Программирование на Java

Лекция 11. Пакет java.awt

20апреля 2003 года

Авторы документа:

Николай Вязовик (Центр Sun технологий МФТИ) <vyazovick@itc.mipt.ru>
Евгений Жилин (Центр Sun технологий МФТИ) < gene@itc.mipt.ru>

Аннотация

Лекция посвящена рассмотрению простейшей графической библиотеки java.awt ....

Оглавление

Лекция 11. Пакет java.awt

Содержание лекции.

1. Введение

Трудность при создании независимой от платформы библиотеки заключается в том, что ее разработчикам либо приходится требовать, чтобы все приложения на всех платформах вели себя и выглядели одинаково, либо для поддержки, скажем, трех различных разновидностей интерфейса приходится писать в три раза больше кода. Существуют два взгляда на эту проблему. Один подход заключается в том, что упор делается на графику низкого уровня - рисование пикселей, при этом разработчики библиотеки сами заботятся о внешнем виде каждого компонента. При другом подходе создаются абстракции, подходящие для библиотек каждой из операционных систем, и именно "родные" пакеты данной операционной системы служат подъемной силой для архитектурно-нейтральной библиотеки на каждой из платформ. В Java при создании библиотеки Abstraction Window Toolkit (AWT) выбран второй подход.

Начнем изучение AWT с апплетов.

2. Апплеты

Апплеты (applets) - это маленькие приложения, которые размещаются на серверах Internet, транспортируются клиенту по сети, автоматически устанавливаются и запускаются на месте, как часть документа HTML. Когда апплет прибывает к клиенту, его доступ к ресурсам ограничен.

Ниже приведен исходный код канонической программы HelloWorld, оформленной в виде апплета:

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

public class HelloWorldApplet extends Applet {
   public void paint(Graphics g) {
      g.drawString("Hello World!", 20, 20);
   }
}
```

Этот апплет начинается двумя строками, которые импортируют все пакеты иерархий java.applet и java.awt. Дальше в нашем примере присутствует метод paint, замещающий одноименный метод класса Applet. При вызове этого метода ему передается аргумент, содержащий ссылку на объект класса Graphics. Последний используется для прорисовки нашего апплета. С помощью метода drawString, вызываемого с этим объектом типа Graphics, в позиции экрана (20,20) выводится строка "Hello World".

Для того, чтобы с помощью браузера запустить этот апплет, нам придется написать несколько строк html-текста.

```
<applet code="HelloWorldApplet" width=200 height=40>
</applet>
```

Вы можете поместить эти строки в отдельный html-файл, либо вставить их в текст этой программы в виде комментария и запустить программу appletviewer с его исходным текстом в качестве аргумента. На экране появится строка приветствия.

2.1. Тег HTML <Applet>

Тег <applet> используется для запуска апплета как из HTML-документа, так и из программы appletviewer. Программа appletviewer выполняет каждый найденный ей тег <applet> в отдельном окне, в то время как браузеры позволяют разместить на одной странице несколько апплетов. Синтаксис тэга <APPLET> в настоящее время таков:

```
<APPLET
   CODE = appletFile
   OBJECT = appletSerialFile
   WIDTH = pixels
   HEIGHT = pixels
   [ARCHIVE = jarFiles]
   [CODEBASE = codebaseURL]
   [ALT = alternateText]
   [NAME = appletInstanceName]
   [ALIGN = alignment]
   [VSPACE = pixels]
   [HSPACE = pixels]
>
[< PARAM NAME = AttributeName1 VALUE = AttributeValue1 >]
[< PARAM NAME = AttributeName2 VALUE = AttributeValue2 >]
[HTML-текст, отображаемый при отсутствии поддержки Java]
</APPLET>
```

- CODE = appletClassFile

  CODE - обязательный атрибут, задающий имя файла, в котором содержится оттранслированный код апплета. Имя файла задается относительно codebase, то есть либо от текущего каталога, либо от каталога, указанного в атрибуте CODEBASE. В Java 1.1 вместо этого атрибута может использоваться атрибут OBJECT.

- OBJECT = appletClassSerialFile

  Указывает имя файла, содержащего сериализованный апплет, из которого последний будет восстановлен. При запуске определяемого таким образом апплета должен вызываться не метод init(), а метод start(). Для апплета необходимо задать либо атрибут CODE, либо атрибут OBJECT, но задавать эти атрибуты одновременно нельзя.

- WIDTH = pixels
- HEIGHT = pixels

  WIDTH и HEIGHT - обязательные атрибуты, задающие начальный размер видимой области апплета.

- ARCHIVE = jarFiles

  Этот необязательный атрибут задает список jar-файлов (разделяется запятыми), которые предварительно загружаются в Web-броузер. В этих архивных файлах могут содержаться

файлы классов, изображения, звуки и любые другие ресурсы, необходимые апплету. Для создания архивов используется утилита JAR, синтаксис вызова которой напоминает вызов команды TAR Unix:

```
c:\> jar cf soundmap.jar *.class image.gif sound.wav
```

Очевидно, что передача сжатых jar-файлов повышает эффективность работы. Поэтому многие средства разработки (Lotus JavaBeans, Borland JBuilder) имеют средства для публикации апплетов в виде jar-файлов.

- CODEBASE = codebaseURL

  CODEBASE - необязательный атрибут, задающий базовый URL кода апплета, являющийся каталогом, в котором будет выполняться поиск исполняемого файла апплета (задаваемого в признаке CODE). Если этот атрибут не задан, по умолчанию используется каталог данного HTML-документа. CODEBASE не обязательно должен указывать на тот же узел, с которого был загружен HTML-документ.

- ALT = alternateAppletText

  Признак ALT - необязательный атрибут, задающий короткое текстовое сообщение, которое должно быть выведено (как правило, в виде всплывающей подсказки при нахождении курсора мыши над областью апплета) в том случае, если используемый браузер распознает синтаксис тега <applet> , но выполнять апплеты не умеет. Это не то же самое, что HTML-текст, который можно вставлять между <applet> и </applet> для браузеров, вообще не поддерживающих апплетов.

- NAME = appletInstanceName

  NAME - необязательный атрибут, используемый для задания имени для данного экземпляра апплета. Присвоение апплетам имен необходимо для того, чтобы другие апплеты на этой же странице могли находить их и общаться с ними. Для того, чтобы получить доступ к подклассу MyApplet класса Applet с именем "Duke", нужно написать: MyApplet a = getAppletContext().getApplet("Duke"); После того, как вы получили таким образом дескриптор именованного экземпляра апплета, вы можете вызывать его методы точно так же, как это делается с любым другим объектом.

- ALIGN = alignment

  ALIGN - необязательный атрибут, задающий стиль выравнивания апплета. Этот атрибут трактуется так же, как в теге IMG, возможные его значения - LEFT, RIGHT, TOP, TEXT-TOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM, ABSBOTTOM.

- VSPACE = pixels
- HSPACE = pixels

  Эти необязательные атрибуты задают ширину свободного пространства в пикселях сверху и снизу апплета (VSPACE), и слева и справа от него (HSPACE). Они трактуются точно так же, как одноименные атрибуты тега IMG.

- PARAM NAME = appletAttribute1 VALUE = value1

  Этот тег дает возможность передавать из HTML-страницы апплету необходимые ему аргументы. Апплеты получают эти атрибуты, вызывая метод getParameter(), описываемый ниже.

2.2. Передача параметров

- getParameter(String)

Метод getParameter возвращает значение типа String, соответствующее указанному имени параметра. Если вам в качестве параметра требуется значение какого-либо другого типа, вы должны преобразовать строку-параметр самостоятельно. Вы сейчас увидите некоторые примеры использования метода getParameter для извлечения параметров из приведенного ниже примера:

```
<applet code=Testing width=40 height=40>
<param name=fontName value=Univers>
<param name=fontSize value=14>
<param name=leading value=2>
<param name=accountEnabled value=true>
```

Ниже показано, как извлекается каждый из этих параметров:

```
String FontName = getParameter("fontName");
String FontSize = Integer.parseInt(getParameter("fontSize"));
String Leading = Float.valueOf(getParameter("leading"));
String PaidUp = Boolean.valueOf(getParameter("accountEnabled"));
```

2.3. Контекст апплета

- getDocumentBase и getCodeBase

  Возможно, Вы будете писать апплеты, которым понадобится явно загружать данные и текст. Java позволяет апплету загружать данные из каталога, в котором располагается HTML-документ, запустивший апплет (база документа - getDocumentBase), и из каталога, из которого был загружен class-файл с кодом апплета (база кода - getCodeBase).

- AppletContext и showDocument

  AppletContext представляет собой средства, позволяющие получать информацию об окружении работающего апплета. Метод showDocument приводит к тому, что заданный его параметром документ отображается в главном окне браузера или фрейме.

2.4. Отладочная печать

Отладочную печать можно выводить в два места: на консоль и в статусную строку программы просмотра апплетов. Для того, чтобы вывести сообщение на консоль, надо написать:

```
  System.out.println("Hello there, welcome to Java");
```

Сообщения на консоли очень удобны, поскольку консоль обычно не видна пользователям апплета, и в ней достаточно места для нескольких сообщений. В браузере Netscape консоль Java доступна из меню Options, пункт "Show Java Console".

Метод showStatus выводит текст в статусной области программы appletviewer или браузера с поддержкой Java. В статусной области можно вывести только одну строку сообщения.

2.5. Порядок инициализации апплета

Ниже приведен порядок, в котором вызываются методы класса Applet, с пояснениями, нужно или нет переопределять данный метод.

- init

  Метод init вызывается первым. В нем вы должны инициализировать свои переменные.

- start

  Метод start вызывается сразу же после метода init. Он также используется в качестве стартовой точки для возобновления работы после того, как апплет был остановлен. В то время, как метод init вызывается только однажды - при загрузке апплета, start вызывается каждый раз при выводе HTML-документа, содержащего апплет, на экран. Так, например, если пользователь перейдет к новой WWW-странице, а затем вернется назад к странице с апплетом, апплет продолжит работу с метода start.

- paint

  Метод paint вызывается каждый раз при повреждении апплета. AWT следит за состоянием окон в системе и замечает такие случаи, как, например, перекрытие окна апплета другим окном. В таких случаях, после того, как апплет снова оказывается видимым, для восстановления его изображения вызывается метод paint.

- update

  Используемый по умолчанию метод update класса Applet сначала закрашивает апплет цветом фона по умолчанию, после чего вызывает метод paint. Если вы в методе paint заполняете фон другим цветом, пользователь будет видеть вспышку цвета по умолчанию при каждом вызове метода update - то есть, всякий раз, когда вы перерисовываете апплет. Чтобы избежать этого, нужно заместить метод update. В общем случае нужно выполнять операции рисования в методе update, а в методе paint, к которому будет обращаться AWT, просто вызвать update.

- stop

  Метод stop вызывается в тот момент, когда браузер покидает HTML-документ, содержащий апплет. При вызове метода stop апплет еще работает. Вы должны использовать этот метод для приостановки тех подпроцессов, работа которых необязательна при невидимом апплете. После того, как пользователь снова обратится к этой странице, вы должны будете возобновить их работу в методе start.

- destroy

  Метод destroy вызывается тогда, когда среда (например, браузер Netscape) решает, что апплет нужно полностью удалить из памяти. В этом методе нужно освободить все ресурсы, которые использовал апплет.

2.6. Перерисовка

Возвратимся к апплету HelloWorldApplet. В нем мы заместили метод paint, что позволило апплету выполнить отрисовку. В классе Applet предусмотрены дополнительные методы рисования, позволяющие эффективно закрашивать части экрана. При разработке первых апплетов порой непросто понять, почему метод update никогда не вызывается. Для инициации update предусмотрены три варианта метода repaint.

repaint

Метод repaint используется для принудительного перерисовывания апплета. Этот метод, в свою очередь, вызывает метод update. Однако, если ваша система медленная или сильно загружена, метод update может и не вызваться. Близкие по времени запросы на перерисовку могут объединяться AWT, так что метод update может вызываться спорадически. Если вы хотите добиться ритмичной смены кадров изображения, воспользуйтесь методом repaint(time) - это позволит уменьшить количество кадров, нарисованных не вовремя.

repaint(time)

Вы можете вызывать метод repaint, устанавливая крайний срок для перерисовки (этот период задается в миллисекундах относительно времени вызова repaint).

repaint(x, y, w, h)

Эта версия ограничивает обновление экрана заданным прямоугольником, изменены будут только те части экрана, которые в нем находятся.

repaint(time, x, y, w, h)

Этот метод - комбинация двух предыдущих.

2.7. Задание размеров графических изображений

Графические изображения вычерчиваются в стандартной для компьютерной графики системе координат, в которой координаты могут принимать только целые значения, а оси направлены слева направо и сверху вниз. У апплетов и изображений есть метод size, который возвращает объект Dimension. Получив объект Dimension, вы можете получить и значения его переменных width и height:

```
Dimension d = size();
System.out.println(d. width + "," + d.height);
```

2.8. Простые методы класса Graphics

У объектов класса Graphics есть несколько простых функций рисования. Каждую из фигур можно нарисовать заполненной, либо прорисовать только ее границы. Каждый из методов drawRect, drawOval, fillRect и fillOval вызывается с четырьмя параметрами: int x, int y, int width и int height. Координаты x и y задают положение верхнего левого угла фигуры, параметры width и height определяют ее границы.

- drawLine

```
drawline(int x1, int y1, int x2, int y2)
```

Этот метод вычерчивает отрезок прямой между точками с координатами (x1,y1) и (x2,y2). Эти линии представляют собой простые прямые толщиной в 1 пиксель. Поддержка разных перьев и разных толщин линий не предусмотрена.

- drawArc и fillArc

  Сигнатура методов drawArc и fillArc следующая:

  ```
  drawArc(int x, int y, int width, int height, int startAngle, int sweepAngle)
  ```

  Эти методы вычерчивают (fillArc заполняет) дугу, ограниченную прямоугольником (x,y,width, height), начинающуюся с угла startAngle и имеющую угловой размер sweepAngle. Ноль градусов соответствует положению часовой стрелки на 3 часа, угол отсчитывается против часовой стрелки (например, 90 градусов соответствуют 12 часам, 180 - 9 часам, и так далее).

- drawPolygon и fillPolygon

  Прототипы для этих методов:

  ```
  drawPolygon(int[], int[], int)
  fillPolygon(int[], int[], int)
  ```

  Метод drawPolygon рисует контур многоугольника (ломаную линию), задаваемого двумя массивами, содержащими x и y координаты вершин, третий параметр метода - число пар координат. Метод drawPolygon не замыкает автоматически вычерчиваемый контур. Для того, чтобы прямоугольник получился замкнутым, координаты первой и последней точек должны совпадать.

Рассмотрим поддержку цвета в Java и вернемся к рассмотрению методов Graphics.

2.9. Цвет

Цветовая система AWT разрабатывалась так, чтобы была возможность работы со всеми цветами. После того, как цвет задан, Java отыскивает в диапазоне цветов дисплея тот, который ему больше всего соответствует. Вы можете запрашивать цвета в той семантике, к которой привыкли - как смесь красного, зеленого и голубого, либо как комбинацию оттенка, насыщенности и яркости. Вы можете использовать статические переменные класса Color.black для задания какого-либо из общеупотребительных цветов - black, white, red, green, blue, cyan, yellow, magenta, orange, pink, gray, darkGray и lightGray.

Для создания нового цвета используется один из трех описанных ниже конструкторов.

- Color(int, int, int)

  Параметрами для этого конструктора являются три целых числа в диапазоне от 0 до 255 для красного, зеленого и голубого компонентов цвета.

- Color(int)

  У этого конструктора - один целочисленный аргумент, в котором в упакованном виде заданы красный, зеленый и голубой компоненты цвета. Красный занимает биты 16-23, зеленый - 8-15, голубой - 0-7.

- Color(float, float, float)

  Последний из конструкторов цвета, Color(float, float, float), принимает в качестве параметров три значения типа float (в диапазоне от 0.0 до 1.0) для красного, зеленого и голубого базовых цветов.

2.9.1. Методы класса Color

- HSBtoRGB(float, float, float)
- RGBtoHSB(int, int, int, float[])

  HSBtoRGB преобразует цвет, заданный оттенком, насыщенностью и яркостью (HSB), в целое число в формате RGB, готовое для использования в качестве параметра конструктора Color(int). RGBtoHSB преобразует цвет, заданный тремя базовыми компонентами, в массив типа float со значениями HSB, соответствующими данному цвету.

  Цветовая модель HSB (Hue-Saturation-Brightness, оттенок-насыщенность-яркость) является альтернативой модели Red-Green-Blue для задания цветов. В этой модели оттенки можно представить как круг с различными цветами (оттенок может принимать значения от 0.0 до 1.0, цвета на этом круге идут в том же порядке, что и в радуге - красный, оранжевый, желтый, зеленый, голубой, синий, фиолетовый). Насыщенность (значение в диапазоне от 0.0 до 1.0) - это шкала глубины цвета, от легкой пастели до сочных цветов. Яркость - это также число в диапазоне от 0.0 до 1.0, причем меньшие значения соответствуют более темным цветам, а большие - более ярким.

- getRed(), getGreen(), getBlue()

  Каждый из этих методов возвращает в младших восьми битах результата значение соответствующего базового компонента цвета.

- getRGB()

  Этот метод возвращает целое число, в котором упакованы значения базовых компонентов цвета, причем

```
red = Oxff & (getRGB() >> 16);
green = Oxff & (getRGB() >> 8);
blue = Oxff & getRGB();
```

Продолжаем описание методов класса Graphics:

- setPaintMode() и setXORMode(Color)

  Режим отрисовки paint - используемый по умолчанию метод заполнения графических изображений, при котором цвет пикселей изменяется на заданный. XOR устанавливает режим рисования, когда результирующий цвет получается выполнением операции XOR (исключающее или) для текущего и указанного цветов (особенно полезно для анимации).

2.10. Шрифты

Библиотека AWT обеспечивает большую гибкость при работе со шрифтами благодаря предоставлению соответствующих абстракций и возможности динамического выбора

шрифтов. Вот очень короткая программа, которая печатает на консоли Java имена всех имеющихся в системе шрифтов.

```
/*
 * <applet code="WhatFontsAreHere" width=100 height=40>
 * </applet>
 *
 */
import java.applet.*;
import java.awt.*;

public class WhatFontsAreHere extends Applet {
   public void init() {
      String fontList[];

      // Устаревший способ получения набора шрифтов:
      // Toolkit.getDefaultToolkit().getFontList()
      fontList = GraphicsEnvironment.getLocalGraphicsEnvironment().
         getAvailableFontFamilyNames();
      for (int i=0; i < fontList.length; i++) {
         System.out.println(i + ": " + fontList[i]);
      }
   }
}
```

- drawString

  В предыдущих примерах использовался метод drawString(String, x, y). Этот метод выводит строку с использованием текущих шрифта и цвета. Точка с координатами (x,y) соответствует левой границе базовой линии символов, а не левому верхнему углу, как это принято в других методах рисования. Для того, чтобы понять, как при этом располагается описывающий строку прямоугольник, ниже будет рассмотрен класс FontMetrics.

2.10.1. Использование шрифтов

Конструктор класса Font создает новый шрифт с указанным именем, стилем и размером в пунктах:

```
Font StrongFont = new Font("Helvetica", Font.BOLD|Font.ITALIC, 24);
```

В настоящее время доступны следующие имена шрифтов: Dialog, Helvetica, TimesRoman, Courier и Symbol. Для указания стиля шрифта внутри данного семейства предусмотрены три статические переменные. - Font.PLAIN, Font.BOLD и Font.ITALIC, что соответствует обычному стилю, курсиву и полужирному.

Теперь давайте посмотрим на несколько дополнительных методов.

- getFamily и getName

Метод getFamily возвращает строку с именем семейства шрифтов. С помощью метода getName можно получить логическое имя шрифта.

- getSize

  Этот метод возвращает целое число, представляющее собой размер шрифта в пунктах.

- getStyle

  Этот метод возвращает целое число, соответствующее стилю шрифта. Полученный результат можно побитово сравнить со статическими переменными класса Font: - PLAIN, BOLD и ITALIC.

- isBold, isItalic, isPlain

  Эти методы возвращают true в том случае, если стиль шрифта - полужирный (bold), курсив (italic) или обычный (plain), соответственно.

2.10.2. Позиционирование и шрифты: FontMetrics

В Java используются различные шрифты, а класс FontMetrics позволяет программисту точно задавать положение выводимого в апплете текста. Прежде всего нам нужно понять кое-что из обычной терминологии, употребляемой при работе со шрифтами:

- Высота (height) - размер от верхней до нижней точки самого высокого символа в шрифте.
- Базовая линия (baseline) - линия, по которой выравниваются нижние границы символов (не считая снижения (descent)).
- Подъем (ascent) - расстояние от базовой линии до верхней точки символа.
- Снижение (descent) - расстояние от базовой линии до нижней точки символа.

2.10.3. Использование FontMetrics

Ниже приведены некоторые методы класса FontMetrics:

stringWidth

Этот метод возвращает длину заданной строки для данного шрифта.

bytesWidth, charsWidth

Эти методы возвращают ширину указанного массива байтов для текущего шрифта.

getAscent, getDescent, getHeight

Эти методы возвращают подъем, снижение и ширину шрифта. Сумма подъема и снижения дают полную высоту шрифта. Высота шрифта - это не просто расстояние от самой нижней точки букв g и y до самой верхней точки заглавной буквы T и символов вроде скобок. Высота включает подчеркивания и т.п.

getMaxAscent и getMaxDescent

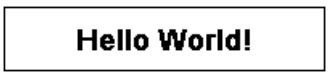
Эти методы служат для получения максимальных подъема и снижения всех символов в шрифте.

2.10.4. Центрирование текста

Давайте теперь воспользуемся методами объекта FontMetrics для получения подъема, снижения и длины строки, которую требуется нарисовать, и с помощью полученных значений отцентрируем ее в нашем апплете.

```
/*
 * <applet code="HelloWorld" width=200 height=100>
 * </applet>
 *
 */
import java.applet.*;
import java.awt.*;

public class HelloWorld extends Applet {
   final Font f = new Font("Helvetica", Font.BOLD, 18);
   public void paint(Graphics g) {
      Dimension d = this.size();
      g.setColor(Color.white);
      g.fillRect(0,0,d.width,d.height);
      g.setColor(Color.black);
      g.setFont(f);
      drawCenteredString("Hello World!", d.width, d.height, g);
      g.drawRect(0,0,d.width-1,d.height-1);
   }
   public void drawCenteredString(String s, int w, int h, Graphics g) {
      FontMetrics fm = g.getFontMetrics();
      int x = (w - fm.stringWidth(s)) / 2;
      int y = (fm.getAscent() + (h - (fm.getAscent() + fm.getDescent()))/2);
      g.drawString(s, x, y);
   }
}
```

Вот как выглядит апплет в действии:



3. Базовые классы

Теперь, когда рассмотрены классы Graphics и Fonts изучим базовую архитектуру AWT, касающуюся интерфейсных объектов.

- Компоненты

  Component - это абстрактный класс, который инкапсулирует все атрибуты визуального интерфейса - обработка ввода с клавиатуры, управление фокусом, взаимодействие с мышью, уведомление о входе/выходе из окна, изменения размеров и положения окон, прорисовка своего собственного графического представления, сохранение текущего

текстового шрифта, цветов фона и переднего плана (более 100 методов). Перейдем к некоторым конкретным подклассам класса Component.

- Container

  Container - это абстрактный подкласс класса Component, определяющий дополнительные методы, которые дают возможность помещать в него другие компоненты, что дает возможность построения иерархической системы визуальных объектов. Container отвечает за расположение содержащихся в нем компонентов с помощью интерфейса LayoutManager, описание которого будет позднее в этой главе.

- Panel

  Класс Panel - это очень простая специализация класса Container. В отличие от последнего, он не является абстрактным классом. Поэтому о Panel можно думать, как о допускающем рекурсивную вложенность экранном компоненте. С помощью метода add в объекты Panel можно добавлять другие компоненты. После того, как в него добавлены какие-либо компоненты, можно вручную задавать их положение и изменять размер с помощью методов setLocation, setSize и setBounds класса Component.

  В предыдущей главе мы уже использовали один из подклассов Panel - Applet. Каждый раз, когда мы создавали Applet, методы paint и update рисовали его изображение на поверхности объекта Panel. Прежде, чем мы углубимся в методы Panel, давайте познакомимся с основными компонентами AWT, которые можно вставлять в пустую Panel при создании графических приложений.

4. Основные компоненты

- Canvas

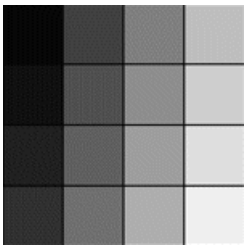
Основная идея использования объектов Canvas в том, что они являются семантически свободными компонентами. Вы можете придать объекту Canvas любое поведение и любой желаемый внешний вид. Его имя подразумевает, что этот класс является пустым холстом, на котором вы можете "нарисовать" любой компонент - такой, каким вы его себе представляете.

Произведем от Canvas подкласс GrayCanvas, который будет просто закрашивать себя серым цветом определенной насыщенности. Наш апплет будет создавать несколько таких объектов, каждый со своей интенсивностью серого цвета.

```
/* <applet code = "PanelDemo"
width=300
height=300>
   </applet>
 */
import java.awt.*;
import java.applet.*;

class GrayCanvas extends Canvas {
   Color gray;
   public GrayCanvas(float g) {
```

```
      gray = new Color(g, g, g);
   }

   public void paint(Graphics g) {
      Dimension size = size();
      g.setColor(gray);
      g.fillRect(0, 0, size.width, size.height);
      g.setColor(Color.black);
      g.drawRect(0, 0, size.width-1, size.height-1);
   }
}

public class PanelDemo extends Applet {
   static final int n = 4;
   public void init() {
      setLayout(null);
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
            float g = (i * n + j) / (float) (n * n);
            Canvas c = new GrayCanvas(g);
            add(c);
            c.setSize(width / n, height / n);
            c.setLocation(i * width / n, j * height / n);
         }
      }
   }
}
```

Вот как этот апплет выглядит на экране:



Мы устанавливаем размер каждого из объектов Canvas на основе значения, полученного с помощью метода size, который возвращает объект класса Dimension. Обратите внимание на то, что для размещения объектов Canvas в нужные места используются методы resize и move. Такой способ станет очень утомительным, когда мы перейдем к более сложным компонентам и более интересным вариантам расположения. А пока в нашем апплете для выключения упомянутого механизма использован вызов метода setLayout(null).

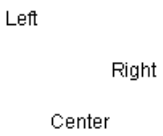
- Label

Функциональность класса Label сводится к тому, что он знает, как нарисовать объект String - текстовую строку, выровняв ее нужным образом. Шрифт и цвет, которыми отрисовывается строка метки, являются частью базового определения класса Component. Для работы с

этими атрибутами предусмотрены пары методов getFont/setFont и getForeground/setForeground. Задать или изменить текст строки после создания объекта с помощью метода setText. Для задания режимов выравнивания в классе Label определены три константы - LEFT, RIGHT и CENTER. Ниже приведен пример, в котором создаются три метки, каждая - со своим режимом выравнивания.

```
/* <applet code = "LabelDemo" width=100 height=100>
   </applet>
 */
import java.awt.*;
import java.applet.*;

public class LabelDemo extends Applet {
   public void init() {
      setLayout(null);
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      Label left = new Label("Left", Label.LEFT);
      Label right = new Label("Right", Label.RIGHT);
      Label center = new Label("Center", Label.CENTER);
      add(left);
      add(right);
      add(center);
      left.setBounds(0, 0, width, height / 3);
      right.setBounds(0, height / 3, width, height / 3);
      center.setBounds(0, 2 * height / 3, width, height / 3);
   }
}
```

На этот раз, чтобы одновременно переместить и изменить размер объектов Label, мы использовали метод reshape. Ширина каждой из меток равна полной ширине апплета, высота - 1/3 высоты апплета. Вот как этот апплет должен выглядеть, если его запустить:



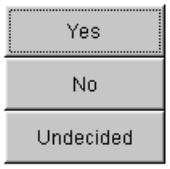
- Button

Объекты-кнопки помечаются строками, причем эти строки нельзя выравнивать подобно строкам объектов Label (они всегда центрируются внутри кнопки). Позднее в данной главе речь пойдет о том, как нужно обрабатывать события, возникающие при нажатии и отпускании пользователем кнопки. Ниже приведен пример, в котором создаются три расположенные по вертикали кнопки.

```
/* <applet code = "ButtonDemo" width=100 height=100>
   </applet>
 */
import java.awt.*;
```

```
import java.applet.*;

public class ButtonDemo extends Applet {
   public void init() {
      setLayout(null);
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      Button yes = new Button("Yes");
      Button no = new Button("No");
      Button maybe = new Button("Undecided");
      add(yes);
      add(no);
      add(maybe);
      yes.setBounds(0, 0, width, height / 3);
      no.setBounds(0, height / 3, width, height / 3);
      maybe.setBounds(0, 2 * height / 3, width, height / 3);
   }
}
```

Вот как выглядит работающий апплет:



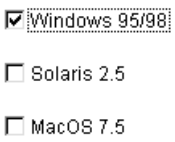
- Checkbox

Класс Checkbox часто используется для выбора одной из двух возможностей. При создании объекта Checkbox ему передается текст метки и логическое значение, чтобы задать исходное состояние окошка с отметкой. Программно можно получать и устанавливать состояние окошка с отметкой с помощью методов getState и setState. Ниже приведен пример с тремя объектами Checkbox, задаваемое в этом примере исходное состояние соответствует отметке в первом объекте.

```
/* <applet code = "CheckBoxDemo" width=120 height=100>
   </applet>
 */
import java.awt.*;
import java.applet.*;

public class CheckboxDemo extends Applet {
   public void init() {
      setLayout(null);
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      Checkbox win95 = new Checkbox("Windows 95/98", null, true);
      Checkbox Solaris = new Checkbox("Solaris 2.5");
      Checkbox mac = new Checkbox("MacOS 7.5");
```

```
      add(win95);
      add(solaris);
      add(mac);
      win95.setBounds(0, 0, width, height / 3);
      Solaris.setBounds(0, height / 3, width, height / 3);
      mac.setBounds(0, 2 * height / 3, width, height / 3);
   }
}
```

Ниже приведен внешний вид работающего апплета:



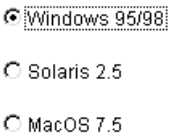
- CheckboxGroup

Второй параметр конструктора Checkbox (в предыдущем примере мы ставили там null) используется для группирования нескольких объектов Checkbox. Для этого сначала создается объект CheckboxGroup, затем он передается в качестве параметра любому количеству конструкторов Checkbox, при этом предоставляемые этой группой варианты выбора становятся взаимоисключающими (только один может быть задействован). Предусмотрены и методы, которые позволяют получить и установить группу, к которой принадлежит конкретный объект Checkbox - getCheckboxGroup и setCheckboxGroup. Вы можете пользоваться методами getCurrent и setCurrent для получения и установки состояния выбранного в данный момент объекта Checkbox. Ниже приведен пример, отличающийся от предыдущего тем, что теперь различные варианты выбора в нем взаимно исключают друг друга.

```
/* <applet code = "CheckboxGroupDemo" width=120 height=100>
   </applet>
 */
import java.awt.*;
import java.applet.*;

public class CheckboxGroupDemo extends Applet {
   public void init() {
      setLayout(null);
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      CheckboxGroup g = new CheckboxGroup();
      Checkbox win95 = new Checkbox("Windows 95/98", g, true);
      Checkbox solaris = new Checkbox("Solaris 2.5", g, false);
      Checkbox mac = new Checkbox("MacOS 7.5", g, false);
      add(win95);
      add(solaris);
      add(mac);
      win95.setBounds(0, 0, width, height / 3);
      solaris.setBounds(0, height / 3, width, height / 3);
```

```
      mac.setBounds(0, 2 * height / 3, width, height / 3);
   }
}
```

Обратите внимание - окошки изменили свою форму, теперь они не квадратные, а круглые:



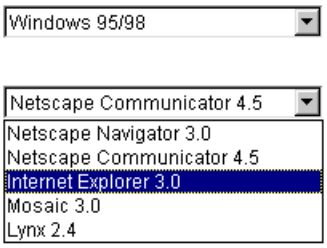
- Choice

Класс Choice (выбор) используется при создании раскрывающихся списочных меню (выпадающих списков типа ComboBox в Windows). Компонент Choice занимает ровно столько места, сколько требуется для отображения выбранного в данный момент элемента, когда пользователь щелкает мышью на нем, раскрывается меню со всеми элементами, в котором можно сделать выбор. Каждый элемент меню - это строка, которая выводится, выровненная по левой границе. Элементы меню выводятся в том порядке, в котором они были добавлены в объект Choice. Метод countItems возвращает количество пунктов в меню выбора. Вы можете задать пункт, который выбран в данный момент, с помощью метода select, передав ему либо целый индекс (пункты меню перечисляются с нуля), либо строку, которая совпадает с меткой нужного пункта меню. Аналогично, с помощью методов getSelectedItem и getSelectedIndex можно получить, соответственно, строку-метку и индекс выбранного в данный момент пункта меню. Вот очередной простой пример, в котором создается два объекта Choice.

```
/* <applet code = "ChoiceDemo" width=200 height=100>
   </applet>
 */
import java.awt.*;
import java.applet.*;

public class ChoiceDemo extends Applet {
   public void init() {
      setLayout(null);
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      Choice os = new Choice();
      Choice browser = new Choice();
      os.addItem("Windows 95/98");
      os.addItem("Solaris 2.5");
      os.addItem("MacOS 7.5");
      browser.addItem("Netscape Navigator 3.0");
      browser.addItem("Netscape Communicator 4.5");
      browser.addItem("Internet Explorer 3.0");
      browser.addItem("Mosaic 3.0");
      browser.addItem("Lynx 2.4");
      browser.select("Netscape Communicator 4.5");
      add(os);
```

```
      add(browser);
      os.setBounds(0, 0, width, height / 2);
      browser.setBounds(0, height / 2, width, height / 2);
   }
}
```

А вот как выглядят эти выпадающие списки:

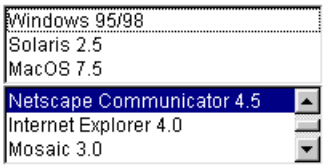


- List

Класс List представляет собой компактный список с возможностью выбора нескольких вариантов и с прокруткой (аналог ListBox в Windows). Ниже приведен пример с двумя списками выбора, один из которых допускает выбор нескольких элементов, а второй - выбор единственного элемента.

```
/* <applet code = "ListDemo" width=200 height=100>
   </applet>
 */
import java.awt.*;
import java.applet.*;

public class ListDemo extends Applet {
   public void init() { setLayout(null);
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      List os = new List(0, true);
      List browser = new List(0, false);
      os.addItem("Windows 95/98");
      os.addItem("Solaris 2.5");
      os.addItem("MacOS 7.5");
      browser.addItem("Netscape Navigator 3.0");
      browser.addItem("Netscape Communicator 4.5");
      browser.addItem("lnternet Explorer 4.0");
      browser.addItem("Mosaic 3.0");
      browser.addItem("Lynx 2.4");
      browser.select(1);
      add(os);
      add(browser);
      os.setBounds(0, 0, width, height / 2);
      browser.setBounds(0, height / 2, width, height / 2);
   }
}
```

Заметьте, что у нижнего списка имеется линейка прокрутки, поскольку все его элементы не уместились в заданный нами размер:



- Scrollbar

Объекты Scrollbar (линейки прокрутки) используются для выбора подмножества значений между заданными минимумом и максимумом. Визуально у линейки прокрутки есть несколько органов управления, ориентированных либо вертикально, либо горизонтально. Стрелки на каждом из ее концов показывают, что, нажав на них, вы можете продвинуться на один шаг в соответствующем направлении. Текущее положение отображается с помощью движка линейки прокрутки, которым пользователь также может управлять, устанавливая требуемое положение линейки.

Конструктор класса Scrollbar позволяет задавать ориентацию линейки прокрутки - для этого предусмотрены константы VERTICAL и HORIZONTAL. Кроме того, с помощью конструктора можно задать начальное положение и размер движка, а так же минимальное и максимальное значения, в пределах которых линейка прокрутки может изменять параметр. Для получения и установки текущего состояния линейки прокрутки используются методы getValue и setValue. Кроме того, воспользовавшись методами getMinimum и getMaximum, вы можете получить рабочий диапазон объекта. Ниже приведен пример, в котором создается и вертикальная, и горизонтальная линейки прокрутки.

```
/* <applet code = "ScrollbarDemo" width=200 height=100>
   </applet>
 */
import java.awt.*;
import java.applet.*;

public class ScrollbarDemo extends Applet {
   public void init() {
      setLayout(null);
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer. parseInt(getParameter("height"));
      Scrollbar hs = new Scrollbar(Scrollbar.HORIZONTAL, 50, width / 10, 0,
100);
      Scrollbar vs = new Scrollbar(Scrollbar.VERTICAL, 50, height / 2, 0,
100);
      add(hs);
      add(vs);
      int thickness = 16;
      hs.setBounds(0, height - thickness, width - thickness, thickness);
      vs.setBounds(width - thickness, 0, thickness, height - thickness);
   }
}
```

В этом примере скроллируется, конечно, пустая область:

- TextField

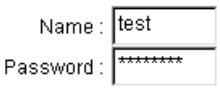
Класс TextField представляет собой реализацию однострочной области для ввода текста. Такие области часто используются в формах для пользовательского ввода. Вы можете "заморозить" содержимое объекта TextField с помощью метода setEditable, а метод isEditable сообщит вам, можно ли редактировать текст в данном объекте. Текущее значение объекта можно получить методом getText и установить методом setText. С помощью метода select можно выбрать фрагмент строки, задавая его начало и конец, отсчитываемые с нуля. Для выбора всей строки используется метод selectAll.

Метод setEchoChar задает символ, который будет выводиться вместо любых вводимых символов. Вы можете проверить, находится ли объект TextField в этом режиме, с помощью метода echoCharIsSet, и узнать, какой именно символ задан для эхо-печати, с помощью метода getEchoChar. Вот пример, в котором создаются классические поля для имени пользователя и пароля.

```
/* <applet code = "TextFieldDemo" width=200 height=100>
</applet>
*/
import java.awt.*;
import java.applet.*;

public class TextFieldDemo extends Applet {
   public void init() {
      setLayout(null);
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      Label namep = new Label("Name : ", Label.RIGHT);
      Label passp = new Label("Password : ", Label.RIGHT);
      TextField name = new TextField(8);
      TextField pass = new TextField(8);
      pass.setEchoChar('*');
      add(namep);
      add(name);
      add(passp);
      add(pass);
      int space = 25;
      int w1 = width / 3;
      namep.setBounds(0, (height - space) / 2, w1, space);
      name.setBounds(w1, (height - space) / 2, w1, space);
      passp.setBounds(0, (height + space) / 2, w1, space);
      pass.setBounds(w1, (height + space) / 2, w1, space);
```

```
    }
}
```

Ниже приведен внешний вид работающего апплета:

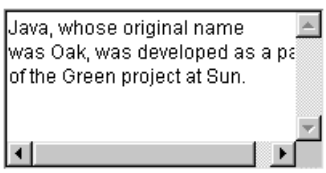


- TextArea

Порой одной строки текста оказывается недостаточно для конкретной задачи. AWT включает в себя очень простой многострочный редактор обычного текста, называемый TextArea. Конструктор класса TextArea воспринимает значение типа String в качестве начального текста объекта. Кроме того, в конструкторе указывается число колонок и строк текста, которые нужно выводить. Есть три метода, которые позволяют программе модифицировать содержимое объекта TextArea: appendText добавляет параметр типа String в конец буфера; insertText вставляет строку в заданное отсчитываемым от нуля индексом место в буфере; replaceText копирует строку-параметр в буфер, замещая ею текст, хранящийся в буфере между первым и вторым параметрами-смещениями. Ниже приведена программа, создающая объект TextArea и вставляющая в него строку.

```
/* <applet code = "TextAreaDemo" width=200 height=100>
   </applet>
 */
import java.awt.*;
import java.applet.*;

public class TextAreaDemo extends Applet {
   public void init() {
      setLayout(null);
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      String val = "Java, whose original name\n"+
         "was Oak, was developed as a part\n"+
         "of the Green project at Sun.\n";
      System.out.println(val);
      TextArea text = new TextArea(val, 80, 40);
      add(text);
      text.setBounds(0, 0, width, height);
   }
}
```

Ниже приведен внешний вид работающего апплета:



5. Менеджеры компоновки

- Layout

Все компоненты, с которыми мы работали до сих пор в этой главе, размещались "вручную". И в каждом примере мы вызывали загадочный метод setLayout(null). Этот вызов запрещал использование предусмотренного по умолчанию механизма управления размещением компонентов. Для решения подобных задач в AWT предусмотрены диспетчеры размещения (layout managers).

- LayoutManager

Каждый класс, реализующий интерфейс LayoutManager, следит за списком компонентов, которые хранятся с именами типа String. Всякий раз, когда вы добавляете компонент в Panel, диспетчер размещения уведомляется об этом. Если требуется изменить размер объекта Panel, то идет обращение к диспетчеру посредством методов minimumLayoutSize и preferredLayoutSize. В каждом компоненте, который приходится обрабатывать диспетчеру, должны присутствовать реализации методов preferredSize и minimumSize. Эти методы должны возвращать предпочтительный и минимальный размеры для прорисовки компонента, соответственно. Диспетчер размещения по возможности будет пытаться удовлетворить эти запросы, в то же время заботясь о целостности всей картины взаимного расположения компонентов.

В Java есть несколько предопределенных классов - диспетчеров размещения, описываемых ниже.

- FlowLayout

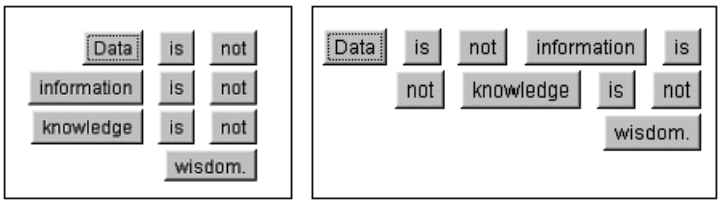
Класс FlowLayout реализует простой стиль размещения, при котором компоненты располагаются, начиная с левого верхнего угла, слева направо и сверху вниз. Если в данную строку не помещается очередной компонент, он располагается в левой позиции новой строки. Справа, слева, сверху и снизу компоненты отделяются друг от друга небольшими промежутками. Ширину этого промежутка можно задать в конструкторе FlowLayout. Каждая строка с компонентами выравнивается по левому или правому краю, либо центрируется в зависимости от того, какая из констант LEFT, RIGHT или CENTER была передана конструктору. Режим выравнивания по умолчанию - CENTER, используемая по умолчанию ширина промежутка - 5 пикселей.

Ниже приведен пример, в котором в Panel включается несколько компонентов Label. Объект Panel использует FlowLayout с выравниванием RIGHT.

```
/* <applet code = "FlowLayoutDemo" width=200 height=100>
   </applet>
   <applet code = "FlowLayoutDemo" width=250 height=100>
   </applet>
 */

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

public class FlowLayoutDemo extends Applet {
   public void init() {
```

```
      setLayout(new FlowLayout(FlowLayout.RIGHT, 10, 3));
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      String val = "Data is not information " +
      "is not knowledge is not wisdom.";
      StringTokenizer st = new StringTokenizer(val);
      while (st.hasMoreTokens()) {
         add(new Button(st.nextToken()));
      }
   }
}
```

Необходимо вызвать пример для двух различных страниц (ширина апплета 200 и 250 пикселей) для того, чтобы проиллюстрировать, как объекты Label перетекают из строки в строку, и при этом строки выравниваются по правому краю:



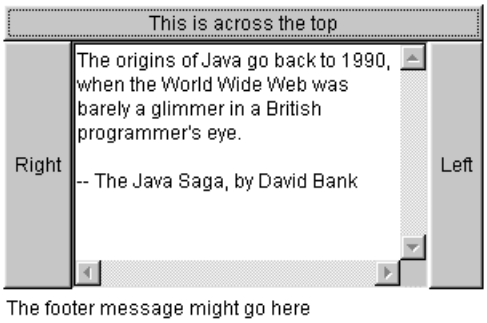
- BorderLayout

Класс BorderLayout реализует обычный стиль размещения для окон верхнего уровня, в котором предусмотрено четыре узких компонента фиксированной ширины по краям, и одна большая область в центре, которая может расширяться и сужаться в двух направлениях, занимая все свободное пространство окна. У каждой из этих областей есть строки-имена: String.North, String.South, String.East и String.West соответствуют четырем краям, а Center - центральной области. Ниже приведен пример BorderLayout с компонентом в каждой из названных областей.

```
/* <applet code = "BorderLayoutDemo" width=300 height=200>
   </applet>
 */
import java.awt.*;
import java.applet.*;
import java.util.*;

public class BorderLayoutDemo extends Applet {
   public void init() {
      setLayout(new BorderLayout());
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      add("North", new Button("This is across the top"));
      add("South", new Label("The footer message might go here"));
      add("East", new Button("Left"));
      add("West", new Button("Right"));
      String msg = "The origins of Java go back to 1990,\n"+
         "when the World Wide Web was\n"+
```

```
         "barely a glimmer in a British\n"+
         "programmer's eye.\n\n"+
         "-- The Java Saga, by David Bank";
      add("Center", new TextArea(msg));
   }
}
```

Ниже приведен внешний вид работающего апплета:

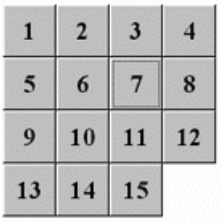


- GridLayout

Класс GridLayout размещает компоненты в простой равномерной сетке. Конструктор этого класса позволяет задавать количество строк и столбцов. Ниже приведен пример, в котором GridLayout используется для создания сетки 4x4, 15 квадратов из 16 заполняются кнопками, помеченными соответствующими индексами. Как вы уже, наверное, поняли, это - панель для игры в "пятнашки".

```
/* <applet code = "GridLayoutDemo" width=200 height=200>
   </applet>
 */
import java.awt.*;
import java.applet.*;

public class GridLayoutDemo extends Applet {
   static final int n = 4;
   public void init() {
      setLayout(new GridLayout(n, n));
      setFont(new Font("Helvetica", Font.BOLD, 24));
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
            int k = i * n + j;
            if (k > 0)
            add(new Button("" + k));
         }
      }
   }
}
```

Если доработать этот пример - получится неплохая игра:



На самом деле, если подобное игровое поле состоит из большого количества активных полей (как например, стандартная игра "Сапер" в Windows), то использовать AWT-кнопки будет уже не рационально, так как множество собственных компонент Windows будут заметно замедлять работу приложения.

- Insets

Класс Insets используется для того, чтобы вставлять в объект Panel границы, напоминающие горизонтальные и вертикальные промежутки между объектами, которые делает диспетчер размещения. Для того, чтобы добиться вставки границ в объект Panel, нужно заместить метод Insets реализацией, возвращающей новый объект Insets с четырьмя целыми значениями, соответствующими ширине верхнего, нижнего, левого и правого краев.

```
public Insets insets() {
   return new Insets(10, 10, 10, 10);
}
```

- CardLayout

Класс CardLayout по своему уникален. Он отличается от других программ управления размещением компонентов тем, что представляет несколько различных вариантов размещения, которые можно сравнить с колодой карт. Колоду можно тасовать так, чтобы в данный момент времени наверху была только одна из карт. Это может быть полезно при создании интерфейсов пользователя, в которых есть необязательные компоненты, включаемые и выключаемые динамически в зависимости от реакции пользователя.

6. Окна

- Window

Класс Window во многом напоминает Panel за тем исключением, что он создает свое собственное окно верхнего уровня. Большая часть программистов скорее всего будет использовать не непосредственно класс Window, а его подкласс Frame.

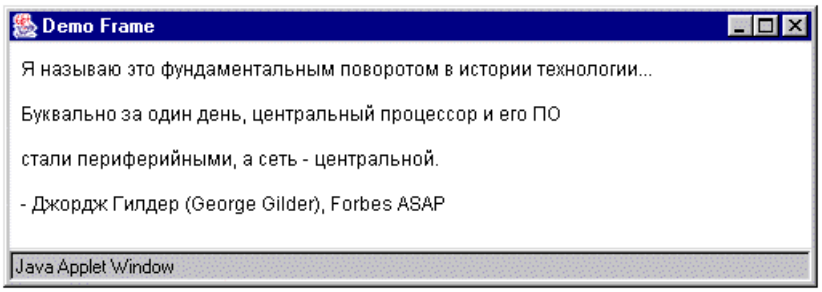
- Frame

Frame - это как раз то, что обычно и считают окном на рабочей поверхности экрана. У объекта Frame есть строка с заголовком, управляющие элементы для изменения размера и линейка меню. Для того чтобы вывести/спрятать изображение объекта Frame, нужно использовать методы show и hide. Ниже приведен пример апплета, который показывает объект Frame с содержащимся в нем компонентом TextArea.

```
/* <applet code = "FrameDemo" width=200 height=200>
   </applet>
```

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

public class FrameDemo extends Applet {
   public void init() {
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));

      Frame f = new Frame("Demo Frame");
      f.setSize(width, height);
      f.setLayout(new FlowLayout(FlowLayout.LEFT));

      f.add(new Label("Я называю это фундаментальным поворотом в истории
технологии..."));
      f.add(new Label("Буквально за один день, центральный процессор и его
ПО"));
      f.add(new Label("стали периферийными, а сеть - центральной."));
      f.add(new Label("- Джордж Гилдер (George Gilder), Forbes ASAP"));

      f.show();
   }
}
```

Вот как выглядит такой фрейм:



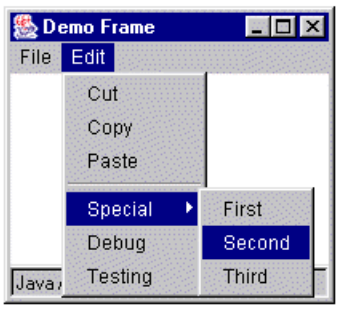
7. Меню

С каждым окном верхнего уровня может быть связана линейка меню. Объект MenuBar может включать в себя несколько объектов Menu. Последние, в свою очередь, содержат в себе список вариантов выбора - объектов MenuItem. Menu - подкласс MenuItem, так что объекты Menu также могут включаться в этот список, что позволяет создавать иерархически вложенные подменю. Вот пример, в котором к окну добавлены несколько вложенных меню.

```
/* <applet code = "MenuDemo" width=200 height=200>
   </applet>
 */
import java.awt.*;
import java.applet. *;
```

```
public class MenuDemo extends Applet {
   public void init() {
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      Frame f = new Frame("Demo Frame");
      f.setSize(width, height);
      MenuBar mbar = new MenuBar();
      f.setMenuBar(mbar);
      Menu file = new Menu("File");
      file.add(new MenuItem("New... "));
      file.add(new MenuItem("Open..."));
      file.add(new MenuItem("Close"));
      file.add(new MenuItem("-"));
      file.add(new MenuItem("Quit..."));
      mbar.add(file);
      Menu edit = new Menu("Edit");
      edit.add(new MenuItem("Cut"));
      edit.add(new MenuItem("Copy"));
      edit.add(new MenuItem("Paste"));
      edit.add(new MenuItem("-"));
      Menu sub = new Menu("Special");
      sub.add(new MenuItem("First"));
      sub.add(new MenuItem("Second"));
      sub.add(new MenuItem("Third"));
      edit.add(sub);
      edit.add(new CheckBoxMenuItem("Debug"));
      edit.add(new CheckBoxMenuItem("Testing"));
      mbar.add(edit);
      f.show();
   }
}
```

Посмотрим на практически классическое меню:



8. Обработка событий

Модель обработки событий в AWT представляет собой, по существу, модель обратных вызовов (callback). При создании GUI-элемента ему сообщается, какой метод или методы

он должен вызывать при возникновении в нем определенного события (нажатия кнопки, мыши и т.п.). Эту модель очень легко использовать в C++, поскольку этот язык позволяет оперировать указателями на методы (чтобы определить обратный вызов, необходимо всего лишь передать указатель на функцию). Однако в Java это недопустимо (методы не являются объектами). Поэтому для реализации модели необходимо определить класс, реализующий некоторый специальный интерфейс. Затем можно передать экземпляр такого класса GUI-элементу, обеспечивая таким образом обратный вызов. Когда наступит ожидаемое событие, GUI-элемент вызовет соответствующий метод объекта, определенного ранее.

Модель обработки событий Java используется как в пакете AWT, так и в JavaBeans API. В этой модели разным типам событий соответствуют различные классы Java. Каждое событие является подклассом класса java.util.EventObject. События пакета AWT, которые и рассматриваются в данной главе, являются подклассом java.awt.AWTEvent. Для удобства события различных типов пакета AWT (например, MouseEvent или ActionEvent) помещены в новый пакет java.awt.event.

Для каждого события существует порождающий его объект, который можно получить с помощью метода getSource(), и каждому событию пакета AWT соответствует определенный идентификатор, который позволяет получить метод getID(). Это значение используется для того, чтобы отличать события различных типов, которые могут описываться одним и тем же классом событий. Например, для класса FocusEvent возможны два типа событий: FocusEvent.FOCUS_GAINED и FocusEvent.FOCUS_LOST. Подклассы событий содержат информацию, связанную с данным типом события. Например, в классе MouseEvent существуют методы getX(), getY() и getClickCount (). Этот класс наследует, в числе прочих, и методы getModifiers() и getWhen().

Модель обработки событий Java базируется на концепции слушателя событий. Слушателем события является объект, заинтересованный в получении данного события. В объекте, который порождает событие (в источнике событий), содержится список слушателей, заинтересованных в получении уведомления о том, что данное событие произошло, а также методы, которые позволяют слушателям добавлять или удалять себя из этого списка. Когда источник порождает событие (или когда объект источника зарегистрирует событие, связанное с вводом информации пользователем), он оповещает все объекты слушателей событий о том, что данное событие произошло.

Источник события оповещает объект слушателя путем вызова специального метода и передачи ему объекта события (экземпляра подкласса EventObject). Для того чтобы источник мог вызвать данный метод, он должен быть реализован для каждого слушателя. Это объясняется тем, что все слушатели событий определенного типа должны реализовывать соответствующий интерфейс. Например, объекты слушателей событий ActionEvent должны реализовывать интерфейс ActionListener. В пакете java.awt.event содержатся интерфейсы слушателей для каждого из определенных в нем типов событий (например, для событий MouseEvent здесь определено два интерфейса слушателей: MouseListener и MouseMotionListener). Все интерфейсы слушателей событий являются расширениями интерфейса java.util.EventListener. В этом интерфейсе не определяется ни один из методов, но он играет роль интерфейса-метки, в котором однозначно определены все слушатели событий как таковые.

В интерфейсе слушателя событий может определяться несколько методов. Например, класс событий, подобный MouseEvent, описывает несколько событий, связанных с мышью,

таких как события нажатия и отпускания кнопки мыши. Эти события вызывают различные методы соответствующего слушателя. По установленному соглашению, методам слушателей событий может быть передан один единственный аргумент, являющийся объектом того события, которое соответствует данному слушателю. В этом объекте должна содержаться вся информация, необходимая программе для формирования реакции на данное событие. В таблице 6 приведены определенные в пакете java.awt.event типы событий, соответствующие им слушатели, а также методы, определенные в каждом интерфейсе слушателя.

Таблица 1. Типы событий, слушатели и методы слушателей в Java

Класс события	Интерфейс слушателя	Методы слушателя
ActionEvent	ActionListener	actionPerformed()
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged()
ComponentEvent	ComponentListener	componentHidden() componentMoved() componentResized() componentShown()
ContainerEvent	ContainerListener	componentAdded() componentRemoved()
FocusEvent	FocusListener	focusGained() focusLost ()
ItemEvent	ItemListener	itemStateChanged()
KeyEvent	KeyListener	keyPressed() keyReleased() keyTyped()
MouseEvent	MouseListener	mouseClicked() mouseEntered() mouseExited() mousePressed() mouseReleased()
MouseMotionEvent	MouseMotionListener	mouseDragged() mouseMoved()
TextEvent	TextListener	textValueChanged()
WindowEvent	WindowListener	windowActivated() windowClosed() windowClosing() windowDeactivated() windowDeiconified() windowIconified() windowOpened()

Для каждого интерфейса слушателей событий, содержащего несколько методов, в пакете java.awt.event определен простой класс-адаптер, который обеспечивает пустое тело для каждого из методов соответствующего интерфейса. Когда нужен только один или два таких метода, иногда проще получить подкласс класса-адаптера, чем реализовать интерфейс самостоятельно. При получении подкласса адаптера требуется лишь переопределить те методы, которые нужны, а при прямой реализации интерфейса необходимо определить все методы, в том числе и ненужные в данной программе. Заранее определенные классы-адаптеры называются так же, как и интерфейсы, которые они реализуют, но в этих названиях Listener заменяется на Adapter: MouseAdapter, WindowAdapter и т.д.

Как только реализован интерфейс слушателя или получены подклассы класса-адаптера, необходимо создать экземпляр нового класса, чтобы определить конкретный объект слушателя событий. Затем этот слушатель должен быть зарегистрирован соответствующим

источником событий. В программах пакета AWT источником событий всегда является какой-нибудь элемент пакета. В методах регистрации слушателей событий используются стандартные соглашения об именах: если источник событий порождает события типа X, в нем существует метод addXListener () для добавления слушателя и метод removeXListener() для его удаления. Одной из приятных особенностей модели обработки событий Java является возможность легко определять типы событий, которые могут порождаться данным элементом. Для этого следует просто просмотреть, какие методы зарегистрированы для его слушателя событий. Например, из описания API для объекта класса Button следует, что он порождает события ActionEvent. В таблице 7 приведен список элементов пакета AWT и событий, которые они порождают.

Таблица 2. Элементы пакета AWT и порождаемые ими события в Java1.1

Элемент	Порождаемое событие	Значение
Button	ActionEvent	Пользователь нажал кнопку
CheckBox	ItemEvent	Пользователь установил или сбросил флажок
CheckBoxMenuItem	ItemEvent	Пользователь установил или сбросил флажок рядом с пунктом меню
Choice	ItemEvent	Пользователь выбрал элемент списка или отменил его выбор
Component	ComponentEvent	Элемент либо перемещен, либо он стал скрытым, либо видимым
	FocusEvent	Элемент получил или потерял фокус ввода
	KeyEvent	Пользователь нажал или отпустил клавишу
	MouseEvent	Пользователь нажал или отпустил кнопку мыши, либо курсор мыши вошел или покинул область, занимаемую элементом, либо пользователь просто переместил мышь или переместил мышь при нажатой кнопке мыши
Container	ContainerEvent	Элемент добавлен в контейнер или удален из него
List	ActionEvent	Пользователь выполнил двойной щелчок мыши на элементе списка
	ItemEvent	Пользователь выбрал элемент списка или отменил выбор
MenuItem	ActionEvent	Пользователь выбрал пункт меню
Scrollbar	AdjustmentEvent	Пользователь осуществил прокрутку
TextComponent	TextEvent	Пользователь внес изменения в текст элемента
TextField	ActionEvent	Пользователь закончил редактирование текста элемента
Window	WindowEvent	Окно было открыто, закрыто, представлено в виде пиктограммы, восстановлено или требует восстановления

8.1. Рисование "каракулей" в Java

Классический апплет, в котором используется модель обработки событий Java. В этом примере реализованы интерфейсы MouseListener и MouseMotionListener, регистрирующие себя с помощью своих же методов addMouseListener() и addMouseMotionListener().

```
/* <applet code = "Scribble2" width=200 height=200>
 * </applet>
 */
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Scribble2 extends Applet implements
MouseListener, MouseMotionListener {
   private int last_x, last_y;

   public void init() {
      // Сообщает данному апплету о том, какие объекты
      // классов MouseListener и MouseMotionListener он должен оповещать
      // о событиях, связанных с мышью и ее перемещением.
      // Поскольку интерфейс реализуется в самом апплете,
      // при этом будут вызываться методы апплета.
      this.addMouseListener(this) ;
      this.addMouseMotionListener(this);
   }

   // Метод интерфейса MouseListener. Вызывается при нажатии
   // пользователем кнопки мыши.
   public void mousePressed(MouseEvent e) {
      last_x = e.getX();
      last_y = e.getY();
   }

   // Метод интерфейса MouseMotionListener. Вызывается при
   // перемещении мыши с нажатой кнопкой.
   public void mouseDragged(MouseEvent e) {
      Graphics g = this.getGraphics();
      int x = e.getX(), y = e.getY();
      g.drawLine(last_x, last_y, x, y);
      last_x = x; last_y = y;
   }

   // Другие, не используемые методы интерфейса MouseListener.
   public void mouseReleased(MouseEvent e) {;}
   public void mouseClicked(MouseEvent e) {;}
   public void mouseEntered(MouseEvent e) {;}
   public void mouseExited(MouseEvent e) {;}
```

```
   // Другой метод интерфейса MouseMotionListener.
   public void mouseMoved(MouseEvent e) {;}
}
```

Экран вроде бы пустой - но на нем можно рисовать:

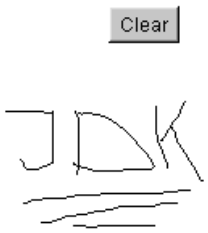
8.2. Рисование "каракулей" с использованием встроенных классов

Модель обработки событий Java разработана с учетом того, чтобы хорошо сочетаться с другой особенностью Java: встроенными классами. В следующем примере показано, как изменится данный апплет, если слушатели событий будут реализованы в виде анонимных встроенных классов. Обратите внимание на компактность данного варианта программы. Новая особенность, добавленная в апплет - кнопка Clear. Для этой кнопки зарегистрирован объект ActionListener, а сама она выполняет очистку экрана при наступлении соответствующего события.

```
/* <applet code = "Scribble3" width=200 height=200>
 * </applet>
 */
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Scribble3 extends Applet {
   int last_x, last_y;

   public void init() {
      // Определяет, создает и регистрирует объект MouseListener.
      this.addMouseListener(new MouseAdapter() {
         public void mousePressed(MouseEvent e) {
            last_x = e.getX(); last_y = e.getY();
         }
      });

      // Определяет, создает и регистрирует объект MouseMotionListener.
      this.addMouseMotionListener(new MouseMotionAdapter() {
         public void mouseDragged(MouseEvent e) {
            Graphics g = getGraphics();
            int x = e.getX(), y= e.getY();
            g.setColor(Color.black);
            g.drawLine(last_x, last_y, x, y);
            last_x = x; last_y = y;
         }
```

```
        });

        // Создает кнопку Clear.
        Button b = new Button("Clear");
        // Определяет, создает и регистрирует объект слушателя
        // для обработки события, связанного с нажатием кнопки.
        b.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              // стирание каракулей
              Graphics g = getGraphics();
              g.setColor(getBackground());
              g.fillRect(0, 0, getSize().width, getSize().height);
           }
        });

        // Добавляет кнопку в апплет.
        this.add(b);
    }
}
```

Теперь апплет выглядит так:



Обратите внимание, что в этот пример порождает 3 вспомогательных класса: Scribble3$1,2,3.

9. Заключение

В этой главе изучается построение графического интерфейса пользователя (GUI) с помощью Java, в которой для этой цели предназначена библиотека AWT.

Рассмотрение начинается с апплетов, небольших программ, которые предназначены для работы в браузерах как небольшие части HTML-страниц. Во-первых, необходимо использовать специальный тег, чтобы разместить апплет на странице. В частности, можно указывать специальные параметры, чтобы апплет можно было настраивать без перекомпиляции кода. Во-вторых, рассматриваются этапы жизненного цикла апплета, который отличается от цикла обычного приложения, которое запускается методом main. Наконец, рассматриваются способы рисования в Java – абстрактный класс Graphics, работа с цветами, шрифтами.

Затем описываются стандартные компоненты AWT, которые иерархически упорядочены в дерево наследования с классом Component в вершине. Важным его наследником является класс Container, который может хранить набор компонент. Прямые наследники Component

составляют набор управляющих элементов («контролов» от англ. controls), а наследники Container – набор контейнеров для группировки и расположения компонент. Для упрощения размещения отдельных элементов пользовательского интерфейса применяются менеджеры компоновки (Layout managers).

Один из наследников Container – класс Window, который представляет собой самостоятельное окно в многооконной операционной системе. Два его наследника – Dialog и Frame. Для работы с файлами определен наследник Dialog – FileDialog.

Для построения меню используется свое небольшое дерево наследования с MenuComponent в качестве вершины.

Наконец, излагаются принципы модели событий от пользователя, позволяющей обрабатывать все действия, которые производит клиент, работая с программой. 11 событий и соответствующих им интерфейсов предоставляют все необходимое для написания полноценной GUI-программы.

10. Контрольные вопросы

11-1. От какого класса наследуется класс Applet?

a.) java.awt.Panel

11-2. Может ли быть дважды вызван метод init у апплета? Метод start?

a.) Метод init вызывается только один раз при конструировании апплета. Метод start может быть вызван многократно, если пользователь покидал и возвращался на страницу.

11-3. Чем различаются методы paint, update, repaint?

a.) paint определяет внешний вид компоненты, в нем описывается отрисовка всех внешних элементов

update сначала закрашивает всю компоненту фоновым (background) цветом, а затем вызывает paint

repaint не перерисовывает компоненту напрямую, он инициирует вызов метода update через указанный промежуток времени

11-4. Как создать объект класса Color, описывающий чистый синий цвет?

a.) new Color(0,0,255). Также можно воспользоваться константой Color.blue

11-5. Какими параметрами в Java характеризуется шрифт?

a.) Имя семейства шрифта, размер (в пунктах), стиль (обычный, жирный, наклонный).

11-6. Для чего нужен класс FontMetrics?

a.) Поскольку размер шрифта задается в пунктах, а отображение текста делается с помощью шрифтов, которые поддерживаются операционной системой, необходима специальная утилита для вычисления размера

шрифта в пикселах. Это может потребоваться для точного позиционирования текста в компоненте.

Класс FontMetrics предоставляет набор методов для получения отдельных параметров шрифта, таких как ширина слова, высота шрифта и другие.

11-7. Напишите класс-компоненту, у которого по центру рисуется квадрат размерами 10x10.

a.)
```
public class SquareComponent extends Canvas {
   public void paint(Graphics g) {
      g.drawRect(getWidth()-5, getHeight()-5, 10, 10);
   }
}
```

11-8. Как в AWT создаются компоненты чекбокс (check-box)? Радио-кнопка (radio-button)?

a.) чек бокс порождается компонентой Checkbox:

```
Checkbox chbox = new Checkbox("название");
```

Для создания радио-кнопок необходимо связать несколько компонент Checkbox с помощью класса CheckboxGroup:

```
CheckboxGroup group = new CheckboxGroup();
Checkbox  rb1 = new Checkbox("режим 1",group, true);
Checkbox  rb2 = new Checkbox("режим 2",group, false);
```

11-9. В чем разница между компонентами List и Choice?

a.) List отображает несколько элементов списка сразу, а Choise только один. Так же в List может быть выбрано несколько элементов (если установлено свойство multiselect), а в Choice только один.

11-10. Для чего нужны менеджеры компоновки? Исходя из каких параметров они выполняют свою работу?

a.) Для автоматического расположения компонент внутри контейнера. Менеджер компоновки может установить размер и местоположение компоненты, и в дальнейшем не придется проводить дополнительную работу, если изменился размер окна или компонент, или их количество.

При компоновке учитываются следующие параметры:

- размер контейнера
- количество компонент и порядок их следования
- начальный размер и положение компонент
- constraints, устанавливаемый при добавлении компоненты
- дополнительные свойства самого менеджера (отступы между компонентами и т.п.)

11-11. В чем разница между Dialog и Frame?

a.) Основное различие заключается в том, что Frame – самостоятельное окно, а Dialog всегда привязан к Frame. Только Dialog обладает свойством модальности. Только Frame может иметь главное меню. Dialog нельзя минимизировать или максимизировать.

11-12. Какие действия необходимо произвести, чтобы создать компонент и подписаться на событие, которое он генерирует?

a.) Сначала создается сама компонента. Затем создается класс-слушатель, реализующий соответствующий Listener-интерфейс, который будет реагировать на появление события. Наконец, вызывается метод add<…>Listener, который регистрирует слушателя.

11-13. Как узнать, какие события генерирует стандартный компонент?

a.) Необходимо посмотреть, каким методы add<…>Listener в нем объявлены или унаследованы от родительского класса.