1.1. Введение в AWT: работа с окнами, графикой и текстом

AWT содержит классы и методы, которые позволяют создавать окна и управлять ими.

Хотя основное предназначение AWT состоит в поддержке окон апплета, ее можно также использовать для создания автономных окон, которые работают в среде GUI операционной системы Windows. Большинство примеров содержится в апплетах, так что для их выполнения нужно использовать программу просмотра апплетов или java-совместимый Web-браузер. Несколько примеров демонстрируют создание автономных оконных программ.

Классы AWT

Классы AWT содержатся в пакете java.awt. Это один из самых больших пакетов Java. К счастью, он организован нисходящим, иерархическим способом, поэтому легок в понимании и использовании. Табл. 14.1 перечисляет некоторые из классов AWT.

Таблица 14.1. Некоторые AWT-классы

Класс	Описание
`AWTEvent`	Инкапсулирует AWT-события
`AWTEventMulticaster`	Рассылает события множеству слушателей
`BorderLayout`	Менеджер граничной (Border) компоновки. Граничная компоновка использует пять компонентов: North, South, East, West и Center (Север, Юг, Восток, Запад и Центр)
`Button`	Создает элемент управления командная кнопка
`Canvas`	Пустое, свободное от семантики окно
`CardLayout`	Менеджер карточной (Card) компоновки. Карточная компоновка моделируют пронумерованную колоду карт. Показывается только карта, находящаяся сверху
`Checkbox`	Создает элемент управления флажок
`CheckboxGroup`	Создает группу элементов управления флажок
`CheckboxMenuItem`	Создает помеченный пункт меню
`Choice`	Создает раскрывающийся (pop-up) список
`Color`	Управляет цветами переносимым, независимым от платформы способом
`Component`	Абстрактный суперкласс для различных AWT-компонентов

`Container`	Подкласс Component, который может содержать другие компоненты
`Cursor`	Инкапсулирует растровый курсор
`Dialog`	Создает окно диалога
`Dimension`	Определяет измерения объекта. Ширина сохраняется в width, а высота — в height
`Event`	Инкапсулирует события
`EventQueue`	Организует очереди событий
`FileDialog`	Создает окно, из которого может быть выбран файл
`FlowLayout`	Менеджер поточной (Flow) компоновки. Поточная компоновка размещает компоненты слева направо, сверху вниз
`Font`	Инкапсулирует шрифт печати
`FontMetrics`	Инкапсулирует различную информацию, связанную с шрифтом. Эта информация помогает отображать текст в окне
`Frame`	Создает стандартное окно (фрейм), которое имеет строку заголовка, углы, изменяющие размеры и строку меню
`Graphics`	Инкапсулирует графический контекст. Этот контекст используется различными методами вывода для отображения вывода в окне
`GraphicsDevice`	Описывает графическое устройство типа экрана или принтера
`GraphicsEnvironment`	Описывает коллекцию доступных объектов классов Font и GraphicsDevice
`GridBagConstraints`	Определяет различные ограничения, касающиеся класса
`GridBagLayout`	Менеджер ячеистой (Grid Bag) компоновки. Ячеистая компоновка отображает компоненты в ячейках, подчиненных ограничениям, указанным в GridBagConstraints
`GridLayout`	Менеджер сеточной (Grid) компоновки. Сеточная компоновка отображает компоненты в двумерной таблице
`Image`	Инкапсулирует графические изображения
`Insets`	Инкапсулирует границы контейнера
`Label`	Создает метку, которая отображает строку
`List`	Создает список, из которого пользователь может делать выбор. Подобен стандартному списку Windows
`MediaTracker`	Управляет объектами среды

Menu	Создает выпадающее (pull-down) меню
MenuBar	Создает строку меню
MenuComponent	Абстрактный класс, реализованный различными классами меню
MenuItem	Создает пункт меню
MenuShortCut	Инкапсулирует быструю клавишу (сочетание клавиш) для пункта меню
Panel	Самый простой конкретный подкласс класса Container
Point	Инкапсулирует пару декартовых координат, сохраняемых в переменных x и y
Polygon	Инкапсулирует многоугольник
PopupMenu	Инкапсулирует раскрывающееся (pop-up) меню
PrintJob	Абстрактный класс, который представляет задание для печати
Rectangle	Инкапсулирует прямоугольник
Scrollbar	Создает элемент управления полоса прокрутки
ScrollPanel	Контейнер, который обеспечивает горизонтальные и/или вертикальные полосы прокрутки для другого компонента
SystemColor	Содержит цвета GUI элементов управления окном, таких как окна, полосы прокрутки, текст и пр.
TextArea	Создает элемент управления с многострочным редактированием
TextComponent	Суперкласс для TextArea и TextField
TextField	Создает элемент управления с однострочным редактированием
Toolkit	Абстрактный класс, реализованный в AWT
Window	Создает окно без границы, строки меню и заголовка

1.2. Основы оконной графики

AWT определяет окна согласно иерархии классов, которая с каждым уровнем добавляет функциональные возможности и специфику. Два наиболее общих типа окон являются производными от типа Panel, который пользуется апплетами, и от типа Frame, который создает стандартное окно. Многое из функциональных возможностей этих окон получено от их родительских классов. Иерархия классов, связанных с классами Panel и Frame, показана на рис. 1.

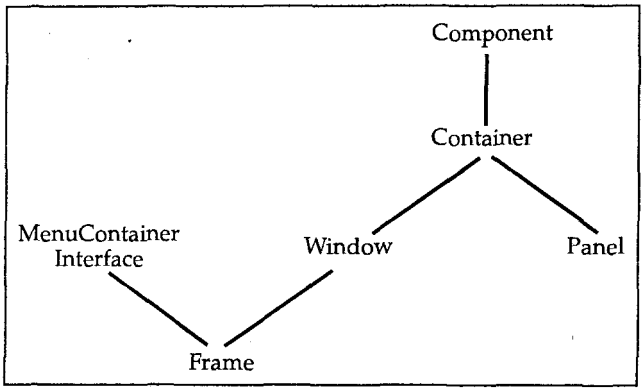


Рис. 1. Иерархия классов для Panel и Frame

Рассмотрим каждый из этих классов.

Класс Component

Самый верхний в AWT-иерархии — класс Component. Это абстрактный класс, который инкапсулирует все атрибуты визуального компонента. Все элементы интерфейса пользователя, которые отображены на экране и взаимодействуют с пользователем, — это подклассы Component. В классе Component определено более сотни public-методов, которые являются ответственными за управление событиями, такими как ввод с помощью мыши и клавиатуры, позиционирование и изменение размеров окна, перерисовка и т. д. (многие из этих методов уже использовались при создании апплетов). Объект класса Component отвечает за запоминание текущих цветов переднего плана и фона и выбранного текстового шрифта.

Класс Container

Класс Container является подклассом Component. Он содержит дополнительные методы, которые позволяют вкладывать в него другие Component-объекты. Внутри класса Container могут храниться и его собственные объекты. Это делает Container *системой многоуровневого включения*. Контейнер отвечает за размещение (т. е. позиционирование) любых компонентов, которые он содержит. Он делает это с помощью различных менеджеров компоновки (размещения).

Класс Panel

Класс Panel — конкретный подкласс Container. Он не добавляет каких-либо новых методов. это просто реализация класса Container. О

Panel можно мыслить как о рекурсивно вкладываемом конкретном экранном компоненте. Panel — суперкласс для Applet. Когда экранный вывод направляется к апплету, он рисуется на поверхности объекта Panel. В сущности, объект Panel — это окно, которое не содержит области заголовка, строки меню и обрамления. Вот почему, когда апплет выполняется внутри браузера, вы не видите эти элементы. Если апплет выполняется с помощью утилиты просмотра апплетов (appletviewer), то заголовок и обрамление такого окна обеспечивается самой утилитой.

К Panel-объекту можно добавить другие компоненты с помощью метода add() (унаследованного из Container). Как только эти компоненты добавлены, можно позиционировать и изменять их размеры вручную, используя методы setLocation(), setSize() или setBounds(), определенные в Component.

Класс Window

Класс Window создает окно верхнего уровня. *Окно верхнего уровня* не содержится в любом другом объекте. Оно находится непосредственно на Рабочем столе. Объекты класса Window не создаются непосредственно. Вместо этого используется подкласс Window с именем Frame.

Класс Frame

Класс Frame инкапсулирует то, что обычно представляют как "окно", то есть прямоугольную область экрана в управляемой рамке. Это подкласс Window и его окно имеет строку заголовка, строку меню, обрамление и углы, изменяющие размеры окна. Если создать в апплете объект типа Frame, то он будет содержать сообщение вида "Warning: Applet Window", говорящее о том, что окно было создано апплетом. Это сообщение предупреждает пользователей, что окно, которое они видят, было запущено апплетом, а не программным обеспечением, выполняющимся на их компьютере. (Чтобы получать пароли и другую секретную информацию, неизвестную пользователям, можно было бы использовать апплет, который способен маскироваться под хост-приложение.) Когда фрейм создается обычной программой, а не апплетом, то строится нормальное окно.

Класс Canvas

Хотя Canvas не является частью Container-иерархии, существует еще один тип окна, который может оказаться полезным. Это — класс Canvas. Canvas-объект моделирует *пустое окно,* в котором можно рисовать.

1.3. Работа с фреймовыми окнами

Тип окна, который вы будете чаще всего создавать, является производным от Frame. Он используется для создания окон верхнего уровня и дочерних окон для апплетов и приложений. Как говорилось выше, *фрейм* — это окно со стандартным стилем (с заголовком, меню, обрамлением, управляющими уголками). Frame поддерживает два конструктора:

```
Frame ()
Frame (String заголовок)
```

Первая форма создает стандартное окно, которое не содержит заголовка. Вторая форма создает окно с заголовком, указанным в параметре заголовок. Обратите внимание, что вы не можете определить размеры окна, поэтому следует устанавливать размеры окна уже после того, как оно было создано.

Существует несколько методов, которые можно использовать при работе с Frame-окнами. Рассмотрим их.

Установка размеров окна

Чтобы установить размеры окна, используется метод setSize(). Существует две формы этого метода (с разными списками параметров):

```
void setSize(int newWidth, int newHeight)
void setSize(Dimension newSize)
```

Новый размер окна специфицируется параметрами newWidth и newHeight (первая форма), или полями width и height объекта класса Dimension, передаваемыми параметру newSize. Размеры задаются в пикселах.

Метод getSize() используется для получения текущего размера окна. Его сигнатура:

```
Dimension getSize()
```

Данный метод возвращает текущий размер окна в полях width и height объекта класса Dimension.

Скрытие и показ окна

После создания фрейм-окно остается невидимым до тех пор, пока вы не вызовете метод setVisible() . Сигнатура этого метода имеет вид:

```
void setVisible (boolean visibleFlag)
```

Компонент становится видимым, если параметр этого метода получает значение true, иначе он остается скрытым (невидимым).

Установка заголовка окна

Можно изменить заголовок фрейм-окна, если вызвать метод setTitle(). Он имеет следующий формат:

```
void setTitle(String newTitle)
```

где newTitle — новый заголовок окна.

Закрытие фрейм-окна

Когда фрейм-окно закрывается, программа должна удалить это окно с экрана, вызывая setVisible() с аргументом false:

```
setVisible(false);
```

Чтобы перехватить событие закрытия окна, нужно реализовать метод windowClosing() интерфейса WindowListener. Внутри windowClosing() необходимо удалить окно с экрана с помощью вызова setVisible(false). Эту методику иллюстрирует пример, приведенный в следующем разделе.

Создание фрейм-окна в апплете

Хотя и возможно построить фрейм-окно, просто создавая Frame-объект, но в таком окне нельзя принимать или обрабатывать события, которые происходят внутри него, или выводить в него информацию. Если создать подкласс Frame, в нем можно переопределить методы класса Frame и предусмотреть обработку событий.

Для создания фрейм-окна внутри апплета нужно создать подкласс Frame. Затем переопределить необходимые для работы с окном стандартные методы, такие как init(), start(), stop() и paint() и реализовать метод windowClosing() интерфейса WindowListener, вызывая setVisible(false), чтобы закрыть окно.

Определив подкласс Frame, нужно создать объект этого класса. При создании окна задают высоту и ширину по умолчанию. Чтобы установить необходимый размер окна, требуется вызвать метод setSize(). Однако вновь построенное фрейм-окно первоначально не будет видимым (на экране). Чтобы сделать его видимым, нужно вызвать setVisible() с аргументом true.

Следующий апплет создает подкласс Frame с именем SampleFrame. Оконный объект этого подкласса создается внутри метода init() класса AppletFrame.

Обратите внимание, что SampleFrame вызывает Frame-конструктор. Он создает стандартное фрейм-окно с заголовком, передаваемым конструктору в аргументе title. Методы start() и stop() окна апплета переопределяются так, чтобы они показывали и скрывали дочернее

окно. Это приводит к автоматическому удалению окна при завершении апплета, закрытии окна или при переходе к другой странице браузера. При возврате браузера к данному апплету показ дочернего окна возобновляется.

Программа 97. Дочернее окно в апплете

```
// Файл AppletFrame.java
// Создает дочернее фрейм-окно внутри апплета.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="AppletFrame" width=300 height=50> </applet>
*/

// Создать подкласс Frame

class SampleFrame extends Frame {
   SampleFrame(String title) {
      super(title);      // Вызов конструктора суперкласса Frame
// Создать объект для обработки window-событий
      MyWindowAdapter adapter = new MyWindowAdapter(this);
// Регистрировать его для приема этих событий
      addWindowListener(adapter);
   }
   public void paint(Graphics g) {
      g.drawString("This is in frame window", 10, 40);
   }
}

class MyWindowAdapter extends WindowAdapter {
   SampleFrame sF;    // Ссылка на Frame-окно
   public MyWindowAdapter (SampleFrame s) { // В конструктор передается
      sF = s;                               // ссылка на Frame-окно
   }
   public void windowClosing(WindowEvent we) { // Обработка соб. закр. окна
      sF.setVisible(false);       // Делаем окно невидимым
   }
}

// Создать фрейм-окно.

public class AppletFrame extends Applet {
   Frame f;                       // Ссылка на Frame-окно
   public void init() {
      f = new SampleFrame ("A Frame Window");
      f.setSize(250, 250);     // Установка размеров
      f.setVisible(true);
   }
   public void start() {       // Вызывается при запуске или возобновлении
      f.setVisible(true);
   }
   public void stop() {        // Приостанавливает выполнение апплета
      f.setVisible(false);
   }
```

```
    public void paint(Graphics g) {
       g.drawString("This is in applet window", 10, 20);
    }
}
```

Пример вывода этой программы представлен на рис.2.

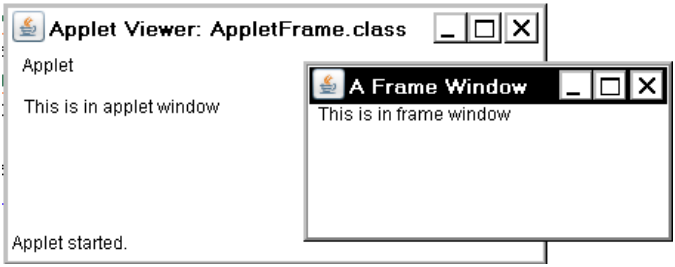


Рис. 2.Окно апплета и дочернее фрейм-окно

1.4. Обработка событий фрейм-окна

Класс Frame наследует все возможности своего суперкласса (т. е. класса Component). Это означает, что можно управлять (и пользоваться) создаваемым фрейм-окном точно так же, как главным окном апплета. Например, можно переопределить paint(), чтобы отобразить вывод, вызвать repaint(), когда нужно восстановить окно, а также переопределить все обработчики событий. Всякий раз, когда происходит событие, связанное с окном, будут вызываться обработчики событий, определенные для этого окна. Каждое окно обрабатывает свои собственные события. Например, следующая программа создает окно, которое откликается на события мыши. Главное окно апплета также реагирует на события мыши. События от мыши посылаются к тому окну, в котором они происходят.

Программа 98. Обработка событий мыши в окне апплета и дочернем окне

```
// Обработка событий мыши как в дочернем окне, так и окне апплета.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code = "WindowEvents" width = 300 height = 50>
</applet>
*/

// Создать подкласс Frame

class SampleFrame extends Frame
   implements MouseListener, MouseMotionListener {
```

```
   String msg = "";
   int mouseX = 10, mouseY = 40;
   int movX = 0, movY = 0;
   SampleFrame(String title) { // Конструктор
      super(title);
// Регистрировать этот объект, чтобы получать свои собственные
// события мыши
      addMouseListener(this);
      addMouseMotionListener(this);
// Создать объект для обработки событий мыши
      MyWindowAdapter adapter = new MyWindowAdapter(this);
// Регистрировать его для получения таких событий
      addWindowListener(adapter);
   }
// Обработать событие "Щелчок мыши".
   public void mouseClicked(MouseEvent me) { }
// Обработать событие "Мышь введена",
   public void mouseEntered(MouseEvent evtOb) {
// сохранить координаты
      mouseX = 10;
      mousey = 54;
      msg = "Mouse just entered child.";
      repaint();
   }
// Обработать событие "Мышь выведена",
   public void mouseExited(MouseEvent evtObj) {
// сохранить координаты
      mouseX = 10;
      mouseY = 54;
      msg = "Mouse just left child window.";
      repaint();
   }
// Обработать событие "Кнопка мыши нажата",
   public void mousePressed(MouseEvent me) {
// сохранить координаты
      mouseX = me.getX();
      mouseY = me.getY();
      msg = "Down";
      repaint();
   }
// Обработать событие "Кнопка мыши отпущена",
   public void mouseReleased(MouseEvent me) {
   // сохранить координаты
      mouseX = me.getX();
      mouseY = me.getY();
      msg = "Up";
      repaint ();
   }
// Обработать событие "Мышь перетащена",
   public void mouseDragged(MouseEvent me) {
// сохранить координаты
      mouseX = me.getX();
      mouseY = me.getY();
      movX = me.getX();
      movY = me.getY();
```

```
      msg = "*";
   repaint ();
   }
// Обработать событие "Мышь передвинута",
   public void mouseMoved(MouseEvent me)   {
// сохранить координаты
      movX = me.getX();
      movY = me.getY();
      repaint(0,  0,  100,  60);
   }
   public void paint(Graphics g) {
      g.drawString(msg, mouseX, mouseY);
      g.drawString("Mouse at " + movX + ", " + movY, 10, 40);
   }
}
class MyWindowAdapter extends WindowAdapter {
   SampleFrame sampleFrame;
   public MyWindowAdapter(SampleFrame sampleFrame) {
      this.sampleFrame = sampleFrame;
   }
   public void windowClosing(WindowEvent we) {
      sampleFrame.setVisible(false);
   }
}

// Окно апплета.
 public class WindowEvents extends Applet
      implements MouseListener, MouseMotionListener {
   SampleFrame f;
   String msg = "";
   int mouseX = 0, mouseY = 0;
   int movX = 0, movY = 0;
// Создать фрейм-окно
   public void init () {
      f = new SampleFrame("Handle Mouse Events");
      f.setSize (300, 200);
      f.setVisible(true);
// Регистрировать этот объект, чтобы получить его собственные
// события мыши
      addMouseListener(this);
      addMouseMotionListener(this);
   }
// Удалить фрейм-окно при останове апплета.
   public void stop() {
      f.setVisible(false);
   }
// Показать фрейм-окно при старте апплета.
   public void start() {
      f.setVisible(true);
   }
// Обработать событие "Кнопка мыши нажата",
   public void mouseClicked(MouseEvent me) {
   }
// Обработать событие "Мышь введена",
   public void mouseEntered(MouseEvent me) {
```

```
// сохранить координаты
      mouseX = 0;
      mouseY = 24;
      msg = "Mouse just entered applet window.";
      repaint();
   }
// Обработать событие "Мышь выведена",
   public void mouseExited(MouseEvent me) {
// сохранить координаты
      mouseX = 0;
      mouseY = 24;
      msg = "Mouse just left applet window."; repaint();
   }
// Обработать событие "Кнопка мыши нажата",
   public void mousePressed(MouseEvent me) {
// сохранить координаты-
      mouseX = me.getX();
      mouseY = me.getY();
      msg = "Down";
      repaint();
   }
// Обработать событие "Кнопка мыши отпущена",
   public void mouseReleased(MouseEvent me) {
// сохранить координаты
      mouseX = me.getX();
      mouseY = me.getY();
      msg = "Up";
      repaint();
   }
// Обработать событие "Мышь перетащена",
   public void mouseDragged(MouseEvent me) {
// сохранить, координаты
      mouseX = me.getX();
      mouseY = me.getY();
      movX = me.getX();
      movY = me.getY();
      msg = "*";
      repaint();
   }
// Обработать событие "Мышь передвинута",
   public void mouseMoved(MouseEvent me)   {
// сохранить координаты
      movX = me.getX();
      movY = me.getY();
      repaint(0, 0, 100, 20);
   }
// Вывести msg в окне апплета.
   public void paint(Graphics g) {
      g.drawString (msg, mouseX, mouseY);
      g.drawString("Mouse at " + movX + ",  " + movY, 0, 10);
   }
}
```

Пример вывода этой программы представлен на рис. 3.

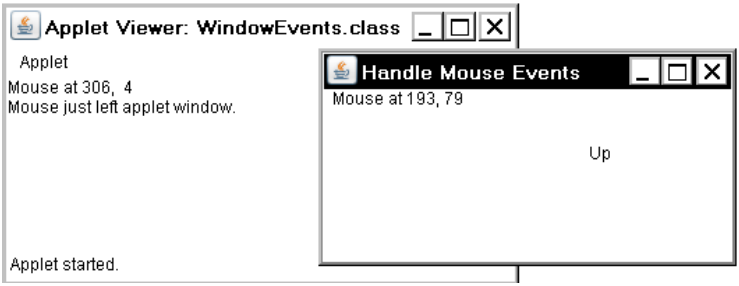


Рис. 3. Окно апплета и дочернее фрейм-окно

Для запуска из браузера создадим в папке bin проекта файл RunWindowEvents.html:

```
<html>
<body>
<applet code = WindowEvents.class width="200" height="200" >
</applet>
</body>
</html>
```

Результат открытия этого файла в браузере Mozilla Firefox показан на рис.4.

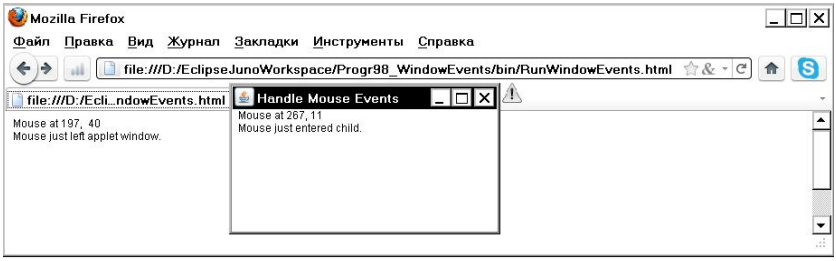


Рис. 4. Запуск апплета и дочернего окна браузером

1.5. Создание оконной программы

Хотя AWT чаще всего используется для создания апплетов, с его помощью можно также проектировать и автономные приложения. Для этого нужно просто организовать один или несколько оконных объектов внутри метода main(). Следующая автономная программа создает фрейм-окно, которое отвечает на щелчки мыши и нажатия клавиш.

Программа 99. Автономное оконное приложение

```
// Файл AppWindow.java
// Создать AWT-приложение.
```

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
// Определить класс фрейм-окна.
public class AppWindow extends Frame {      // Оконный класс
   String keymsg = "";
   String mousemsg = "";
   int mouseX = 30, mouseY = 30;
   public AppWindow() {
      addKeyListener(new MyKeyAdapter(this));
      addMouseListener(new MyMouseAdapter(this));
      addWindowListener(new MyWindowAdapter());
   }
   public void paint(Graphics g) {
      g.drawString(keymsg, 10, 40);
      g.drawString(mousemsg, mouseX, mouseY);
   }
// Создать фрейм-окно для приложения.
   public static void main(String args[]) {
      AppWindow appwin = new AppWindow();
      appwin.setSize(new Dimension(300, 200));
      appwin.setTitle("An AWT-Based Application");
      appwin.setVisible(true);
   }
}
class MyKeyAdapter extends KeyAdapter {  // Класс для обработки клавиш
   AppWindow appW;                       // Ссылка на окно
   public MyKeyAdapter(AppWindow appWnd) {
      appW = appWnd;
   }
   public void keyTyped(KeyEvent ke) {   // Обработчик нажатия клавиши
      appW.keymsg += ke.getKeyChar();
      appW.repaint();
   };
}
class MyMouseAdapter extends MouseAdapter {
   AppWindow appW;
   public MyMouseAdapter(AppWindow appWnd) {
      appW = appWnd;
   }
   public void mousePressed(MouseEvent me) {
      appW.mouseX = me.getX();
      appW.mouseY = me.getY();
      appW.mousemsg = "Mouse Down at " + appW.mouseX +
         ",   " + appW.mouseY;
      appW.repaint();
   }
}
class MyWindowAdapter extends WindowAdapter {
   public void windowClosing(WindowEvent we) {
      System.exit(0);
   }
}
```

Пример вывода этой программы представлен на рис. 5.

Рис. 5. Работа оконного приложения

После создания фрейм-окно живет своей собственной жизнью. Оно реагирует на нажатие клавиш и нажатие мыши. Обратите внимание, что main() завершается обращением к appwin.setVisible (true).

Программа поддерживается в рабочем состоянии, пока не будет закрыто ее окно. При создании оконного приложения main() используется, чтобы запустить для него окно верхнего уровня. После этого программа будет функционировать как GUI-приложение, а не как консольные программы, использовавшиеся ранее.

На рис.6 показано окно командной строки и окно работающей программы.

Для перехода в каталог с откомпилированным class-файлом выполнена команда:

```
>cd /d D:\EclipseJunoWorkspace\Progr99_AppWindow\bin
```

Следующая команда включает в переменную окружения classpath текущий каталог, который обозначается точкой (.):

```
>set classpath=%classpath%;.;
```

Запуск программы осуществляется командой:

```
>java AppWindow
```

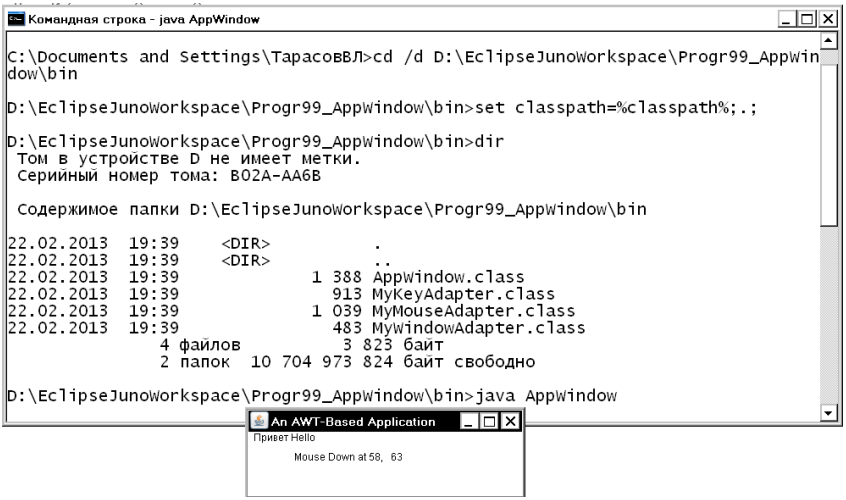


Рис. 6. Запуск оконного приложения из командной строки

1.6. Отображение информации в окне

В самом общем смысле окно является контейнером для разнообразной информации. Хотя в предшествующих примерах мы уже выводили небольшие порции текста в окно, мы еще не начали пользоваться способностью окна представлять высококачественный текст и графику. В действительности, большая часть средств AWT ориентирована на поддержку именно этих возможностей. Ниже обсуждаются возможности обработки текста, графики и шрифтов на языке Java. Эти возможности являются достаточно мощными и гибкими.

Работа с графикой

AWT поддерживает богатый набор графических методов. Вся графика рисуется относительно окна. Это может быть главное или дочернее окно апплета, а также окно автономного приложения. Начало координат каждого окна — в его верхнем левом углу и обозначается как (0, 0). Координаты определяются в пикселах. Весь вывод в окно выполняется через графический контекст. *Графический контекст* инкапсулирован в классе и получается двумя способами;

□ передается апплету, когда вызывается один из его многочисленных методов, таких как paint() или update();

□ возвращается методом getGraphics() класса Component.

В приводимых далее примерах графика демонстрируется в главном окне апплета. Однако та же техника применима к любому другому типу окна.

Класс Graphics определяет ряд функций рисования. Каждая форма может быть рисованной или заполненной. Объекты рисуются и заполняются выбранным в текущий момент графическим цветом, который по умолчанию является черным. Когда графический объект превышает размеры окна, вывод автоматически усекается. Рассмотрим несколько методов рисования.

Рисование линий

Линии рисуются методом drawLine() формата:

```
void drawLine(int startX, int startY, int endX, int endY)
```

DrawLine() отображает линию (в текущем цвете рисования), которая начинается в координатах startX, startY и заканчивается в endX, endY.

Программа 100. Рисование линий в окне

```
// Файл Lines.java
// Рисует линии.
import java.awt.*;
import java.applet.*;
/*
<applet code="Lines" width-300 height=200>
</applet>
*/
public class Lines extends Applet {
   public void paint(Graphics g) {
      g.drawLine(0, 0, 100, 100);
      g.drawLine(0, 100, 100, 0);
      g.drawLine(40, 25, 250, 180);
      g.drawLine(75, 90, 400, 400);
      g.drawLine(20, 150, 400, 40);
      g.drawLine(5, 290, 80, 19);
   }
}
```

Пример вывода этой программы представлен на рис. 721.5.

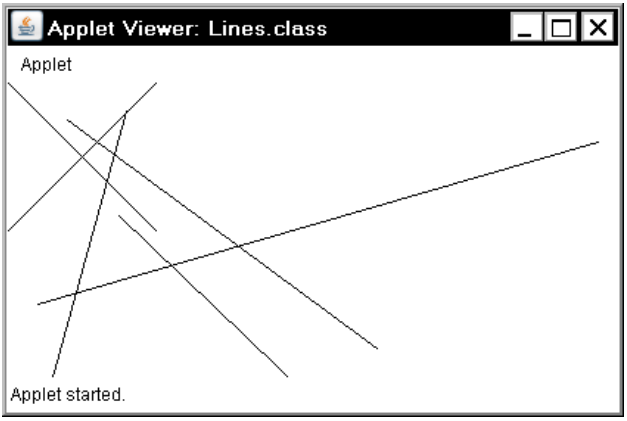


Рис. 7. Рисование линий в окне

Рисование прямоугольников

Методы drawRect() и fillRect() отображают соответственно рисованный и заполненный прямоугольник. Их формат:

```
void drawRect (int top, int left, int width, int height)
void fillRect(int top, int left, int width, int height)
```

Координаты левого верхнего угла прямоугольника — в параметрах top и left, width и height — указывают размеры прямоугольника (в пикселах).

Чтобы рисовать округленный прямоугольник, используйте drawRoundRect() или fillRoundRect() с форматами:

```
void drawRoundRect (int top, int left, int width, int height,
                    int xDiam, int yDiam)
void fillRoundRect (int top,  int left, int width, int height,
                    int xDiam, int yDiam)
```

Округленный прямоугольник имеет закругленные углы. Левый верхний угол прямоугольника задается параметрами top, left. Размеры прямоугольника определяются в width и height. Диаметр округляющейся дуги по оси X определяется в xDiam. Диаметр округляющейся дуги по оси Y определяется в yDiam.

Программа 101. Рисование прямоугольников

```
// Файл Rectangles.java
// Рисует прямоугольники
import java.awt.*;
import java.applet.*;
/*
<applet code="Rectangles" width = 300 height = 200>
```

```
</applet>
*/
public class Rectangles extends Applet {
   public void paint(Graphics g) {
      g.drawRect(10, 10, 60, 50);
      g.fillRect(100, 10, 60, 50);
      g.drawRoundRect(190, 10, 60, 50, 15, 15);
      g.fillRoundRect(70, 90, 140, 100, 30, 40);
   }
}
```

Пример вывода этой программы представлен на рис.8.

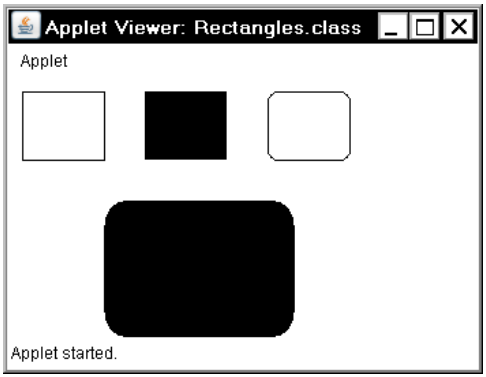


Рис. 8. Рисование прямоугольников

Рисование эллипсов и кругов

Для рисования эллипса используйте drawOval(), а для его заполнения — fillOval(). Эти методы имеют формат:

```
void drawOval (int top, int left, int width, int height)
void fillOval (int top, int left, int width, int height)
```

Эллипс рисуется в пределах ограничительного прямоугольника, чей левый верхний угол определяется параметрами top и left, а ширина и высота указываются в width и height. Чтобы нарисовать круг, в качестве ограничительного прямоугольника указывайте квадрат.

Программа 102. Рисование эллипсов

```
// Файл Ellipses.java
// Рисует эллипсы.
import java.awt.*;
import java.applet.*;
/*
<applet cod e= "Ellipses" width = 300 height = 200>
</applet>
```

```
*/
public class Ellipses extends Applet {
   public void paint(Graphics g) {
      g.drawOval(10, 10, 50, 50);
      g.fillOval(100, 10, 75, 50);
      g.drawOval(190, 10, 90, 30);
      g.fillOval(70, 90, 140, 100);
   }
}
```

Пример вывода этой программы представлен на рис9.

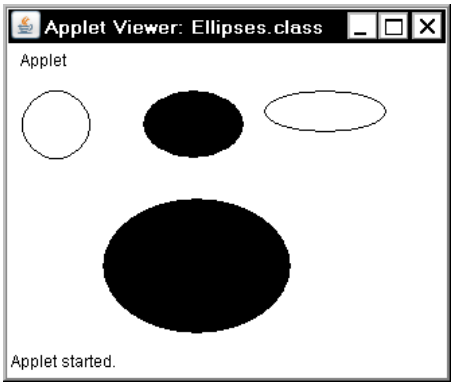


Рис. 9. Рисование эллипсов и окружностей

Рисование дуг

Дуги можно рисовать методами drawArc() и fillArc(), используя форматы:

```
void drawArc(int top, int left, int width, int height,
             int начало, int конец)
void fillArc(int top, int left, int width, int height,
             int начало, int конец)
```

Дуга ограничена прямоугольником, чей левый верхний угол определяется параметрами top, left, а ширина и высота — параметрами width и height.

Дуга рисуется от угла начало до углового расстояния, указанного в конец. Углы указываются в градусах и отсчитываются от горизонтальной оси против часовой стрелки. Дуга рисуется против часовой стрелки, если конец положителен, и по часовой стрелке, если конец отрицателен. Поэтому, чтобы нарисовать дугу от двенадцатичасового до шестичасового положений, начальный угол должен быть 90° и угол развертки 180°.

Программа 103. Рисование дуг

```
// Файл Arcs.java
// Рисует дуги.
import java.awt.*;
import java.applet.*;
/*
<applet code="Arcs" width = 300 height = 200>
</applet>
*/
public class Arcs extends Applet {
   public void paint(Graphics g) {
      g.drawArc(10, 40, 70, 70, 0, 75);
      g.fillArc(100, 40, 70, 70, 0, 75);
      g.drawArc(10, 100, 70, 80, 0, 120);
      g.fillArc(100, 100, 70, 90, 0, 270);
      g.drawArc(200, 80, 80, 80, 0, 180);
   }
}
```

Пример вывода этой программы представлен на рис10.

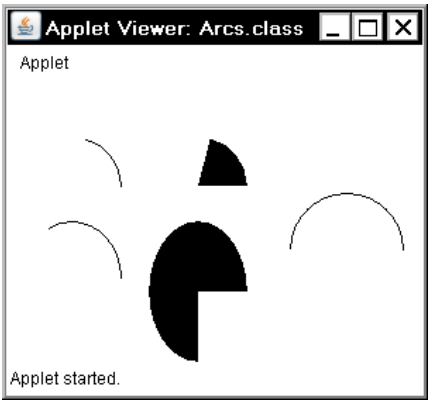


Рис. 10. Рисование дуг

Рисование многоугольников

Фигуры произвольной формы можно рисовать, используя методы drawPolygon() и fillPolygon() со следующими форматами:

```
void drawPolygon (int x[], int y[], int numPoints)
void fillPolygon (int x[], int y[], int numPoints)
```

Оконечные точки многоугольника определяются координатными парами, содержащимися в массивах x[ ] и y[ ]. Число точек, определенных в этих массивах, указывается параметром numPoints. Имеются альтернативные формы этих методов, в которых

многоугольник определяется объектом класса Polygon. Следующий апплет рисует форму песочных часов.

Программа 104. Рисование многоугольников

```
// Файл HourGlass.java
// Рисует многоугольник.
import java.awt.*;
import java.applet.*;
/*
<appiet code = "HourGlass" width = 230 height = 210>
</applet>
*/
public class HourGlass extends Applet {
   public void paint(Graphics g) {
      int x[] = {30, 200, 30, 200, 30};
      int y[] = {30, 30, 200, 200, 30};
      int num = 5;
      g.drawPolygon(x, y, num);
   }
}
```

Пример вывода этой программы представлен на рис.11.

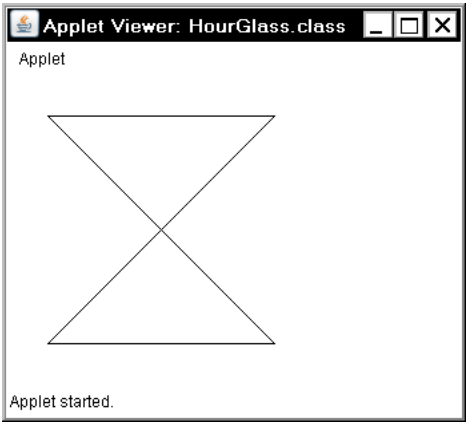


Рис. 11. Рисование многоугольника

1.7. Установка размеров графики

Часто нужно установить размеры графического объекта в определенной пропорции с текущими размерами окна, в котором он рисуется. Для этого сначала получают текущие размеры окна, вызывая метод getSize() для оконного объекта. Он возвращает размеры окна, инкапсулированные в объект класса Dimension. Используя текущие

размеры окна можно нужным образом отмасштабировать графический вывод.

Для демонстрации этой методики ниже представлен апплет, который сначала рисует квадрат размером 200×200 пикселов и затем увеличивает его по 25 пикселов в ширину и.высоту с каждым щелчком мыши, пока он не станет больше чем 500×500. В этой точке следующий щелчок мыши возвратит его к размеру 200×200, и процесс начнется с начала. В окне рисуется прямоугольник вокруг внутренней границы окна; а внутри этого прямоугольника — две перекрещивающихся линии диагоналей. Апплет работает с программой appletviewer, но может не работать в окне браузера.

Программа 105. Изменение размеров рисунка

```
// Файл ResizeMe.java
// Изменяет размеры вывода для установки текущих размеров окна.
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*
<applet code = "ResizeMe" width = 200 height = 200>
</applet>
*/
public class ResizeMe extends Applet {
   final int inc = 25;
   int max = 500;
   int min = 200;
   Dimension d;
   public
   ResizeMe() {
      addMouseListener(new MouseAdapter() {
         public void mouseReleased(MouseEvent me) {
            int w = (d.width  + inc) > max ? min : (d.width  + inc);
            int h = (d.height + inc) > max ? min : (d.height + inc);
            setSize(new Dimension(w, h));
         }
      });
   }
   public void paint(Graphics g) {
      d = getSize();                    // Текущие размеры окна
      g.drawLine(0, 0, d.width - 1, d.height - 1);
      g.drawLine(0, d.height - 1, d.width - 1, 0);
      g.drawRect(0, 0, d.width - 1, d.height - 1);
   }
}
```

1.8. Работа с цветом

Java обеспечивает переносимость цвета, вне зависимости от устройства вывода объекта. Цветовая система AWT позволяет

указывать в программе любой желаемый цвет. Более того, AWT находит наилучшее согласование этого цвета с заданными аппаратными ограничениями дисплея, выполняющего программу или апплет. Поэтому код не имеет никакого отношения к различиям в способах поддержки цветов разными аппаратными устройствами. Работа с цветом поддерживается классом Color.

В классе Color определено несколько цветовых констант (например, Color.black), специфицирующих ряд обычных цветов. Можно также создавать собственные цвета, применяя один из цветовых конструкторов. Обычно используются следующие его формы:

```
Color (int red, int green, int blue)
Color(int rgbValue)
Color(float red, float green, float blue)
```

Первый конструктор получает (через указанные параметры) три целых числа, которые определяют цвет как смесь красного, зеленого и синего цвета. Эти значения должны быть между 0 и 255, как в следующем примере:

```
new Color(255, 100, 100); // светло-красный
```

Второй цветовой конструктор получает одиночное целое число, которое содержит смесь красного, зеленого и синего цветов, упакованную в целое число. Целое число организовано с красным цветом — в битах от 16 до 23, зеленым цветом — в битах от 8 до 15 и синим цветом — в битах от 0 до 7. Пример этого конструктора:

```
int newRed = (0xff000000 | (0xc0 << 16) | (0x00 << 8) | 0x00);
Color darkRed = new Color(newRed);
```

Третий конструктор получает три float-значения (между 0.0 и 1.0), в которых определяется относительная смесь красного, зеленого и синего цвета.

После создания цветового объекта его можно использовать для установки цвета переднего плана и/или фона с помощью методов setForeground() и setBackground(). Его можно также использовать для установки текущего цвета рисования.

Цветовые методы

Класс Color определяет несколько методов, которые помогают манипулировать цветом. Они рассматриваются ниже.

Использование тона, насыщенности и яркости

Для определения специфических цветов используется две альтернативные цветовые модели: HSB (Hue-Saturation-Brightness,

цветовой тон-насыщенность-яркость) и RGB (Red-Green-Blue, красный-зеленый-синий). *Тон* (оттенок) определяется числом между 0.0 и 1.0 (для цветов радуги, расположенных в порядке возрастания: красный, оранжевый, желтый, зеленый, голубой, синий (индиго) и фиолетовый). *Насыщенность* — другая шкала, ранжированная от 0.0 до 1.0, представляющая изменения тона от светлого (пастельного) к интенсивному. Значения *яркости* также ранжированы от 0.0 до 1.0, где 1 — ярко-белый, а 0 — черный. Класс color поставляет два метода, которые выполняют взаимные преобразования RGB- и HSB-моделей:

```
static int HSBtoRGB (float hue, float saturation, float brightness)
static float[ ] RGBtoHSB (int red, int green, int blue, float values[ ])
```

Метод HSBtoRGB() возвращает упакованное RGB-значение, совместимое с конструктором Color (int). Метод RGBtoHSB() возвращает массив HSB-значений с плавающей точкой, соответствующих целым числам RGB. Если параметр values — не null, то этот массив содержит HSB-значения, которые возвращаются в вызывающую программу. В противном случае создается новый массив, и в нем возвращаются HSB-значения. В любом случае массив содержит тон в элементе с индексом 0, насыщенность —- в элементе с индексом 1 и яркость — в элементе с индексом 2.

Методы getRed(), getGreen(), getBlue()

Вы можете получить красные, зеленые и синие компоненты цвета, независимо используя методы getRed(), getGreen() и getBlue(), показанные ниже:

```
int getRed()
int getGreen()
int getBlue()
```

Каждый из этих методов возвращает цветовой компонент RGB, найденный в вызывающем color-объекте в нижних восьми битах целого числа.

Метод getRGB()

Метод getRGB() используется для получения упакованного RGB-представления цвета. Формат метода:

```
int getRGB()
```

Возвращаемое значение организовано так же, как описано ранее.

Установка текущего цвета графики

По умолчанию, графические объекты рисуются в текущем цвете переднего плана. Можно изменить этот цвет, вызывая метод setColor() класса Graphics:

```
void setColor (Color newColor)
```

где параметр newColor определяет новый цвет рисунка.

Текущий цвет можно получить методом getColor():

```
Color getColor()
```

Программа 106. Демонстрация цветов

Следующий апплет создает несколько цветов и рисует различные объекты, используя эти цвета.

```
// Файл ColorDemo.java
// Демонстрирует цвета.
import java.awt.*;
import java.applet.*;
/*
<applet code = "ColorDemo" width = 300 height = 200>
</applet>
*/
public class ColorDemo extends Applet { // Рисовать линии
   public void paint(Graphics g) {
      Color c1 = new Color(255, 100, 100);  // Красный
      Color c2 = new Color(100, 255, 100);  // Зеленый
      Color c3 = new Color(100, 100, 255);  // Синий
      g.setColor(c1);
      g.drawLine(0, 0, 100, 100);
      g.drawLine(0, 100, 100, 0);
      g.setColor(c2);
      g.drawLine(40, 25, 250, 180);
      g.drawLine(75, 90, 400, 400);
      g.setColor(c3);
      g.drawLine(20, 150, 400, 40);
      g.drawLine(5, 290, 80, 19);
      g.setColor(Color.red);              // Чисто красный цвет
      g.drawOval(10, 10, 50, 50);
      g.fillOval(70, 90, 140, 100);
      g.setColor(Color.blue);
      g.drawOval(190, 10, 90, 30);
      g.drawRect(10, 10, 60, 50);
      g.setColor(Color.cyan);
      g.fillRect(100, 10, 60, 50);
      g.drawRoundRect(190, 10, 60, 50, 15, 15);
   }
}
```

Окно апплета показано на рис.12.

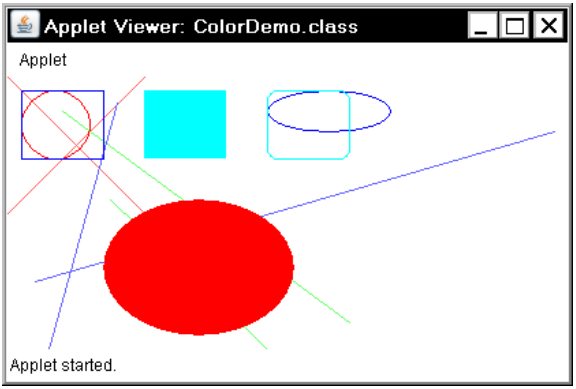


Рис. 12. Цветные линии и фигуры

1.9. Установка режима рисования

Режим рисования (paint mode) определяет, как объекты рисуются в окне. По умолчанию, новый вывод в окно записывается поверх любого предварительно существующего содержания, то есть исходный цвет точки замещается цветом накладываемого на него рисунка. Однако можно выводить новые объекты с помощью метода setXORMode() со следующей сигнатурой:

```
void setXORMode (Color xorColor)
```

Здесь параметр xorColor определяет цвет, который будет использован в окне в режиме XOR, когда объект выводится. В XOR-режиме цвет результирующей точки определяется как результат операции ИСКЛЮЧАЮЩЕГО ИЛИ (XOR) над цветом рисования и исходным цветом той же точки. Преимущество XOR-режима состоит в том, что новый объект всегда будет видимым, независимо от того, какого цвета объект на нем нарисован. Например, белая линия на черном фоне в итоге дает белый цвет, а белая линия на белом фоне — черный.

Чтобы вернуться в режим перезаписи, вызовите метод setPaintMode() с форматом:

```
void setPaintMode ()
```

Вообще, следует использовать режим перезаписи для нормального вывода, и режим XOR — для специальных целей. Например, следующая программа отображает тонкий крестик, который прослеживает указатель мыши. Такой крестик выводится в окно в режиме XOR и всегда видим, независимо от того, каков лежащий под ним цвет.

Программа 107. Режим рисования XOR

```
// Файл XOR.java
// Демонстрирует режим XOR.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code = "XOR" width = 400 height = 200>
</applet>
*/
public class XOR extends Applet {
   int chsX = 100, chsY = 100;
   public XOR() {
      addMouseMotionListener(new MouseMotionAdapter() {
         public void mouseMoved(MouseEvent me) {// Переопределение
            int x = me.getX();
            int y = me.getY();
            chsX = x - 10;
            chsY = y - 10;
            repaint ();
         }
      });
   }
   public void paint(Graphics g) {
      g.drawLine(0, 0, 100, 100);
      g.drawLine(0, 100, 100, 0);
      g.setColor(Color.blue);
      g.drawLine(40, 25, 230, 180);
      g.drawLine(75, 90, 400, 400);
      g.setColor(Color.green);
      g.drawRect(10, 10, 60, 50);
      g.fillRect(100, 10, 60, 50);
      g.setColor(Color.red);
      g.drawRoundRect(190, 10, 60, 50, 15, 15);
      g.fillRoundRect(70, 90, 140, 100, 30, 40);
      g.setColor(Color.cyan);
      g.drawLine(20, 150, 400, 40);
      g.drawLine(5, 290, 90, 19);
// Режим XOR для вывода перекрестья
      g.setXORMode(Color.black);
      g.drawLine(chsX - 10, chsY, chsX + 10, chsY);
      g.drawLine(chsX, chsY - 10, chsX, chsY + 10);
      g.setPaintMode();     // Восстановление режима рисования
   }
}
```

Пример вывода програмы приведен на рис.13.

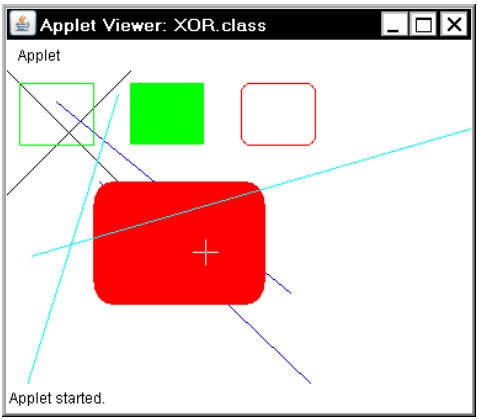


Рис. 13. Отслеживание движения мыши в режиме XOR

1.10. Работа со шрифтами

Пакет AWT поддерживает множество типов шрифтов. Шрифты появились из области традиционного набора текстов и стали важной частью компьютерных документов и дисплеев. AWT обеспечивает гибкость программирования за счет того, что берет на себя операции манипулирования шрифтами и допускает их динамический выбор.

Начиная с версии Java 2, для шрифтов различают три имени: имя семейства, логическое имя и имя гарнитуры (face name). Гарнитура определяет характер рисунка литер шрифта.

Имя семейства — общее название шрифта, например, Courier (Курьер). *Логическое имя* определяет категорию шрифта, например Monospaced (Фиксированной ширины). *Имя гарнитуры* специфицирует определенный шрифт, например, Courier Italic (Курьер курсивный).

Шрифты инкапсулированы в классе Font. Некоторые методы, определенные в Font, перечислены в табл. 14.2.

Таблица 14.2. Некоторые методы, определенные в Font

Метод	Описание
`static Font decode (String str)`	Возвращает шрифт по заданному (в параметре) имени
`boolean equals (Object FontObj)`	Возвращает true, если вызывающий объект содержит тот же самый шрифт, что указан в FontObj, Иначе возвращает false
`String getFamily()`	Возвращает имя семейства шрифта, которому вызывающий шрифт принадлежит
`static Font getFont(String property)`	Возвращает шрифт, связанный с системным свойством, указанным в параметре property. Возвращает указатель null, если свойство не существует
`static Font getFont (String property, Font defaultFont)`	Возвращает шрифт, связанный с системным свойством, указанным в параметре property. Возвращает шрифт, указанный в defaultFont, если свойство не существует
`string getFontName()`	Возвращает имя гарнитуры вызывающего шрифта. (Добавлен в Java 2)
`String getName()`	Возвращает логическое имя вызывающего шрифта
`int getSize()`	Возвращает размер, в пунктах, вызывающего шрифта
`int getStyle()`	Возвращает значения стиля (начертания) вызывающего шрифта
`int hashCode()`	Возвращает код мусора, связанный с вызывающим объектом
`boolean isBold()`	Возвращает true, если шрифт имеет Bold-начертание, иначе — false
`boolean isItalic()`	Возвращает true, если шрифт имеет Italic-начертание, иначе — false
`boolean isPlain()`	Возвращает true, если шрифт имеет Plain-начертание, иначе — false
`String toString()`	Возвращает строчный эквивалент вызывающего шрифта

В классе Font определены переменные, представленные в табл. 21.3.

Таблица 14.3. Переменные класса Font

Переменная	Значение
`String name`	Имя шрифта
`float pointSize`	Размер шрифта в пунктах (дробный)
`int size`	Размер шрифта в пунктах (целый)
`int style`	Стиль (начертание) шрифта

Определение доступных шрифтов

При работе со шрифтами часто необходимо знать, какой шрифт доступен на компьютере. Чтобы получить эту информацию, можно использовать метод getAvailableFontFamilyNames(), определенный классом GraphicsEnvironment. Формат заголовка этого метода:

```
String [ ] getAvailableFontFamilyNames ()
```

Метод возвращает массив строк, который содержит имена доступных семейств шрифтов.

Кроме того, в классе GraphicsEnvironment определен метод getAllFonts (). Его формат:

```
Font [ ] getAllFonts()
```

Этот метод возвращает массив Font-объектов для всех доступных шрифтов.

Так как перечисленные методы — члены класса GraphicsEnvironment, то для их вызова нужна ссылка на этот класс. Можно получить ссылку, используя статический метод getLocalGraphicsEnvironment(), который определен в GraphicsEnvironment. Его формат:

```
static GraphicsEnvironment getLocalGraphicsEnvironment()
```

В следующей программе апплет показывает имена доступных семейств шрифтов.

Программа 108. Список шрифтов

```
// Файл ShowFonts.java
// Показ шрифтов.
/*
<applet code="ShowFonts" width=550 height=60>
</applet>
*/
import java.applet.*;
import java.awt.*;
```

```
public class ShowFonts extends Applet {
   public void paint(Graphics g) {
      String msg = "";
      String FontList[];
      GraphicsEnvironment ge =
         GraphicsEnvironment.getLocalGraphicsEnvironment();
      FontList = ge.getAvailableFontFamilyNames();
      for(int i = 0; i < FontList.length; i++)
         msg += FontList[i] + " ";
      g.drawString(msg, 4, 16);
      msg = "Число шрифтов = " + FontList.length;
      g.drawString(msg, 4, 36);
   }
}
```

На рис. 14 показан пример вывода. При выполнении программы на различных компьютерах список шрифтов может отличаться.

Рис. 14. Перечень доступных шрифтов

Создание и выбор шрифта

Перед выбором нового шрифта нужно сначала создать объект класса Font, который описывает этот шрифт. Одна из форм конструктора класса Font имеет формат:

```
Font (String fontName, int fontStyle, int pointSize)
```

Здесь fontName определяет имя желательного шрифта. Имя можно указывать, используя либо логическое имя, либо имя гарнитуры. Все среды Java поддерживают следующие шрифты: Dialog, DialogInput, Sans Serif, Serif, Monospaced и Symbol. Шрифт Dialog используется диалоговыми окнами системы и применяется по умолчанию. Можно также использовать любые другие шрифты, поддерживающиеся средой, но они могут быть не всегда доступными.

Стиль шрифта указывается параметром fontStyle. Он может состоять из одной или нескольких констант: Font.PLAIN, Font.BOLD и Font.ITALIC. Стили можно комбинировать, объединяя эти константы операцией OR. Например, выражение Font.BOLD | Font.ITALIC определяет стиль *полужирный курсив*.

Размер шрифта указывается параметром pointSize в пунктах. Напомним, что типографский *пункт* (point) равен 1/72 дюйма (дюйм - 2.54 см).

Чтобы использовать созданный шрифт, следует *выбрать* его с помощью метода setFont(). Он определен в классе Component и имеет общую форму:

```
void setFont(Font fontObj)
```

Здесь fontObj — объект, который содержит желательный шрифт.

Следующая программа выводит пример любого стандартного шрифта. Каждый раз, когда вы щелкаете кнопку мыши внутри ее окна, выбирается новый шрифт, и его имя отображается на экране.

Программа 109. Выбор шрифтов

```
// Файл SampleFont.java
// Показывает шрифты.
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*
<applet code = "SampleFonts" width = 200 height = 100>
</applet>
*/
public class SampleFonts extends Applet {
   int next = 0;
   Font f;
   String msg;
   public void init() {
      f = new Font("Dialog", Font.PLAIN, 12);
      msg = "Dialog";
      setFont(f);
      addMouseListener(new MyMouseAdapter(this));
   }
   public void paint(Graphics g) {
      g.drawString(msg, 4, 20);
   }
}
class MyMouseAdapter extends MouseAdapter {
   SampleFonts sF;
   public MyMouseAdapter(SampleFonts smplFnts) {
      sF = smplFnts;
   }
   public void mousePressed(MouseEvent me) {
 // Переключает шрифт каждым щелчком мыши.
      sF.next++;
      switch(sF.next) {
         case 0:
            sF.f = new Font("Dialog", Font.PLAIN, 12);
            sF.msg = "Dialog";
         break;
```

```
            case 1:
               sF.f = new Font("DialogInput", Font.PLAIN, 12);
               sF.msg = "Dialoglnput";
            break;
            case 2:
               sF.f = new Font("SansSerif", Font.PLAIN, 12);
               sF.msg = "SansSerif";
            break;
            case 3:
               sF.f = new Font("Serif", Font.PLAIN, 12);
               sF.msg = "Serif";
            break;
            case 4:
               sF.f = new Font("Monospaced", Font.PLAIN, 12);
               sF.msg = "Monospaced";
            break;
         }
         if (sF.next == 4)
            sF.next = -1;
         sF.setFont(sF.f);
         sF.repaint();
      }
}
```

Пример вывода этой программы представлен на рис. 15,

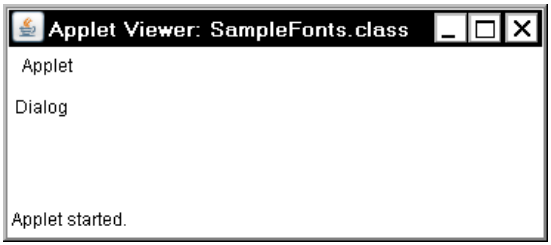


Рис. 15. Выбор шрифта щелчком мыши

Получение информации о шрифте

Текущий шрифт можно получить методом getFont(). Он определен в классе Graphics как:

```
Font getFont()
```

Из полученного текущего шрифта, можно извлекать информацию, используя различные методы, определенные в классе Font. Например, следующий апплет отображает имя, семейство, размер и стиль текущего шрифта.

Программа 110. Свойства шрифта

```
// Файл FontInfo.java
// Показывает информацию о шрифте,
```

```
import java.applet.*;
import java.awt.*;
/*
<applet code = "FontInfo" width = 350 height = 60>
</applet>
*/
public class FontInfo extends Applet {
   public void paint(Graphics g) {
      Font f = g.getFont();
      String fontName = f.getName();
      String fontFamily = f.getFamily();
      int fontSize = f.getSize();
      int fontStyle = f.getStyle();
      String msg = "Family:  " + fontName;
      msg += ", Font: " + fontFamily;
      msg += ", Size: " + fontSize + ", Style: ";
      if((fontStyle & Font.BOLD) == Font.BOLD)
         msg += "Bold ";
      if((fontStyle & Font.ITALIC) == Font.ITALIC)
         msg += "Italic ";
      if((fontStyle & Font.PLAIN) == Font.PLAIN)
         msg  += "Plain ";
      g.drawString(msg, 4, 16);
   }
}
```

Вывод программы показан на рис.16.

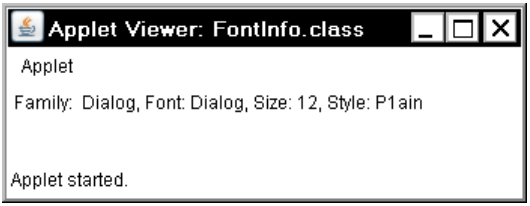


Рис. 16. Свойства текущего шрифта

Управление текстовым выводом с помощью класса FontMetrics

Для большинства шрифтов не все символы имеют одинаковую ширину (такие шрифты называют *пропорциональными).* Кроме того, высота каждого символа, длина *выносных элементов* (свисающих частей, как у символов g или p, например) и величина пробела между горизонтальными строками изменяется от шрифта к шрифту. Далее, может быть изменен размер шрифта (в пунктах). Переменный характер этих (и других) атрибутов не имел бы слишком больших последствий, если бы Java не требовал от программиста ручного управления фактически всем текстовым выводом.

Учитывая, что размеры каждого шрифта могут отличаться и что шрифты могут быть изменены во время выполнения программы, должен существовать некоторый способ для определения размеров и различных других атрибутов текущего шрифта. Например, для записи одной строки текста после другой необходимо как-то узнать, какова высота шрифта и сколько пикселов необходимо иметь между строками. Чтобы заполнить эту потребность, AWT включает класс FontMetrics, который инкапсулирует различную информацию о шрифте. Начнем с определения общей терминологии, используемой при описании шрифтов:

□ Высота (Height) — размер (от верха до низа) самого высокого символа в шрифте.

□ Базовая линия (Baseline) — линия, по которой выровнен низ всех символов (не считая десцендера).

□ Высота надстрочного элемента, асцендер (Ascent) — расстояние от базовой линии до верха символа.

□ Высота подстрочного элемента, десцендер (Descent) — расстояние от базовой линии до низа символа.

□ Интерлиньяж (Leading) — расстояния между самым низом одной строки текста и самым верхом следующей строки.

Во многих из предыдущих примеров использовался метод drawString(). Он выводит строку в текущем шрифте и цвете, начиная с определенного положения в окне. Однако точка вывода находится слева на базовой линии, а не в левом верхнем углу окна, как обычно в других рисующих методах. Общая ошибка — рисовать строку в той же самой координате, в которой вы рисовали бы прямоугольник. Например, при рисовании прямоугольника начиная с точки 0, 0 в окне апплета будет изображен полный прямоугольник. Но при рисовании строки "Typesetting" с координатами 0, 0 будут изображены только нижние выносные элементы (или десцендеры) символов y, p, и g. Используя метрику шрифта, можно определять надлежащее размещение каждой отображаемой строки.

Класс FontMetrics определяет несколько методов, которые помогают управлять текстовым выводом. Наиболее используемые методы перечислены в табл. 14.4. Они помогают должным образом отобразить текст в окне.

Таблица 14.4. Некоторые методы класса FontMetrics

Метод	Описание
`int bytesWidth (byte b[ ], int start, int numBytes)`	Возвращает ширину строки, состоящей из numBytes символов, содержащихся в массиве b. Параметр start указывает номер начального символа этой строки в массиве b

`int charWidth(char c[], int start, int numChars)`	Возвращает ширину строки, состоящей из numChars символов, содержащихся в массиве c. Параметр start указывает номер начального символа этой строки в массиве c
`int charWidth (char c)`	Возвращает ширину c
`int charWidth (int c)`	Возвращает ширину c
`int getAscent()`	Возвращает асцендер шрифта
`int getDescent ()`	Возвращает десцендер шрифта
`Font getFont()`	Возвращает шрифт
`int getHeight()`	Возвращает высоту строки текста. Это значение можно использовать для вывода в окно многострочного текста
`int getLeading()`	Возвращает размер интерлиньяжа
`int getMaxAdvance()`	Возвращает ширину самого широкого символа. Возвращает -1, если это значение недоступно
`int getMaxAscent()`	Возвращает максимальный асцендер
`int getMaxDescent()`	Возвращает максимальный десцендер
`int[ ] getWidths()`	Возвращает ширины первых 256 символов
`int stringWidth(String str)`	Возвращает ширину строки, указанной в параметре str
`String toString()`	Возвращает строчный эквивалент вызывающего объекта

Рассмотрим несколько примеров.

Отображение многострочного текста

Возможно, самое обычное использование FontMetrics — для определения интервала между строками текста. Второе — для определения длины отображаемой строки.

Для отображения многострочного текста программа должна вручную отслеживать текущую позицию вывода. Когда требуется вывести новую строку, координата Y должна быть смещена к началу следующей строки. Когда строка отображается, координата X должна быть установлена в точку, где заканчивается предыдущая строка. Это позволяет записывать следующую строку, начиная с конца предыдущей.

Для определения интерлиньяжа можно использовать значение, возвращаемое методом getLeading(). Чтобы определять полную высоту шрифта, прибавьте значение, возвращенное методом getAscent(), к значению, возвращенному методом getDescent(). Эти значения можете использовать, чтобы позиционировать каждую строку выводимого текста. Однако во многих случаях нет нужды использовать эти индивидуальные значения. Часто все, что нужно знать — полную

высоту строки, которая является суммой межстрочного пробела, асцендера и десцендера. Самый простой способ получить эти значения — вызвать getHeight(). Просто увеличивайте координату Y на это значение, каждый раз, когда нужно продвинуться к следующей строке при выводе текста.

Чтобы начать вывод с конца предыдущего вывода на той же строке, нужно знать длину (в пикселах) каждой отображаемой строки. Для получения этого значения вызовите метод stringWidth(). Значение можно использовать для продвижения координаты X при отображении очередной строки.

В следующем апплете показано, как можно вывести в окно множество строк текста. Кроме того, демонстрируется, как можно вывести несколько строк текста на одной строке. Обратите внимание на переменные curX и curY. Они следят за текущей позицией текстового вывода.

Программа 111. Многострочный вывод

```
// Файл MultiLine.java
// Демонстрирует многострочный вывод.
import java.applet.*;
import java.awt.*;
/*
<applet code = "MultiLine" width = 300 height = 100>
</applet>
*/
public class MultiLine extends Applet {
   int curX = 0, curY = 0;      // Текущая позиция
   public void init() {
      Font f = new Font("SansSerif", Font.PLAIN, 12);
      setFont(f);
   }
   public void paint(Graphics g) {
      FontMetrics fm = g.getFontMetrics();
      nextLine("This is on line one.", g);
      nextLine("This is on line two.", g);
      sameLine(" This is on same line.", g);
      sameLine(" This,  too.", g);
      nextLine("This is on line three.", g);
   }

// Продвинуться к следующей строке.

   void nextLine(String s, Graphics g) {
      FontMetrics fm = g.getFontMetrics();
      curY += fm.getHeight();         // Продвижение к следующей строке
      curX = 0;
      g.drawString(s, curX, curY);
      curX = fm.stringWidth(s);       // Продвижение к концу строки
   }
```

```
// Показать на той же линии.
   void sameLine(String s, Graphics g) {
      FontMetrics fm = g.getFontMetrics();
      g.drawString(s, curX, curY);
      curX += fm.stringWidth(s);  // Продвижение к концу строки
   }
}
```

Воспользуемся утилитой appletviewer для запуска апплета. Подготовим файл RunMultiline.html:

```
<applet code = "MultiLine" width = 300 height = 100>
</applet>
```

Этот файл сохраним в папке ...\bin, в которой находится откомпилированный файл MultiLine.class. Выполним в командной строке команды показанные на рис.17. Последняя команда:

```
appletviewer RunMultiLine.html
```

запускает апплет.

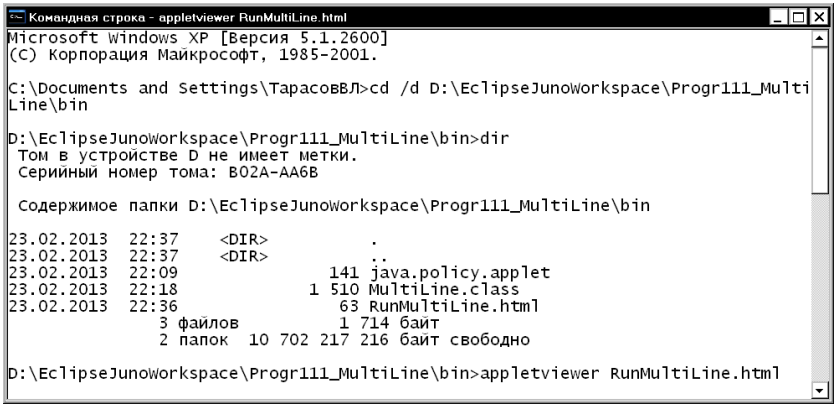


Рис. 17. Запуск апплета утилитой appletviewer

Пример вывода этой программы представлен на рис. 18.

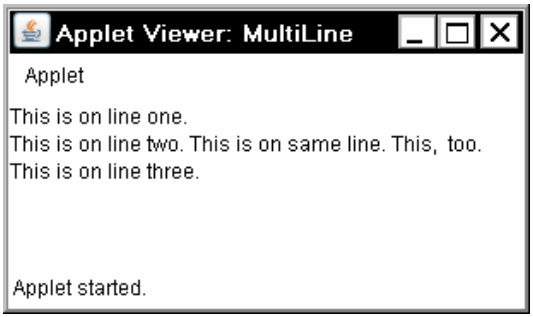


Рис. 18. Вывод многострочного текста

Выравнивание текста по центру

Здесь показан пример, в котором текст выравнивается по центру в окне апплета (по горизонтали и вертикали). Программа получает асцендер, десцендер и ширину строки и с их помощью вычисляет позицию, с которой нужно отобразить в центре окна.

Программа 112. Центрирование текста

```
// Файл CenterText.java
// Выравнивает текст по центру.
import java.applet.*;
import java.awt.*;
/*
<applet code = "CenterText" width = 200 height = 100>
</applet>
*/

public class CenterText extends Applet {
   final Font f = new Font("SansSerif", Font.BOLD, 18);
   public void paint(Graphics g) {
      Dimension d = this.getSize();  // Размеры окна
      g.setColor(Color.white);
      g.fillRect(0, 0, d.width, d.height);// Закраска окна белым цветом
      g.setColor(Color.black);
      g.setFont(f);
      drawCenteredString("This is centered.", 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) {
// w, h - ширина и высота окна
      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);
   }
```

}

Пример вывода этой программы представлен на рис. 19.

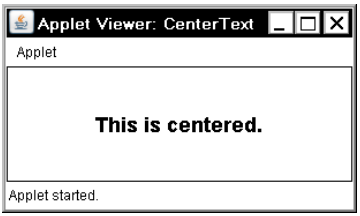


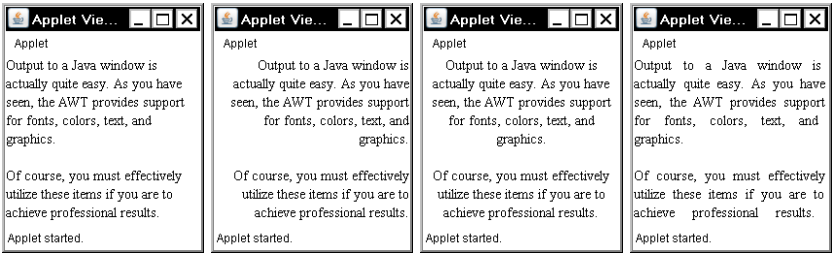
Рис. 19. Центрирование текста

Выравнивание многострочного текста

Большинство текстовых процессоров могут выравнивать текст влево и/или вправо, по центру. В следующей программе вы увидите, как можно выполнить эти действия.

Программа 113. Выравнивание многострочного текста

В программе происходит следующее: строка, которая будет выравниваться, разбивается на отдельные слова. Для каждого слова программа следит за его длиной в текущем шрифте и автоматически продвигается к следующей строке, если слово не будет помещаться в текущую строку. Каждая законченная строка отображается в окне в выбранном в текущий момент стиле выравнивания. После каждого щелчка мыши в окне апплета, стиль выравнивания изменяется. Пример вывода этой программы представлен на рис.20.



а б в г

Рис. 20. Выравнивание текста: а – влево, б – вправо, в – по центру, г – по ширине

```
// Файл TextLayout.java
// Демонстрирует выравнивание текста.
import java.applet.*;
import java.awt.*;
```

```
import java.awt.event.*;
import java.util.*;
/*
<title> Text Layout </title>
<applet code = "TextLayout" width = 200 height = 200>
<param name = "text" value =
                        "Output to a Java window is actually quite easy.
      As you have seen, the AWT provides support for fonts,
      colors, text, and graphics.
<p> Of course, you must effectively utilize these items if you are to
 achieve professional results.">
 <param name = "fontname" value = "Serif">
 <param name = "fontSize" value = "14">
</applet>
*/
public class TextLayout extends Applet {
   final int LEFT = 0;
   final int RIGHT = 1;
   final int CENTER = 2;
   final int LEFTRIGHT = 3;
   int align;
   Dimension d;
   Font f;
   FontMetrics fm;
   int fontSize;
   int fh, bl;
   int space;
   String text;
   public void init() {
      setBackground(Color.white);
      text = getParameter("text");
      try {
         fontSize = Integer.parseInt(getParameter("fontSize"));
      }
      catch (NumberFormatException e)
      {
         fontSize = 14;
      }
      align = LEFT;
      addMouseListener(new MyMouseAdapter(this));
   }
   public void paint(Graphics g){
      update(g);
   }
   public void update(Graphics g) {
      d = getSize();
      g.setColor(getBackground());
      g.fillRect(0, 0, d.width, d.height);
      if(f == null)
         f = new Font(getParameter("fontname"), Font.PLAIN, fontSize);
      g.setFont(f);
      if(fm == null) {
         fm = g.getFontMetrics();
         bl = fm.getAscent();
```

```
            fh = bl + fm.getDescent();
            space = fm.stringWidth(" ");
        }
        g.setColor(Color.black);
        StringTokenizer st = new StringTokenizer(text);
        int x = 0;
        int nextx;
        int y = 0;
        String word, sp;
        int wordCount = 0;
        String line = "";
        while (st.hasMoreTokens()) {
            word = st.nextToken();
            if(word.equals("<p>")) {
                drawString(g, line, wordCount, fm.stringWidth(line), y + bl);
                line = "";
                wordCount = 0;
                x = 0;
                y = y + (fh * 2);
            }
            else {
                int w = fm.stringWidth(word);
                if(( nextx = (x + space + w)) > d.width) {
                    drawString(g, line, wordCount, fm.stringWidth(line), y + bl);
                    line = "";
                    wordCount = 0;
                    x = 0;
                    y = y + fh;
                }
                if(x != 0)
                    sp = " ";
                else
                    sp = "";
                line = line + sp + word;
                x = x + space + w;
                wordCount++;
            }
        }
        drawString(g, line, wordCount, fm.stringWidth(line), y+bl);
    }
    public void drawString(Graphics g, String line,
                           int wc, int lineW, int y) {
        switch(align) {
            case LEFT:
                g.drawString(line, 0, y);
            break;
            case RIGHT:
                g.drawString(line, d.width - lineW, y);
            break;
            case CENTER:
                g.drawString(line, (d.width - lineW)/2, y);
            break;
            case LEFTRIGHT:
                if (lineW < (int) (d.width * .75))
                    g.drawString(line, 0, y);
```

```
            else {
               int toFill = (int)((d.width - lineW)/wc);
               int nudge = d.width - lineW - (toFill * wc);
               int s = fm.stringWidth(" ");
               StringTokenizer st = new StringTokenizer(line);
               int x = 0;
               while(st.hasMoreTokens()) {
                  String word = st.nextToken();
                  g.drawString(word, x, y);
                  if(nudge > 0) {
                     x = x + fm.stringWidth(word) + space + toFill + 1;
                     nudge--;
                  }
                  else
                     x = x + fm.stringWidth(word)  + space + toFill;
               }
            }
         break;
      }
   }
}
class MyMouseAdapter extends MouseAdapter {
   TextLayout tl;
   public MyMouseAdapter (TextLayout tl) {
      this.tl = tl;
   }
   public void mouseClicked(MouseEvent me) {
      tl.align = (tl.align + 1) % 4;
      tl.repaint();
   }
}
```

Посмотрим, как работает апплет. Сначала он создает несколько констант, которые будут использоваться для определения стиля выравнивания, и затем объявляет несколько переменных. Метод init() получает текст, который будет отображен. Затем апплет инициализирует размер шрифта в блоке try-catch, который установит размер шрифта в 14 пунктов, если в документе HTML отсутствует параметр fontSize. Параметр text является длинной строкой текста с HTML-тегом <p> в качестве абзаца-разделителя.

Двигателем этого примера является метод update(). Он устанавливает шрифт и получает базовую линию и высоту шрифта от объекта шрифтовой метрики (FontMetrics fm). Затем создается объект StringTokenizer, который используется, чтобы извлечь следующую лексему (строку, ограниченную пробелами) из строки, определенной в переменной text. Если следующая лексема есть <p>, метод увеличивает интервал строк. Иначе, он выясняет, выходит ли длина этой лексемы за ширину колонки. Если строка заполнена текстом или нет больше лексем, строка выводится методом drawString().

Три первых направления в переключателе метода drawString() работают идентично: каждое выравнивает строку, которая передана через параметр line, по левому, правому краю или по центру окна, в зависимости от стиля выравнивания. Направление LEFTRIGHT выравнивает как левую, так и правую стороны строки. Для этого вычисляется остаточный пробел (разность между шириной строки и шириной колонки) и распределяется между всеми словами. Последний метод в этом классе изменяет стиль выравнивания каждый раз, когда вы щелкаете мышью в окне апплета.

Для запуска программы надо подготовить файл RunTextLayout.html:

```
<title> Text Layout </title>
<applet code = "TextLayout" width = 200 height = 200>
<param name = "text" value =
                    "Output to a Java window is actually quite easy.
      As you have seen, the AWT provides support for fonts,
      colors, text, and graphics.
<P> Of course, you must effectively utilize these items if you are to
 achieve professional results.">
 <param name = "fontname" value = "Serif">
 <pararn name = "fontSize" value = "14">
</applet>
```

Запускается ааплет или утилитой appletviewer или браузером. На рис.21 показано окно браузера Mozilla Firefox с апплетом с выравнивание текста по левому и по ширине.

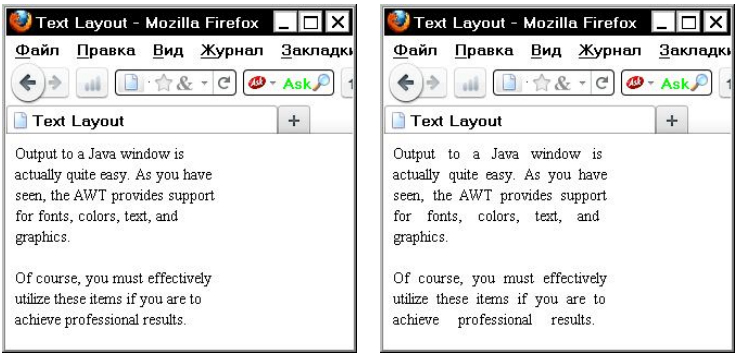


Рис. 21. Апплет в окне браузера с выравниванием текста по левому краю и по ширине

Исследование текста и графики

Хотя эта глава описывает наиболее важные аспекты и общую методику, которые используются при отображении текста или графики, она не охватывает всех возможностей Java. В этой области ожидаются

дальнейшие усовершенствования и улучшения, поскольку Java и вычислительная среда продолжают развиваться. Например, Java 2 добавляет к AWT новую подсистему, названную *Java 2D*. Java 2D поддерживает улучшенное управление графикой, включая трансляцию координат, вращение и масштабирование, расширенные свойства изображений. Если обработка расширенной графики представляет для вас определенный интерес, то вы, несомненно, захотите детально исследовать Java 2D.

Задачи 13-17. Графика

13. Напишите программу, изображающую в окне правильную пятиугольную звезду.

14. Напишите программу, изображающую в окне окружность и вписанный в нее правильный шестиугольник.

15. Напишите программу, изображающую в окне угол, опирающийся на некоторую дугу. Из вершины угла проведите два луча так, чтобы угол делился на три равные части.

16. Напишите программу, отображающую в окне текущие размеры окна по щелчку мыши.

17. Перепишите программы 100-113, для фреймового окна вместо окна апплета, выполнив те же самые действия.