Библиотека Swing

Swing – это набор инструментов, предназначенный для создания графических пользовательских интерфейсов GUI (в том числе для Web).

В начале существования java классов Swing не было. Причиной разработки классов Swing было слабое место в исходной подсистеме GUI AWT.

AWT определяет базовый набор элементов управления, окон и диалогов, которые поддерживают пригодный к использованию, но ограниченный в возможностях графический интерфейс. Одной из причин ограниченности AWT является платформенно-ориентированная поддержка визуальных компонентов. Внешний вид этих компонентов определяется используемой платформой, а не закладывается в java. Такие компоненты AWT называются ***тяжеловесными***.

В основном компоненты Swing являются ***легковесными.*** Они написаны на java и не зависят от средств платформы, на которой выполняется программа.

Swing поддерживает принцип ***подключаемого внешнего вида.*** Поскольку каждый компонент Swing визуализируется кодом java, внешний вид компонента находится под контролем Swing.

Swing поддерживает настраиваемые стили. Изначально доступны три стиля: **metal, Windows** и **Motif**.

Классы Swing построены на основе AWT. Swing использует тот же механизм обработки событий, что и AWT.

Подключение библиотеки:

import javax.swing.*;

Архитектура «Модель – представление – контроллер»

В общем случае визуальный компонент определяется тремя отдельными аспектами:

- как компонент выглядит во время его визуализации на экране;
- как компонент взаимодействует с пользователем;
- информацией о состоянии компонента.

Модель соответствует информации о состоянии, связанной с компонентом. Например, в случае флажка модель содержит поле, которое показывает, отмечен ли флажок.

Представление определяет, как компонент отображается на экране, включая любые аспекты представления, зависящие от текущего состояния модели.

Контроллер определяет, как компонент будет реагировать на действия пользователя. Например, когда пользователь щелкает на флажке, контроллер реагирует изменением модели, чтобы отразить выбор пользователя (отметка флажка или снятие отметки).

Подход, применяемый в Swing, называется архитектурой **«модель-делегат».** Большинство компонентов Swing содержат два объекта. Один из них представляет модель, а другой делегата пользовательского интерфейса.

Графический пользовательский интерфейс Swing состоит из двух ключевых элементов: **компонентов и контейнеров**. Все контейнеры тоже являются компонентами.

Различие: компонент представляет собой независимый визуальный элемент управления вроде кнопки, а контейнер вмещает группу компонентов. Таким образом, контейнер является особым типом компонента и предназначен для содержания других компонентов. Более того, чтобы можно было отобразить компонент, он должен находиться внутри контейнера. Во всех пользовательских графических интерфейсах Swing имеется как минимум один контейнер. Поскольку контейнеры являются компонентами, контейнер может содержать другие контейнеры.

Компоненты Swing происходят от класса **JComponent**. JComponent наследует AWT - классы Container и Component.

JApplet JButton JFrame JLabel JTextField JWindow

В Swing определены два типа контейнеров:

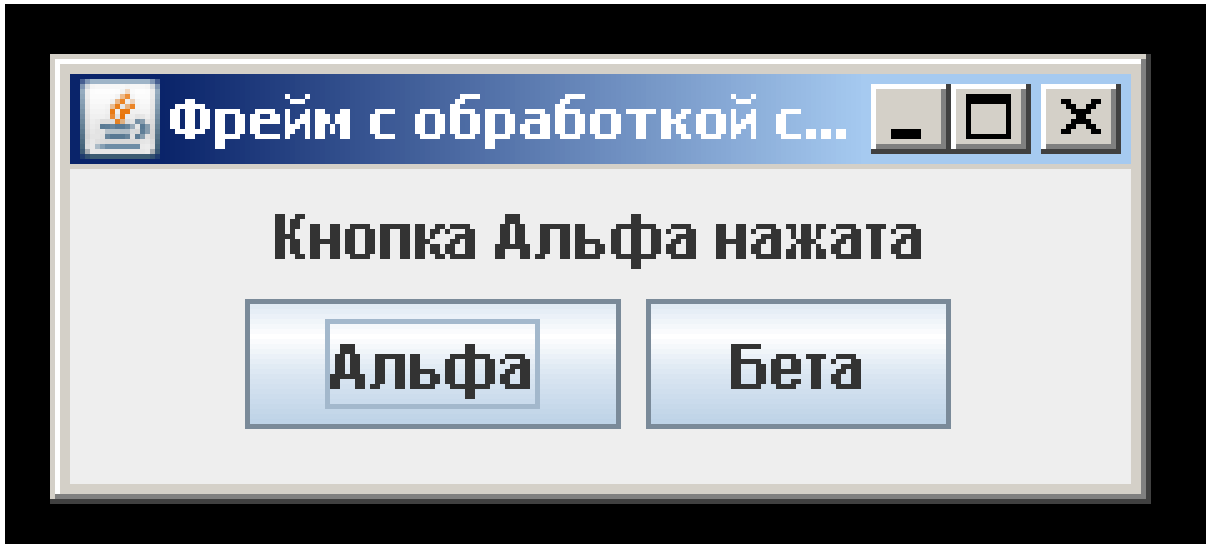
1. Контейнеры верхнего уровня.
   К ним относятся контейнеры JFrame, JApplet, JWindow и JDialog, эти контейнеры не являются наследниками класса JComponent. Однако они наследуют AWT классы Component и Container.

2. Облегченные контейнеры.
   Они наследуются от JComponent. Примером облегченного контейнера является JPanel, который является контейнером общего назначения.

JFrame - контейнер верхнего уровня, который обычно используется в Swing-приложениях.

JLabel – Swing-компонент, создающий метку, которая является компонентом, отображающим информацию (это самый простой Swing-компонент, пассивный компонент, т.е. метка не реагирует на действия пользователя).

```
import java.awt.*; import java.awt.event.*;  import javax.swing.*;

class EventDemo {
        JLabel jl;

        EventDemo() {
              JFrame jf = new JFrame("Фрейм с обработкой событий");
                        jf.setLayout(new FlowLayout());
                        jf.setSize(220,90);
                        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               JButton Alpha = new JButton ("Альфа") ;
               JButton  Beta = new JButton("Бета");

              Alpha.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent ae) {
                                        jl.setText("Кнопка Альфа нажата"); } } );

             Beta.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent ae) {
                                         jl.setText("Кнопка Beta нажата");  } }  );
             jl = new JLabel("Нажми на кнопку");
             jf.add(jl);    jf.add(Alpha);   jf.add(Beta);   jf.setVisible(true);
  }
public static void main(String args[ ]) {
                                        new EventDemo(); }          }
```


Фрейм с обработкой с...
Кнопка Альфа нажата
Альфа
Бета

Рисование в Swing

Swing позволяет выводить информацию в область отображения фрейма, панели или одного из компонентов Swing, таких как JLabel (в большинстве случаях использования Swing рисование не происходит на поверхности компонента) .

Чтобы вывести данные прямо на поверхность компонента, нужно использовать один или несколько методов рисования, определенных в AWT (например, drawLine() или drawRect()). AWT-класс Component содержит метод paint(), который используется для рисования данных на поверхности компонента. Метод paint() вызывается системой времени выполнения в процессе визуализации компонента, т.к. окно, в котором отображается компонент, может быть перекрыто другим окном, а затем вновь появиться на экране, окно может быть свернуто, а затем восстановлено. Метод paint() вызывается также в тех случаях, когда программа начинает свою работу. При написании кода, основанного на AWT, нужно переопределить метод paint().

Поскольку класс JComponent унаследован от Component, все легковесные компоненты Swing наследуют метод paint(), но не требуют переопределять его, чтобы вывести информацию на поверхность компонента.

Swing использует три различных метода:
paintComponent(), paintBorder() и paintChildren() .

Эти методы рисуют заданную часть компонента и делят процесс рисования на три разных логических действия. В swing-компоненте исходный метод AWT paint() выполняет вызовы этих методов в указанном порядке.

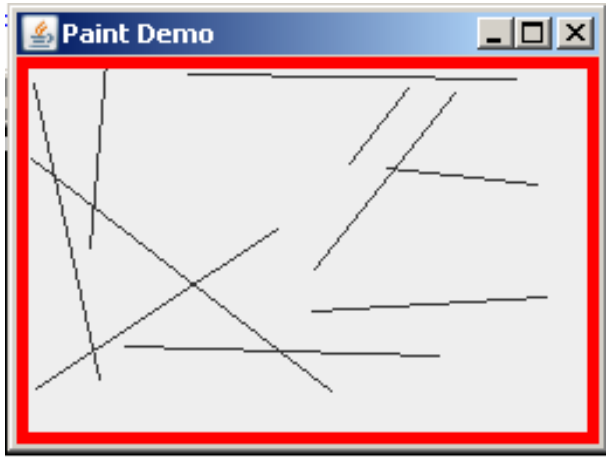
Чтобы нарисовать поверхность компонента Swing, нужно создать подкласс компонента, а затем пере определить метод paintComponent(). При переопределении метода paintComponent() сначала вызывается метод **super.paintComponent()**, после выводятся данные, которые требуется отобразить.

protected void paintComponent(Graphics g)

Чтобы нарисовать компонент, нужно вызвать метод repaint(), который вызывает paint() сразу же, как только представляется возможность. В Swing вызов метода paint() приводит к вызову метода paintComponent().

```
import java.awt.*;          import java.awt.event.*;
import javax.swing.*;     import java.util.*;

class PaintPanel extends JPanel {
    Insets ins;
    Random  rand;
PaintPanel() {
          setBorder( BorderFactory. createLineBorder (Color.RED, 5));
          rand = new Random();  }

protected void paintComponent(Graphics g) {
    super.paintComponent(g);
       int    x,   y,   x2,  y2;
       int   height = getHeight();
       int   width = getWidth();
       ins = getInsets();
    for (int i=0; i < 10; i++) {
         x = rand.nextInt(width - ins.left);
         y = rand.nextInt(height - ins.bottom);
         x2 = rand.nextInt(width - ins.left);
         y2 = rand.nextInt(height - ins.bottom);
        g.drawLine(x, y, x2, y2);    }
}  }
```

```
class PaintDemo {
Jlabel  jlab;
PaintPanel  pp;

PaintDemo () {
 JFrame jfrm = new JFrame (“Paint Demo”);
            jfrm.setSize(200, 150);
            jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            pp = new PaintPanel();
            jfrm.add(pp) ;
            jfrm.setVisible(true);  }

public static void main(String args[]) {
                                    new PaintDemo();  }
}
```



Insets

Класс Insets используется для того, чтобы вставлять в объект Panel границы, соответствующие ширине верхнего, нижнего, левого и правого краев.

Иногда требуется, чтобы между контейнером, в котором содержатся компоненты, и окном, где они находятся, оставался небольшой промежуток. Для этого можно использовать метод getInsets() из класса Container. Метод возвращает объект Insets, который содержит верхнюю, нижнюю, левую и правую вставки (inset), используемые при отображении объекта. Эти значения диспетчер компоновки применяет для вставки компонентов во время компоновки окна.

Конструктор Insets:

Insets (int top, int left, int bottom, int right)

Значения top, left, bottom и right, определяют промежуток между контейнером и окном, в которое он помещен.

Insets getInsets()

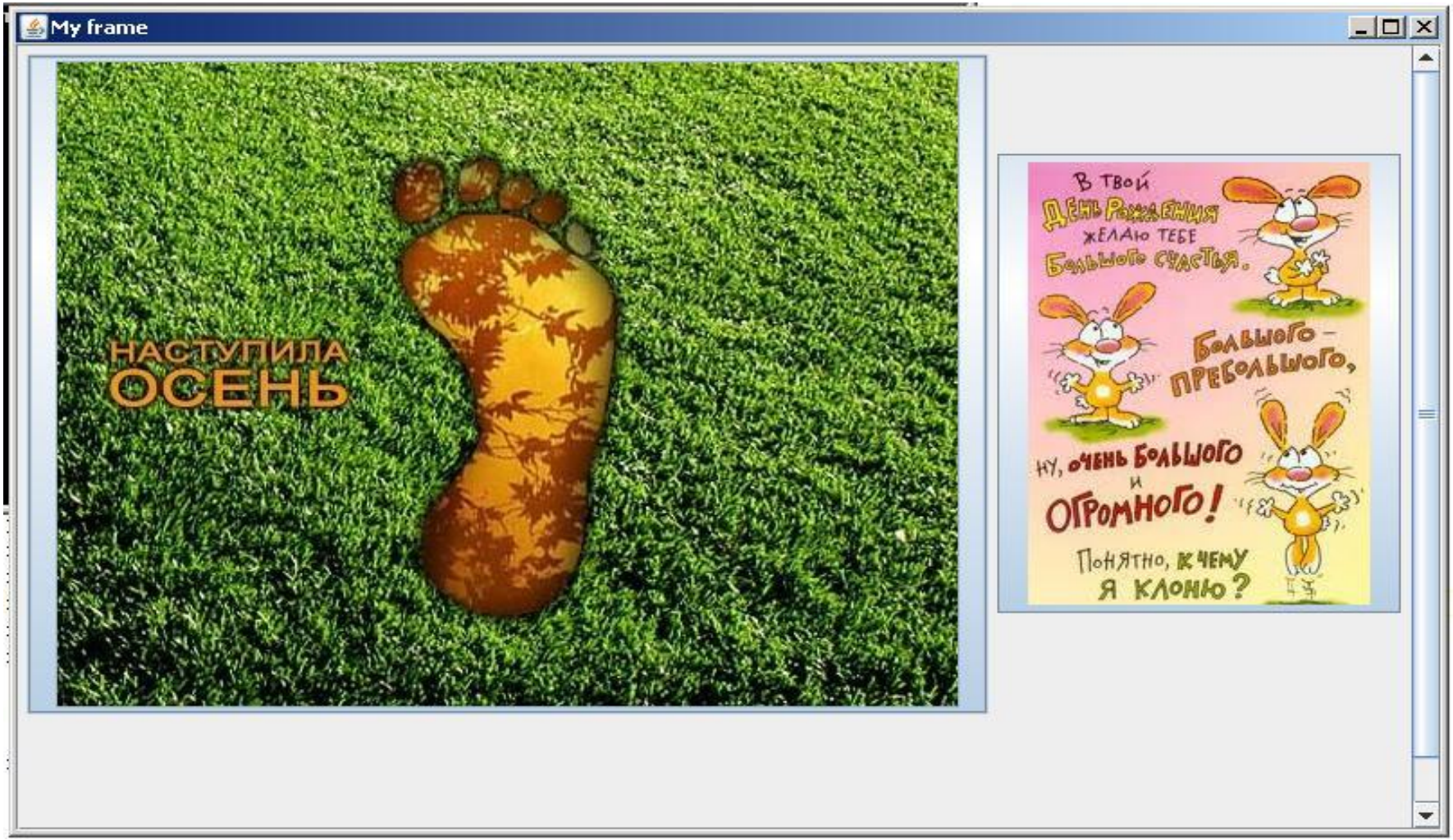
```
import javax.imageio.ImageIO;
import javax.swing.*;
import java.awt.*;
import java.io.IOException;
import java.awt.image.*;

class ImagePanel extends JPanel {
   private Image image;

   public ImagePanel(BufferedImage Im) {
      image = Im.getScaledInstance(200, 200, Image.SCALE_DEFAULT);
   }

public void paintComponent(Graphics g) {
     super.paintComponent(g);
    if(image != null){
         g.drawImage(image, 0, 0, 1000, 1000, null);
                                          //getWidth(), getHeight(), null);
      }   }   }
```

```
public class TestPicture {
    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setTitle("My Panel");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ImagePanel p =null;
 try {
p = new ImagePanel(ImageIO.read(TestPicture.class.getResource("pict.jpg")));
                    }
 catch (IOException e) {  e.printStackTrace();      }

        p.setPreferredSize(new Dimension(1000, 1000));
        f.add(new JScrollPane(p));
        f.setSize(800, 600);
        f.setVisible(true);
    }
}
```


My Panel
НАСТУПИЛА
ОСЕНЬ

```
import javax.imageio.ImageIO;
import java.awt.*;
import java.io.IOException;
import java.awt.image.*;

BufferedImage  B =null;

 try {
        B=ImageIO.read(TestPicture.class.getResource("pict.jpg")));
                }
 catch (IOException e) {
                                e.printStackTrace();      }


Image   image =
                B.getScaledInstance(200, 200, Image.SCALE_DEFAULT);
   }
```

```
import java.awt.Dimension;  import javax.swing.*;

public class TestFrame extends JFrame {

   public TestFrame() {
      JFrame f = new JFrame("My frame");
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JPanel p = new JPanel();
      JButton b = new JButton();
      ImageIcon icon= new ImageIcon(TestFrame.class.getResource("pict.jpg"));
            b.setIcon(icon);
            p.add(b);
            p.setPreferredSize(new Dimension(500, 500));
            f.add(p);
            f.add(new JScrollPane(p));
            f.setSize(1000, 1000);
            f.setLocationRelativeTo(null);
            f.setVisible(true);
            }

   public static void main(String[] args) {
                                                new TestFrame();         }
}
```



Получить изображение: Image im = icon.getImage();

```
import javax.swing.*;   import javax.imageio.*;   import java.awt.*;
import java.io.*; import java.lang.*; import java.util.*;
import java.awt.geom.*; import java.lang.Thread.*; import
java.lang.Runnable.*;
import java.util.Random.*; import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


public class SuperMen {

    public static void main(String[] args) {

       Frame frame = new Frame();
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setVisible(true);

  }
}
```

```
class Frame extends JFrame implements Runnable {

    static int    WIDTH = 640 ,   HEIGHT = 480;
    int   X = 350,   Y = 250,   flag =1;
    ImageBox    background,   men;
    MyButton   buttonStart,   buttonStop;
    Thread th;

public Frame() {

        setSize(WIDTH, HEIGHT);          // размер фрейма
        setLocation(X,Y);
        setLayout(null);
        background = new ImageBox("nightsky.jpg", 0,0);
        add(background);
        buttonStart = new MyButton(0, 430, 70, 20, "Start");
        buttonStop = new MyButton(71, 430, 70, 20, "Stop");

       background.add(buttonStart);
       background.add(buttonStop);

      men = new ImageBox("SuperMen.png", 50,50);
```

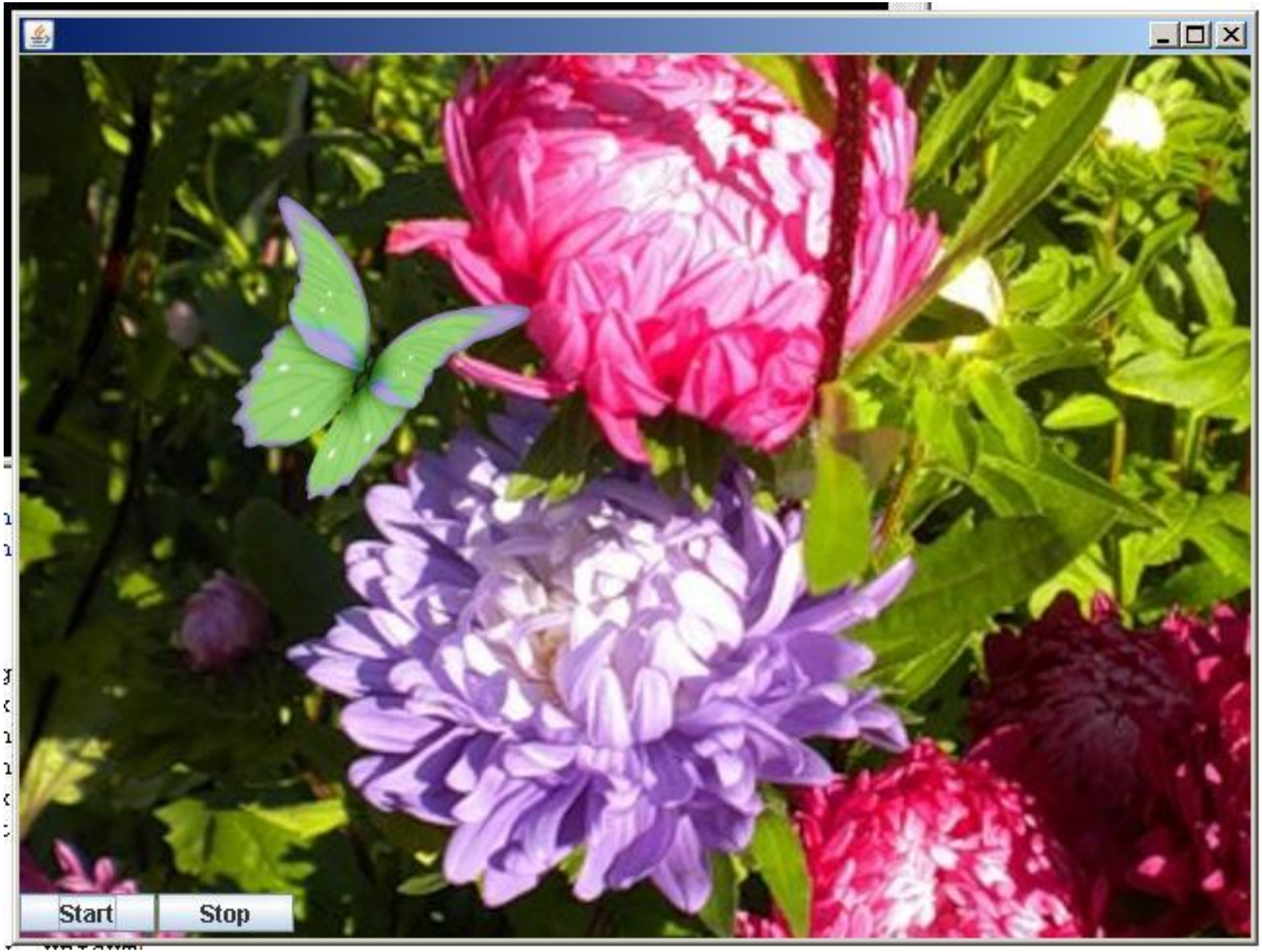
```
 th = new Thread(this, "Дочерний поток - супермен летает");

buttonStart.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                      if (flag== 1)
                            th.start();
                       flag = 2;                              }         }  );

buttonStop.addActionListener(new ActionListener() {
                   public void actionPerformed(ActionEvent e) {
                        flag = 3;
                                                          }       }  );

}
```

```
public void run() {
    int x, y;
    background.add(men);
    x = 750;
    y = 800;

   try {
      for (;;) {
       Thread.sleep(1);
              if (flag == 2) {
                 Thread.sleep(20);
                 men.setBounds(y, x, Frame.WIDTH, Frame.HEIGHT);
                 men.repaint();
                 x--;  y--;
                 if (x == -140) {
                    x = 750;    y = 800;     }
            }    }     }
     catch(InterruptedException e) {
           System.out.println("Дочерний поток прерван");
     }   }   }
```

```
class ImageBox extends JComponent {

    private Image image;

     ImageBox(String s, int x, int y) {

        try {
            image = ImageIO.read(new File(s));
        }
        catch (IOException e) {
         System.out.println("Ошибка! - " + e);
        }

       this.setLayout(null);
       this.setBounds(x, y, Frame.WIDTH, Frame.HEIGHT );
   }

   public void paintComponent(Graphics g) {
        g.drawImage(image, 0, 0, null);

}   }
```

```
class MyButton extends JButton {
int X;
int Y;
int SIZEX;
int SIZEY;

    MyButton(int x, int y, int sx, int sy, String s) {
    X = x;
    Y = y;
    SIZEX = sx;
    SIZEY = sy;

     this.setText(s);
     this.setBounds(X, Y, SIZEX, SIZEY);

    }
}
```


Start
Stop


Start
Stop

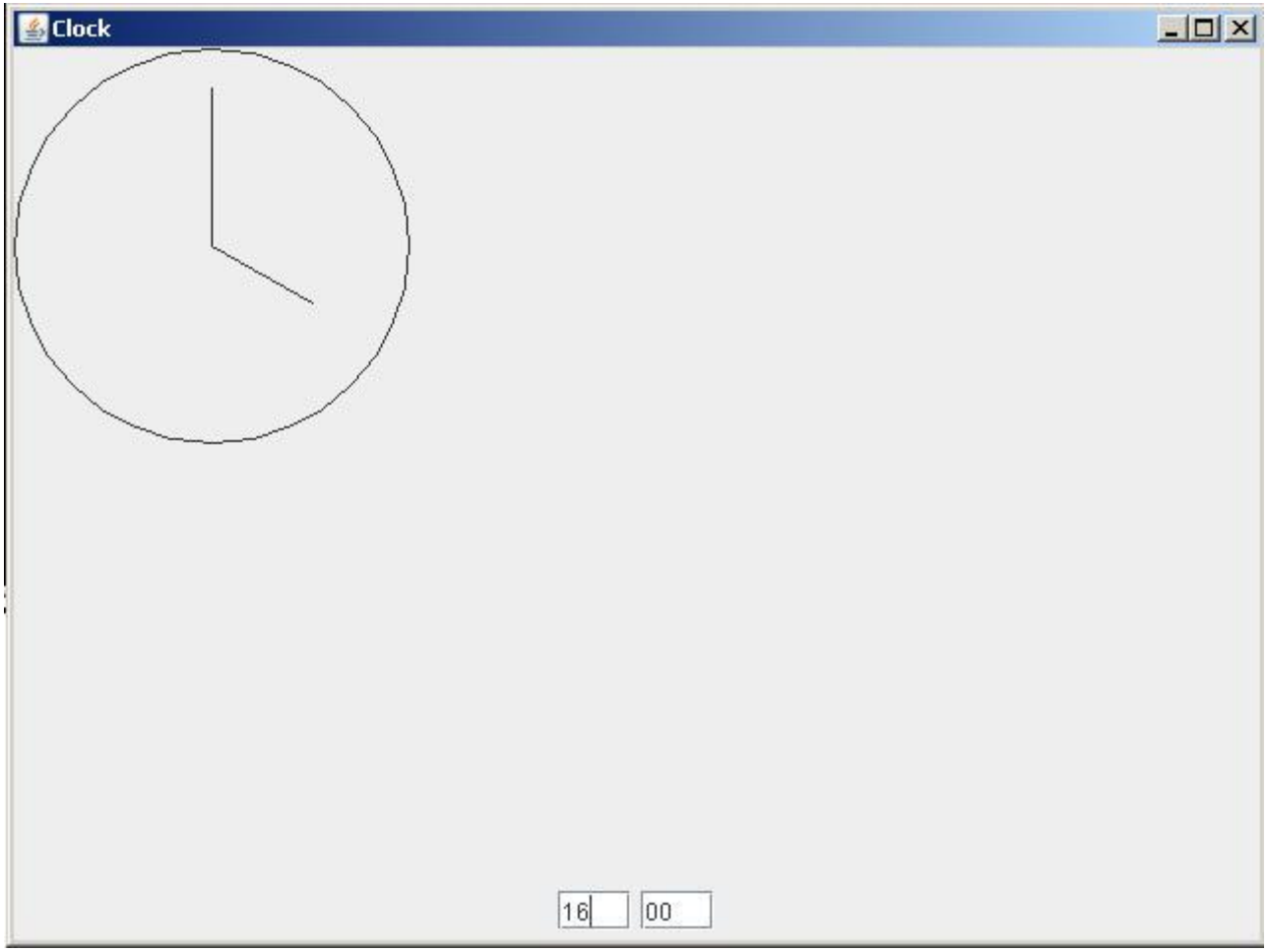

Start
Stop

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class Clock {
    public static void main(String[] args) {
       Frame frame = new Frame();
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setVisible(true);
    }
}

class Frame extends JFrame {

 static int WIDTH = 640;
 static int HEIGHT = 480;
 JTextField hour, minute;
 ClockPanel clock;
```

```
public Frame() {

    setSize(WIDTH, HEIGHT);
    setTitle("Clock");

            DocumentListener listener = new ClockListener();
            JPanel panel =new JPanel();

            hour = new JTextField("12", 3);
            panel.add(hour);
            hour.getDocument().addDocumentListener(listener);

            minute = new JTextField("00", 3);
            panel.add(minute);
            minute.getDocument().addDocumentListener(listener);

            add(panel, BorderLayout.SOUTH);

            clock = new ClockPanel();
            add(clock,BorderLayout.CENTER);
}
```

```
public void setClock(){
              try{
                  int hours = Integer.parseInt(hour.getText().trim());
                  int minutes =Integer.parseInt(minute.getText().trim());
                  clock.setTime(hours,minutes);  }
                catch(NumberFormatException e) { /*неверный формат времени  */  }
             }

             class ClockListener implements DocumentListener{
                 public void insertUpdate(DocumentEvent e){ setClock();
System.out.println("nnnnn");}
             public void removeUpdate(DocumentEvent e){ setClock();
System.out.println("nttttt");}
             public void changedUpdate(DocumentEvent e) { }
             }

             }
```

```
class ClockPanel extends JPanel{
double minutes =0, RADIUS =100, MIN_ST=0.8*RADIUS, HOUR_ST=0.6*RADIUS;

public void setTime(int h, int m){     minutes = h*60 + m;     repaint(); }

public void paintComponent(Graphics g){
   super.paintComponent(g);
   Graphics2D g2 = (Graphics2D)g;
   Ellipse2D circle = new Ellipse2D.Double(0,0,2*RADIUS,2*RADIUS);
   g2.draw(circle);
   double hAngle = Math.toRadians(90-360*minutes/(12*60));
   drawHand(g2,hAngle,HOUR_ST);
   double mAngle = Math.toRadians(90-360*minutes/60);
   drawHand(g2,mAngle,MIN_ST);
 }
public void drawHand(Graphics2D g2, double angle, double stLength){

Point2D end = new Point2D.Double(RADIUS+stLength*Math.cos(angle),
                                                    RADIUS-stLength*Math.sin(angle));

Point2D center = new Point2D.Double(RADIUS,RADIUS);
g2.draw(new Line2D.Double(center,end));
                        }           }
```


Clock
16
00

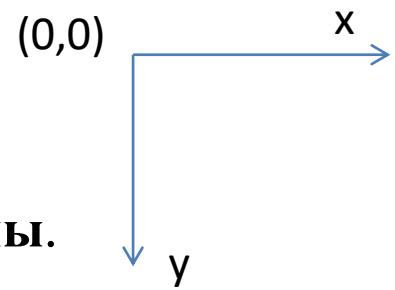
Программирование графики с помощью Java2D

Java2D предоставляет расширенные возможности программирования двумерной графики, включает функции для отображения линий, текста и изображений в пакетах:
java.awt.image
java.awt.color
java.awt.font
java.awt.geom
java.awt.print
java.awt.image.renderable.

Класс **Graphics2D** – подкласс абстрактного класса Graphics, используется для отображения двухмерных фигур, текста и изображений.

Программирование графики с помощью Java 2D

1. Java 2D



Система координат:
единица измерений – **пикселы**.

Графический контекст **Java** дает возможность рисовать на экране. Управление графическим контекстом происходит с помощью класса **Graphics**. **Graphics** содержит методы для рисования, работы со шрифтами, цветом и т.п.

Graphics – абстрактный класс (нельзя создать объект), который для каждой платформы реализует все возможности рисования. Мы используем графику **Java** без учета различий платформ.

Метод **paint(Graphics g)** из класса **Component** вызывается для отображения объекта **Component** и содержит ссылку на объект типа **Graphics**. Метод **paint(Graphics g)** вызывается системой, разработчик для перерисовки должен использовать метод **repaint()**, который вызывает метод **update()**, который предварительно по умолчанию очищает экран и вызывает **paint()**. Поэтому можно переопределить **update()** без очистки экрана, в некоторых случаях это помогает избавиться от мерцания изображения при перерисовке без использования двойной буферизации.

Класс **JComponent** – подкласс **Component** – суперкласс компонентов из **Swing** (пакет **javax.swing**) . При рисовании в **Swing** вызывается метод **paintComponent(Graphics g)**. Разработчик вызывает этот метод с помощью **repaint()**.

Java 2D предоставляет усовершенствованные возможности двумерной графики.

Пакеты, интерфейсы и классы Java 2D:

1. Пакет **java.awt**

Класс **Graphics2D** – подкласс класса **Graphics** для отображения двумерных фигур, текста и изображений.

Класс **BasicStroke** – набор атрибутов для отображения линий.

Класс **GradientPaint** – градиентная заливка фигур.

Класс **TexturePaint** – заливка двумерных фигур с помощью текстур.

Интерфейс **Shape** – определения геометрических объектов.

Интерфейс **Stroke** – методы для отображения контуров геометрических фигур.

Интерфейс **Paint** - определяет отображение графики.

2. Пакет **java.awt.geom**

Класс **GeneralPath** – траектория, построенная с помощью прямых линий, кубических и квадратичных кривых (кривые Безье).

Класс **Line2D** –линия в пространстве координат.

Класс **RectangularShape** – базовый класс для геометрических фигур, заключенных в прямоугольники.

Класс **AffineTransform -** аффинное преобразование, которое выполняет линейное отображение координат, сохраняя прямолинейность и параллельность линий (поворот фигуры вокруг заданной точки).

3. Пакет **java.awt.image**

Класс **AffineTransformOp -** класс использует аффинное преобразование для линейного отображения координат исходного изображения или растра в требуемые изображение или растр.

Класс **ColorModel** – набор методов для определения цвета пиксела.

Класс **Raster** – класс, описывающий прямоугольный массив пикселов.

Класс **BufferedImage** – изображение Image с буфером пикселов, описываемых **ColorModel** и **Raster**.

Интерфейс **BufferedImageOp** – определяет методы для работы с объектами **BufferedImage**, например, размывка изображения.

Интерфейс **RasterOp -** определяет методы для работы с объектами класса Raster.

```
import javax.swing.*;
import java.awt.*;

public class Gradients {
    public static void main ( String[] args )  {
        JFrame f = new JFrame ();
          f.setBackground ( Color.WHITE );
          f.add ( new JComponent()  {

          public void paintComponent ( Graphics g ) {
             super.paintComponent ( g );
            Graphics2D   g2d = ( Graphics2D ) g;
            g2d.setPaint (
            new GradientPaint ( 0, 0, Color.WHITE, 0, getHeight (), Color.BLACK ) );
            g2d.fillRect ( 0, 0, getWidth () / 3 - 5, getHeight () );
              g2d.setPaint ( new LinearGradientPaint ( 0, 0, 0, getHeight (),
                 new float[]{ 0f, 0.25f, 0.5f, 0.75f, 1f },
                 new Color[]{ Color.WHITE, Color.RED, Color.GREEN, Color.BLUE,
                 Color.BLACK },
                 MultipleGradientPaint.CycleMethod.NO_CYCLE ) );
              g2d.fillRect ( getWidth () / 3, 0, getWidth () / 3 - 5, getHeight () );
```

```
g2d.setPaint ( new RadialGradientPaint ( getWidth () * 2 / 3 + getWidth () / 3 / 2,
                    getHeight () / 2, getHeight () / 2,
                    new float[]{ 0f, 0.25f, 0.5f, 0.75f, 1f },
                    new Color[]{ Color.WHITE, Color.RED, Color.GREEN,
                    Color.BLUE,  Color.BLACK },
                    MultipleGradientPaint.CycleMethod.REFLECT ) );
 g2d.fillRect ( getWidth () * 2 / 3, 0, getWidth () / 3, getHeight () );
          }
       } );

       f.setDefaultCloseOperation ( JFrame.EXIT_ON_CLOSE );
       f.setSize ( 500, 200 );
       f.setVisible ( true );
    }
}
```