Дмитриева Марина Валерьевна

ОСНОВЫ ПРОГРАММИРОВАНИЯ ДЛЯ ИНТЕРНЕТ. АППЛЕТЫ. ЗАНЯТИЕ 2. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС. ГРАФИЧЕСКИЕ МЕТОДЫ

В статье рассмотрены графические методы, позволяющие рисовать различные фигуры. Приведены Java-апплеты, демонстрирующие работу с графическими объектами.

Пакет AWT (Abstract Windowing Toolkit – оконный пользовательский интерфейс) поддерживает большой набор графических методов. Вся графика рисуется относительно окна. Это может быть главное или дочернее окно апплета или окно автономного приложения. В приведенных примерах графика рассматривается в главном окне апплета, однако та же техника применима к любому типу окна.

В Java принята система координат, используемая практически во всех графических системах. Начало координат каждого окна – в его верхнем левом углу и обозначается как (0,0), ось *Ox* идет вправо, ось *Oy* – вниз, координаты вычисляются в пикселях.

Весь вывод в окно выполняется через графический контекст. Графический контекст передаётся апплету, когда вызывается один из его методов. Метод (рисование) производит собственно вывод изображения в отведенное место на экране, методу **`paint`** передается в качестве параметра объект **`Graphics`**, который апплет может использовать для рисования различных фигур. Метод **`repaint`** (перерисовка) вызывается в тот момент, когда апплет должен заново вывести свое изображение на экран. Метод **`update`** (обновление) вызывается методом **`repaint`** для обновления изображения.

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 чёрным. Когда графический объект превышает размеры окна, вывод автоматически усекается.

РИСОВАНИЕ ЛИНИЙ

Простейшая геометрическая фигура, которую можно изобразить на экране с использованием объекта **`Graphics`** – прямая линия. Метод **`drawLine(int startX, int startY, int endX, int endY)`** рисует на экране отрезок, соединяя точки с координатами **`(startX,startY)`** и **`(endX,endY)`**.

РАССТОЯНИЕ МЕЖДУ ТОЧКАМИ

Создадим апплет, работа с которым осуществляется следующим образом: пер-

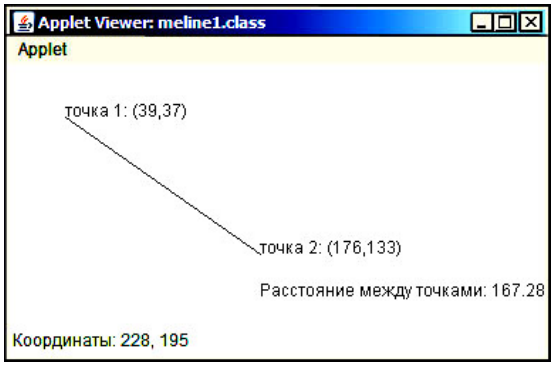
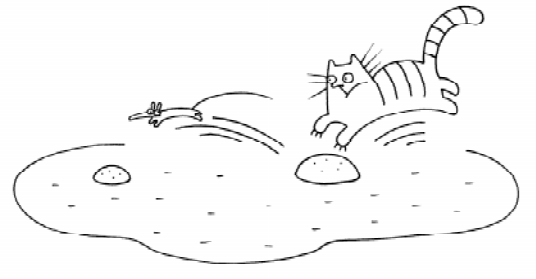


Рис. 1. Расстояние между двумя точками

вый щелчок мыши фиксирует первую точку на экране, выдаются координаты этой точки, второй щелчок мышью фиксирует вторую точку, выдаются координаты второй точки и расстояние между двумя точками (в пикселях). При следующем щелчке мышью процесс повторяется сначала: фиксируется первая точка и т. д.

Для обработки событий от мыши реализуются интерфейсы **MouseListener** и **MouseMotionListener**. Апплет является как источником, так и слушателем событий от мыши. Переменная **k** фиксирует состояние, какой точке соответствует щелчок мыши. В зависимости от этого формируется либо сообщение **msg1**, либо **msg2** с указанием координат текущей точки. При вычислении расстояния между двумя точками используется метод **sqrt** класса **Math**, содержащего набор методов математических функций. В листинге 1 текст программы, рисующий отрезок прямой с координатами начала и конца и расстоянием между точками.



Расстояние между точками

В тексте программы используется условный оператор, который имеет вид:

if (B) S или **if (B) S else S1**

где **B** – выражение типа **Boolean**, **S** и **S1** – операторы. Выражение логического типа может быть получено в результате применения так называемых операций отношения. К ним относятся операции меньше (<), меньше или равно (<=), больше (>), больше либо равно (>=), равно (==), не равно (!=). Следует обратить внимание на операцию равно, часто вместо нее применяют просто знак равенства, забывая, что знак равенства в современных языках используется в операторах присваивания.

В языке Java, как во многих других языках, определены операции над логическими значениями. Логическое отрицание обозначается восклицательным знаком (!), логическое И – (&&) и логическое ИЛИ – (||).

В листинге 1 приведен текст программы, решающей задачу.

В строке статуса фиксируются текущие координаты курсора мыши. Изображение апплета представлено на рис. 1.

Листинг 1. Расстояние между двумя точками

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class meline1 extends Applet
       implements   MouseMotionListener, MouseListener {
// сообщения
          String msg1 ="";
          String msg2 ="";
// текущие координаты мыши
          int curX = 0,
              curY = 0;
```

```
// переключатель для точек
        int k=0;
// координаты двух точек
        int x1,y1,x2,y2;
// расстояние
        double r;
// блок прослушивания событий от мыши - апплет
        public void init() {
          addMouseMotionListener(this);
          addMouseListener(this);
     }
 // Обработка события щелчка мыши
        public void mouseClicked(MouseEvent e) {
             curX = e.getX();
             curY = e.getY();
             if (k==0)k=1;
             if (k==1)
               {x1= curX; y1= curY;
                msg1 = "точка 1: ("+x1+","+ y1+")";
               }
             else
               if (k==2)
                 {x2 = curX; y2 = curY;
                  msg2 = "точка 2: ("+x2+","+ y2+")";
                  }
             repaint();
        }
 // другие события от мыши
        public void mouseEntered(MouseEvent e) {}
        public void mouseExited(MouseEvent e) {}
        public void mousePressed(MouseEvent e) {}
        public void mouseReleased(MouseEvent e) {}
        public void mouseDragged(MouseEvent e) {}
 // обработка события перемещения мыши
        public void mouseMoved(MouseEvent e) {
 //отобразить текущие координаты строке  статуса
        showStatus("Координаты: " +  e.getX() + ", " +  e.getY());
        }
 // вывод сообщений о координатах точек и расстоянии между ними
        public void paint (Graphics g) {
             if (k==1)
             {g.drawString(msg1, x1, y1); k=2;}
             else
              if (k==2)
             {  g.drawString(msg1, x1, y1);
                g.drawString(msg2, x2, y2);
                g.drawLine(x1, y1, x2, y2);
                r = Math.sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
                g.drawString("Расстояние между точками: "+r, x2, y2+30);
                k=1;
             }
        } // paint
} // meline1
```

РИСОВАНИЕ ПРЯМОУГОЛЬНИКОВ

Рассмотрим методы, позволяющие нарисовать прямоугольник со сторонами, параллельными сторонам экрана. Язык Java обеспечивает несколько различных способов рисования прямоугольников. Метод **drawRect()** рисует контур прямоугольника, если требуется нарисовать закрашенный прямоугольник, то следует применить метод **fillRect()**. Оба эти методы имеют четыре параметра, первые два параметра задают координаты левого верхнего угла, третий параметр задает ширину, а четвертый – высоту прямоугольника.

ПРЯМОУГОЛЬНИК С ЗАКРУГЛЕННЫМИ УГЛАМИ

Кроме обычных прямоугольников можно рисовать прямоугольники с закругленными углами. Чтобы нарисовать округлённый прямоугольник, используется метод **drawRoundRect()** или **fillRoundRect()**, при их использовании, кроме четырех параметров, требуется задать еще два, пятый параметр определяет диаметр округляющей дуги по оси *x* и шестой параметр – диаметр округляющей дуги по оси *y*.

В листинге 2 приведен текст программы, которая рисует прямоугольник заданного размера. Можно изменять размеры прямоугольника с помощью клавиш. Нажатие на клавишу «M» увеличивает размер прямоугольника, на клавишу «L» – уменьшает прямоугольник. С помощью клавиши «C» можно рисовать только контур прямоугольника, либо рисовать закрашенный прямоугольник. И, наконец, с помощью клавиши «R» рисуется либо обычный прямоугольник, либо прямоугольник с закругленными краями. Вид прямоугольника и закраска в программе определяются с помощью логических значений.

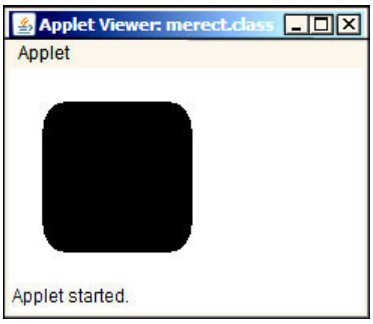


Рис. 2. Закрашенный прямоугольник с закругленными углами

Организовать ветвление в программе можно не только с помощью условного оператора, но и оператора **switch**. Оператор **switch** имеет следующий синтаксис:

```
switch (B)
  case e1: S1;
  case e2: S2;
...
  case en: Sn;
defolt: S;
```

где **B** – выражение целого типа или типа **char**, **e1**, **e2**, ..., **en** – константы, **S1**, **S2**, ..., **Sn**, **S** – операторы.

Выполнение оператора происходит следующим образом. Вычисляется значение выражения **B**. Если значение выражения равно **ei**, то выполнение передается оператору **Si**. Если значение выражения **B** не совпало ни с одним значением **ei** (**i=1,2,…,n**), то управление передается оператору **S**. Часть, начинающаяся с **defolt**, может отсутствовать, управление будет передано оператору, следующему за оператором **switch**.

Оператор **break** прерывает выполнение текущего оператора и передает управление следующему оператору.

Обратите внимание на использование оператора **switch** в тексте программы. После каждой из альтернатив используется оператор **break**, если его не поставить, то произойдет так называемое «проваливание», то есть продолжит выполнение оператор следующей альтернативы.

На рис. 2 закрашенный прямоугольник с закругленными углами и текущими размерами.

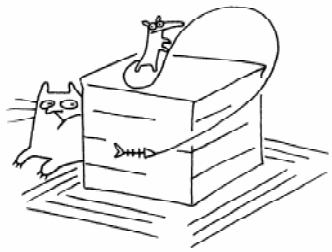
ПРЯМОУГОЛЬНИК С ОБЪЕМНЫМ ЭФФЕКТОМ

Кроме простых прямоугольников, класс **Graphics** предоставляет методы для создания прямоугольников с объемным эф-

Листинг 2. Управление размером и видом прямоугольника с помощью клавиш

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class merect  extends Applet
       implements  KeyListener  {
          int x = 20, // координаты левого верхнего
              y = 20, // угла прямоугольника
              w = 50, // ширина
              h = 60; // высота
          int dw = this.getWidth(),  //  предельные размеры
              dh = this.getHeight(), //  при увеличении
              d = 10; // минимальный размер
          boolean flag = false, // контур или закрашенный
                  flag1 = false;
// начальные действия
          public void init() {
               addKeyListener(this);
               dw = this.getWidth()/2;
                 dh = this.getHeight()/2;
     }
//нажатие клавиши на клавиатуре
      public void keyPressed (KeyEvent e) {
// от нажатой клавиши зависят размеры прямоугольника, закраска и вид
               switch ((char)e.getKeyCode()) {
                case 'L': w -= 10; h-=10; break;
                case 'M': w += 10; h+=10; break;
                case 'C': flag=!flag; break;
                case 'R': flag1=!flag1;
               }
// Случай, когда приблизились к границам
                if (w > dw)  w =  dw;
                if (w < d) w = d;
                if (h > dh - d) h = dh;
                if (h < d) h = d;
                    repaint();
      }
//другие события клавиатуры
      public void keyReleased (KeyEvent evt) {}
      public void keyTyped(KeyEvent evt) {}
// рисование прямоугольника
      public void paint (Graphics g) {
          if (flag1) {
            if (flag)
               g.fillRect(x+2, y+2, w, h);
            else
               g.drawRect(x+2, y+2, w, h);
          }
          else {
               if (flag)
                    g.fillRoundRect(x+2, y+2, w, h, 30, 40);
                 else
                    g.drawRoundRect(x+2, y+2, w, h, 30, 40);
          }
      } // paint
} // merect
```

фектом, то есть с тенями, которые выглядят приподнятыми или вдавленными, в зависимости от значения одного из передаваемых параметров. Чтобы создать прямоугольник с тенями, используют один из методов: **draw3DRect (x, y, w, h, raised)** или **fill3DRect (x, y, w, h, raised)**. Целые значения **x** и **y** являются координатами левого верхнего угла, целые значения **w**, **h** задают ширину и высоту прямоугольника,



...методы для создания прямоугольников с объемным эффектом...

Листинг 3. Прямоугольники с объемным эффектом

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class merect2  extends Applet
       implements    MouseListener {
// текущие координаты мыши
       int curX = 0,
           curY = 0;
// координаты левого верхнего угла прямоугольника и размеры
       int x1=10,y1=10,w1=100,h1=110,
           x2=120,y2=10,w2=100,h2=110;
// эффект объемности по состоянию переменных flag1 и flag2
       boolean flag1 = false,
               flag2 = false;
// слушиватель событий от мыши - апплет
       public void init() {
            addMouseListener(this);
     }
 // обработка события щелчка мыши
       public void mouseClicked(MouseEvent e) {
          curX = e.getX();
          curY = e.getY();
          flag1 =  !(curX >= x1 && curX <= x1+w1 && curY >= y1 && curY <= y1+h1);
          flag2 =  !(curX >= x2 && curX <= x2+w2 && curY >= y2 && curY <= y2+h2);
          repaint();
       }
// другие, не используемые методы интерфейса MouseListener
         public void mouseEntered(MouseEvent e) {}
         public void mouseExited(MouseEvent e) {}
         public void mousePressed(MouseEvent e) {}
         public void mouseReleased(MouseEvent e) {}
// рисование прямоугольников с тенью и без объемного эффекта
         public void paint (Graphics g) {
              int w=getSize().width;
              int h=getSize().height;
              g.setColor(Color.lightGray);
              g.fill3DRect(0,0,w-1,h-1,true);
              g.setColor(Color.pink);
              g.fill3DRect(10,10,w1,h1,flag1);
              g.fill3DRect(120,10,w2,h2,flag2);
         } // paint
} // merect2
```

логическое значение **raised** указывает на то, является прямоугольник приподнятым или вдавленным. Если значение параметра **raised** равно **true**, то прямоугольник приподнят, если **false**, то вдавлен.

Создадим апплет (листинг 3), в котором нарисованы два прямоугольника. Видом прямоугольников будем управлять с помощью курсора мыши. При щелчке по прямоугольнику он становится вдавленным, при щелчке вне его прямоугольник становится приподнятым, то есть прямоугольник рисуется с эффектом тени.

Для того чтобы определить, находится ли точка с текущими координатами внутри левого прямоугольника, логической переменной **flag1** присваивается значение логического выражения:

```
flag1 =  !(curX >= x1 && curX <= x1+w1 && curY >= y1 && curY <= y1+h1);
```

При формировании выражения использовались операции отношения, логическое отрицание и логическое И.

На рис. 3 отражена ситуация, когда один из прямоугольников приподнят, а другой утоплен.

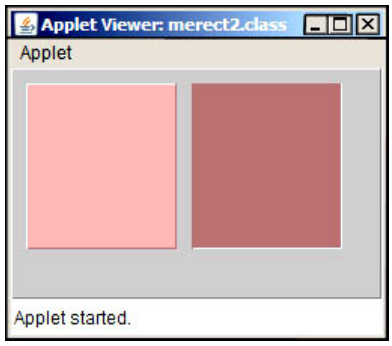


Рис. 3. Прямоугольники с эффектом объемности

РИСОВАНИЕ ЭЛЛИПСОВ И ОКРУЖНОСТЕЙ

Эллипс рисуется в пределах ограничительного прямоугольника, чей левый верхний угол определяется параметрами **top** и **left**, а ширина и высота указываются в **width** и **height**. Эллипс вписывается в прямоугольник. Для рисования эллипса используется **drawOval()**, а для его заполнения – **fillOval()**.

РИСОВАНИЕ ОКРУЖНОСТИ

Чтобы нарисовать круг, в качестве ограничительного прямоугольника нужно указать квадрат. Для этого достаточно задать одинаковые значения для ширины и высоты прямоугольника, в который вписывается эллипс.

Напишем программу, при выполнении которой при щелчке кнопкой мыши по области окна рисуется круг, центр которого зависит от текущих координат. Сразу после загрузки апплета круг располагается в центре области. В листинге 4 представлен текст программы.

УПРАВЛЕНИЕ ДИСКОМ С ПОМОЩЬЮ КЛАВИШ

Немного усложним задачу. Создадим апплет, позволяющий управлять движением диска, являющегося кругом, с помощью клавиш. Диск может перемещаться вниз, вверх, влево, вправо. В программе проверяется ситуация, когда диск приблизился к границе области. Создаваемый класс должен реализовать два интерфейса: методы работы с мышью и методы работы с клавиатурой.

Сразу после загрузки апплета диск расположен в центре области. Далее пользователь может изменить положение диска, щелкнув кнопкой мыши. Управлять движением диска можно с помощью клавиш. В строке статуса отражается на-

Для рисования эллипса используется drawOval()...

Листинг 4. Рисование круга по щелчку кнопкой мыши с текущих координат

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class medisk1 extends Applet
        implements  MouseListener {
//координаты центра диска и радиус
 int xC, yC, r;
 public void init(){
  addMouseListener(this);
  xC=this.getWidth()/2;
  yC=this.getHeight()/2;
  r=40;
}// init
//обработка событий от мыши
public void mouseClicked(MouseEvent e){
   xC=e.getX();
   yC=e.getY();
   repaint();
 }//mouseClicked
// другие методы интерфейса MouseListener
 public void mousePressed(MouseEvent e){}
 public void mouseReleased(MouseEvent e){}
 public void mouseEntered(MouseEvent e){}
 public void mouseExited(MouseEvent e){}
//рисование
public void paint(Graphics g) {
  g.setColor(Color.blue);
  g.fillOval(xC-r,yC-r,2*r,2*r);
  g.setColor(Color.red);
  g.drawOval(xC-r,yC-r,2*r,2*r);
 } // paint
}// medisk1
```

правление движения. В листинге 5 содержится исходный текст программы.

На рис. 4 показан диск после перемещений по экрану. Строка статуса содержит данные о последнем перемещении.

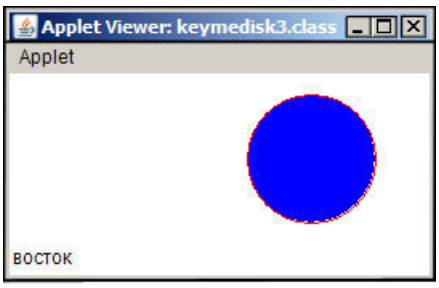


Рис. 4. Управление движением диска с помощью клавиш

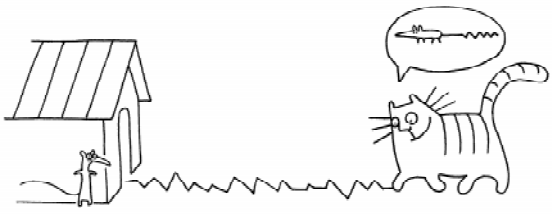
Создадим апплет, позволяющий управлять движением диска, являющегося кругом...

РИСОВАНИЕ МНОГОУГОЛЬНИКОВ

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

Создадим апплет, при загрузке которого на экране рисуется треугольник. При нажатии на клавишу клавиатуры «R» прямая, соединяющая одну из вершин треугольника с противоположной стороной, смещается вправо, при нажатии на клавишу «L» прямая смещается влево. Первоначально прямая линия совпадает с одной из сторон треугольника.

При построении треугольника использовался метод **drawPolygon**, первый и вто-



Метод drawPolygon рисует контур многоугольника (ломаную линию)...

Листинг 5. Управление диском с помощью клавиш

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class keymedisk3  extends Applet
       implements  KeyListener, MouseListener {
 String msg="";
 int d=30; // приближение к границе области
//координаты центра диска и радиус
 int xC, yC, r;
// установка начальных значений, диск в середине области
  public void init(){
    addKeyListener(this);
    addMouseListener(this);
    xC=this.getWidth()/2;
    yC=this.getHeight()/2;
    r=20;
}// init
//нажатие клавиши на клавиатуре
  public void keyPressed (KeyEvent e) {
// от нажатой клавиши зависит, в какую сторону двигаться
      switch ((char)e.getKeyCode()) {
         case 'W': xC -= 10; msg="запад";  break;
         case 'E': xC += 10; msg="восток"; break;
         case 'N': yC -= 10; msg="север"; break;
         case 'S': yC += 10; msg="юг"; break;
        }
// Случай, когда приблизились к границам
       if (xC > getWidth() - d)  xC = getWidth() - d;
       if (xC < d) xC = d;
       if (yC > getHeight() - d) yC = getHeight() - d;
       if (yC < d) yC = d;
       showStatus(msg);
        repaint();
     }
//другие методы интерфейса MouseListener
  public void keyReleased (KeyEvent evt) {}
  public void keyTyped(KeyEvent evt) {}
//обработка событий от мыши
  public void mouseClicked(MouseEvent e){
     xC=e.getX();
     yC=e.getY();
     repaint();
  }//mouseClicked
// методы обработки событий от мыши интерфейса MouseListener
      public void mousePressed(MouseEvent e){}
      public void mouseReleased(MouseEvent e){}
      public void mouseEntered(MouseEvent e){}
      public void mouseExited(MouseEvent e){}
//рисование в новом положении
  public void paint(Graphics g) {
    g.setColor(Color.blue);
    g.fillOval(xC-r,yC-r,2*r,2*r);
    g.setColor(Color.red);
    g.drawOval(xC-r,yC-r,2*r,2*r);
 } // paint
}// keymedisk3
```

рой параметры которого – массивы, содержащие координаты вершин треугольника, причем последние значения в массиве совпадают с первыми. В массиве четыре элемента. При нажатии на клавишу вычисляются новые координаты прямой, соединяющей вершину треугольника с основанием, и рисуется отрезок прямой.

В тексте программы будет использован массив. В массив объединяются элементы одного типа, которые для дальнейшей работы удобно перенумеровать и обращаться к отдельному элементу массива, используя так называемый индекс массива. Для обозначения массивов в Java используется конструкция: **T[]**, где **T** – тип элемента массива. Элементы массива нумеруются, начиная с нуля. Можно задать значения элементам массива при описании, как это сделано в тексте программы:

Листинг 6. Перемещение линии в треугольнике

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class ml2 extends Applet
                implements   KeyListener{
// координаты вершин треугольника
  int leftx=20;
  int lefty=220;
  int rightx=leftx+210;
  int righty=lefty;
  int upx=(leftx+rightx)/2;
  int upy=lefty-210;
// точка на основании треугольника
  int x = leftx;
  int y = lefty;
  int p = 10; // величина шага
  public void init(){
   addKeyListener(this);
   resize(300,300);
  }
  public void keyPressed (KeyEvent e) {
     switch ((char)e.getKeyCode()) {
       case 'R': if(x >= leftx && x < rightx){ x=x+p;};break;
       case 'L': if(x > leftx && x <= rightx){ x=x-p;};break;
   }
   repaint();
 }
// другие методы интерфейса KeyListener
  public void keyReleased (KeyEvent evt) {}
  public void keyTyped(KeyEvent evt) {}
// прорисовка
  public void paint(Graphics g){
  int xnode[]={leftx,rightx,upx,leftx};
  int ynode[]={lefty,righty,upy,lefty};
  g.setColor(Color.red);
  g.drawPolygon(xnode,ynode,xnode.length);
  g.setColor(Color.pink);
  g.fillPolygon(xnode,ynode,xnode.length);
  g.setColor(Color.blue);
  g.drawLine(x,y,upx,upy);
}//paint
}//ml2
```

```
int xnode[]={leftx,rightx,upx,leftx};
int ynode[]={lefty,righty,upy,lefty};
```

Далее в программе эти массивы координат будут использоваться для построения треугольника (листинг 6).

На рис. 5 представлен апплет, в котором после действий пользователя прямая линия, соединяющая вершину треугольника с основанием, переместилась вправо.

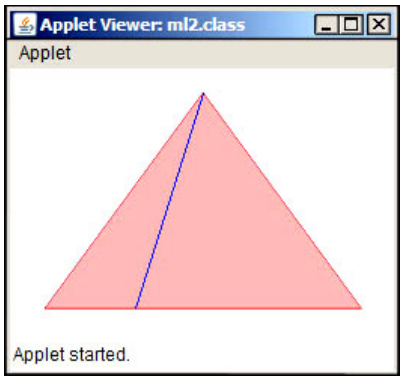


Рис. 5. Движение прямой линии внутри треугольника

РИСОВАНИЕ КОНТУРА МНОГОУГОЛЬНИКА

Напишем программу, которая строит контур многоугольника в результате действий пользователя. После щелчка кнопкой мыши по экрану добавляется отрезок прямой, соединяющий точку с текущими координатами с последней построенной вершиной.

Для очистки области рисования надо нажать на клавишу C. Далее можно начать рисовать новый многоугольник. Текст программы представлен в листинге 7.

Массивы можно создавать с помощью конструкторов, отличающихся от конструкторов других объектов. В конструкторе после ключевого слова **new** должен быть задан тип элементов и количество элементов, например

```
int x[]=new int[100];
int y[]=new int[100];
```

В массиве задано максимальное число элементов. Реальное число элементов определяется переменной **num**. После очередного щелчка кнопкой мыши к построенным элементам добавляется отрезок прямой, соединяющий две последние точки.

При рисовании ломаной используется оператор цикла. В языке Java определены циклы различных типов. Рассмотрим цикл, который используется в тексте программы. Синтаксис оператора цикла **for**:

```
for (инициализация; выражение; шаг)
оператор;
```

При выполнении цикла **for** сначала выполняются операторы инициализации, затем вычисляется выражение. Если выражение истинно, то выполняется оператор цикла, а затем третье выражение, обозначенное *шаг*.

Если значение выражения ложно, то выполнение цикла завершится. Рассмотрим оператор цикла, приведенный в тексте программы.

```
for(int i=0; i<num; ++i){
    g.drawLine(oldx, oldy, x[i], y[i]);
    oldx=x[i];
    oldy=y[i];
  }
```

В части инициализации описывается переменная целого типа **i**, которой присваивается значение 0. Если значение **i** меньше **num**, то выполняется оператор цикла, затем значение переменной **i** увеличивается на 1 (см. листинг 7).

На рис. 6 представлен апплет после действий пользователя. В строке статуса отражены координаты последней присоединенной к ломаной вершины.

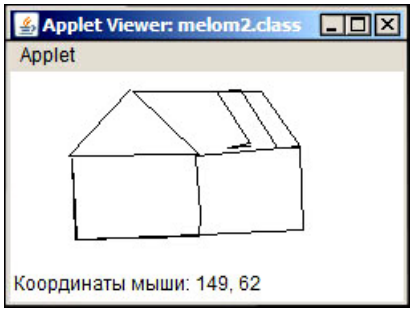


Рис. 6. Построение ломаной по щелчку кнопкой мыши

Листинг 7. Построение ломаной при щелчке кнопкой мыши

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class melom2 extends Applet
       implements   MouseListener, KeyListener{
// массив координат вершин ломаной
     int x[];
     int y[];
// число вершин
     int num;
     boolean cl=false;
// начальные действия
     public void init() {
       addMouseListener(this);
       addKeyListener(this);
       num = 0;
       x = new int[100];
       y = new int[100];
       resize(400,300);
     } // init
// нажатие на клавишу клавиатуры
      public void keyPressed (KeyEvent e) {
          cl = (char)e.getKeyCode()==’C’;
        repaint();
       }
//  другие методы интерфейса KeyListener
      public void keyReleased (KeyEvent evt) {}
      public void keyTyped(KeyEvent evt) {}
// Обработка события щелчок мыши
      public void mouseClicked(MouseEvent e) {
          if(num < 100) {
             x[num]=e.getX();
             y[num]=e.getY();
             ++num;
             showStatus("Координаты мыши: "+e.getX()+", "+e.getY());
             repaint();
           }
       }
// другие методы интерфейса MouseListener
       public void mouseEntered(MouseEvent e) {}
       public void mouseExited(MouseEvent e) {}
       public void mousePressed(MouseEvent e) {}
       public void mouseReleased(MouseEvent e) {}
// рисование
       public void paint (Graphics g) {
          if (!cl) {
            int oldx=x[0];
            int oldy=y[0];
            for(int i=0; i<num; ++i){
               g.drawLine(oldx, oldy, x[i], y[i]);
               oldx=x[i];
               oldy=y[i];
             }
          }
```

```
          else {
             g.drawRect(0, 0, getWidth(), getHeight());
             num = 0;
             cl=false;
            }
        } // paint
} // melom2
```

УПРАЖНЕНИЯ

1. С помощью примитивных геометрических фигур нарисовать светофор.
2. С помощью примитивных геометрических фигур нарисовать циферблат часов.
3. С помощью примитивных геометрических фигур нарисовать картинку, например, дом, дерево, солнышко, елку, светофор.
4. С помощью примитивных геометрических фигур изобразить генеалогическое дерево своей семьи.
5. Определите стратегию, согласно которой строится прямоугольник, определите периметр и площадь построенного прямоугольника.
6. Определите стратегию, согласно которой строится треугольник, определите периметр и площадь построенного треугольника.
7. Определите стратегию, согласно которой строится окружность, определите длину окружности и площадь круга.
8. Определите стратегию, согласно которой строится эллипс, предусмотрите изменение цвета заливки эллипса с помощью клавиш клавиатуры.

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