Programming of Interactive Systems

Anastasia.Bezerianos@lri.fr

(Nolwenn Maudet)

Week 4:
Graphics & Images

Anastasia.Bezerianos@lri.fr

Java 2D API

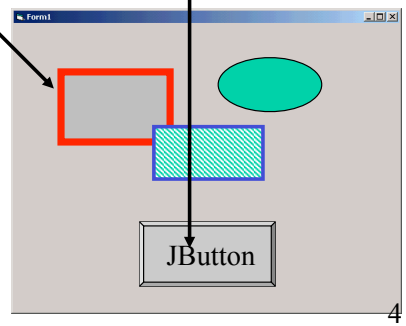
Provides 2D graphics, text & image capabilities

- Wide range of geometric primitives
- Mechanisms for hit detection of shapes, text, images
- Color & transparency
- Transformations
- Printing
- Control of the quality of rendering

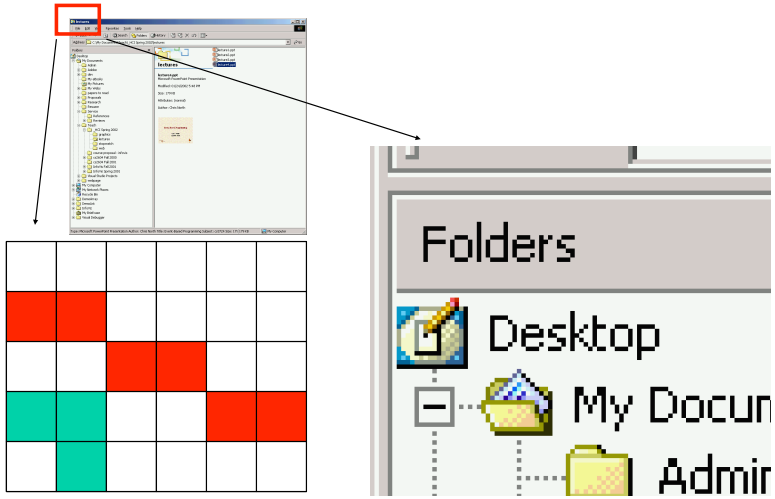
Designing components

A window (panel) is a canvas on which applications draw:

- API components (e.g., buttons), done by Java
- The rest (that is up to you!)



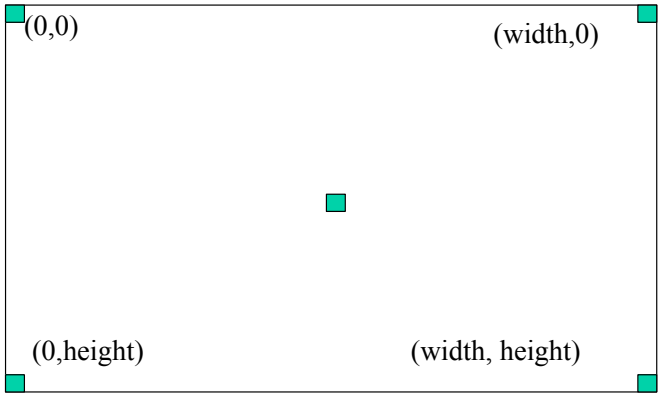
Pixel = picture element



coordinate system

Almost cartesian :

(0,0) top left

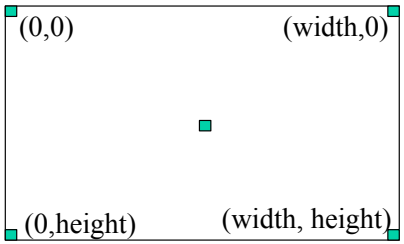


windows and subwindows

Each component has its own design space: its *subwindow*

Subwindow = the rectangular space inside the parent o the component, where the component is designed.

It has its own coordinate system

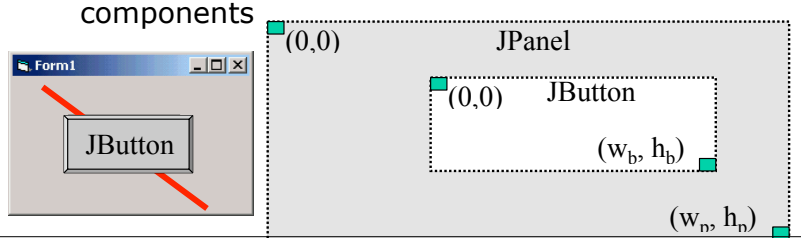


windows and subwindows

Each component has its own design space: its *subwindow*

Subwindow = the rectangular space inside the parent of the component, where the component is designed.
It has its own coordinate system

Clipping, rules: a component can not draw outside its subwindow, nor on one of its own components



my own graphical component

- inherits from Component, JComponent ou **JPanel**
- redefines the function paint

```
void paint (Graphics G_Arg) {
    Graphics2D g2 = (Graphics2D) G_Arg;
}
```

- inherits the function repaint(), that in turn calls paint(…)

  We call repaint() if we want to update our component

  Asynchronous, Java deals with paint in *Graphics*

- inherits methods (that we need to keep track of)

  setSize(), setEnable(), setBackground(), setFont(), setForeground()...

drawing

```
import java.awt.Graphics
import java.awt.Graphics2D // Java2
```

1. get a hold of the "graphics context" of your component

```
Graphics g = myJPanel.getGraphics( );
Graphics2D g2 = (Graphics2D) g;
```

2. draw different shapes

```
g2.drawLine(x1,y1, x2,y2);
```


new component, an example

```
public class MyPanel extends JPanel {

  public void paintComponent(Graphics g){
    super.paintComponents(g);        // erases background
    Graphics2D g2 = (Graphics2D)g;   //cast for java2

    // my graphics:
    g2.setColor(new Color(255,0,0));
    g2.fillRect(10,10,200,50); // left, top, width, height
    g2.setColor(new Color(0,0,0));
    g2.drawString("Hello World", 20, 20);// S, left, BOTTOM
  }
}
```

new component, an example

```
public class MyPanel extends JPanel {

    public void paintComponent(Graphics g) {
        super.paintComponent(g); // erases background
        Graphics2D g2 = (Graphics2D) g; // cast for java2
        // my graphics:
        g2.setColor(new Color(255, 0, 0));
        g2.fillRect(10, 10, 200, 50);
        g2.setColor(new Color(0, 0, 0));
        g2.drawString("Hello World", 20, 20);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("my panel");

        JPanel jp = new MyPanel();
        frame.getContentPane().add(jp);

        frame.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
        frame.setSize(250, 200);
        frame.setVisible(true);
    }
}
```

my own graphical component

- inherits from Component, JComponent ou **JPanel**
- redefines the function paint

```
void paint (Graphics G_Arg) {
    Graphics2D g2 = (Graphics2D) G_Arg;
}
```

- inherits the function repaint(), that in turn calls paint(…)
  - We call repaint() if we want to update our component
  - Asynchronous, Java deals with paint in *Graphics*
- inherits methods (that we need to keep track of)
  - setSize(), setEnable(), setBackground(), setFont(), setForeground()...

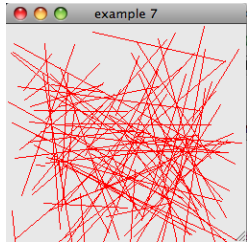
void paint (Graphics G_Arg) {

An instance of *Graphics* is provided by Java for components to draw on it

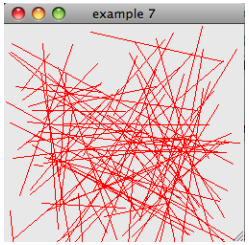
The Graphics object has a state:

- Translation from the origin for rendering: translate()
  - 0,0 = top left by default
- Effect Zone (!= rectangle) = Clip
  - by default : entire component, but we can constraint it
- Color of design
  - Color col1 = new Color (255, 0, 0); RGB but can have HSV
- Character font
  - Font font1 = new Font("SansSerif", Font.BOLD, 12);


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

public class SwingDemo7 extends JFrame {

    public JPanel panel;

    public void init() {
           Container cp = getContentPane();
           this.setTitle("example 7");
           this.setDefaultCloseOperation(EXIT_ON_CLOSE);
           panel = new JPanel();
           cp.add(panel);
    }

    public static void main(String[] args)
    {
            SwingDemo7 frame = new SwingDemo7();
            frame.init();
            frame.setSize(250,250);
            frame.setVisible(true);

            Graphics g = frame.panel.getGraphics();
            Graphics2D g2 = (Graphics2D) g;

            g2.setColor( Color.RED );

            for (int i = 0; i <100 ;++i) {
              g2.drawLine(
                  (int)(250*Math.random()), (int)(250*Math.random()),
                  (int)(250*Math.random()), (int)(250*Math.random()) );
              }
    }
}
```




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

public class SwingDemo7 extends JFrame {

    public JPanel panel;

    public void init() {
           Container cp = getContentPane();
           this.setTitle("example 7");
           this.setDefaultCloseOperation(EXIT_ON_CLOSE);
           panel = new JPanel();
           cp.add(panel);
    }

    public static void main(String[] args)
    {
            SwingDemo7 frame = new SwingDemo7();
            frame.init();
            frame.setSize(250,250);
            frame.setVisible(true);

            Graphics g = frame.panel.getGraphics();
            Graphics2D g2 = (Graphics2D) g;

            g2.setColor( Color.RED );

            for (int i = 0; i <100 ;++i) {
              g2.drawLine(
                  (int)(250*Math.random()), (int)(250*Math.random()),
                  (int)(250*Math.random()), (int)(250*Math.random()) );
              }
    }
}
```



What is going on here?

when to repaint() ?

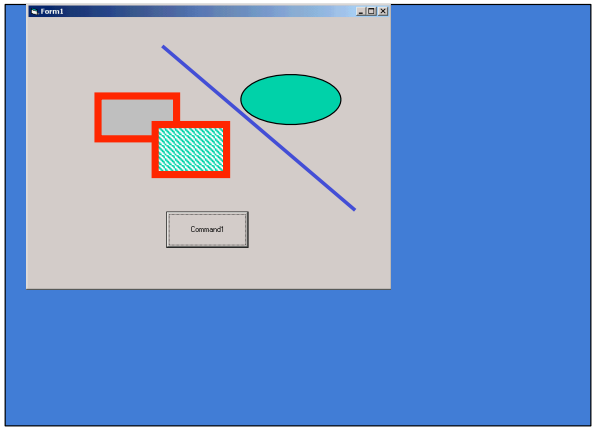
The screen is a single painting layer

- All windows are drawn on the same layer
- Windows do not know what they may hide
- Need to repaint every time a new zone of the screen becomes visible

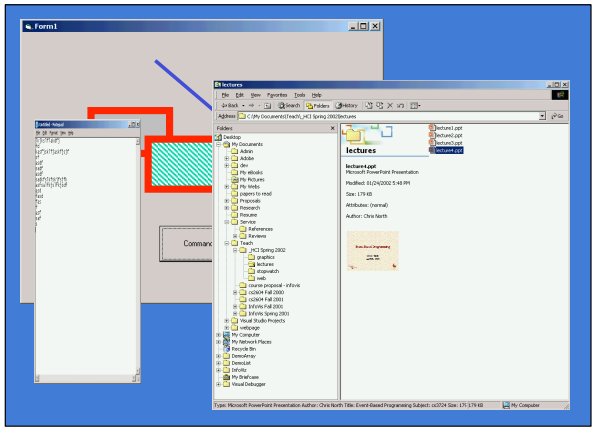
repainting events (handled by the system)

- window open, resizing, bring to front/back, minimize
- when other windows modify the screen

a screen with an application

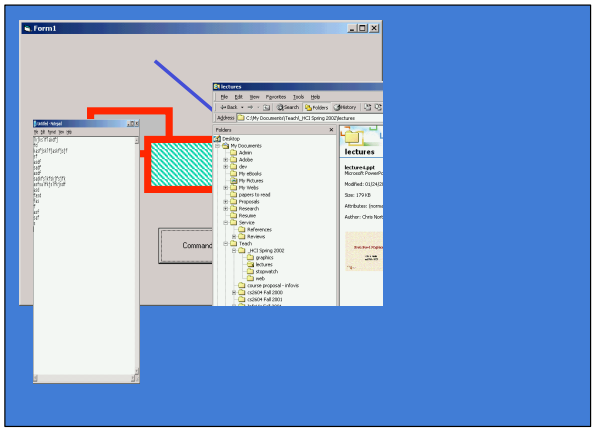


a screen with 3 applications



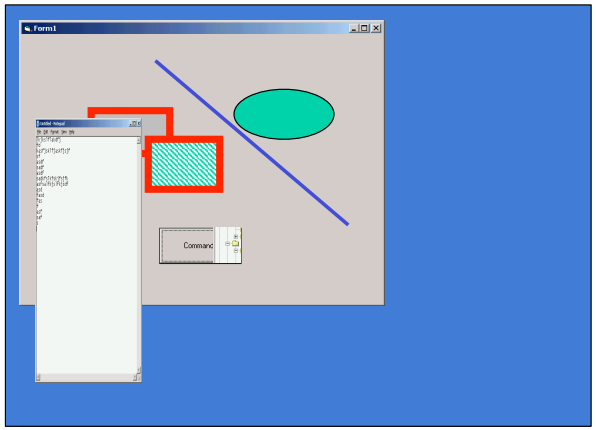
a screen, after closing one application -1

closing window notifies system of state, screen sends msg for repaint



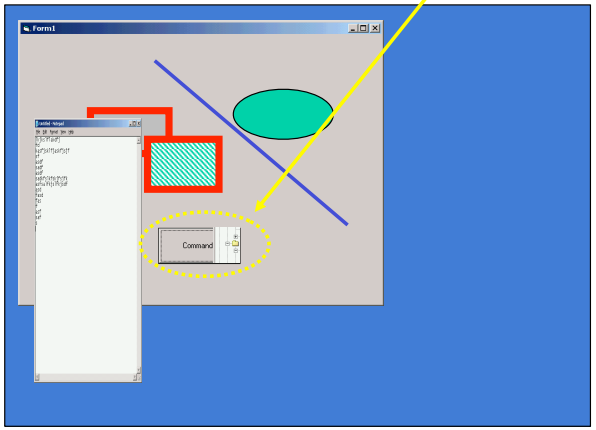
a screen, after closing one application - 2

remaining windows receive the message to repaint themselves



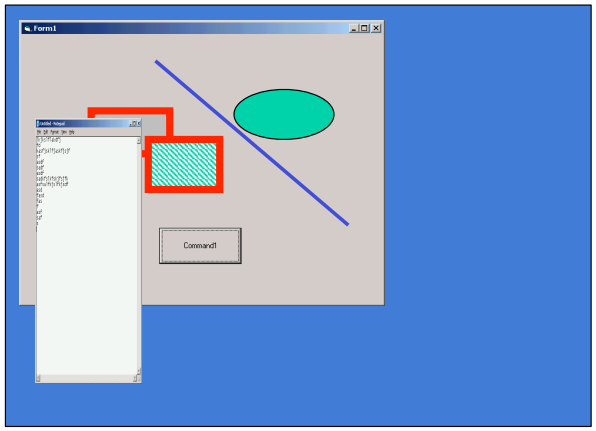
a screen, after closing one application - 3

each window asks their components to repaint themselves



a screen, after closing one application - 4

all updated!



painting in Java : *repaint()*

Repaint event:

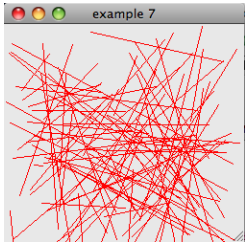
- Java Swing components can receive *repaint* events
- They call their function ***paintComponent( )***
- *E*ach component can inherit and redefine its *paintComponent()*

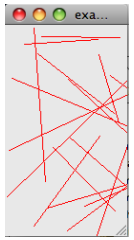
We call *repaint( ),* it calls paintComponent() or paint()

paint() vs paintComponent()

- paint() comes from AWT, in Swing paint() triggers:
  - paintComponent(), paintBorder(), and paintChildren()

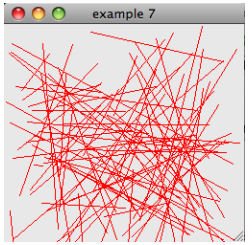
```
import javax.swing.*;
import java.awt.*;

public class SwingDemo7 extends JFrame {

    public JPanel panel;

    public void init() {
           Container cp = getContentPane();
           this.setTitle("example 7");
           this.setDefaultCloseOperation(EXIT_ON_CLOSE);
           panel = new JPanel();
           cp.add(panel);
    }

    public void paint(Graphics g) {
          g = this.panel.getGraphics();
          Graphics2D g2 = (Graphics2D) g;

          g2.setColor(Color.RED);

          for (int i = 0; i < 100; ++i) {
               g2.drawLine((int) (250 * Math.random()),
                     (int) (250 * Math.random()), (int) (250 * Math.random()),
                     (int) (250 * Math.random()));
          }
    }

    public static void main(String[] args)  {
             SwingDemo7 frame = new SwingDemo7();
             frame.init();
             frame.setSize(250,250);
             frame.setVisible(true);
    }
}
```



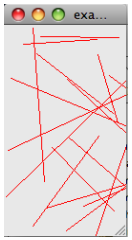


what happens after resizing?


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

public class SwingDemo7 extends JFrame {

    public JPanel panel;

    public void init() {
           Container cp = getContentPane();
           this.setTitle("example 7");
           this.setDefaultCloseOperation(EXIT_ON_CLOSE);
           panel = new JPanel();
           cp.add(panel);
    }

    public void paint(Graphics g) {
          g = this.panel.getGraphics();
          Graphics2D g2 = (Graphics2D) g;

          g2.setColor(Color.RED);

          for (int i = 0; i < 100; ++i) {
               g2.drawLine((int) (250 * Math.random()),
                     (int) (250 * Math.random()), (int) (250 * Math.random()),
                     (int) (250 * Math.random()));
          }
    }

    public static void main(String[] args)  {
             SwingDemo7 frame = new SwingDemo7();
             frame.init();
             frame.setSize(250,250);
             frame.setVisible(true);
    }
}
```



What is going on here?



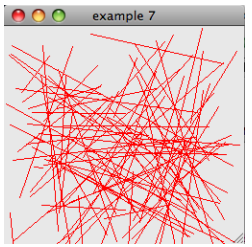
what happens after resizing?

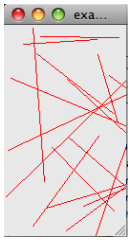
painting in Java : *repaint()*

Automatic repainting of windows can be intelligent, and not repaint if things have not changed in the window.

As a designer it is best to be have the state of your graphical objects ready to repaint at any given time


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

public class SwingDemo7 extends JFrame {

    public JPanel panel;

    public void init() {
           Container cp = getContentPane();
           this.setTitle("example 7");
           this.setDefaultCloseOperation(EXIT_ON_CLOSE);
           panel = new JPanel();
           cp.add(panel);
    }

    public void paint(Graphics g) {
          super.paintComponents(g);
          g = this.panel.getGraphics();
          Graphics2D g2 = (Graphics2D) g;

          g2.setColor(Color.RED);

          for (int i = 0; i < 100; ++i) {
               g2.drawLine((int) (250 * Math.random()),
                     (int) (250 * Math.random()), (int) (250 * Math.random()),
                     (int) (250 * Math.random()));
          }
    }

    public static void main(String[] args)  {
             SwingDemo7 frame = new SwingDemo7();
             frame.init();
             frame.setSize(250,250);
             frame.setVisible(true);
    }
```





what happens after resizing?

painting in Java : *repaint()*

When to call repaint (in **your** code) :

- when the user changes the visual properties of an object (e.g., move)
- when the user does an action that has a graphical result (e.g., adds a new object)
- when a thread (or animation clock) that controls a graphical element progresses to the next step

design functions in Graphics

Example : public void drawLine (x1, y1, x2, y2)
- properties depend on current color (i.e., last defined color)

fill*() / draw*() = draw filled shape or just contour
- * = { Rect, Oval, String, Arc, Polygon, PolyLine }

Function **clear()** to remove all drawn shapes

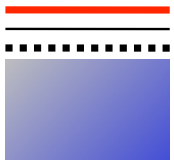
Function FontMetrics getFontMetrics()
- Returns an instance/object with info on the size of the text

Function **drawImage()** to draw an image
- Needs an instance/object "Image"
- Asynchronous. Possibility to listen: ImageObserver

design with Graphics2D

Function *public void paint(Graphics g)* called by Java
- But we use *Graphics* = ***Graphics2D*** since v1.1
- We always typecast : `Graphics2D g2 = (Graphics)g;`

Why? Graphics2D has advanced drawing



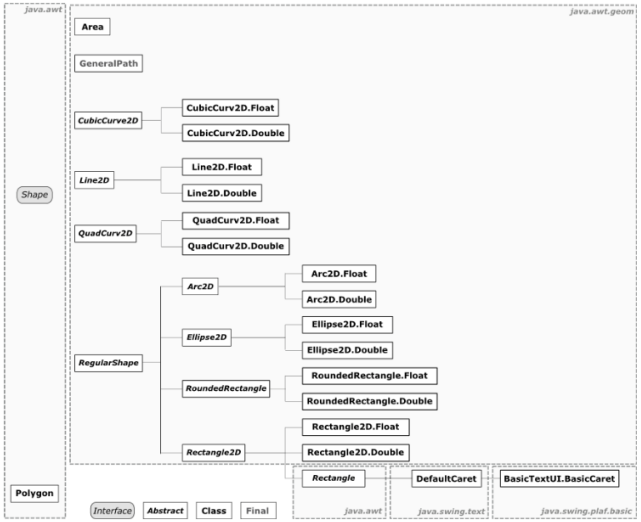
- Paint: Color, GradientPaint or TexturePaint
- Font
- Clip
- Stroke: forme, thickness (1p), corners
- Transformations: affine transformation matrix
  - Translation, rotation, zoom, penchant (sheer)
- Composite: rules on how to blend pixel colors
- List of RenderingHint that defines rendering quality

shape examples with « draw » and « fill »

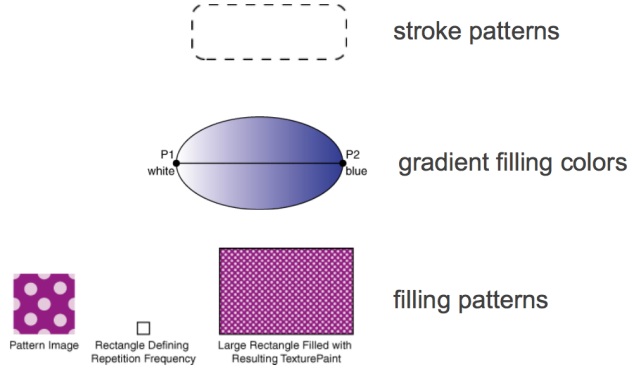
Quadratic Bézier curve Cubic Bézier curve

Arbitrary shapes (*GeneralPath*)

Geometric primitives



stroking and painting

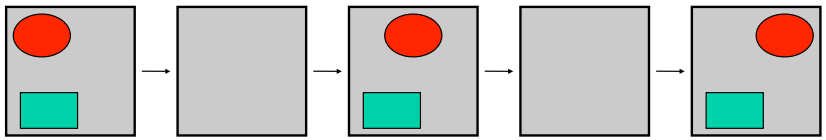


Flickering, un problème

"Ugly flickering" with *repaint*:

- Paint background
- Redraw shapes

This usually results in a flickering effect

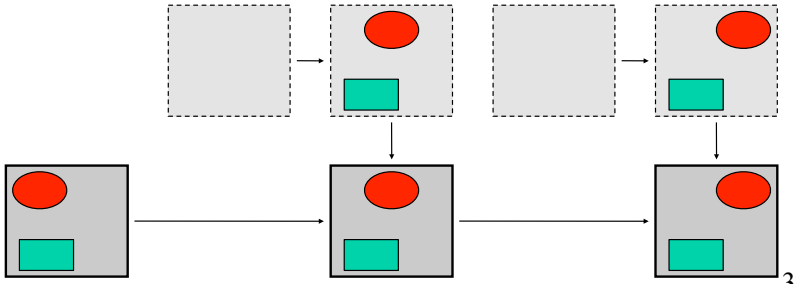


DoubleBufferring

design components in an image outside the screen

draw the image on *JPanel*

Swing does this for you with setDoubleBuffered(b) should be active by default



drawString and Antialiasing

drawString() designs each glyph in a text chain, assiging a color to each pixel that is "on" that glyph

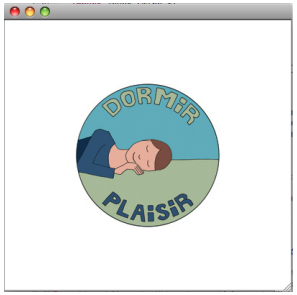
Antialiasing is a way to "smooth" the edges of text on screen (by assigning blended colors to pixels)

drawString and Antialiasing

```
  public void paintComponent(Graphics g){
    super.paintComponent(g);          // erases background
    Graphics2D g2 = (Graphics2D)g;    //cast for java2

    g2.setRenderingHint(
            RenderingHints.KEY_TEXT_ANTIALIASING,
            RenderingHints.VALUE_TEXT_ANTIALIAS_ON);

// my graphics:
    g2.setColor(new Color(255,0,0));
    g2.fillRect(10,10,200,50); // left, top, width, height
    g2.setColor(new Color(0,0,0));
    g2.drawString("Hello World", 20, 20);// S, left, BOTTOM
  }
```

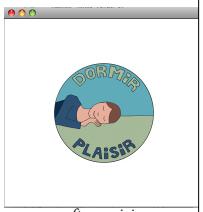

images


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

public class SwingDemo8 extends JPanel {

          BufferedImage image;

          public SwingDemo8(BufferedImage image) {
                    this.image = image;
          }

          protected void paintComponent(Graphics g) {
                    int x = (getWidth() - image.getWidth()) / 2;
                    int y = (getHeight() - image.getHeight()) / 2;
                    g.drawImage(image, x, y, this);
          }

          public static void main(String[] args) throws IOException {
                    BufferedImage image = ImageIO.read(new File("image.jpg"));

                    SwingDemo8 myDemo = new SwingDemo8(image);
                    JFrame f = new JFrame();
                    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                    f.add(new JScrollPane(myDemo));

                    f.setSize(400, 400);
                    f.setLocation(200, 200);
                    f.setVisible(true);
          }
}
```



after resizing (content unchanged)

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

public class SwingDemo9 extends JPanel {

        BufferedImage image;

        public SwingDemo8(BufferedImage image) {
                this.image = image;
        }

        protected void paintComponent(Graphics g) {
                g.drawImage(image,
                0, 0, getWidth(), getHeight(),
                0, 0, image.getWidth(), image.getHeight(), this);
        // drawImage (I, dst, src, null/this) :
        // dst how many pixels we'll draw, src part of the image to draw
        }

        public static void main(String[] args) throws IOException {
                BufferedImage image = ImageIO.read(new File("image.jpg"));

                SwingDemo9 myDemo = new SwingDemo9(image);
                JFrame f = new JFrame();
                f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                f.add(new JScrollPane(myDemo));

                f.setSize(400, 400);
                f.setLocation(200, 200);
                f.setVisible(true);
        }
}
```



after resizing change size

Clipping (for images and more)

Restricts the drawing area to be rendered

Clipping (for images and more)

Restricts the drawing area to be rendered

```
rect.setRect(x+marginx, y + marginy, w,h);
 g2.clip(rect);
 g2.drawImage(image,x,y,null); // could be other shapes
```

Transformations

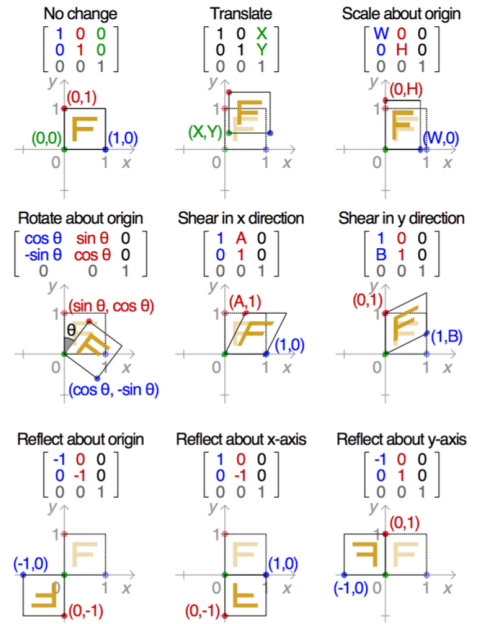
rotate, scale, translate, shear methods on Graphics2D

```
g2.translate (100,200);
```

AffineTransform class

```
AffineTransform atransf = new Affintetransform();
atransf.rotate(Math.PI/2); // rotate 90 degrees
g2.transform(atransf);
```

Affine Transformations



more on Java2D
http://docs.oracle.com/javase/tutorial/2d/

more on image manipulation
http://docs.oracle.com/javase/tutorial/2d/images/index.html
http://www.javalobby.org/articles/ultimate-image/