопке удаления окна, возникает событие `Event.WINDOW_DESTROY`. Мы его обрабатываем следующим образом:

```
if (evt.id == Event.WINDOW_DESTROY)
```

```
{
```

```
    dispose();
```

```
    return true;
```

```
}
```

```
else
```

```
    return super.handleEvent(evt);
```

Вызывая метод `dispose`, мы удаляем окно диалоговой панели и освобождаем все связанные с ним ресурсы.

Метод `action` класса `MessageBox`

Если пользователь нажимает кнопку ОК, расположенную в окне диалоговой панели, метод `action` вызывает для панели метод `dispose`, удаляя эту панель с экрана и из памяти:

```
if (evt.target.equals(btnOK))
```

```
{
```

```
    dispose();
```

```
}
```

ЛИТЕРАТУРА

1. Фролов А.В., Фролов Г.В. Библиотека системного программиста. М.: ДИАЛОГ-МИФИ
Т.11 - 13. Операционная система Microsoft Windows 3.1 для программиста, 1994
Т.14. Графический интерфейс GDI в Microsoft Windows, 1994
Т.15. Мультимедиа для Windows, 1994
Т.22. Операционная система Windows 95 для программиста, 1996
Т.23. Глобальные сети компьютеров. Практическое введение в Internet, E-Mail, FTP, WWW и HTML, программирование для Windows Sockets
Т.29. Сервер Web своими руками. Язык HTML, приложения CGI и ISAPI, установка серверов Web для Windows, 1997
2. Д. Родли, Создание Java-апплетов: Пер. с англ., К: НИИПФ "ДиаСофт Лтд.", 1996
3. S. Davic, Learn Lava Now, Microsoft Press, One Microsoft Way, 1996
4. К. Джамса, Java: Пер. с англ., Мн.: ООО "Поппури", 1996
5. Баженова И.Ю., Язык программирования Java, М.: ДИАЛОГ-МИФИ, 1997

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

</APPLET>, 27
 <APPLET>, 19; 25; 27; 47
 <PARAM>, 28; 47
 Abstract Window Toolkit, 12
 action, 73
 ACTION_EVENT, 55
 add, 116; 138; 139
 addItem, 83; 88
 addLayoutComponent, 108; 114
 addPoint, 37
 addSeparator, 139
 ALIGN, 27
 allowsMultipleSelections, 88
 ALT, 27
 ALT_MASK, 56
 appendText, 102
 Applet, 24; 72; 107
 arg, 55
 AWT, 12; 107
 BOLD, 32
 boolean, 9; 10
 BorderLayout, 107; 111; 137
 brighter, 32
 Button, 71; 72
 byte, 9
 bytesWidth, 34
 Canvas, 71
 CardLayout, 108; 114
 CENTER, 108
 CGI, 150
 char, 9
 Character, 10
 charsWidth, 34
 charWidth, 34
 CheckBox, 71; 77
 CheckboxGroup, 78
 Choice, 83
 class, 14
 clear, 88
 clearRect, 29
 clickCount, 55
 clipRect, 29; 33; 39
 CODE, 27
 CODEBASE, 27
 Color, 34
 Color black, 31
 Color blue, 31
 Color cyan, 31
 Color darkGray, 31
 Color gray, 31
 Color green, 31
 Color lightGray, 31
 Color magenta, 31
 Color orange, 31
 Color pink, 31
 Color red, 31
 Color white, 31
 Color yellow, 31
 Component, 72; 107
 Container, 72; 107; 116; 136
 Control Panel, 44
 copyArea, 29; 39
 countItems, 83; 88; 139
 countMenus, 138
 create, 29
 CROSSHAIR_CURSOR, 136
 CTRL_MASK, 56
 darker, 32
 DEFAULT_CURSOR, 136
 DefWindowProc, 56
 delete, 14
 delItem, 88
 deselect, 88
 destroy, 25
 Dialog, 136; 140
 disable, 139
 dispose, 29; 137
 double, 9; 11
 DOWN, 56
 draw3DRect, 29
 drawArc, 29; 38
 drawBytes, 29
 drawChars, 29
 drawImage, 30
 drawLine, 30; 35
 drawOval, 30; 38

drawPolygon, 30; 36; 37
drawRect, 30; 35
drawRoundRect, 30; 36
drawString, 26; 30
E_RESIZE_CURSOR, 136
echoCharIsSet, 98
enable, 140
END, 56
equals, 32
Event, 55
Event.LIST_SELECT, 94
Event.WINDOW_DESTROY, 138; 141
evt, 55
extends, 24
F1-F12, 56
FileDialog, 136
fill3DRect, 30; 36
fillArc, 30
fillOval, 30; 38
fillPolygon, 30; 37
fillRect, 30; 35
fillRoundRect, 30; 36
final, 14
finalize, 30
first, 114
float, 9; 11
FlowLayout, 107; 108
FlowLayout.CENTER, 108
FlowLayout.LEFT, 108
FlowLayout.RIGHT, 108
Font, 32
Frame, 136
getAlignment, 95
getAppletInfo, 25
getAscent, 34
getBlue, 32
getBoundingBox, 37
getCheckboxGroup, 77
getClipRect, 30; 33
getColor, 30; 34
getColumns, 98; 102
getCurrent, 78
getCursorType, 137
getDescent, 34
getEchoChar, 98
getFamily, 32
getFont, 31; 32; 34
getFontMetrics, 31; 34
getGraphics, 117
getGreen, 32
getHeight, 34
getHelpMenu, 138
getHSBColor, 32
getIconImage, 137
getItem, 83; 139
getLabel, 72; 77; 140
getLeading, 34
getMaxAdvance, 34
getMaxAscent, 34
getMaxDescent, 34
getMenu, 138
getMenuBar, 137
getName, 33
getParameter, 47
getRed, 32
getRGB, 32
getRows, 88; 102
getSelectedIndex, 83
getSelectedIndexes, 88
getSelectedItem, 83
getSelectedItems, 88
getSelectedText, 99
getSelectionEnd, 99
getSelectionStart, 99
getSize, 33
getState, 77
getStyle, 33
getText, 95; 99
getTitle, 137; 140
getVisibleIndex, 88
getWidths, 34
GIF, 12
GOT_FOCUS, 55
Graphics, 26; 29; 35; 117
GridBagLayout, 107; 115
GridLayout, 107; 109
HAND_CURSOR, 136
handleEvent, 55; 73
hashCode, 33
HEIGHT, 27
hide, 138

HOME, 56
 HSBtoRGB, 32
 HSPACE, 27
<http://www.dials.ccas.ru/frolov>, 6
 id, 55
 IEEE 754, 10
 init, 25
 inline, 15
 insertText, 102
 inside, 37
 int, 9
 Integer, 10
 IP, 11
 ISAPI, 150
 isBold, 33
 isEditable, 99
 isEnabled, 140
 isItalic, 33
 isModal, 140
 isPlain, 33
 isResizable, 137; 140
 isSelected, 88
 isTearOff, 139
 ITALIC, 32
 Java Applet Wizard, 16; 19
 Java Development Kit, 3
 Java Workspace, 16
 java.applet, 12; 23; 24; 26; 28; 40; 45; 49; 59; 63; 67; 74; 79; 84; 90; 96; 100; 103; 109; 112; 118; 123; 132; 142
 java.awt, 12; 24
 java.awt.Checkbox, 77
 java.awt.CheckboxGroup, 78
 java.awt.Choice, 83
 java.awt.Component, 26
 java.awt.Container, 26
 java.awt.Dialog, 140
 java.awt.Graphics, 29
 java.awt.Image, 12
 java.awt.List, 88
 java.awt.Menu, 139
 java.awt.MenuBar, 138
 java.awt.MenuComponent, 138
 java.awt.MenuContainer, 136; 138
 java.awt.MenuItem, 139
 java.awt.Panel, 26
 java.awt.peer, 12
 java.awt.TextArea, 102
 java.awt.TextComponent, 99
 java.awt.Window, 140
 java.io, 11
 java.lang, 10
 java.lang.Object, 29
 java.net, 11
 java.util, 11
 java.util.Vector, 65
 JDK, 3
 JIT, 3
 Just-in-Time Compilation, 3
 key, 55
 KEY_ACTION, 55
 KEY_ACTION_RELEASE, 55
 KEY_PRESS, 55
 KEY_RELEASE, 55
 keyDown, 66
 keyUp, 66
 Label, 71; 94
 last, 114
 Layout Manager, 71; 107
 layoutContainer, 108; 114
 LEFT, 56; 108
 List, 71; 87
 LIST_DESELECT, 55
 LIST_SELECT, 55
 LOAD_FILE, 55
 long, 9; 11
 LOST_FOCUS, 55
 makeVisible, 88
 Math, 11
 Menu Bar, 138
 MenuBar, 138
 MenuItem, 139
 META_MASK, 56
 MFC, 3
 Microsoft Foundation Classes, 3
 Microsoft Internet Explorer, 16
 Microsoft Visual J++, 4
 minimumLayoutSize, 108; 115
 minimumSize, 88; 98; 102
 MM_TEXT, 27
 modifiers, 55
 MOUSE_DOWN, 55
 MOUSE_DRAG, 55

MOUSE_ENTER, 55
MOUSE_EXIT, 55
MOUSE_MOVE, 55
MOUSE_UP, 56
mouseDown, 57
mouseDrag, 57
mouseEnter, 57
mouseExit, 57
mouseMove, 57
mouseUp, 57
MOVE_CURSOR, 136
MS-DOS, 26
N_RESIZE_CURSOR, 136
NAME, 27; 47
NE_RESIZE_CURSOR, 136
Netscape Navigator, 16
new, 14
next, 114
NW_RESIZE_CURSOR, 136
Object, 14
out, 19
OWL, 10
paint, 25; 26
Panel, 107; 116; 117
PGDN, 56
PGUP, 56
PLAIN, 32
Polygon, 37
preferredLayoutSize, 108; 115
preferredSize, 89; 98; 102
previous, 114
println, 19
PrintStream, 19
public, 14
remove, 137; 138; 139
removeLayoutComponent, 108
replaceItem, 89
replaceText, 103
resize, 25
RGBtoHSB, 32
RIGHT, 56; 108
S_RESIZE_CURSOR, 136
SAVE_FILE, 56
SCROLL_ABSOLUTE, 56
SCROLL_LINE_DOWN, 56
SCROLL_LINE_UP, 56
SCROLL_PAGE_DOWN, 56
SCROLL_PAGE_UP, 56
Scrollbar, 71
SDK-Java, 4
SE_RESIZE_CURSOR, 136
select, 83; 89; 99
selectAll, 99
setAlignment, 95
setCheckboxGroup, 77
setColor, 31
setCurrent, 78
setCursor, 137
setEchoCharacter, 98
setEditable, 99
setFont, 31; 32
setHelpMenu, 138
setIconImage, 137
setLabel, 72; 77; 140
setLayout, 116
setMenuBar, 137
setMultipleSelections, 89
setPaintMode, 31; 33
setResizable, 137; 140
setState, 77
setText, 95; 99
setTitle, 137; 140
setXORMode, 31; 33
SHIFT_MASK, 56
short, 9
show, 115
showStatus, 57; 60; 61; 62; 75; 76; 80; 82; 92; 94; 110; 113; 119; 120; 121; 122
start, 25
static, 14
stop, 25
String, 11; 15
stringWidth, 34
super, 15
SW_RESIZE_CURSOR, 136
switch, 73
Symantec Cafe, 4
System, 19
target, 55
TCP, 11
TEXT_CURSOR, 136
TextArea, 71; 102

TextComponent, 71; 99
TextField, 71; 98
throws, 15
TITLE, 27
toString, 31; 32; 33
translate, 31; 33
UDP, 11
union, 14
UP, 56
URL, 11
VALUE, 47
W_RESIZE_CURSOR, 136
WAIT_CURSOR, 136
when, 55
WIDTH, 27
Window, 136
WINDOW_DEICONIFY, 56
WINDOW_DESTROY, 56
WINDOW_EXPOSE, 56
WINDOW_ICONIFY, 56
WINDOW_MOVED, 56
Windows Sockets, 4
WM_CHAR, 55
WM_LBUTTONDOWN, 55
WM_PAINT, 26
wrapper classes, 10
x, 55
y, 55
апплет, 19
апплеты, 3
атрибуты контекста отображения, 33
базовые типы данных, 9
библиотеки динамической загрузки DLL, 9
виртуальный процессор Java, 9
вложенные классы, 14
главное меню, 138
деструктор, 14
диалоговые панели, 136
замещающие классы, 10
интерфейсы, 15
контекст отображения, 26; 29
линии, 35
массив, 13
наследование, 15
область ограничения, 39
однострочное текстовое поле, 94
перезагрузка (переопределение) операторов, 11
перезагрузка операторов, 11
переопределение операторов, 15
прямоугольник, 35
размещение компонент в окне контейнера, 107
сборка мусора, 14
система Layout Manager, 107
система координат, 27
события, 55
создание классов, 14
списки, 83
список шрифтов, доступных апплету, 44
ссылки, 13
статические методы и поля, 14
указатели, 13

ОГЛАВЛЕНИЕ

АННОТАЦИЯ	2
ВВЕДЕНИЕ	3
БЛАГОДАРНОСТИ	5
КАК СВЯЗАТЬСЯ С АВТОРАМИ	6
1 НОВЫЙ ЯЗЫК ПРОГРАММИРОВАНИЯ	7
Мобильность Java	7
Базовые типы данных	8
Библиотеки классов Java	9
Встроенные классы	9
Замещающие классы	9
Класс String	10
Другие встроенные классы	10
Подключаемые библиотеки классов	10
Библиотека классов java.util	10
Библиотека классов java.io	10
Библиотека классов java.net	10
Библиотека классов java.awt	11
Библиотека классов java.awt.image	11
Библиотека классов java.awt.peer	11
Библиотека классов java.applet	11
Указатели, которых нет	12
Массивы в Java	12
Сборка мусора	13
Особенности реализации классов в Java	13
Определение класса	13
Определение методов	13
Переопределение операторов	14
Интерфейсы	14
Ссылки на методы класса	14
Наследование	14
2 ПЕРВОЕ ПРИЛОЖЕНИЕ И ПЕРВЫЙ АППЛЕТ	15
Приложение Hello	15
Подготовка и запуск приложения	15
Взгляд на исходный текст приложения Hello	17
Простейший апплет	18
Исходные файлы апплета HelloAp	23
Файл HelloAp.java	23
Конструктор HelloAp	23
Метод getAppletInfo	24
Метод init	24
Метод destroy	24
Метод start	24
Метод stop	25
Метод paint	25
Файл HelloApp.html	26
Упрощаем исходный текст апплета	27
3 РИСОВАНИЕ В ОКНЕ АППЛЕТА	29
Контекст отображения	29
Полотно для рисования	29
Методы класса Graphics	29
Установка атрибутов контекста отображения	31
Выбор цвета	31
Выбор шрифта	33
Установка режима рисования	34
Установка маски для рисования	34
Сдвиг начала системы координат	34
Определение атрибутов контекста отображения	34
Определение границ области ограничения вывода	34
Определение цвета, выбранного в контекст отображения	34
Определение шрифта, выбранного в контекст отображения	34
Определение метрик текущего шрифта	34

Определение метрик заданного шрифта.....	35
Рисование геометрических фигур	35
Линии	35
Прямоугольники и квадраты	36
Многоугольники.....	37
Овалы и круги	39
Сегменты	39
Задание области ограничения.....	40
Копирование содержимого прямоугольной области.....	40
Приложение Painter	40
Исходные файлы приложения Painter.....	41
Метод init.....	44
Метод getAppletInfo	44
Метод paint.....	44
Приложение FontList.....	46
Исходный текст приложения.....	46
Описание исходного текста.....	48
Метод init	48
Метод paint.....	49
Приложение TextOut	49
Исходные тексты приложения TextOut.....	51
Описание исходных текстов.....	55
Поля класса TextOut.....	55
Метод getParameterInfo	56
Метод init	56
Метод paint	57
4 ОБРАБОТКА СОБЫТИЙ.....	58
Как обрабатываются события.....	58
События от мыши	60
Нажатие клавиши мыши.....	60
Отпускание клавиши мыши.....	60
Перемещение курсора мыши.....	60
Выполнение операции Drag and Drop.....	60
Вход курсора мыши в область окна апплета	60
Выход курсора мыши из области окна апплета.....	60
Приложение MouseClick	60
Исходные тексты приложения	62
Описание исходного текста.....	64
Метод getAppletInfo.....	64
Метод paint	65
Метод mouseDown.....	65
Методы mouseUp, mouseDrag, mouseEnter, mouseExit	65
Метод mouseMove	65
Приложение LineDraw.....	65
Исходные тексты приложения	66
Описание исходного текста.....	69
Поля класса LineDraw	69
Метод getAppletInfo.....	69
Метод init	69
Метод paint	69
Метод mouseDown.....	70
Метод mouseUp	70
Метод mouseDrag	70
Метод mouseMove	70
События от клавиатуры.....	70
Приложение KeyCode	71
Исходные тексты приложения KeyCode.....	71
Описание исходного текста.....	73
Поля класса KeyCode.....	73
Метод getAppletInfo.....	73
Метод init	73
Метод paint.....	74
Метод keyDown	74
Метод keyUp.....	74
5 КОМПОНЕНТЫ В ОКНЕ АППЛЕТА.....	75
Кнопки	76
Обработка событий от кнопки.....	77
Приложение ButtonPress	77
Исходные тексты приложения ButtonPress	78
Описание исходного текста.....	80
Поля класса ButtonPress.....	80

Метод getAppletInfo.....	81
Метод init	81
Метод action	81
Метод paint	82
Переключатели	82
Создание переключателей с независимой фиксацией	82
Создание переключателей с зависимой фиксацией	83
Приложение CheckBoxes.....	83
Исходные тексты приложения CheckBoxes.....	84
Описание исходного текста.....	87
Поля класса CheckBoxes	87
Метод getAppletInfo.....	87
Метод init	87
Метод action	88
Метод paint	88
Списки класса Choice	88
Приложение ChoiceList.....	89
Исходные тексты приложения ChoiceList.....	90
Описание исходного текста.....	92
Поля класса ChoiceList.....	92
Метод getAppletInfo.....	92
Метод init	92
Метод action	93
Метод paint	93
Списки класса List.....	93
Описание класса List.....	94
Обработка событий от списка класса List	96
Приложение ListBox.....	96
Исходные тексты приложения	96
Описание исходного текста.....	100
Поля класса ListBox.....	100
Метод getAppletInfo.....	100
Метод init	100
Метод action	100
Метод handleEvent.....	101
Метод paint	101
Текстовое поле класса Label.....	102
Приложение TextLabel.....	102
Исходные тексты приложения	103
Описание исходного текста.....	105
Поля класса TextLabel.....	105
Метод getAppletInfo.....	105
Метод init	105
Метод action	105
Метод paint	105
Текстовое поле класса TextField.....	105
Приложение TxtField	107
Исходные тексты приложения	107
Описание исходного текста.....	110
Поля класса TxtField.....	110
Метод getAppletInfo.....	110
Метод init	110
Метод action	110
Метод paint	110
Многострочное текстовое поле класса TextArea	110
Приложение TextEdit.....	111
Исходные тексты приложения	112
Описание исходного текста.....	114
Поля класса TxtField.....	114
Метод getAppletInfo.....	114
Метод init	114
Метод action	115
Метод paint	115
6 НАСТРОЙКА СИСТЕМЫ LAYOUT MANAGER.....	116
Режимы системы Layout Manager.....	116
Режим FlowLayout	117
Режим GridLayout.....	118
Приложение Grid.....	118
Исходные тексты приложения	119
Описание исходного текста.....	121
Режим BorderLayout.....	121
Приложение Border.....	121
Исходные тексты приложения	122

Описание исходного текста.....	124
Режим CardLayout.....	124
Режим GridBagLayout.....	125
7 РАБОТА С ПАНЕЛЯМИ.....	127
Создание панелей.....	127
Добавление панелей.....	127
Добавление компонент в панели.....	128
Рисование в окне панели.....	128
Приложение PanelDemo.....	128
Исходные тексты приложения.....	129
Описание исходного текста.....	132
Поля класса PanelDemo.....	132
Метод getAppletInfo.....	132
Метод init.....	132
Метод action.....	133
Приложение Notebook.....	133
Исходные тексты приложения.....	135
Описание исходного текста.....	140
Поля класса Notebook.....	140
Метод getAppletInfo.....	141
Метод init.....	141
Метод action.....	142
Метод paint.....	143
Создание нового класса на базе класса Panel.....	143
Приложение Panel2.....	143
Исходные тексты приложения.....	144
Описание исходного текста.....	146
Поля класса Panel2.....	147
Метод getAppletInfo класса Panel2.....	147
Метод init класса Panel2.....	147
Метод paint класса Panel2.....	147
Класс FirstPanel.....	147
Метод paint класса FirstPanel.....	147
Класс SecondPanel.....	148
Метод paint класса SecondPanel.....	148
8 ОКНА И ДИАЛоговые ПАНЕЛИ.....	149
Окна класса Frame.....	149
Меню в окне класса Frame.....	151
Классы Menu и MenuItem.....	152
Создание диалоговых панелей.....	153
Приложение FrameWnd.....	154
Исходные тексты приложения.....	155
Описание исходного текста.....	161
Поля класса FrameWnd.....	161
Метод getAppletInfo класса FrameWnd.....	161
Метод init класса FrameWnd.....	161
Метод destroy класса FrameWnd.....	161
Метод action класса FrameWnd.....	161
Класс MainFrameWnd.....	161
Поля класса MainFrameWnd.....	162
Конструктор класса MainFrameWnd.....	162
Метод paint класса MainFrameWnd.....	162
Метод handleEvent класса MainFrameWnd.....	162
Метод action класса MainFrameWnd.....	163
Класс MessageBox.....	163
Поля класса MessageBox.....	163
Конструктор класса MessageBox.....	163
Метод handleEvent класса MessageBox.....	164
Метод action класса MessageBox.....	164
ЛИТЕРАТУРА.....	165
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ.....	166
ОГЛАВЛЕНИЕ.....	171