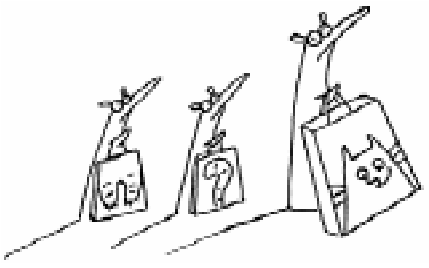
Дмитриева Марина Валерьевна

ОСНОВЫ ПРОГРАММИРОВАНИЯ ДЛЯ ИНТЕРНЕТ. АППЛЕТЫ. ЗАНЯТИЕ 4. ИНСТРУМЕНТАРИЙ ДЛЯ СОЗДАНИЯ ГРАФИЧЕСКИХ ИНТЕРФЕЙСОВ. ТЕКСТ И КНОПКИ

Статья посвящена средствам создания и управления визуальными компонентами: кнопками и информационными строками.

Графический интерфейс пользователя (graphical user interface, *GUI*) – система средств взаимодействия пользователя с компьютером, основанная на представлении доступных пользователю системных объектов в виде графических компонентов экрана (окон, кнопок, списков и т. п.). При разработке инструментов создания GUI важным является удобство использования визуальных интерфейсов и эффективность реализации графических средств.

Разработчики Java создали удобные инструменты для поддержки GUI, один из них – Abstract Windowing Toolkit (AWT), реализованный в виде пакета **`java.awt`** и его подпакетов. AWT позволяет управлять визуальными компонентами, кнопками, окнами, списками и др., управлять шрифтами и цветами, предоставляет возможности создания и обработки событий, связанных с визуальными компонентами и генерируемых пользователем при выборе тех или иных компонентов управления, и многое другое.



...реализованный в виде... его подпакетов.

В AWT базовые классы образуют иерархическую структуру. Класс **`Component`** определяет самые общие свойства элементов AWT. Его потомок – класс **`Container`** определяет свойства визуальной компоненты как контейнера других компонент. В AWT контейнером может быть либо окно, либо панель. Размеры окна можно менять с помощью мыши. Панель – область экрана, в которой Java-программа может размещать свои компоненты. Основные методы класса **`Container`** – добавление подкомпоненты в контейнер и удаление подкомпоненты из контейнера. AWT содержит большое число элементов управления. Мы начнем рассмотрение с самых простых элементов: текста (метка или пояснение) и кнопки.

ЭЛЕМЕНТЫ УПРАВЛЕНИЯ. ОСНОВНЫЕ ПОНЯТИЯ

Элементы управления (**`controls`**) – это компоненты, которые предоставляют пользователю различные способы взаимо-

действия с приложением. Для включения элемента управления в окно нужно добавить его к окну. Для этого необходимо сначала создать экземпляр элемента управления и затем добавить его к окну вызовом метода **add()**, который определён в классе **Container**. Метод **add()** имеет несколько форм. Мы пока будем использовать следующую форму:

```
Component add(Component compObj),
```

где **compObj** – экземпляр элемента управления, который мы хотим добавить. Метод **add** возвращает ссылку на объект, который передаётся параметром **compObj**. Сразу после добавления элемент управления будет автоматически выводиться на экран каждый раз, когда отображается его родительское окно.

Если нужно удалить элемент управления из окна, когда он больше не нужен, то следует вызвать метод **remove()**, который определён в классе **Container**. Его общая форма:

```
void remove(Component obj),
```

где **obj** – ссылка на элемент управления, который нужно удалить. Вызывая метод **removeAll()**, можно удалить все элементы управления.

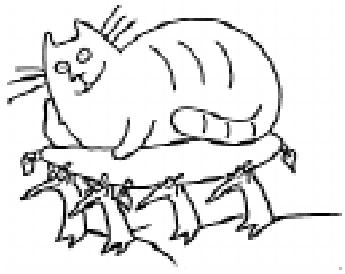
Все элементы управления генерируют события, когда к ним обращается пользователь. Таким образом, программа просто реализует соответствующий интерфейс и затем регистрирует блок прослушивания для каждого элемента, которым нужно управлять.

ТЕКСТ (Label)

Простейшим компонентом AWT является текст. Он представляет собой информационную текстовую строку. Существуют три способа создания информационной строки. Самый простой – создать пустую строку:

```
Label mytext1 = new Label().
```

Второй способ позволяет передать текст в качестве параметра:



...базовые классы образуют иерархическую структуру.

```
Label mytext2 = new Label ('Текст,
                помещаемый в апплет')
```

Текст может быть выровнен влево, по центру и вправо. Для задания выравнивания при создании объектов можно использовать константы: **Label.LEFT**, **Label.CENTER**, **Label.RIGHT**.

Третий способ предполагает создание информационной строки с выравниванием. Текст с выравниванием создается так:

```
Label mytext3 = new Label ('Текст,
выровненный по центру', Label.CENTER).
```

ТЕКСТ С РАЗЛИЧНЫМИ СПОСОБАМИ ВЫРАВНИВАНИЯ

В листинге 1 приведен текст апплета, в котором описаны три текстовых объекта **l1**, **l2**, **l3**. Создаются объекты с помощью конструктора **new**, первый параметр которого задает информационную строку, второй параметр – способ выравнивания. Для добавления объектов в окно используется метод **add**, после применения которого объект появляется на экране.

Апплет показан на рис. 1.

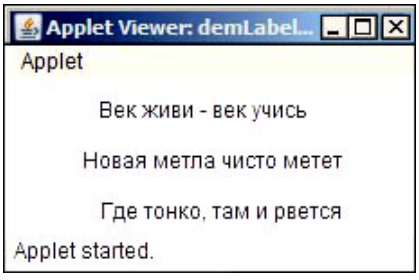


Рис. 1. Текст с различными способами выравнивания

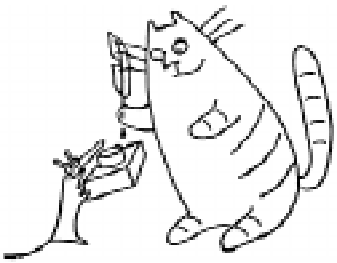
Листинг 1. Текст с различными способами выравнивания

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class  demLabel1 extends Applet  {
// описание текстовых объектов
    Label l1, l2, l3;
    public void init() {
// создание объектов
      l1 = new Label('Век живи - век учись', Label.LEFT);
      l2 = new Label('Новая метла чисто метет', Label.CENTER);
      l3 = new Label('Где тонко, там и рвется', Label.RIGHT);
// добавление объектов в апплет
      add(l1);
      add(l2);
      add(l3);
 }
}
```

КНОПКА (Button)

Наиболее широко используемым элементом управлением является кнопка. Кнопка – это компонент, который содержит текстовую метку и генерирует событие, когда её нажимают. Кнопки являются объектами класса **Button**. В классе **Button** определяются два конструктора: **Button()** и **Button(String str)**.

В первом случае создается пустая кнопка, во втором – кнопка с текстовой меткой, которая передаётся с помощью параметра **str**. После того как кнопка создана, можно установить её метку, вызывая метод **setLabel()**. Извлечь метку кнопки можно вызовом **getLabel()**. Задать метку для кнопки можно с помощью метода **setLabel(String str)**, где параметр **str** указывает новую метку для кнопки.



...кнопка с текстовой меткой...

ИСПОЛЬЗОВАНИЕ КНОПОК. СОБЫТИЕ ActionEvent

После того как кнопка создана, можно с ее помощью выполнять некоторые действия. Каждый компонент из AWT имеет метод **action()**, который вызывается при выполнении над компонентом какого-либо действия. Для кнопки метод **action()** вызывается после нажатия на нее. Метод **action()** для любого компонента имеет вид:

Puplic Boolean action (Event event, Object whatAction),

где **event** – событие, происшедшее с компонентом. Параметр **whatAction** содержит дополнительную информацию о компоненте. Для кнопок параметр **whatAction** задает текст кнопки, которая была нажата. Параметр event содержит информацию о событии, например компонент, к которому относится событие **(event.target)**.

При обработке события следует проверять, соответствует ли тип объекта **event.target** типу, к которому предположительно относится событие. Например, если ожидается, что событие относится к объекту **Button**, необходимо убедиться, что **event.target instanceof Button** имеет значение **true**.

МЕТОД `action`. ИЗМЕНЕНИЕ ЦВЕТА ФОНА АППЛЕТА ПО КНОПКЕ

Создадим апплет, содержащий одну кнопку и текст. При нажатии на кнопку изменяется цвет фона апплета на красный, при повторном нажатии на кнопку цвет фона становится синим. В программе описывается объект класса **`Button (Button but)`**, в методе **`init`** с помощью конструктора создается кнопка с надписью. В окне апплета размещается кнопка и информационная строка с помощью метода **`add`**.

При нажатии на кнопку изменяется цвет фона...

Для обработки события применим метод **`action()`**. Сначала проверяется, относится ли событие к кнопке. Если это не так, то выдается значение **`false`**. Если источником события является кнопка, то проверяется значение переменной **`k`** и устанавливается цвет фона апплета. Меняется текст информационной строки, в строку статуса помещается значение переменной **`k`**. В листинге 2 приведен текст программы, выполняющей описанные действия.

Листинг 2. Изменение цвета фона апплета и текста пословицы при щелчке по кнопке

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class demButton1 extends Applet  {
// описание объктов - кнопки и текстового объекта
    Button but;
    int k = 1;
    Label l;
    public void init() {
     setSize(250,100);
// создание объектов – кнопки и информационной строки
       but = new Button('Изменить цвет фона');
      l = new Label('Без труда не вытащишь и рыбку из пруда', Label.CENTER);
// добавление объектов в окно апплета
        add(but);
        add(l);
 }
// обработка события нажатия кнопки
    public boolean action (Event ae, Object whatAction) {
      if (!(ae.target instanceof Button))
        { return false; }
      else
        if (k==1)
          {setBackground(Color.red);
           l.setText('Век живи - век учись');
           k = 2;   }
        else
          {setBackground(Color.blue);
          l.setText('Делу - время, потехе - час');
          k =1 ;}
      showStatus('k = ' + k);
      repaint();
      return true;
    }
}
```

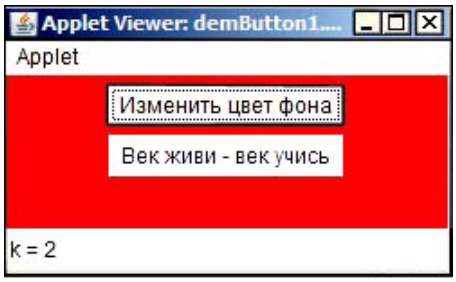


Рис. 2. Изменение цвета фона апплета при нажатии на кнопку

На рис. 2. показан апплет после нажатия на кнопку.

МАНИПУЛИРОВАНИЕ С ТРЕМЯ КНОПКАМИ

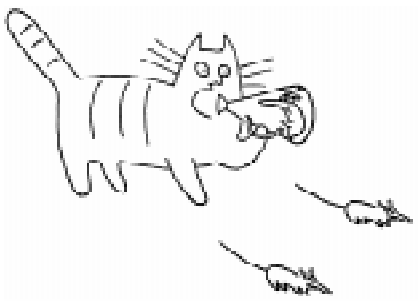
Реализуем апплет, в котором предусмотрены три кнопки. В качестве текстовых меток для кнопок используются названия цветов. При нажатии на кнопку с подписью «Красный» будет устанавливаться красный цвет фона для окна апплета. Аналогично для двух других кнопок (синий и зеленый).

Каждый раз, когда нажимается одна из кнопок, то анализируется текст кнопки, которая была нажата и в соответствии с этим устанавливается цвет фона апплета. В этом примере метка кнопки используется для того, чтобы определить, какая из кнопок была нажата. Как и в предыдущем случае сначала проверяется, генерируется ли событие кнопкой **`(ae.target instanceof Button)`**. Затем надо разобраться, какой именно кнопкой генерируется событие. Для этого используется второй параметр метода **`action`**. Строковая переменная **`str`** получает значение второго параметра метода **`action`** в результате выполнения оператора присваивания:

```
String str = (String) whatAction.
```

Далее по названию кнопки определяется, какая кнопка была нажата и устанавливается соответствующий цвет фона апплета. В строку статуса помещается значение второго параметра. В листинге 3 приведен текст программы, реализующий описанные действия.

На рис. 3 показан апплет, содержащий три кнопки, управляющие цветом фона.



Когда источник порождает событие, он оповещает все объекты...

ИНТЕРФЕЙС `ActionListener`

Интерфейс **`ActionListener`** определяет метод **`actionPerformed()`**, который вызывается при возникновении события от мыши. В качестве аргумента в этот метод передаётся объект **`ActionEvent`**. Он содержит как ссылку на кнопку, которая сгенерировала событие, так и ссылку на строку, которая является меткой кнопки. Для идентификации кнопки можно использовать любую из этих ссылок.

РИСОВАНИЕ ФИГУРЫ ПРИ НАЖАТИИ НА КНОПКУ

Создадим апплет, в котором расположено три кнопки с названием геометрических фигур. При нажатии на кнопку с меткой Квадрат в области апплета рисуется квадрат и формируется об этом сообщение в строковой переменной **`msg`**. Описываемый класс **`demButton3`** реализует интерфейс **`ActionListener`**, содержащий метод **`actionPerformed()`**, который вызывается при возникновении события от мыши.

Для каждого события существует порождающий его объект. Слушателем события является объект, заинтересованный в получении данного события. В объекте, который порождает событие (в источнике событий), содержится список слушателей, заинтересованных в получении уведомления о том, что данное событие про-

изошло, а также методы, которые позволяют слушателям добавлять или удалять себя из этого списка. Когда источник порождает событие, он оповещает все объекты слушателей событий о том, что данное событие произошло.

По установленному соглашению, методам слушателей событий может быть передан один единственный аргумент, являющийся объектом того события, которое соответствует данному слушателю. В этом объекте должна содержаться вся ин-

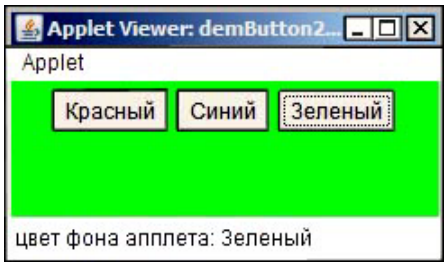


Рис. 3. Апплет с тремя кнопками, управляющими цветом фона

Листинг 3. Пример с тремя кнопками, управляющими цветом фона апплета

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class demButton2 extends Applet  {
// описание кнопок
    Button b1, b2,b3;
    public void init() {
     setSize(200,100);
// создание объектов
        b1 = new Button('Красный');
        b2 = new Button('Синий');
        b3 = new Button('Зеленый');
// добавление объектов в апплет
        add(b1);
        add(b2);
        add(b3);
}
// обработка события
    public boolean action (Event ae, Object whatAction) {
// проверка, что событие относится к кнопке
    if (!(ae.target instanceof Button))
     { return false; }
    else {
            String str = (String) whatAction;
            if(str == 'Красный')
             { setBackground(Color.red);}
            else
              if(str == 'Синий')
                { setBackground(Color.blue);}
              else
                  if(str == 'Зеленый')
                    { setBackground(Color.green);}
            showStatus('цвет фона апплета: ' + str);
            repaint();
            return true;
           }
     }
}// demButton2
```

формация, необходимая программе для формирования реакции на данное событие. В методе **init** с помощью конструктора создаются три объекта – кнопки, далее при помощи метода **add** кнопки добавляются в окно и регистрируются в качестве слушателей событий от мыши. При щелчке кнопкой мыши по одной из трех кнопок порождается событие, которое обрабатывается с помощью метода **actionPerformed** интерфейса **ActionListener**. Переменная **k** используется для запоминания нажатой кнопки, чтобы с помощью метода **paint** нарисовать соответствующую фигуру. В листинге 4 содержится текст программы, реализующей рисование различных геометрических фигур как реакцию на нажатие соответствующей кнопки мыши.

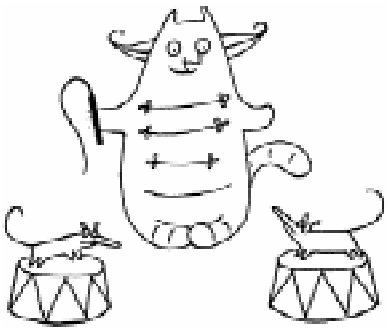
На рис. 4 показан апплет, содержащий три кнопки для управления рисованием фигуры.

Рис. 4. Апплет с тремя кнопками, управляющими рисованием геометрических фигур

ОБМЕН ИЗОБРАЖЕНИЙ

Реализуем апплет, содержащий два изображения и кнопку. При нажатии на кнопку изображения меняются местами. Как и в предыдущем примере, описываемый класс **demButton4** реализует интерфейс **ActionListener**, содержащий метод **actionPerformed**. При реализации метода проверяем, что источником события является кнопка с надписью «Обмен». С использованием переменной **cur** типа **Image**, изображения меняются местами. Повторное нажатие на кнопку приведет к обмену существующих изображений и т. д. В листинге 5 содержится текст программы, реализующей обмен двух изображений.

На рис. 5 показан апплет, содержащий кнопку и два изображения, которые по кнопке меняются местами.



При нажатии на кнопку изображения меняются местами.

СКРЫТИЕ И ОТОБРАЖЕНИЕ РИСУНКА

Реализуем апплет, содержащий кнопку. Первоначально название кнопки «Отобразить». При нажатии на кнопку в окне апплета появляется изображение, а название кнопки меняется, новое название – «Скрыть». При нажатии на кнопку «Скрыть», изображение исчезает из окна апплета, название кнопки изменяется.

Изменение названия кнопки происходит с помощью метода **setLabel**, параметром которого является строка, соответствующая новому названию. Перемен-

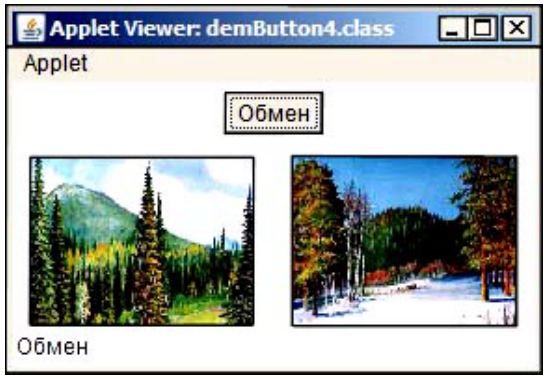


Рис. 5. Апплет с кнопкой, переставляющей два изображения

Листинг 4. Рисование геометрических фигур

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class demButton3_graf  extends Applet implements ActionListener  {
    String msg = '';// сообщение о нажатой кнопке
    Button b1, b2, b3; // кнопки
    int k; // вариант выбора для рисования фигуры
    int r=40;
    int w; // ширина объекта
     int h; // высота объекта
    public void init() {
     w = this.getWidth();
     h = this.getHeight();
// создание объектов- кнопок
        b1 = new Button('Квадрат');
        b2 = new Button('Круг');
        b3 = new Button('Треугольник');
// добавление объектов в апплет
        add(b1);
        add(b2);
        add(b3);
// регистрация событий от кнопок
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
    }
// обработка события щечка кнопкой мыши по кнопке
    public void actionPerformed(ActionEvent ae) {
        String str = ae.getActionCommand();
        if(str.equals('Квадрат')) {
            msg = 'Вы нажали на кнопку Квадрат';  k=1;}
        else if(str.equals('Круг')) {
            msg = 'Вы нажали на кнопку Круг';  k=2;}
        else {
            msg = 'Вы нажали на кнопку Треугольник'; k=3;}
        showStatus(msg);
        repaint();
    }
  // выдача результата
        public void paint(Graphics g) {
            g.drawString(msg, 3, 50);
            g.setColor(Color.pink);
            switch (k) {
              case 1: g.fillRect(w/2-r, h/2-r, 2*r, 2*r); break;
              case 2: g.fillOval(w/2-r, h/2-r, 2*r, 2*r); break;
              case 3: int xnode []={w/2,w-r,r,w/2};
                      int ynode[]={r+15,h-r,h-r,r+15};
                      g.fillPolygon(xnode,ynode,xnode.length);
             }
         }
}
```

Листинг 5. Обмен двух изображений

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
//Апплет меняет местами два изображения
public class demButton4 extends Applet implements ActionListener{
 Image im1, im2;
 Button b;
 int w, h;
//начальные установки
public void init(){
     b = new Button('Обмен');
     im1 = getImage(getCodeBase(),'images/g1.jpg');
     im2 = getImage(getCodeBase(),'images/g2.jpg');
       w=im1.getWidth(this)+250;
       h=im1.getHeight(this)+200;
       w=250;
       h=200;
  // размещение кнопки
     add(b);
     b.addActionListener(this);
 }
//обработка события щелчка по кнопке
public void actionPerformed(ActionEvent ae) {
 String str = ae.getActionCommand();
  if(str.equals('Обмен')) {
     Image cur = im1;
       im1 = im2;
       im2 = cur;
       repaint();
     }
   }
//вывод изображения
public void paint (Graphics g){
     g.drawImage(im1,10, 50, w,h, this);
    g.drawImage(im2,30+w, 50, w,h, this);
     }// paint
}//demButton4
```

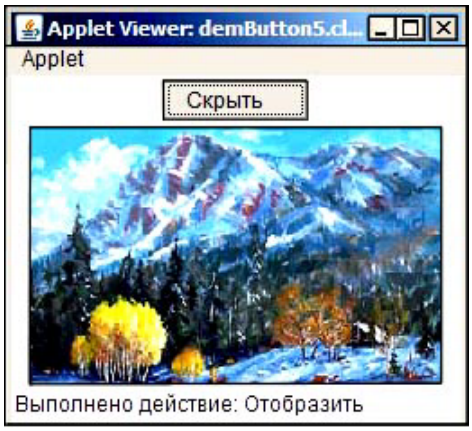


Рис. 6. Апплет с кнопкой, управляющей отображением и скрытием изображения

ная **cur** типа **Image** принимает значение **im**, когда изображение требуется поместить в окно апплета, и значение **null** в случае, когда изображение требуется скрыть. Как и в предыдущем примере, описываемый класс **demButton5** реализует интерфейс **ActionListener**. В строке статуса помещается информация о выполненном действии. В листинге 6 приведен текст программы, в которой с помощью кнопки происходит управление появлением и исчезновением рисунка на странице.

На рис. 6 показан апплет, содержащий кнопку и изображение.

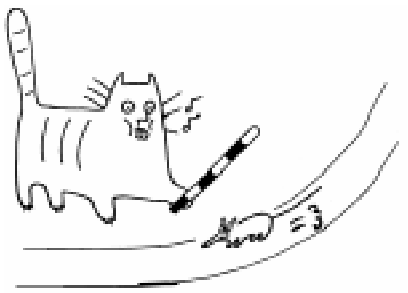
ДВИЖЕНИЕ ПО ОКРУЖНОСТИ С ОСТАНОВКОЙ И ПРОДОЛЖЕНИЕМ

Напишем программу, создающую анимационный эффект, основанный на перемещении изображения по кругу против часовой стрелки. Эффект анимации будет достигаться использованием потока runner, который описывается и создается с помощью конструктора. Логическая переменная **isRunning** будет следить за состоянием потока. В начале просмотра страницы переменной **isRunning** присваивается значение **true**, и с помощью метода **start()** запускается поток **runner**. Метод **start()** сообщает потоку о том, что следует запустить метод **run()**.

В методе **run()** приостанавливается текущий поток на 300 миллисекунд. Для приостановки текущего процесса на указанный промежуток времени использует-

Листинг 6. Скрытие и отображение рисунка

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
// Изображение скрывается и отображается по щелчку по кнопке
public class demButton5 extends Applet implements ActionListener{
 Image im, cur;
 Button b;
 int w, h;
//начальные установки
public void init(){
     setSize(250,200);
     b = new Button('Отобразить');
     im = getImage(getCodeBase(),'images/g3.jpg');
     cur=null;
    w=im.getWidth(this)+230;
    h=im.getHeight(this)+200;
// размещение кнопки
     add(b);
     b.addActionListener(this);
 }
//обработка события щелчка по кнопке
public void actionPerformed(ActionEvent ae) {
 String str = ae.getActionCommand();
  if(str.equals('Скрыть    ')) {
       cur = null;
       b.setLabel('Отобразить');
     }
  else
       if(str.equals('Отобразить')) {
            cur = im;
            b.setLabel('Скрыть    ');
       }
  showStatus('Выполнено действие: '+str);
  repaint();
   }
//вывод изображения
public void paint (Graphics g){
     g.drawImage(cur,10, 50, w,h, this);
     }// paint
}//demButton5
```



Кнопка с названием «Остановить» приостановит движение по кругу...

ся метод **sleep()**, который может завершиться исключительной ситуацией **InterruptedException**, если «заснувший» поток будет «разбужен». Анимация будет приостановлена, если произошло скрытие страницы с апплетом, в этом случае логическая переменная **isRunning** получит значение **false**.

На странице расположены две кнопки. Кнопка с названием «Остановить» приостановит движение по кругу, для продолжения движения следует нажать на кнопку «Продолжить». Какая кнопка была нажата, можно определить по значению логической переменной **p**. В программе используются два изображения, одно в качестве фонового, второе для перемещения по кругу. Траектория движения – окружность также нарисована в окне апплета. В листинге 7 представлен текст программы, использующей поток для получения анимационного эффекта движения по окружности с остановкой и продолжением.

На рис. 7 показан апплет, содержащий две кнопки и изображение коня.

Листинг 7. Движение по окружности с остановкой и продолжением

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class demButtonCircle extends Applet
        implements ActionListener, Runnable{
  Button b1,b2;
  Image pict, background;
  Thread runner = new Thread(this);
// Используется для остановки цикла в методе run
    boolean isRunning;
// переменные для организации движения по кругу
    double angle = 360;
    double r = 100; // радиус окружности
    double xbegin; // координата центра по x
    double ybegin; // координта центра по y
    boolean p = false;
// инициализация
    public void init() {
     resize(300,300);
     b1 = new Button('Продолжить');
     b2 = new Button('Остановить');
     xbegin = getWidth()/2-20;
     ybegin = getHeight()/2-30;
     background = getImage(getDocumentBase(), '../img/oblako01.jpg');
        pict = getImage(getDocumentBase(), '../img/wal.gif');
        add(b1);
        add(b2);
        b1.addActionListener(this);
        b2.addActionListener(this);
      }
```

```
// начало просмотра  страницы с апплетом
     public void start() {
        // запуск анимации
       isRunning = true;
     }
// скрытие страницы с апплетом, например, при переключении
// на другое окно и т.п.
    public void stop() {
        // остановка анимации
       isRunning = false;
   }
// обработка события
    public void actionPerformed(ActionEvent ae) {
        String str = ae.getActionCommand();
        if(str.equals('Продолжить')) {
     // запуск анимации
      isRunning = true;
      p = false;
      showStatus('Движение по кругу ');
        }
        else if(str.equals('Остановить')) {
       isRunning = false;
       p = true;
       showStatus('Движение на месте ');
        }
        repaint();
     }
  // анимация
    public void run() {
        try {
            while (isRunning) {
                Thread.sleep(300);
             }
        } catch(InterruptedException e) {}
   }
    public void update(Graphics g) {
        paint(g);
    }
//рисование
    public void paint(Graphics g) {
      g.drawImage(background, 0, 0, getWidth(), getHeight(), this);
      g.drawOval((int)xbegin/2, (int)ybegin/2, 2*(int)r, 2*(int)r);
      double rad = (angle-270) * Math.PI/180;
      double xcur = (int)Math.round(xbegin + r * Math.sin(rad));
      double ycur = (int)Math.round(ybegin + r * Math.cos(rad));
// поворот на один градус, если не стоим на месте
         if (!p){  angle += 1 ;
              if (angle >= 360) angle = 0;
      }
        g.drawImage(pict, (int)xcur, (int)ycur, 60, 60, this);
     }
 }
```

Рис. 7. Апплет с кнопками, управляющими движением рисунка по кругу против часовой стрелки

УПРАЖНЕНИЯ

Создайте апплеты, реализующие следующие анимационные эффекты:

1. В апплете одно изображение и две кнопки: «Вперед» и «Назад». При нажатии на кнопку «Вперед» появляется следующее изображение, на кнопку «Назад» – предыдущее.

2. В апплете четыре изображения, при нажатии на кнопку «Сдвинуть» происходит циклический сдвиг изображений слева направо.

3. В апплете изображение и две кнопки. При нажатии на одну из кнопок происходит увеличение изображения, при нажатии на другую кнопку – уменьшение изображения.

4. В апплете циферблат часов и кнопка. При нажатии на кнопку положение стрелок сдвигается на 5 минут.

5. В апплете реализуется движение изображения по часовой стрелке.

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