Дмитриева Марина Валерьевна

ОСНОВЫ ПРОГРАММИРОВАНИЯ ДЛЯ ИНТЕРНЕТ. АППЛЕТЫ. ЗАНЯТИЕ 1

Одна из областей применения современного объектно-ориентированного языка программирования Java – программирование для Интернет. Специализированные клиентские программы – апплеты – позволяют решать многие интересные задачи. Статья посвящена первоначальным сведениям, позволяющим создавать простые апплеты.

ВВЕДЕНИЕ

Язык программирования Java разработан программистами фирмы Sun Microsystems. Первая реализация языка Java – Java Development Kit (JDK) 1.0 выпущена в 1995 году. Основным компонентом JDK является компилятор с языка Java в байт-код – платформо-независимый код виртуальной машины. Виртуальная Java-машина (JVM) является интерпретатором байт-кода. Программу на языке Java, откомпилированную в байт-код, можно использовать на любой платформе, на которой реализована JVM. Такой подход обеспечивает мобильность (переносимость) программ, написанных на языке Java.

В основу языка Java положена идея объектно-ориентированного программирования. Основными конструкциями, из которых строится программа на Java, являются классы. Класс представляет собой набор данных и методов их обработки. Механизм наследования позволяет описать новый класс на основе существующего (родительского), при этом свойства и методы (функциональность) родительского класса наследуются новым классом.

Механизм наследования позволяет описать новый класс на основе существующего...

АППЛЕТЫ

В программировании для Интернет язык Java занимает особое положение. Можно создавать апплеты – специального вида Java-программы, выполняющиеся на HTML-страницах с помощью обозревателя, либо выполняющиеся с помощью специальной программы просмотра апплетов – *appletviewer*. В виде апплетов можно реализовать анимационные эффекты, игровые программы, программы контроля знаний, визуализаторы алгоритмов и решать многие другие интересные задачи.

ПЕРВЫЙ АППЛЕТ

Изучение языка программирования принято начинать с простой программы. Далее по мере изучения новых возможностей языка усложняются и программы, использующие введенные конструкции при решении конкретных задач. Самая простая программа – это программа, осуществляющая вывод некоторого сообщения. Создадим апплет, который будет выводить заданную строку в графический контекст апплета на экране. Вместо общепринятого приветствия «Hello, world» выведем для разнообразия текст пословицы. Текст программы приведен в листинге 1.

Рассмотрим текст программы. Каждый апплет, который создается, должен быть подклассом класса Applet. Апплет выпол-

няет вывод в свой графический контекст. Поддержка графического интерфейса осуществляется с помощью пакета AWT (Abstract Windowing Toolkit), содержащего набор классов для создания графического интерфейса пользователя, отображение графики и изображений. Первые две строки текста программы обеспечивают импорт необходимых классов.

Третья строка содержит определение нового класса **begapp**. Программа на языке Java строится из классов. Простая программа состоит из одного класса. Определение класса в простейшем случае должно содержать две обязательные компоненты: ключевое слово **class** и следующее за ним имя класса. Кроме обязательных частей, в определении могут присутствовать модификаторы, характеризующие некоторые свойства класса. Модификатор доступа **public** означает, что класс доступен извне.

Конструкция **extends Applet** означает, что определенный класс является подклассом (наследником) класса Applet. Внутри фигурных скобок располагаются описания переменных и методов данного класса. В описанном классе **begapp** отсутствуют переменные, и имеется только один метод с именем **paint**, который используется для отображения объекта, которому он принадлежит. В тексте программы метод **paint** переопределен таким образом, чтобы выводить изображение строки с заданной позиции экрана. Функция **g.drawString(msg,x,y)** отображает строку на экране. Её аргументы – выводимая строка **msg** и координаты **x,y** строки в окне. Точка с координатами **(0,0)** находится в левом верхнем углу области, ось 0x идет вправо, ось 0y – вниз, координаты вычисляются в пикселях.

КОМПИЛЯЦИЯ АППЛЕТА

Исходный текст программы на языке Java может быть создан в текстовом редакторе (Notepad или WordPad для Windows). В языке Java требуется, чтобы имя файла исходного текста совпадало с именем класса и имело расширении **.java**. В представленном примере текст программы должен быть сохранен в файле с именем **begapp.java** в текущей папке.

Компиляцию и выполнение программы можно осуществлять в режиме командной строки (MS DOS Prompt в Windows). Компиляция программы выполняется командой

```
C:\>javac begapp.java
```

Компилятор **javac** по тексту программы сформирует байт-код и запишет результат в текущую папку под именем **begapp.class**.

ВЫПОЛНЕНИЕ АППЛЕТА

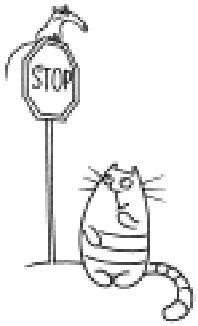
Один из способов выполнения апплета предполагает использование Java-совместимых обозревателей типа Microsoft Internet Explorer или Netscape Navigator. В этом случае нужно создать HTML-документ и встроить в него апплет. Для встраивания апплета можно использовать теги **<applet>** или **<object>**. Тег **<applet>** имеет параметр code, значение которого должно определять имя файла, содержащего файл с расширением **class**:

```
<applet code="begapp.class">
                              </applet>
```

Листинг 1. Апплет для вывода строки

```
import java.applet.*;
import java.awt.*;
public class begapp extends Applet {
     public void paint (Graphics g) {
          g.drawString("Делу - время, потехе - час!", 10, 30);
     }
}
```

После создания документа со встроенным апплетом (**begapp.html**) можно запустить обозреватель и затем загрузить HTML-документ, что приведёт к выполнению апплета. Следует иметь в виду, что тег **<applet>** может иметь и другие параметры.



Метод stop() приостанавливает выполнение апплета...

Второй способ выполнения апплета состоит в использовании специальной программы просмотра апплетов, стандартной утилиты JDK: **appletviewer**. Программа **appletviewer** выполняет апплет в отдельном окне. Для выполнения апплета в командной строке надо написать:

```
C:\>appletviewer begapp.html
```

Если запустить апплет с помощью программы **appletviewer**, появится окно, показанное на рис. 1.

СРЕДЫ РАЗРАБОТКИ ПРОГРАММ НА JAVA

Мы рассмотрели подготовку текстов программ на Java, компиляцию и выполнение в режиме командной строки. Интегральные среды разработки программ на Java обеспечивают набор и редактирование исходных текстов, компиляцию, выполнение и отладку программ.

Разные фирмы-разработчики предлагают интегральные среды, информацию о которых можно получить на Web-сайтах соответствующих фирм: Sun Microsystems, Borland, IBM и др.

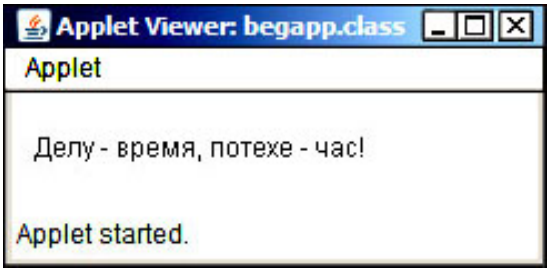


Рис. 1. Апплет выводит пословицу в свой графический контекст

ЖИЗНЕННЫЙ ЦИКЛ АПЛЕТА

Выполнение апплета происходит под управлением обозревателя или специальной программы просмотра апплетов **appletviewer**. Действия, которые выполняются апплетом, зависят от методов, определяющих так называемый *жизненный цикл апплета*. Вызывая эти методы, обозреватель или программа просмотра апплета **appletviewer** управляют поведением апплета.

Метод **init()** вызывается один раз сразу после загрузки документа и создания апплета. Он используется, для того чтобы подготовить условия для выполнения апплета, например создать интерфейс пользователя.

Метод **start()** вызывается каждый раз, когда апплет становится видимым, метод **stop()**, наоборот, когда апплет становится невидимым. Такая смена состояний происходит, например, при просмотре многостраничного документа или переходе по ссылке к другому документу и возвращении обратно. Метод **stop()** приостанавливает выполнение апплета, а метод **start()** возобновляет.

Метод **destroy()** вызывается один раз по завершении работы с HTML-документом.

Перечисленные методы могут переопределяться конкретным классом апплета. Рассмотренный ранее пример с выводом в графический интерфейс пословицы наследует все методы жизненного цикла апплета, переопределяет только один метод **paint()** для вывода изображения строки.

СОБЫТИЯ

Некоторые действия пользователя могут вызвать события. В языке Java каждому типу события соответствует класс, например, событиям, связанным с мышью, – класс **MouseEvent**, событиям, связанным с клавиатурой, – класс **KeyEvent**. Кроме того, что каждому событию соответствует объект-источник, событию соответствует так называемый объект-слушатель (even

listeners). Стать слушателем некоторой группы событий – значит, обрабатывать события данной группы. Для этого апплет должен реализовать *интерфейс*, соответствующий данной группе событий, и занести себя в список слушателей той группы событий, которую он должен обрабатывать.

ИНТЕРФЕЙСЫ

Интерфейс может содержать описание констант и заголовки методов. Сами методы являются пустыми, их реализация в интерфейсе не предусмотрена. Если класс реализует некоторый интерфейс, то он должен явно определить все методы, перечисленные в описании интерфейса.

Применение интерфейсов можно объяснить следующим образом. Класс в Java может наследовать свойства только одного класса. Если родительский класс не содержит необходимых методов, то наследующий класс должен определить и реализовать их самостоятельно. Удобно это делать в соответствии с определенной схемой, содержащей необходимые константы и названия методов, которая и предлагается интерфейсами. Класс может реализовать любое число интерфейсов.

ТЕКУЩИЕ КООРДИНАТЫ КУРСОРА МЫШИ ПРИ ПЕРЕМЕЩЕНИИ

Рассмотрим апплет, демонстрирующий обработку событий, возникающих при перемещении курсора мыши. Для обработки событий мыши можно реализовать интерфейс **MouseMotionListener**, определяющий два метода. Метод **mouseMoved()** вызывается каждый раз, когда мышь перемещается. Метод **mouseDragged()** вызывается каждый раз, когда мышь перемещается с нажатой клавишей (перетаскивается). Их общие формы: **void mouseDragged(MouseEvent e)** и **void mouseMoved(MouseEvent e)**.

Для слежения за этими событиями создается специальный объект при помощи метода **addMouseMotionListener(this)**. Когда этот объект регистрирует одно из перечисленных событий, вызывается метод обработки этого события, указанный

Метод mouseMoved() вызывается каждый раз, когда мышь перемещается.

Листинг 2. Текущие координаты курсора мыши при перемещении

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class mev1 extends Applet
       implements  MouseMotionListener  {
  public void init() {
// регистрация для прослушивания событий от мыши
     addMouseMotionListener(this);
  }
// реакция на событие перетаскивания курсора мыши
  public void mouseDragged(MouseEvent e) {}
// реакция на событие перемещения курсора мыши
  public void mouseMoved(MouseEvent e) {
// отображение текущих координат в строке статуса
   showStatus("Текущие координаты  мыши " + e.getX() + ", " + e.getY());
    }
 }
```

в интерфейсе MouseMotionListener и реализованный в самом классе. Идентификатор **this** обозначает текущий объект, в данном случае – сам апплет.

Для событий, связанных с компонентами AWT, определен пакет классов-источников событий **java.awt.event**, поэтому в тексте программы обеспечен импорт необходимых классов для обработки событий.

В листинге 2 приведен текст программы, в которой при перемещении курсора мыши в строке статуса отображаются текущие координаты. Из методов, составляющих жизненный цикл апплета, переопределен только метод **init**, остальные методы наследуются. Метод **showStatus** помещает свой аргумент в строку статуса. Текущие координаты курсора мыши можно получить с помощью методов **getX()** и **getY()** соответственно.

Если класс реализует некоторый интерфейс, то он должен явно определить все методы, перечисленные в описании интерфейса, поэтому в классе **mev1** присутствует «пустой» метод **mouseDragged**.

ПЕРЕМЕЩЕНИЕ С НАЖАТОЙ КНОПКОЙ МЫШИ (ПЕРЕТАСКИВАНИЕ)

Создадим апплет, в котором при перетаскивании вслед за курсором появляется текст. В строке статуса выдается сообщение, при реализации какого из методов фиксируются текущие координаты мыши, и сами значения текущих координат (листинг 3).

В классе с именем **mev2** описана строковая переменная **msg** и две переменные **curX** и **curY** целого типа для хранения положения курсора мыши, когда проис-

Листинг 3. Обработка событий мыши: перемещение и перетаскивние

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class mev2 extends Applet implements   MouseMotionListener {
     String msg ="";
// текущие координаты мыши
     int curX = 0,
         curY = 0;
     public void init() {
       addMouseMotionListener(this);
     }
// реакция на событие перетаскивание мыши
     public void mouseDragged(MouseEvent e) {
       curX = e.getX();
       curY = e.getY();
       msg = "слово";
       showStatus("Координаты мыши (при перетаскивании): " + curX + ", " + curY);
       repaint();
      }
// реакция на событие перемещение мыши
     public void mouseMoved(MouseEvent e) {
  //отобразить текущие координаты в строке статуса
       showStatus("Координаты мыши (при перемещении): "+e.getX() + ", "+e.getY());
     }
// Вывод сообщения msg в окно апплета с текущих координат
     public void paint (Graphics g) {
       g.drawString(msg, curX, curY);
     }
}
```

...апплет, в котором при перетаскивании вслед за курсором появляется текст.

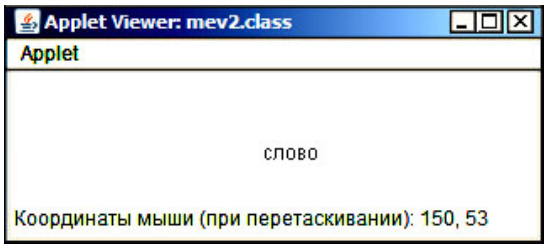


Рис. 2. Апплет с текущими координатами в строке статуса

ходят события. Эти координаты используются методом **paint()**, чтобы отобразить вывод в той точке, где эти события зафиксированы.

Внутри **init()**, как и в предыдущем примере, апплет регистрирует себя для прослушивания событий мыши с помощью метода **addMouseMotionListener(this)**. В тексте программы реализованы оба метода, определённых интерфейсом **MouseMotionListener**. Каждый метод обрабатывает свое событие и затем возвращает управление.

Метод **paint()** полностью перерисовывает объект. Метод **update()** также используется для изображения объекта, но рисует только изменения в изображении объекта, поэтому экономится время для восстановления изображения. Однако метод **update()** никогда не вызывается объектом непосредственно, а только через метод **repaint()**. Методы **paint()** и **update()**, как правило, переопределяются внутри объекта. Апплет показан на рис. 2.

РЕАЛИЗАЦИЯ ВСЕХ МЕТОДОВ РАБОТЫ С МЫШЬЮ

Создадим апплет, реализующий другие события от мыши: щелчок кнопкой мыши, нажатие кнопки мыши, вход курсора мыши в область графического контекста на экране, выход курсора за границу области. При реализации методов обработки событий выполняются однотипные действия: формируется сообщение о вызванном событии и задаются координаты, начиная с которых сообщение будет выведено в окно апплета. Это либо текущие координаты, либо постоянные значения. При реализации двух методов, связанных с перемещением курсора мыши, в строке статуса формируются текущие координаты курсора мыши.

...но рисует только изменения в изображении объекта,

Для обработки событий мыши можно реализовать еще один интерфейс **MouseListener**, определяющий пять методов. Если кнопка мыши нажата и сразу же отпущена, вызывается метод **mouseClicked()**. Когда указатель мыши входит в границы окна апплета, вызывается метод **mouseEntered()**, а когда выходит за границу – вызывается метод **mouseExited()**. При нажатии клавиши мыши вызывается метод **mousePressed()**, и, наконец, при отпускании клавиши – метод **mouseReleased()**.

Как и в предыдущих случаях, внутри класса **mev4** для слежения за этими событиями создается специальный объект при помощи метода **addMouseListener()**, регистрирующий одно из перечисленных событий, вызывается метод обработки этого события, указанный в интерфейсе **MouseListener** и реализованный в описываемом классе.

Как было указано выше, класс может реализовать любое число интерфейсов. В описываемом классе **mev4** реализуются оба интерфейса **MouseMotionListener** и **MouseListener**, поддерживающих работу с мышью. Это отражено в заголовке описания класса **mev4**.

Названия реализуемых интерфейсов перечислены через запятую. В методе **init()** регистрируются блоки прослушивания для этих событий. В обоих случаях как источником, так и получателем событий будет сам апплет (листинг 4).

РЕАЛИЗАЦИЯ СОБЫТИЙ ОТ КЛАВИАТУРЫ

При нажатии клавиши на клавиатуре возникает событие. Для обработки событий клавиатуры можно реализовать интерфейс **KeyListener**, содержащий методы **keyDown()** и **keyUp()**. Метод **keyDown()** вызывается каждый раз, когда пользователь нажимает какую-либо клавишу на клавиатуре, а метод **keyUp()** вызывается тогда, когда клавиша отпускается.

Создадим апплет, при работе которого после нажатия клавиши на клавиатуре в окне апплета появляется изображение нажатой клавиши. В листинге 5 приведен текст программы.

Рассмотрим текст программы. При выполнении метода **init()** создается специальный класс, регистрирующий события от клавиатуры. В качестве источника и слушателя событий от клавиатуры выступает сам апплет. Первоначально переменной **kp** присваивается значение **–1**, значение изменится после того, как будет нажата клавиша. С помощью метода **resize(200,200)** устанавливаются размеры графического интерфейса (окна) апплета.

При реализации метода **keyPressed()** извлекается из объекта, являющегося параметром, код нажатой клавиши. Далее метод **repaint()** заставляет апплет перерисовать свою область изображения.

ОБЪЕКТ FONT

Текст, выводимый в Java-программах, использует текущий шрифт. Можно улучшить вид апплета, используя различные шрифты. Каждый шрифт имеет атрибуты, характеризующие тип шрифта и определяющие его размер.

Класс **Font** позволяет создавать и отображать шрифты. Конструктор класса **Font** имеет три параметра: название гарнитуры, начертание и кегль. Все среды Java поддерживают шрифты: **Dialog**, **DialogInput**, **Sans Serif**, **Serif**, **Monospaced**, **Symbol**. Начертания шрифтов задаются с помощью констант: **Font.PLAIN** (обычный), **Font.BOLD** (полужирный), **Font.ITALIC** (курсив). Третий параметр задает размер шрифта в пунктах. Пункт – единица измерения высоты шрифта, равная 1/72 дюйма. Пример описания и создания шрифта с гарнитурой

Листинг 4. Обработка всех событий мыши

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class mev4 extends Applet
       implements   MouseMotionListener, MouseListener {
// выдаваемое сообщение
          String msg ="";
// текущие координаты мыши
          int curX = 10,
              curY = 20;
// начальные установки
      public void init() {
           addMouseMotionListener(this);
           addMouseListener(this);
       }
```

Продолжение листинга 4

```
//  событие щелчок кнопкой мыши
       public void mouseClicked(MouseEvent e) {
          curX = e.getX();
          curY = e.getY();
          msg = "Щелчок мыши";
          repaint();
       }
// Ввод курсора мыши в область окна апплета
       public void mouseEntered(MouseEvent e) {
          curX = 20;
          curY = 30;
          msg = "Ввод курсора мыши в область окна апплета";
          repaint();
       }
// Вывод курсора мыши из области окна апплета
       public void mouseExited(MouseEvent e) {
          curX = 30;
          curY = 40;
          msg = "Вывод курсора мыши из области окна апплета";
          repaint();
       }
// Нажатие кнопки мыши
       public void mousePressed(MouseEvent e) {
          curX = 40;
          curY = 50;
          msg = "Нажатие кнопки мыши";
          repaint();
       }
//  Отпускание кнопки мыши
       public void mouseReleased(MouseEvent e) {
          curX = 50;
          curY = 60;
          msg = «Отпускание кнопки мыши»;
          repaint();
       }
// Обработка события перетаскивания мыши
       public void mouseDragged(MouseEvent e) {
          curX = e.getX();
          curY = e.getY();
          msg = "слово";
          showStatus("Координаты мыши (при перетаскивании): " + curX + ", "+curY);
          repaint();
       }
// Обработка события перемещения мыши
       public void mouseMoved(MouseEvent e) {
       showStatus("Координаты мыши(при перемещении): " + e.getX() + ", " +e.getY());
       }
// Вывод сообщения msg в окно апплета с текущего положения курсора мыши
       public void paint (Graphics g) {
          g.drawString(msg, curX, curY);
       }
}
```

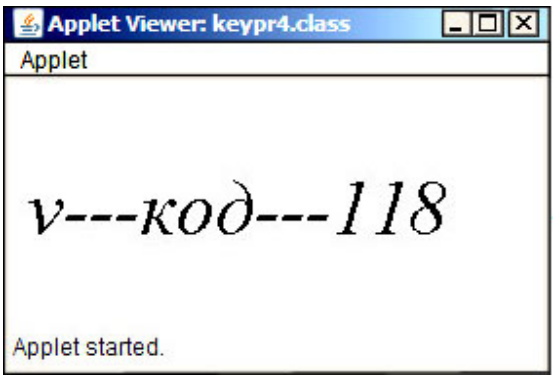


Рис. 3. Апплет с нажатой клавишей клавиатуры и кодом клавиши

"TimeRoman", полужирного начертания, и размера равного 50 пунктам:

```
Font f = new Font("TimeRoman",
                  Font.BOLD,50);
```

После создания шрифта можно его установить с помощью метода **setFont(f)**. После этого текст, выводимый в апплете, будет использовать новый шрифт.

ОТОБРАЖЕНИЕ НАЖАТОЙ КЛАВИШИ РАЗНЫМИ ШРИФТАМИ

Создадим апплет, при работе которого при нажатии на клавишу выводится изображение клавиши и ее код. При отпускании клавиши меняется шрифт, начертание и размеры изображения клавиши в окне апплета (рис. 3). В тексте программы (листинг 6) приведена реализация двух событий, связанных с клавиатурой: нажатие и отпускание клавиши.

При нажатии клавиши создается новый шрифт с полужирным начертанием

Листинг 5. Отображение нажатой клавиши клавиатуры

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class keypr1 extends Applet implements KeyListener{
// код нажатой клавиши
     int kp;
     public void init() {
          addKeyListener(this);
          kp =-1;
          resize(200,200);
     }
// нажатие клавиши на клавиатуре
     public void keyPressed (KeyEvent evt) {
          // извлекается код нажатой клавиши
          kp =  evt.getKeyChar();
          repaint();
     }
// отпускание клавиши
     public void keyReleased (KeyEvent evt) {}
// отпускание клавиши после нажатия
     public void keyTyped(KeyEvent evt) {}
// перерисовка
     public void paint (Graphics g) {
          String s="";
          if (kp != -1){
             s += (char) kp;
               g.drawString(s, 40, 150);
          }
     }
}
```

размером 50 пунктов, устанавливается и рисуется символ с помощью созданного шрифта. При отпускании клавиши создается новый шрифт с другими атрибутами, в частности, начертание шрифта – курсив.

ОБЪЕКТ COLOR

По умолчанию цвет переднего плана (символов) – чёрный, а цвет фона – светло-серый. Цветовая система AWT позволяет указывать в программе любой желаемый цвет. Работа с цветом поддерживает-

Листинг 6. Отображение нажатой клавиши клавиатуры и ее кода

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class keypr4 extends Applet implements KeyListener{
// шрифт
     Font f;
// код нажатой клавиши
     int kp;
// начальные действия
     public void init() {
          addKeyListener(this);
          kp =-1;
          f = new Font("TimeRoman",Font.BOLD,50);
          setFont(f);
          resize(350,200);
     }
// нажатие клавиши на клавиатуре
     public void keyPressed (KeyEvent evt) {
          // извлекается код нажатой клавиши
          kp =  evt.getKeyChar();
          f = new Font("TimeRoman",Font.BOLD,50);
          setFont(f);
          repaint();
     }
// отпускание клавиши
     public void keyReleased (KeyEvent evt) {
          f = new Font("Serif",Font.ITALIC,40);
          setFont(f);
          repaint();
     }
// отпускание клавиши после нажатия
     public void keyTyped(KeyEvent evt) {}
// перерисовка
     public void paint (Graphics g) {
          String s="";
          if (kp != -1){
               s += (char) kp;
               s += "---код---"+ kp;
              g.drawString(s, 10, 100);
          }
     } // paint
} // keypr4
```

ся классом **Color**. Класс **Color** определяет константы, которые можно использовать для указания цвета: **Color.black**, **Color.blue** и др.

Можно также создавать собственные цвета с помощью цветовых конструкторов. Рассмотрим один из них. Конструктор имеет три параметра целого типа, значение каждого из параметров между **0** и **255**. Первый параметр определяет интенсивность красного цвета, второй – интенсивность зеленого, третий – интенсивность синего. Цвет получается как смесь красного, зеленого и синего цветов заданной интенсивности. Например, конструктор **Color c= new Color(255, 100, 100)** задает светло-коричневый цвет. После того как цвет создан, можно установить новый цвет фона с помощью метода **setBackground(Color c)**. Текущий цвет шрифта можно изменить методом **setForeground(Color c)**.

ЦВЕТ ФОНА АППЛЕТА

В листинге 7 приведен текст программы, демонстрирующей изменение цвета фона в зависимости от текущих координат курсора мыши при перемещении курсора. Если курсор перемещается при нажатой кнопке мыши (перетаскивается), то цвет фона остается таким, каким он зафиксирован после последнего перемещения.

В методе **paint()** сначала создается новый цвет. Интенсивность всех трех составляющих цвета зависит от текущих координат курсора мыши и определяется как остаток при делении на **255**. После создания нового цвета, он устанавливается

Листинг 7. Задание цвета фона апплета в зависимости от текущих координат

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class mvecolback1  extends Applet
       implements   MouseMotionListener {
// текущие координаты мыши
       int curX = 0,
          curY = 0;
// блок прослушивания - сам апплет
       public void init() {
            addMouseMotionListener(this);
        }
// перетаскивание курсора мыши
       public void mouseDragged(MouseEvent me) {}
//  перемещение курсора мыши
       public void mouseMoved(MouseEvent me) {
           curX = me.getX();
           curY = me.getY();
//отображение  текущих координат мыши в статусной строке
           showStatus("Координаты мыши(при перемещении): " + curX + ",
                                                          " + curY);
           repaint();
       }
//задание нового цвета фона в окне апплета
       public void paint (Graphics g) {
           Color col= new Color(curX%255,curY%255,(curX+curY)%255+1);
           setBackground(col);
       }
}
```

Цветовая система AWT позволяет указывать в программе любой желаемый цвет…

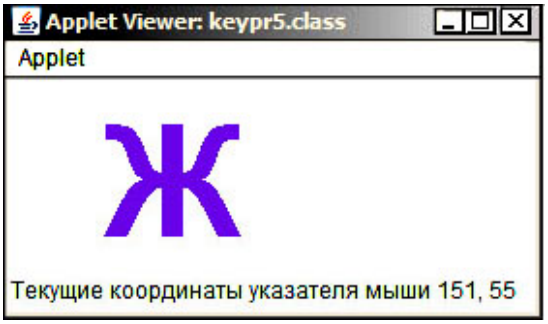


Рис. 4. Апплет с нажатой клавишей с данными и значениями в строке статуса

как цвет фона. При перемещении курсора мыши меняется цвет фона. Если же осуществлять перемещение с нажатой клавишей, то цвет фона не изменится.

ЦВЕТ ШРИФТА

Создадим апплет, при работе которого при нажатии на клавишу клавиатуры, соответствующий символ появляется в окне апплета. Цвет символа зависит от текущих координат курсора мыши. При перемещении курсора мыши меняется цвет символа в окне апплета. При создании класса **`keypr5`** реализуются два интерфейса **`KeyListener`** и **`MouseMotionListener`**, обрабатывающих события как от мыши, так и от клавиатуры (листинг 8).

Как и в предыдущем случае, формирование нового цвета зависит от текущих координат мыши. Интенсивность каждого из цветов определяется как остаток от деления выражений, зависящих от текущих координат курсора мыши, на **`255`**. После создания нового цвета он устанавливается как текущий цвет для символов, или цвет переднего плана.

На рис. 4 показано окно апплета с символом, соответствующим нажатой клавише клавиатуры. Цвет символа управляется текущими координатами курсора мыши. При перемещении курсора меняется цвет символа.

ЗАДАЧИ

1. Создайте апплет, который выводит текст пословицы «Каждый охотник желает знать, где сидит фазан». Первое слово – красным цветом, второе – оранжевым, третье – желтым, четвертое – зеленым, пятое – голубым, шестое – синим, седьмое – фиолетовым.
2. Создайте апплет, который выводит текст пословицы черным цветом, но при попадании курсора мыши на конкретное слово, цвет слова меняется по правилу, описанному в предыдущем примере.
3. Создайте апплет, который выводит в окно текст. При попадании курсора мыши на текст, размер шрифта увеличивается, при покидании курсором мыши текста – размер становится первоначальным.
4. Создайте апплет, в котором текст по щелчку кнопкой мыши выводится с текущей позиции, цвет шрифта и цвет фона зависят от координат курсора мыши.
5. Создайте апплет, содержащий сведения: фамилия, имя и отчество, номер школы, номер класса и по желанию другая информация. Предложите свой анимационный эффект для созданного текста.

Литература

1. *Дж. Вебер* Технология Java в подлиннике: пер. с англ. СПб.: BHV, 1997. 1104 с. ил.
2. *Сафонов В.О.* Введение в Java-технологию. СПб.: Наука, 2002. 187 с.

Листинг 8

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class keypr5 extends Applet
       implements KeyListener, MouseMotionListener {
// код нажатой клавиши
     int kp;
// текущие координаты мыши
     int curX = 0,
         curY = 0;
     public void init() {
          addKeyListener(this);
          addMouseMotionListener(this);
          kp =-1;
          Font f = new Font("TimeRoman",Font.BOLD,80);
          setFont(f);
          resize(300,300);
     }
// нажатие клавиши на клавиатуре
     public void keyPressed (KeyEvent evt) {
          kp =  evt.getKeyChar();
          repaint();
     }
// другие методы интерфейса KeyListener
     public void keyReleased (KeyEvent evt) {}
     public void keyTyped(KeyEvent evt) {}
// обработка события перетаскивания
       public void mouseDragged(MouseEvent e) {}
// обработка события перемещения курсора мыши
     public void mouseMoved(MouseEvent e) {
          curX = e.getX();
          curY = e.getY();
// отображение текущих координат в строке статуса
     showStatus("Текущие координаты указателя мыши "+curX + ", "+curY);
       repaint();
     }
// отображение в графическом интерфейсе
     public void paint (Graphics g) {
          String s="";
          Color col= new Color(curX%255,curY%255,(curX+curY)%255+1);
          setForeground(col);
          if (kp != -1){
                s += (char) kp;
              g.drawString(s, 40, 150);
          }
     }
}
```


Дмитриева Марина Валерьевна, доцент кафедры информатики математико-механического факультета Санкт-Петербургского государственного университета.