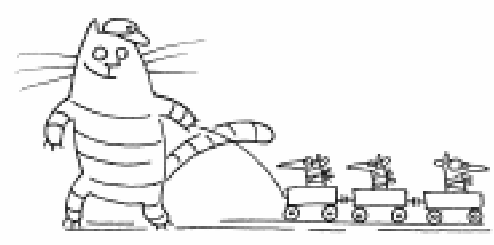
Дмитриева Марина Валерьевна

ОСНОВЫ ПРОГРАММИРОВАНИЯ ДЛЯ ИНТЕРНЕТ. АППЛЕТЫ. ЗАНЯТИЕ 5. МЕНЕДЖЕР РАЗМЕЩЕНИЯ. ПАНЕЛИ И ХОЛСТЫ

Рассматриваются классы, реализующие некоторые общие схемы размещения компонентов внутри контейнера. Приводятся примеры использования панелей и холстов.

Для управления размещением элементов внутри объектов-контейнеров, к которым относятся и апплеты, в языке Java существует специальный тип объектов **`LayoutManager`**. *Менеджер размещения* (или компоновки) – это экземпляр некоторого класса, который реализует интерфейс **`LayoutManager`**, содержащий методы для управления компоновкой элементов внутри контейнера. Реализация методов компоновки осуществляется внутри классов. В состав **`AWP`** входит несколько классов, реализующих некоторые общие схемы размещения. Это классы: **`FlowLayout`**, **`GridLayout`**, **`BorderLayout`**, **`CardLayout`**. Внутри апплета создается объект одного из перечисленных типов, называемый менеджером размещения, который и управляет размещением элементов.



...простой способ размещения – последовательное размещение объектов...

Менеджер размещения устанавливается методом **`setLayout()`**. Если метод не указан, то используется менеджер по умолчанию. Рассмотрим классы менеджеров размещения, которые предоставляет пакет **`java.awt`**.

ПОСЛЕДОВАТЕЛЬНОЕ РАЗМЕЩЕНИЕ ОБЪЕКТОВ (`FlowLayout`)

Класс **`FlowLayout`** реализует простой способ размещения – последовательное размещение объектов с указанным выравниванием и заданными между этими объектами интервалами. Компоненты располагаются, начиная с левого верхнего угла, слева направо и сверху вниз. Если очередной компонент в строке не помещается, то он располагается в следующей, новой строке с левой позиции. Компоненты отделяются друг от друга промежутками как по вертикали (сверху и снизу), так и по горизонтали (слева и справа). По умолчанию ширина промежутка 5 пикселей. Каждая строка с компонентами выравнивается по левому, правому краю или по

центру. По умолчанию строка выравнивается по центру.

В листинге 1 приведен текст программы, в которой при выполнении цикла создаются кнопки, затем кнопки размещаются последовательно в окне апплета. Название кнопок – целые числа от 1 до 15. Когда какая-либо из кнопок нажимается, ее название используется при формировании информационного текста (lab), в строку статуса также поступают данные о нажатой кнопке. В программе задано последовательное размещение компонентов с выравниванием их в строке по левому краю. Менеджер размещения определяется с помощью конструкции:

```
setLayout(new FlowLayout
                (FlowLayout.LEFT)).
```

Рис. 1. Последовательное размещение компонентов в окне апплета

Изображение апплета с последовательным размещением компонентов представлено на рис. 1.

Листинг 1. Последовательное размещение объектов

```
import java.awt.*;
import java.applet.*;
// последовательное размещение элементов
public class fl1 extends Applet {
    int n = 16;
    String s = "Нажата кнопка: ";
    Label lab = new Label(s+"?", Label.CENTER);
    Button b;
  public void init() {
    setLayout(new FlowLayout(FlowLayout.LEFT));
    setFont(new Font("Serif", Font.BOLD, 20));
    for (int i = 1; i < n; i++) {
     b = new Button();
     b.setLabel(""+i);
     add(b);
    }
    Color col= new Color(155,55,255);
    setForeground(col);
    lab.setFont(new Font("Serif", Font.ITALIC+Font.BOLD, 22));
    add(lab);
  }
// проверка, какая кнопка нажата
   public boolean action(Event e, Object obj) {
     if ((e.target instanceof Button))  {
       lab.setText(s+(String) obj);
       showStatus(s+ obj);
       return true;
      }
     return false;
   }
}
```

ТАБЛИЧНОЕ РАЗМЕЩЕНИЕ ОБЪЕКТОВ (GridLayout)

Класс **GridLayout** делит область контейнера на равные прямоугольники и размещает элементы по одному в каждом прямоугольнике, то есть создает табличное размещение объектов с заданным числом строк и столбцов, объекты располагаются последовательно слева направо. Конструктор этого класса позволяет задавать количество строк и столбцов.

В листинге 2 представлен текст программы, в которой менеджер **GridLayout** используется для создания сетки с четырьмя строками и четырьмя столбцами. При выполнении цикла в каждый из прямоугольников помещается кнопка, помеченная значениями от 15 до 1. В последний прямоугольник помещен информацион-

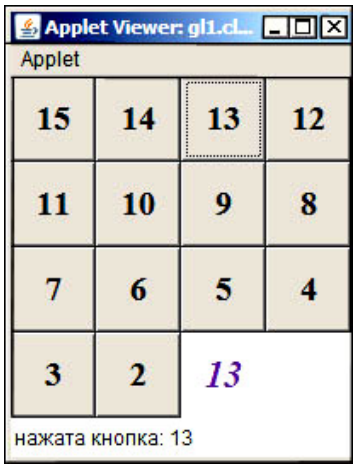
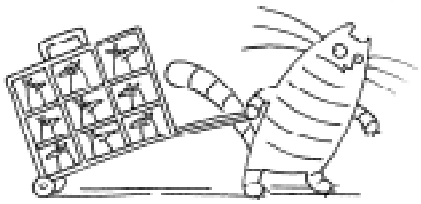


Рис. 2. Табличное размещение объектов в окне апплета



... делит область контейнера на равные прямоугольники и размещает элементы по одному в каждом прямоугольнике...

Листинг 2. Размещение объектов в таблице

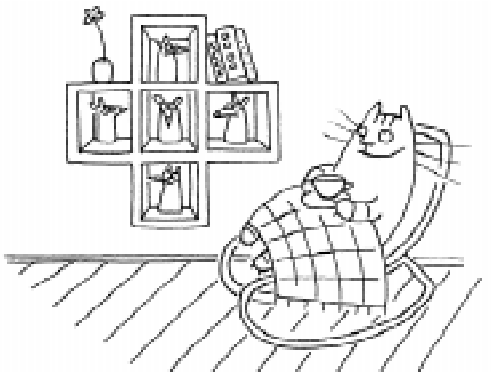
```
import java.awt.*;
import java.applet.*;
public class gl1 extends Applet {
    int n = 4;
    Label lab = new Label("?", Label.CENTER);
    public void init() {
// табличное размещение
    setLayout(new GridLayout(n,n));
    setFont(new Font("Serif", Font.BOLD, 20));
    for (int i = n*n-1; i > 1; i—) {
     add(new Button(""+i));
    }
    Color col= new Color(105,50,155);
    setForeground(col);
    lab.setFont(new Font("Serif", Font.ITALIC+Font.BOLD, 22));
    add(lab);
  }
// проверка, какая кнопка нажата
   public boolean action(Event e, Object obj) {
     if ((e.target instanceof Button))  {
       lab.setText((String) obj);
       showStatus("нажата кнопка: "+obj);
       return true;
      }
     return false;
   }
}
```

ный текст, который меняется в зависимости от нажатой кнопки. Данные о том, какая кнопка нажата, помещаются и в строку статуса.

Табличное размещение элементов показано на рис. 2.

РАЗМЕЩЕНИЕ ОБЪЕКТОВ ПО КРАЮ КОНТЕЙНЕРА (BorderLayout)

Менеджер **BorderLayout** старается поместить элемент в одну из пяти именованных областей контейнера: **NORTH (СЕВЕР)**, **SOUTH (ЮГ)**, **EAST (ВОСТОК)**, **WEST (ЗАПАД)**, **CENTER (ЦЕНТР)** в соответствии со значением аргумента в методе **add()**. В листинге 3 представлена программа, в которой компоненты размещаются с помощью менеджера **BorderLayout**. По краям области размещаются кнопки с названием «Север», «Юг», «Запад», «Восток». В центре области располагается информационный текст, который меняется в зависимости от действий пользователя. При нажатии на кнопку ее название появляется в центре области.



...старается поместить элемент в одну из пяти именованных областей контейнера...

В листинге 3 приведена программа с размещением компонентов по краю области в центре.

Листинг 3. Размещение по краю контейнера

```
import java.awt.event.*;
import java.applet.*;
import java.awt.*;
//размещение по краям
public class bl1 extends Applet {
  Color C=new Color (100, 60, 40);
  Color B=new Color (130, 200, 120);
  Label lab =new Label("Центр",Label.CENTER);
// инициализация и размещение
public void init() {
 resize(200,150);
 setFont(new Font("SansSerif", Font.BOLD+Font.ITALIC, 15));
 setForeground(C);
 setBackground (B);
 setLayout(new BorderLayout());
 add("North",new Button("Север"));
 add("South",new Button("Юг"));
 add("Center",lab);
 add("West",new Button("Запад"));
 add("East",new Button("Восток"));
 } //init
//обработка события щелчка по кнопке
public boolean action(Event e, Object obj) {
     if ((e.target instanceof Button))  {
       lab.setText((String) obj);
       showStatus("нажата кнопка: "+obj);
       return true;
    }
      return false;
  }
 }
```

Рис. 3. Размещение элементов по краям и в центре

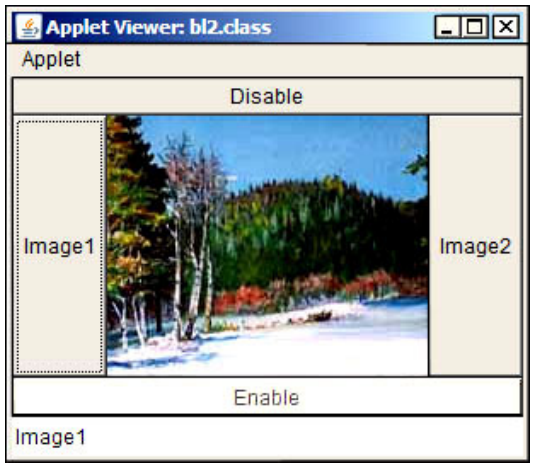


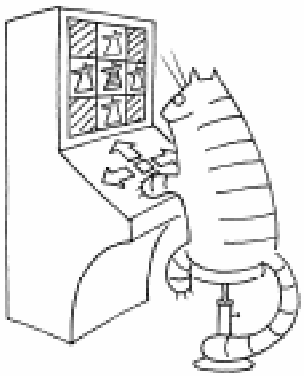
Рис. 4. Размещение кнопок по краям, изображения в центре

Размещение элементов по краям и в центре показано на рис. 3.

РАЗМЕЩЕНИЕ ИЗОБРАЖЕНИЯ В ЦЕНТРЕ АППЛЕТА

Создадим апплет, в котором по краям, как и в предыдущем случае, размещаются кнопки. Сначала три кнопки, кроме нижней, недоступны. Кнопка с именем «Enable» («South») после щелчка по ней разрешает доступ к остальным трем кнопкам, но сама становится недоступной.

Если кнопки доступны, при нажатии на левую кнопку с именем «Image1» («West») в центре апплета появляется изображение, при нажатии на кнопку с именем «Image2» («East») изображение в центральной области заменяется другим. Повторное нажатие кнопки «Image1» приведет к замене существующего изображения. Изображения в центральной части можно менять с помощью нажатия на соответствующую кнопку до тех пор, пока кнопки являются доступными. При нажатии на кнопку «Disable» («North») три остальные кнопки становятся недоступными. Чтобы кнопка **b1** стала недоступной, следует применить метод **b1.setEnabled(false)**.



...изображение в центральной области заменяется другим...

Кнопка вновь станет доступной, если применить метод **b1.setEnabled(true)**. В листинге 4 приведен текст программы, которая осуществляет описанные действия.

Размещение кнопок по краям с изображением в центре показано на рис. 4.

Листинг 4. Размещение кнопок по краю контейнера, а изображения по центру

```
import java.applet.*;
import java.awt.*;
// Менеджер размещения BorderLayout
public class bl2 extends java.applet.Applet{
 Button b1,b2,b3,b4;
 Image im1,im2,im;
// начальные установки
 public void init(){
```

```
  resize(300,200);
  im1=getImage(getCodeBase(),"images/g1.jpg");
  im2=getImage(getCodeBase(),"images/g2.jpg");
  setLayout(new BorderLayout());
  b1=new Button("Disable");
  b2=new Button("Image1");
  b3=new Button("Enable");
  b4=new Button("Image2");
  b1.setEnabled(false);
  b2.setEnabled(false);
  b4.setEnabled(false);
  add("West",b2);
  add("East",b4);
  add("North",b1);
  add("South",b3);
 }
// обработка события от кнопки
public boolean action(Event e, Object arg) {
     if (!(e.target instanceof Button))
          {return false;}
     else
      { String lab=(String)arg;
      showStatus(""+ (String)lab);
      if (lab=="Disable") {
        b2.setEnabled(false);
        b4.setEnabled(false);
        b1.setEnabled(false);
        b3.setEnabled(true);
        im = null;
        repaint();
        return true; }
       else{ if(lab=="Enable"){
              b2.setEnabled(true);
              b4.setEnabled(true);
              b1.setEnabled(true);
              b3.setEnabled(false);
              repaint();
              return true;}
             else {if(lab=="Image1"){
                      im=im1;
                    repaint();
                    return true;}
                    else { if(lab=="Image2") im =im2;  else  im=null;
                       repaint();
                      return true;}
                  }
                }
              }
            }//action
// рисование
   public void paint(Graphics g){
        g.drawImage(im,20,10,this);
    }//paint
}//
```

МЕНЕДЖЕР РАЗМЕЩЕНИЯ CardLayout

Класс **CardLayout** размещает элементы подобно карточкам в картотеке, когда в каждый момент только одна карточка является видимой. Иногда такой способ размещения компонентов сравнивают с колодой карт: колоду можно тасовать так, чтобы в каждый момент времени наверху была только одна карта. Класс **CardLayout** может быть полезен в том случае, когда при создании интерфейсов некоторые компоненты требуется либо скрыть, либо отобразить, в зависимости от действий пользователя.

В листинге 5 приведен текст программы, которая создает несколько кнопок, но менеджер **CardLayout** видимой делает только одну. Первую компоненту можно получить с помощью конструкции **card.first(this)**, каждую следующую с помощью метода **card.next(this)**, а к предыдущей вернуться – **card.previous(this)**.

«Карточное» расположение компонентов после выполнений действий пользователя показано на рис. 5.

КЛАСС PANEL

Класс **Panel** – подкласс класса **Container** и суперкласс для **Applet**. Когда экранный вывод направляется к апплету, он рисуется на поверхности объекта **Panel**, являющегося окном, которое не содержит области заголовка, строки меню и обрамления, поэтому, когда апплет выполняется в обозревателе, эти элементы не видны.

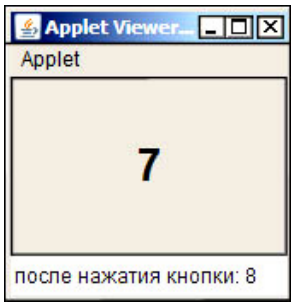


Рис. 5. Отображение одного компонента из нескольких

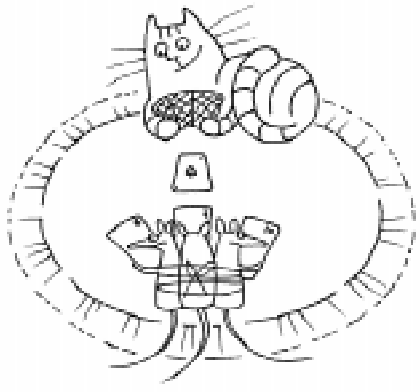
К **Panel**-объекту можно добавить другие компоненты с помощью метода **add()**. Как только элементы добавлены, их можно позиционировать и изменять размеры, используя методы **setLocation()**, **setSize()**, **setBounds()**.

ТРИ ПАНЕЛИ С РАЗНЫМИ СПОСОБАМИ РАЗМЕЩЕНИЯ КОМПОНЕНТОВ

В листинге 6 приведена программа, в которой описаны три панели. На каждой из панелей размещаются объекты с помощью различных менеджеров компоновки. После описания и создания с помощью конструктора панели с именем **p1 (Panel p1 = new Panel())** указывается, что элементы на ней будут размещаться последовательно: **p1.setLayout(new FlowLayout())**. Затем на панель добавляются с помощью метода **add** кнопки и информационная строка. Устанавливается цвет фона и цвет элементов первого плана. Далее описывается вторая панель, в которой элементы должны размещаться по краю, и, наконец, на третьей панели компоненты располагаются табличным способом. Для каждой из панелей устанавливается свой цвет фона и цвет элементов переднего края.

При щелчке по любой из кнопок указывается, какая кнопка была нажата и на какой панели она расположена.

Апплет с тремя панелями, на каждой из которых элементы расположены разными менеджерами компоновки, показан на рис. 6.



...такой способ размещения компонентов сравнивают с колодой карт...

Листинг 5. Размещение с отображением одного элемента

```
//демонстрирует CardLayout
import java.awt.event.*;
import java.awt.*;
import java.applet.*;
//создание кнопок и их  размещение
public class cl1 extends Applet {
 CardLayout card = new CardLayout();
   public void init(){
     setFont(new Font("SansSerif", Font.BOLD, 26));
// создание кнопок
     for(int i = 1; i < 16; i++){
      add(new Button("" + i));
     }
// менеджер размещения
   setLayout(card);
   card.first(this);
   card.next(this);
 //   card.previous(this);
  }//init
// проверка, какая кнопка нажата
   public boolean action(Event e, Object obj) {
     if ((e.target instanceof Button))  {
        String s = (String) obj;
        showStatus("после нажатия кнопки: "+ s);
//      card.next(this);
        card.previous(this);
       return true;
      }
     return false;
   }
}
```

ОТОБРАЖЕНИЕ ТОЛЬКО ОДНОЙ ПАНЕЛИ ИЗ ТРЕХ ОПИСАННЫХ

Создадим программу, в которой, как в предыдущей программе, описано три панели. На каждой из панелей элементы располагаются с помощью различных менеджеров компоновки. На первой панели – последовательное размещение компонентов, на второй – размещение компонентов по краю и в центре, на третьей панели применено размещение с помощью таблицы. А все эти панели размещаются в апплете таким образом, что видимой в каждый момент времени остается только одна панель. Для такого размещения описывается и создается объект **card (CardLayout card = new CardLayout())**.

Далее в программе создаются и заполняются три панели. «Карточное» размещение трех панелей осуществляется конст-

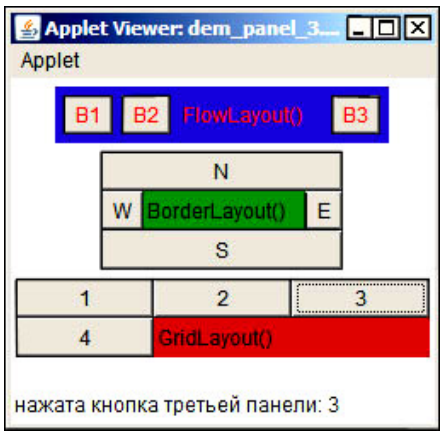


Рис. 6. Элементы на трех панелях

Листинг 6. Три панели с разными способами компоновки элементов

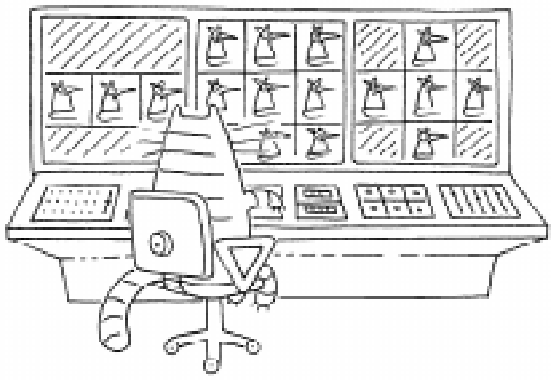
```
import java.awt.event.*;
import java.applet.*;
import java.awt.*;
// на трех панелях разные способы размещения
public class dem_panel_3 extends Applet {
 Color B1=new Color (66, 33, 200);
 Color B2=new Color (0, 150, 0);
 Color B3=new Color (200, 20, 0);
public void init() {
 resize(200,150);
 setForeground(Color.black);
 Panel p1 = new Panel();
  p1.setLayout(new FlowLayout());
       p1.add(new Button("B1"));
       p1.add(new Button("B2"));
       p1.add(new Label("FlowLayout()"));
       p1.add(new Button("B3"));
       p1.setBackground (B1);
       p1.setForeground(Color.red);
 add(p1);
 Panel p2 = new Panel();
  p2.setLayout(new BorderLayout());
       p2.add("North",new Button("N"));
       p2.add("South",new Button("S"));
       p2.add("Center",new Label("BorderLayout()"));
       p2.add("West",new Button("W"));
       p2.add("East",new Button("E"));
       p2.setBackground (B2);
 add(p2);
 Panel p3 = new Panel();
  p3.setLayout(new GridLayout(2,3));
       p3.add(new Button("1"));
       p3.add(new Button("2"));
       p3.add(new Button("3"));
       p3.add(new Button("4"));
       p3.add(new Label("GridLayout()"));
       p3.setBackground (B3);
 add(p3);
  } //init
//проверка нажатия кнопки
public boolean action(Event e, Object obj) {
     if (!(e.target instanceof Button))
                  return false;
     else { String s = (String) obj;
         if (s == "B1" || s == "B2" || s == "B3")
          showStatus("нажата кнопка первой панели: "+ s);
         else
          if (s == "N" || s == "S" || s == "W" || s == "E")
               showStatus("нажата кнопка второй панели: "+ s);
          else
               if (s == "1" || s == "2" || s == "3" || s == "4")
                    showStatus("нажата кнопка третьей панели: "+ s);

           return true;
         }
  } // action
}//
```

рукцией: **setLayout(card)**. При щелчке по любой из кнопок, расположенной на видимой панели, происходит переход к следующей панели, то есть интерфейс пользователя меняется. В листинге 7 представлен текст программы.

Листинг 7. Отображение одной панели из трех описаных

```
import java.applet.*;
import java.awt.*;
//размещение панелей
public class dem_panel_cl extends Applet {
  CardLayout card = new CardLayout();
// инициализация
public void init() {
  resize(200,150);
// первая панель: последовательное размещение элементов
 Panel p1=new Panel();
  p1.setLayout(new FlowLayout());
       p1.add(new Button("Button1"));
       p1.add(new Button("Button2"));
       p1.add(new Button("Button3"));
       p1.add(new Button("Button4"));
       p1.add(new Button("Button5"));
 add(p1);
// вторая панель: табличное размещение элементов
 Panel p2=new Panel();
  p2.setLayout(new BorderLayout());
       p2.add("North",new Button("N"));
       p2.add("South",new Button("S"));
       p2.add("West", new Button("W"));
       p2.add("East", new Button("E"));
  add(p2);
// третья панель: размещение элементов по краю панели
 Panel p3=new Panel();
  p3.setLayout(new GridLayout(2,3));
       p3.add(new Button("Butt1"));
       p3.add(new Button("Butt2"));
       p3.add(new Button("Butt3"));
       p3.add(new Button("Butt4"));
       p3.add(new Button("Butt5"));
 add(p3);
// карточное размещение трех созданных панелей
  setLayout(card);
  } //init
//проверка нажатия кнопки
public boolean action(Event e, Object obj) {
     if (!(e.target instanceof Button))
          return false;
     else { String s = (String) obj;
            showStatus("после нажатия кнопки: "+ s);
            card.next(this);
//          card.previous(this);
            return true;
         }
  } // action
} //
```

Рис. 7. Карточное расположение трех панелей



На каждой из панелей размещаются объекты с помощью различных менеджеров компоновки.

Листинг 8. Холсты с нарисованными эллипсами

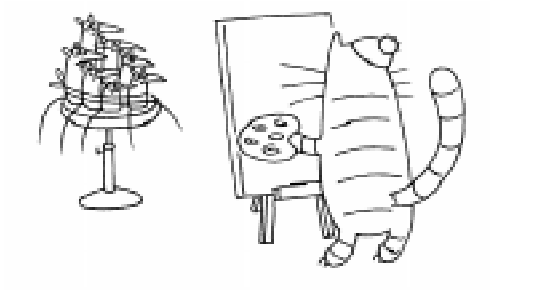
```
import java.awt.*;
import java.applet.*;
public class dem_canvas1 extends Applet {
class  colCanvas extends Canvas {
  Color col;
  colCanvas(float x) {
    col = new Color(x, x, x);
   }
// рисование на холсте
public void paint(Graphics g) {
  Dimension v = getSize();
  int w1 = v.width;
  int h1 = v.height;
  g.setColor(col);
  g.fillOval(0, 0, w1, h1);
  g.setColor(Color.black);
  g.drawRect(0, 0, w1-1, h1-1);
  showStatus("ширина холста: "+ v.width);
  }
}// colCanvas
// инициализация
public void init() {
  int n = 3;
 // размеры окна апплета
  int width = Integer.parseInt(getParameter("width"));
  int height = Integer.parseInt(getParameter("height"));
  int m = Math.min( width, height);
  int d = m/(n*2);
  float y= (float) n*n;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
         float x = (i * n + j) / y;
     Canvas can = new colCanvas(x);
     can.setSize(d,d);
     add(can);
     }
   }
  } // iniit
} // dem_canvas1
```

На рис. 7 показана только одна панель, так как к трем панелям применено карточное расположение.

ХОЛСТ (`Canvas`)

Класс **`Canvas`** можно трактовать как пустой холст, на котором можно «нарисовать» любой компонент в таком виде, как вы его представляете. Объекту **`Canvas`** можно придать желаемый внешний вид с помощью графических средств, предоставляемых пользователю.

Напишем программу, создающую несколько холстов, на каждом из которых нарисован прямоугольник с вписанным эллипсом. Класс **`colCanvas`** является подклассом класса **`Canvas`**, создает холст заданного цвета. Цвет передается конструктору при создании холста с помощью трех параметров, задающих интенсивность красного, зеленого и синего цветов. В листинге 8 текст программы, формирующей различные холсты с изображением фигур.



...пустой холст, на котором можно «нарисовать» любой компонент.

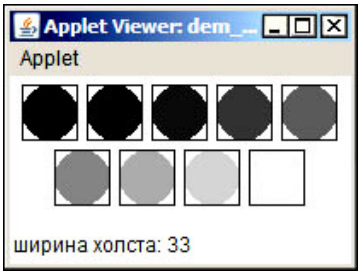


Рис. 8. Холсты с нарисованными овалами

На рис. 8 представлены холсты, на каждом из которых нарисован овал, цвета заливки овалов различны.

УПРАЖНЕНИЯ

1. Создайте апплет, моделирующий часть клавиш клавиатуры.
2. Создайте апплет, моделирующий интерфейс мобильного телефона.

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