Chapter 14

Graphics

Introduction

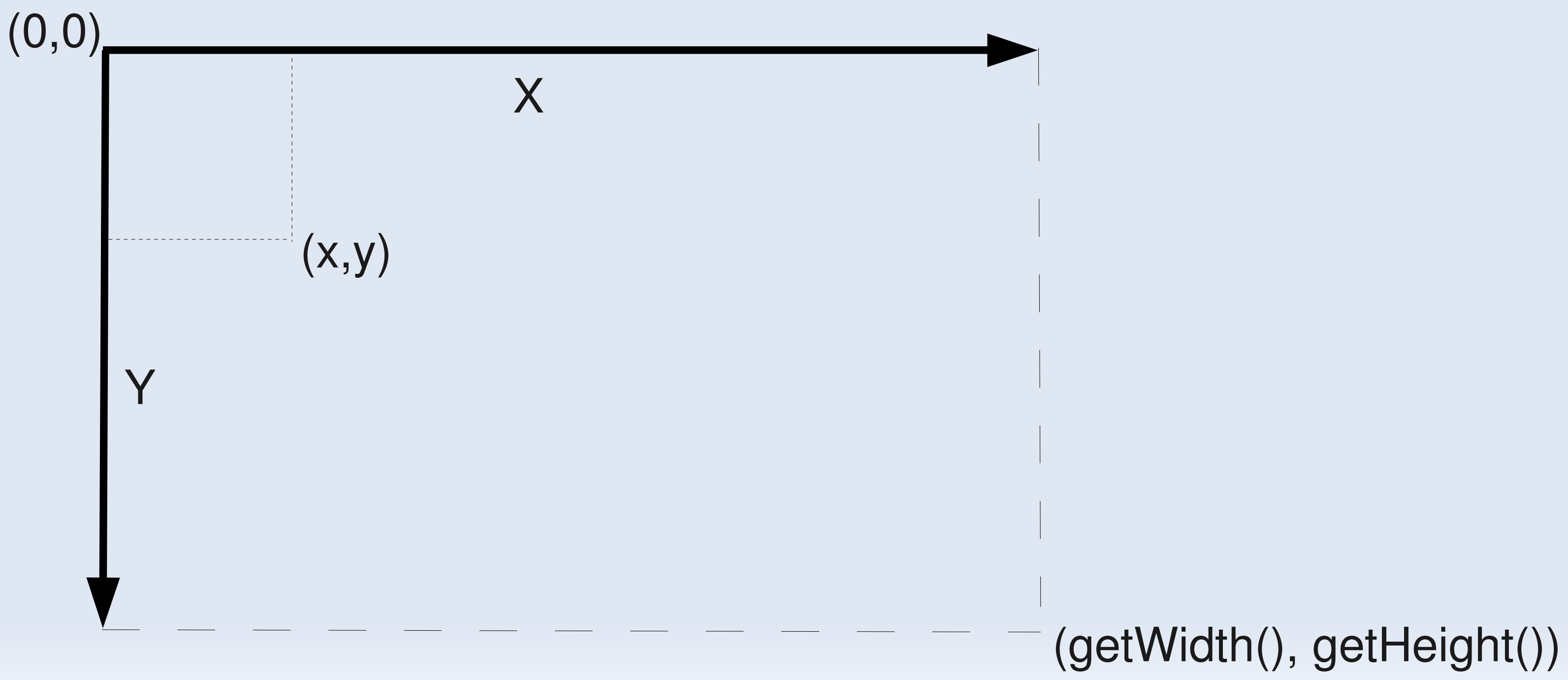
Sometimes we need to represent something in a way that is not available by default in the Swing library. For this we have the ability to create our own custom graphics.

Today we will be covering how to extend JPanel and draw our own custom graphics on top of it.

We will start by examining the Java coordinate system

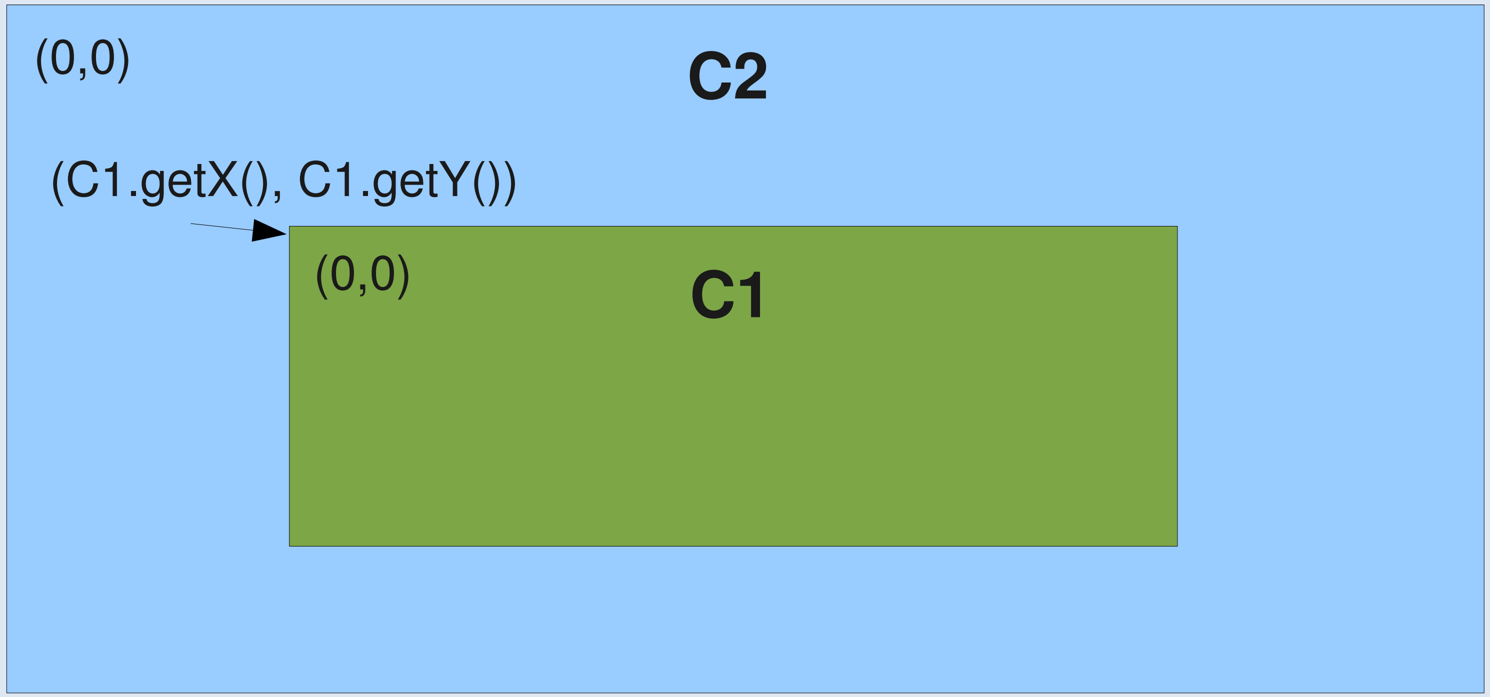
Coordinate System

- Each component in Java has its own coordinate system.
- The origin is not in the center of the component, but is in the upper left hand corner.



Coordinate System

- The location of a component C1 that is contained inside another component C2 is (C1.getX(), C1.getY()) .



Graphics Class

- Now that we are familiar with the coordinate system, lets take a look at what is available to us if we want to draw our own graphics.
- You can draw strings, lines, rectangles, ovals, arcs, polygons, and polylines using the methods in the Graphics class
- We are starting with the java.awt.Graphics class and not the java.awt.Graphics2D

Graphics Class

```
setColor(color: Color)
setFont(font: Font)
drawString(s: String, x: int, y: int)
drawLine(x1: int, y1: int, x2:int, y2: int)
drawRect(x: int, y: int, w: int, h: int)
fillRect(x: int, y: int, w: int, h: int)
drawRoundRect(x: int, y: int, w: int, h: int, aw: int, ah: int)
fillRoundRect(x: int, y: int, w: int, h: int, aw: int, ah: int)
draw3DRect(x: int, y: int, w: int, h: int, raised: boolean)
fill3Drect(x: int, y: int, w: int, h: int, raised: boolean)
drawOval(x: int, y: int, w: int, h: int)
fillOval(x: int, y: int, w: int, h: int)
drawArc(x: int, y: int, w: int, h: int, sAngle: int, arc: int)
fillArc(x: int, y: int, w: int, h: int, sAngle: int, arc: int)
drawPolygon(xPoints: int[], yPoints: int[], nPoints int)
fillPolygon(xPoints: int[], yPoints: int[], nPoints int)
drawPolyline(xPoints: int[], yPoints: int[], nPoints: int)
```

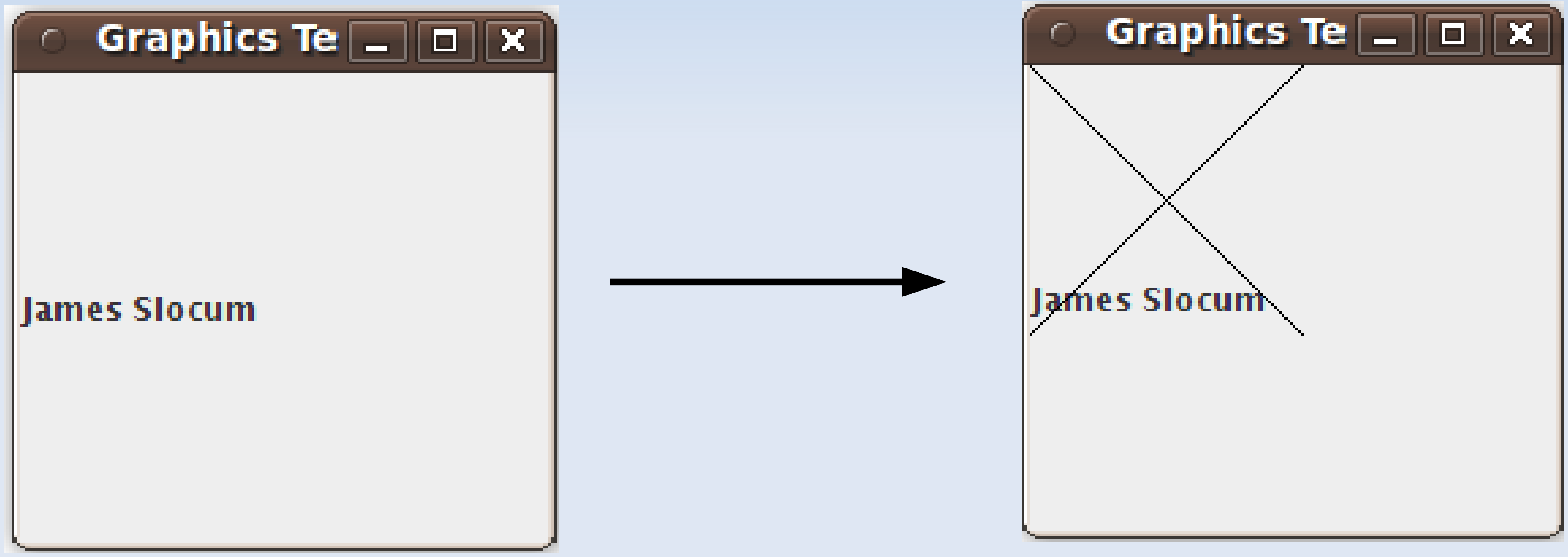
Lets take a look at how to create a Graphics object, and then look at how each one of these works

Creating a Graphics object

- Lets begin exploring the Graphics class by accessing the Graphics object of a component that is already on the screen.

```
import javax.swing.*;
import java.awt.*;
public class TestGraphics{
    public static void main(String[] args) throws Exception{
        JFrame frame = new JFrame("Graphics Test");
        JLabel label = new JLabel("James Slocum");
        frame.add(label);
        frame.setSize(200,200);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        Thread.sleep(1000);
        Graphics g = label.getGraphics();
        g.drawLine(0,0,100,100);
        g.drawLine(0,100,100,0);
    }
}
```

Creating a Graphics object



Problems with this approach

- You cannot access the graphics object until AFTER the frame is visible.
- If you re-size the frame, the lines disappear
- Your code can get really ugly if you are doing a lot of drawing

The solution to this is to extend JPanel and override its paintComponent() method.

Using JPanel

- You can use a JPanel as a blank canvas to draw your graphics on.
- Simply create a new class that extends JPanel.

```
public class CarPanel extends JPanel{
      protected void paintComponent(Graphics g){
```

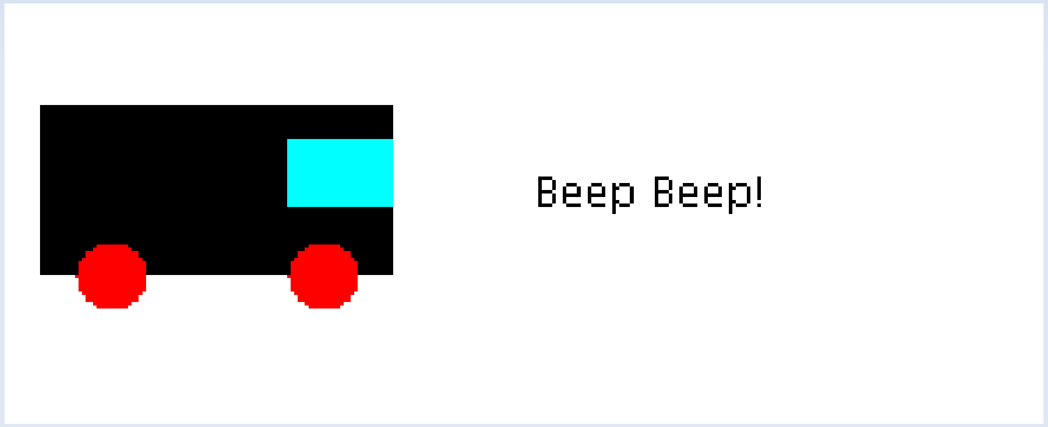
Using JPanel

```
class CarPanel extends JPanel{
    public CarPanel(){
        this.setBackground(Color.WHITE);
    }

    protected void paintComponent(Graphics g){
        super.paintComponent(g);
        g.setColor(Color.BLACK);
        g.fillRect(10,30,100,50);

        g.setColor(Color.RED);
        g.fillOval(20,70,20,20);
        g.fillOval(80,70,20,20);

        g.setColor(Color.CYAN);
        g.fillRect(80,40,30,20);

        g.setColor(Color.BLACK);
        g.drawString("Beep Beep!",150, 60);
    }
}
```

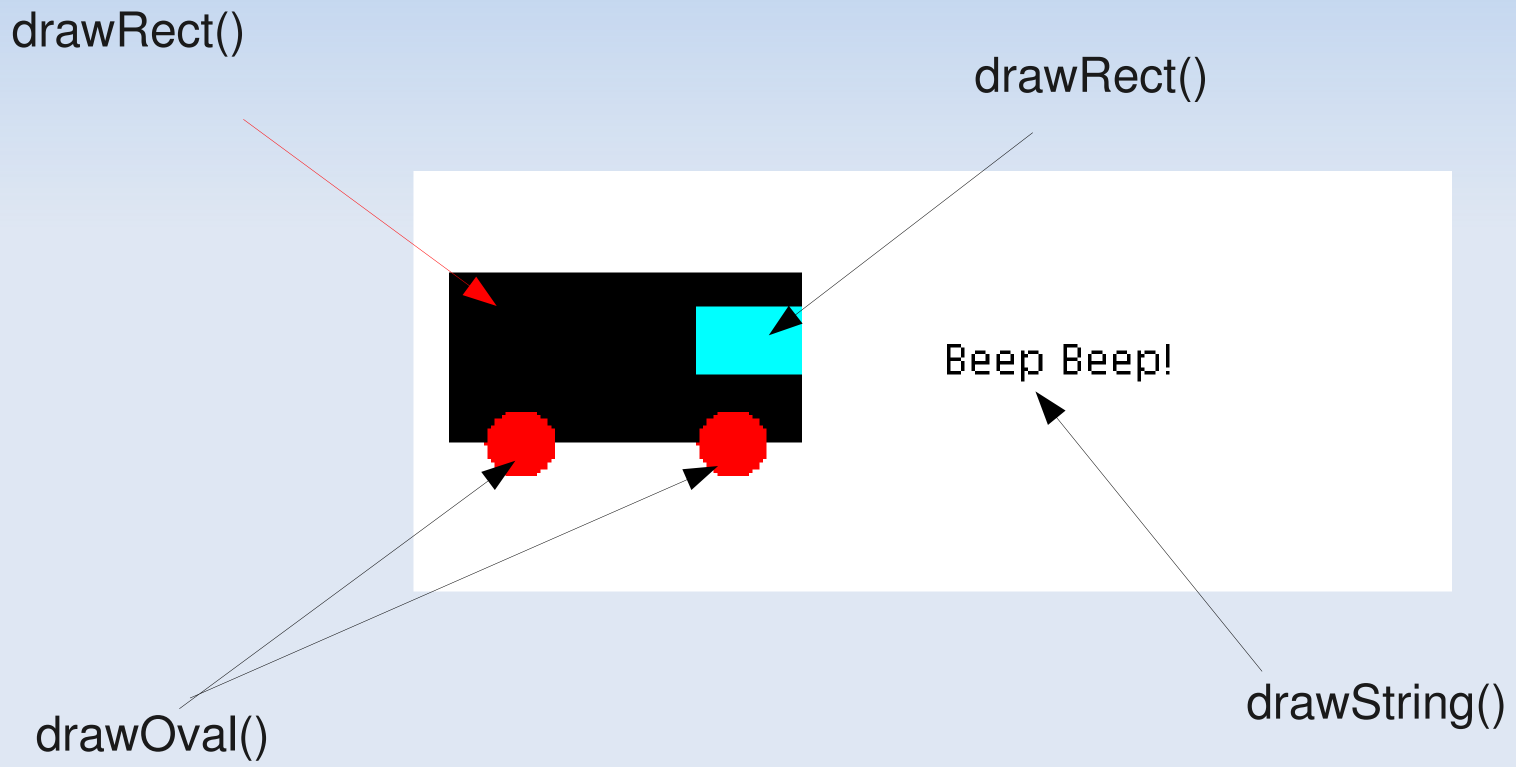
Using JPanel

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

public class TestCustomGraphics extends JFrame{

    public TestCustomGraphics(){
        add(new CarPanel());
    }

    public static void main(String[] args){
        JFrame frame = new TestCustomGraphics();
        frame.setSize(300,150);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Using JPanel

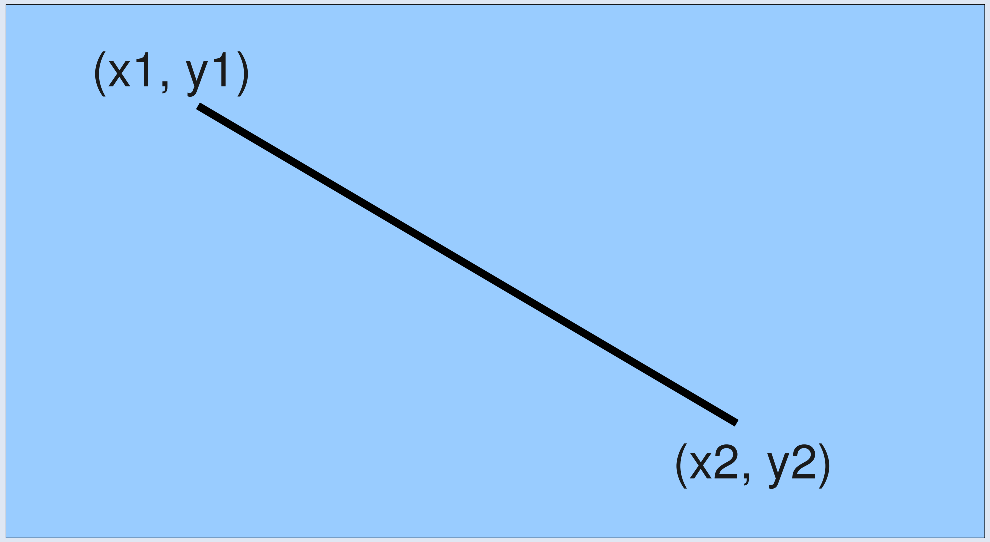


Using JPanel



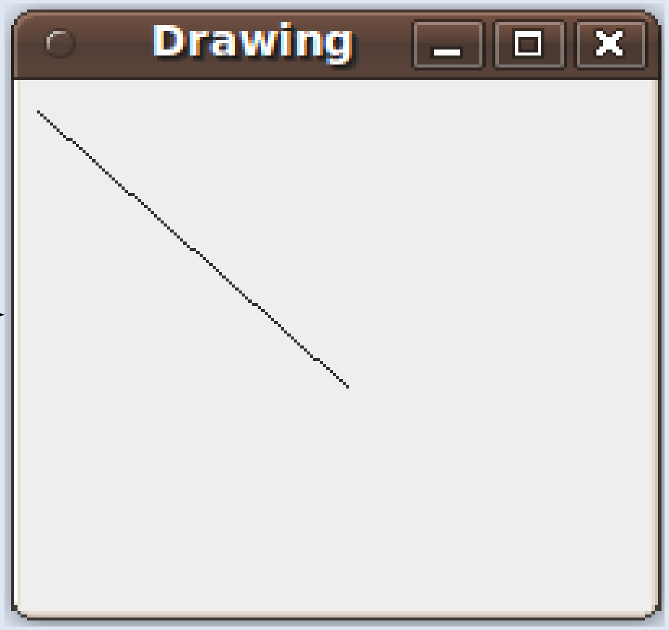
Using the Graphics class methods

- Lets go through each of the Graphics drawing methods and see how they work
- Lets start with drawLine()



Drawing a line

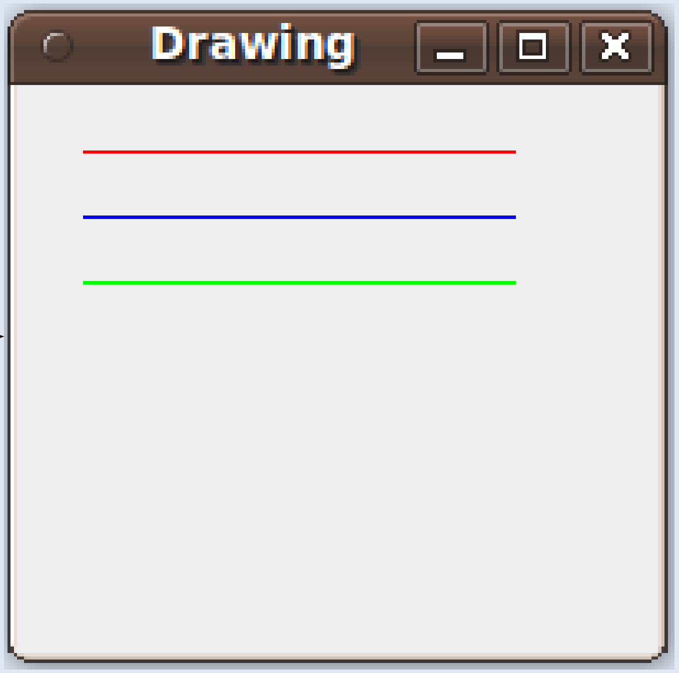
```
import javax.swing.*;
import java.awt.*;

public class DrawPanel extends JPanel{

  protected void paintComponent(Graphics g){
    super.paintComponent(g);

    g.drawLine(5,10,50,50);
  }
}
```



Drawing a line

- You can change the color of a line by using the setColor method before drawing the line

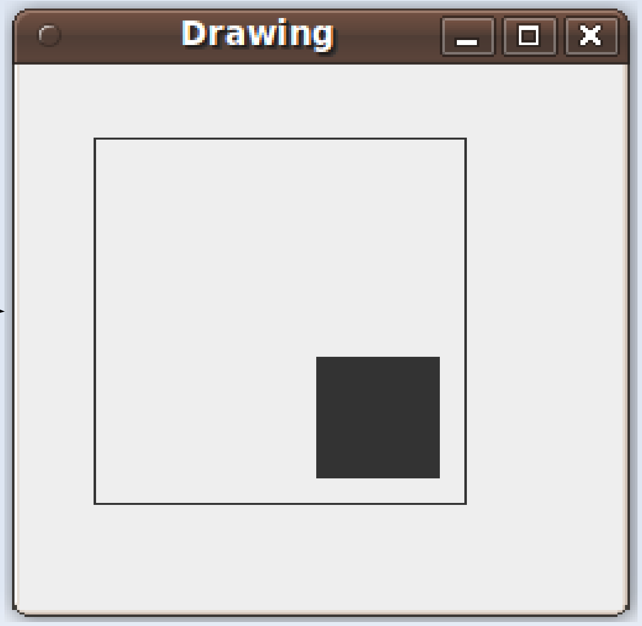
```
import javax.swing.*;
import java.awt.*;

public class DrawPanel extends JPanel{

   protected void paintComponent(Graphics g){
       super.paintComponent(g);

       g.setColor(Color.RED);
       g.drawLine(20, 20, 150, 20);

       g.setColor(Color.BLUE);
       g.drawLine(20, 40, 150, 40);

       g.setColor(Color.GREEN);
       g.drawLine(20, 60, 150, 60);
   }
}
```



Drawing a rectangle

- There are two ways to draw a rectangle, Filled or unfilled.

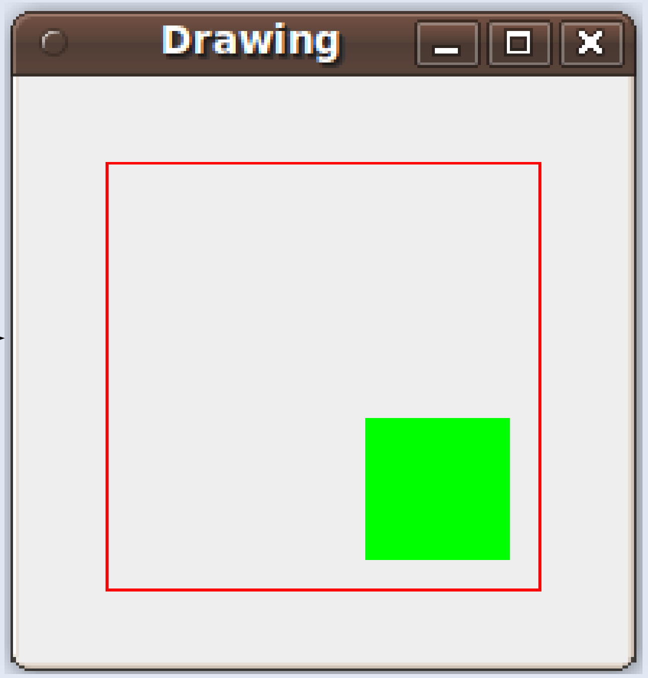
```
import javax.swing.*;
import java.awt.*;

public class DrawPanel extends JPanel{

    protected void paintComponent(Graphics g){
        super.paintComponent(g);

        g.drawRect(30, 30, 150, 150);
        g.fillRect(120, 120, 50, 50);
    }
}
```



Drawing a rectangle

- If you change the color, the drawRect() method will set the line to the color you choose. fillRect() will fill the entire rectangle with the color you choose.

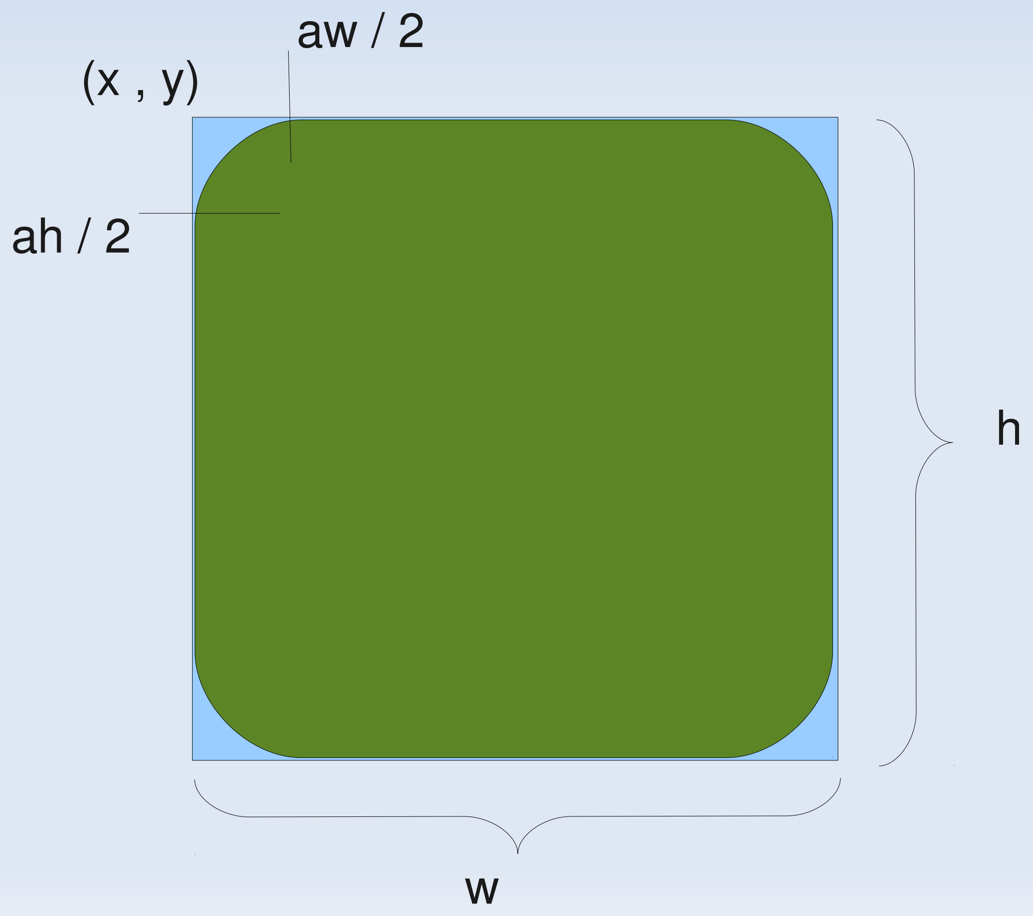
```
import javax.swing.*;
import java.awt.*;

public class DrawPanel extends JPanel{

   protected void paintComponent(Graphics g){
      super.paintComponent(g);

      g.setColor(Color.RED);
      g.drawRect(30, 30, 150, 150);

      g.setColor(Color.GREEN);
      g.fillRect(120, 120, 50, 50);
   }
}
```



Drawing a rounded rectangle

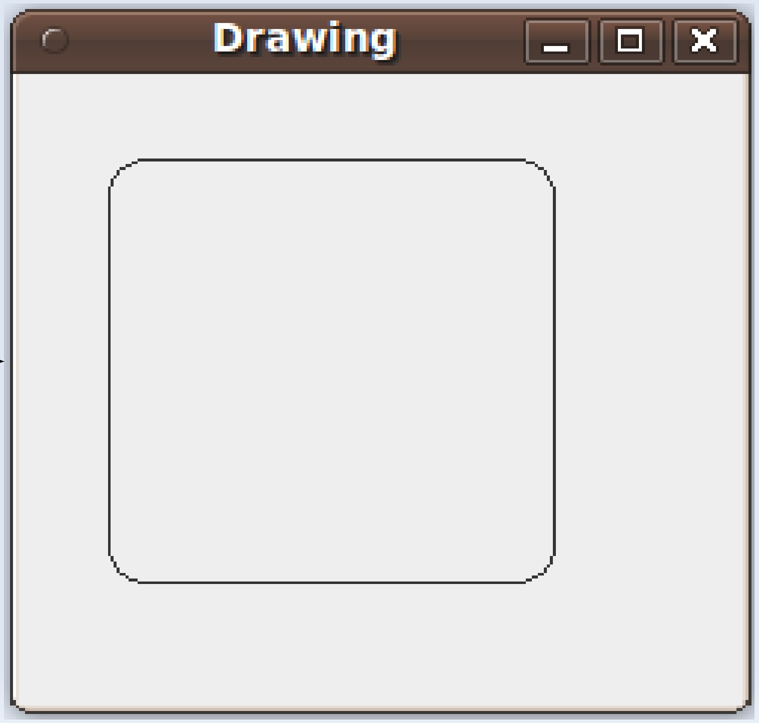
- This is slightly more complicated then the drawRect() and fillRect() methods

```
drawRoundRect(x: int, y: int, w: int, h: int, aw: int, ah: int)
fillRoundRect(x: int, y: int, w: int, h: int, aw: int, ah: int)
```



Drawing a rounded rectangle

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

public class DrawPanel extends JPanel{

   protected void paintComponent(Graphics g){
      super.paintComponent(g);

      g.drawRoundRect(30,30,150,150,25,25);
   }
}
```

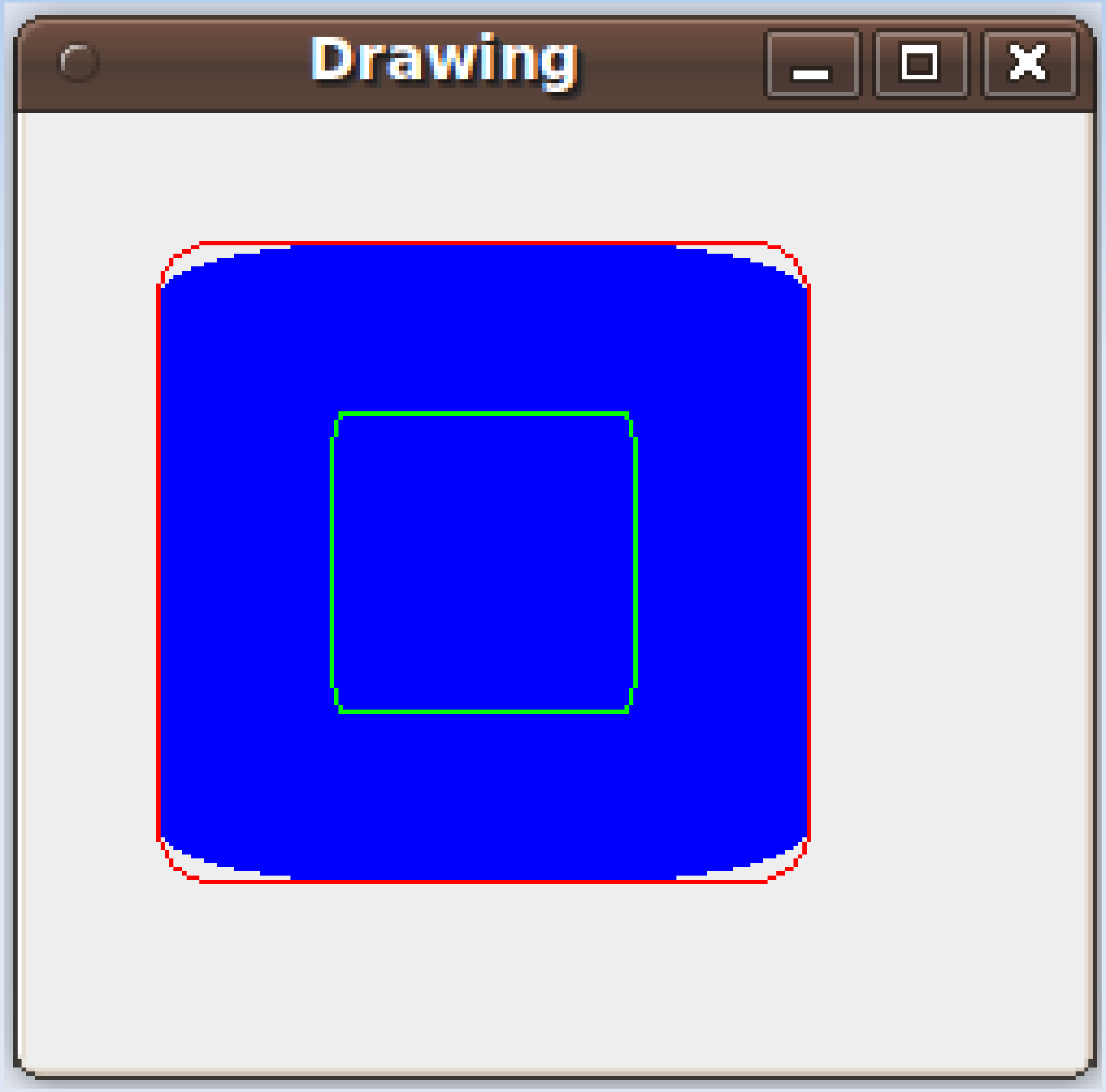


Drawing a rounded rectangle

- For nice round corners keep ah == aw

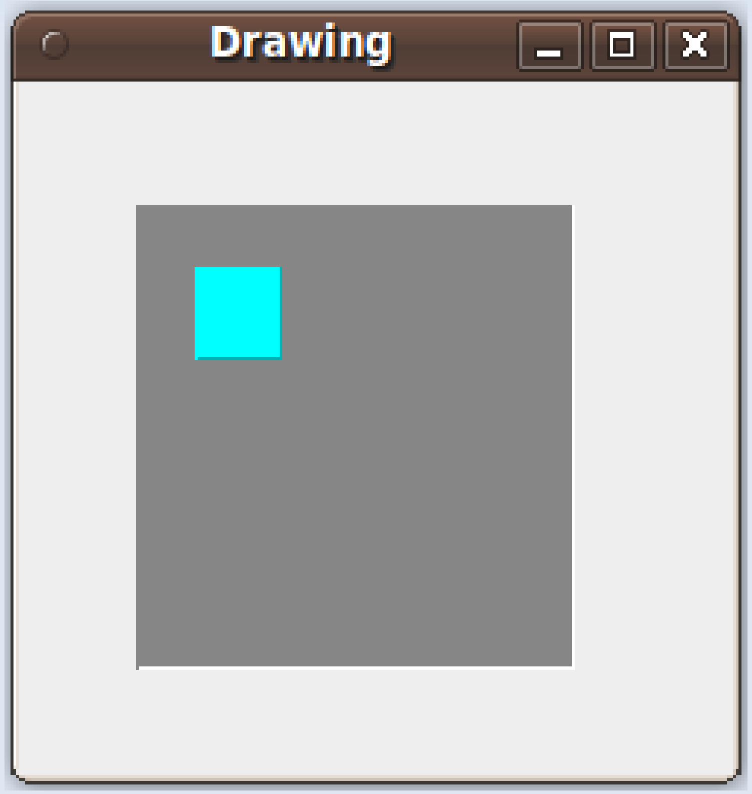
```
import javax.swing.*;
import java.awt.*;

public class DrawPanel extends JPanel{

    protected void paintComponent(Graphics g){
        super.paintComponent(g);

        g.setColor(Color.BLUE);
        g.fillRoundRect(30,30,150,150, 100, 25);

        g.setColor(Color.RED);
        g.drawRoundRect(30,30,150,150,25,25);

        g.setColor(Color.GREEN);
        g.drawRoundRect(70,70, 70,70, 5, 25);
    }
}
```

Drawing a rounded rectangle



Drawing a 3D rectangle

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

public class DrawPanel extends JPanel{

    protected void paintComponent(Graphics g){
        super.paintComponent(g);

        g.setColor(Color.LIGHT_GRAY);
        g.fill3DRect(40,40,150,150, false);

        g.setColor(Color.CYAN);
        g.fill3DRect(60,60,30,30,true);
    }
}
```

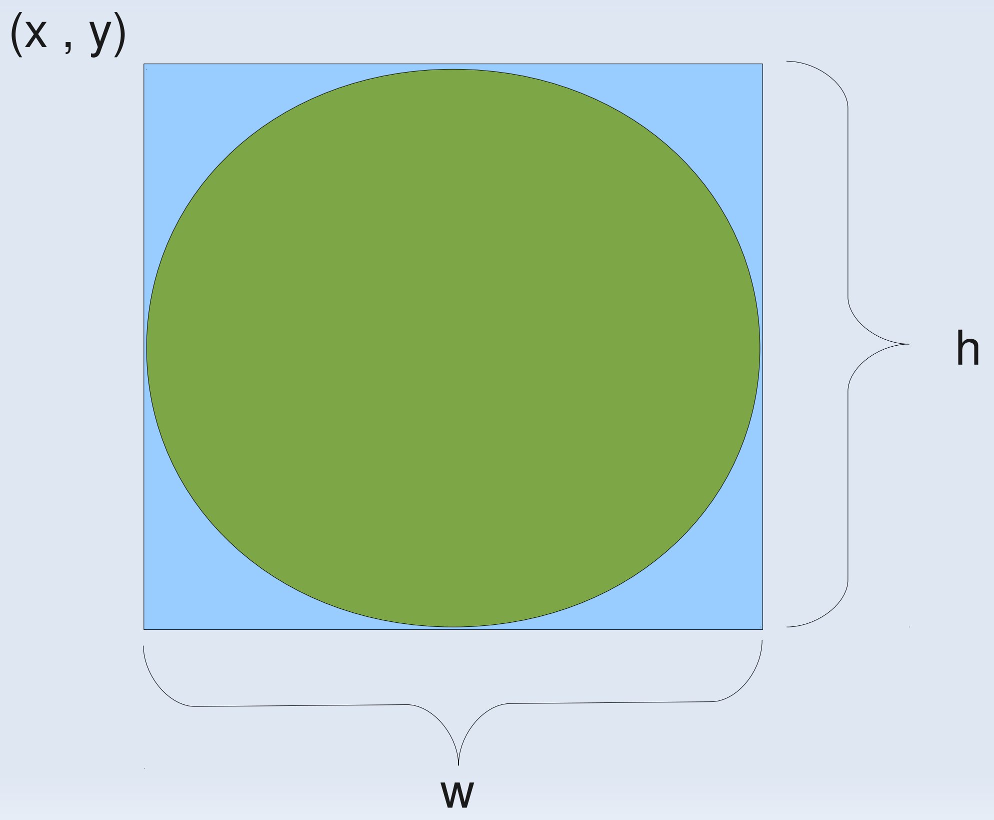


Its hard to see, but there are slight boarders around the boxes to make the box look like its going into, or coming out of the panel

Drawing an oval

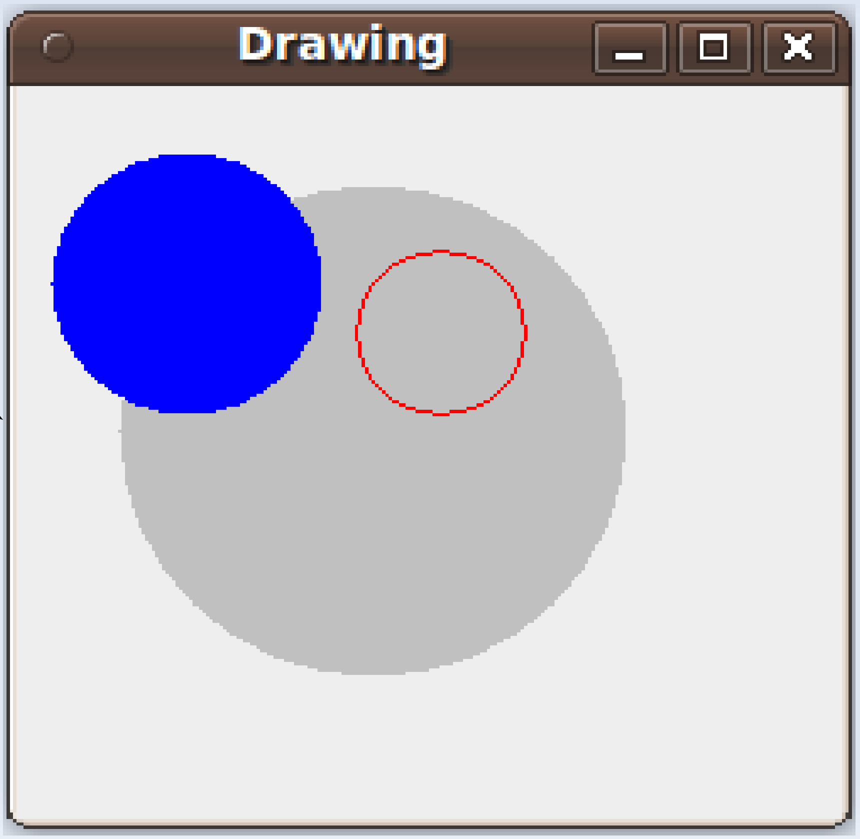
- When drawing a oval, you must specify the dimensions and location of the surrounding box, not the circle itself.

```
drawOval(x: int, y: int, w: int, h: int)
fillOval(x: int, y: int, w: int, h: int)
```



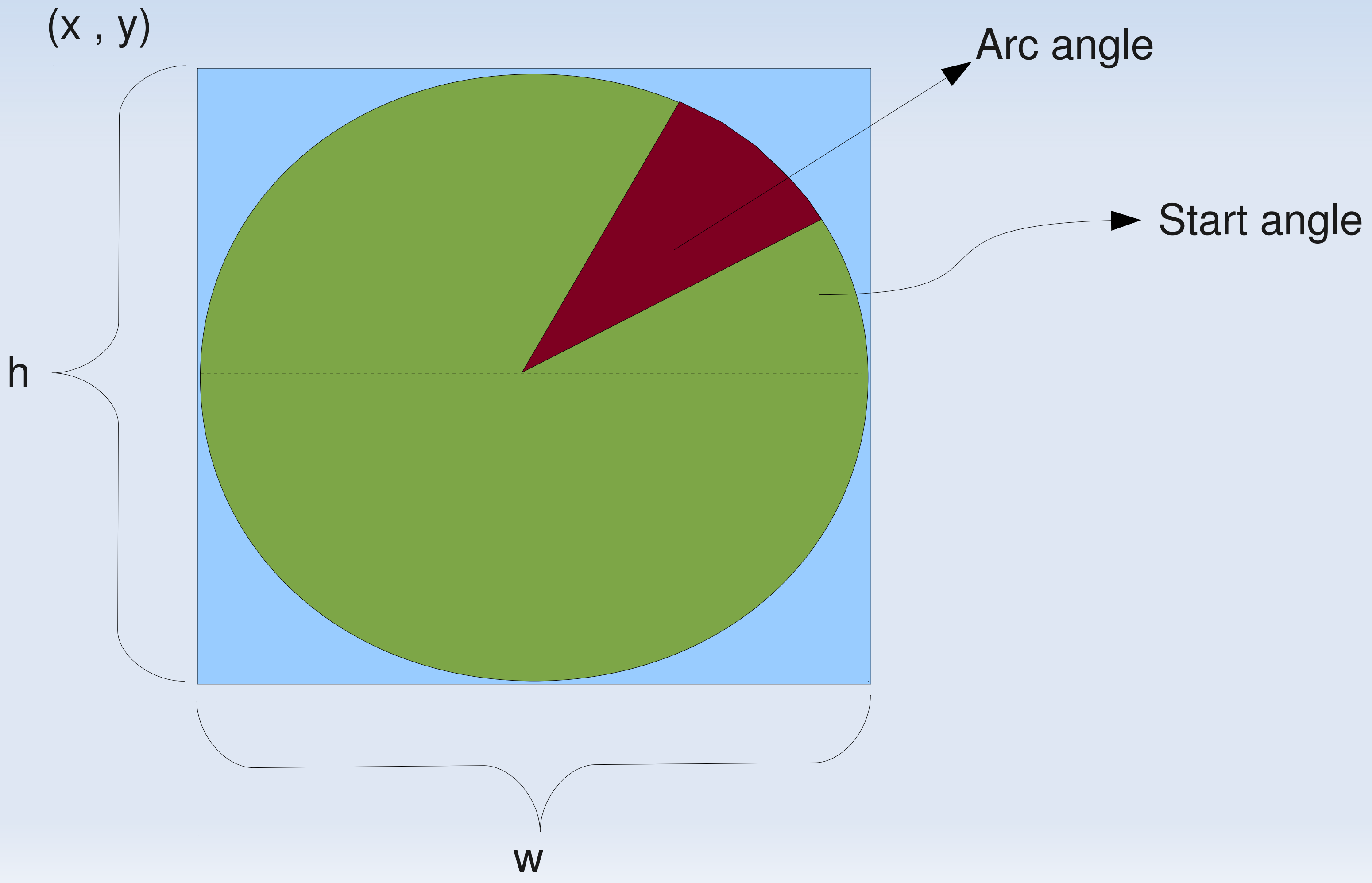
Drawing an oval

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

public class DrawPanel extends JPanel{

   protected void paintComponent(Graphics g){
      super.paintComponent(g);

      g.setColor(Color.LIGHT_GRAY);
      g.fillOval(30,30,150,150);

      g.setColor(Color.RED);
      g.drawOval(100,50,50,50);

      g.setColor(Color.BLUE);
      g.fillOval(10,20,80,80);
   }
}
```



Drawing an arc

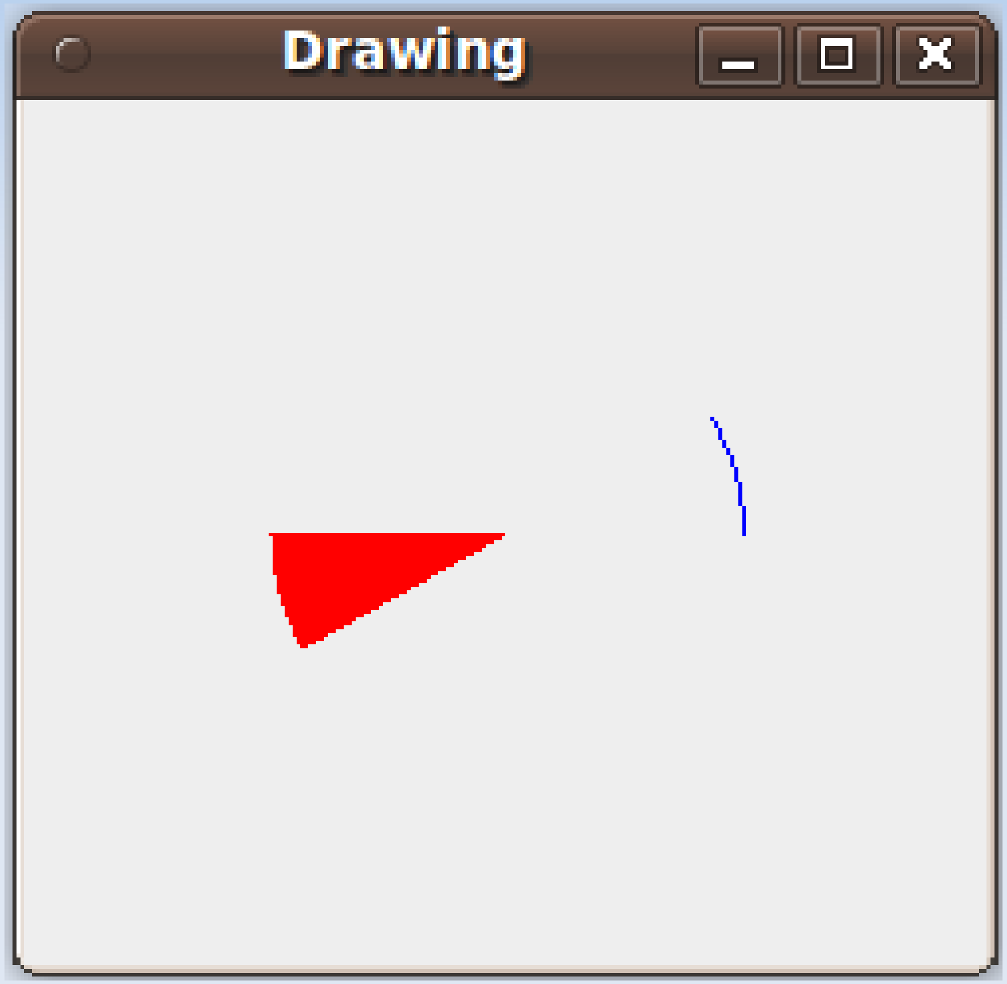
- When drawing an arc, you can imagine drawing a piece of an oval.
- You must provide the dimensions and coordinates of the bounding box, a start angle and an arc angle. This isn't as bad as it sounds!



Drawing an arc

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

public class DrawPanel extends JPanel{

   protected void paintComponent(Graphics g){
       super.paintComponent(g);

       int xCenter = getWidth() / 2;
       int yCenter = getHeight() / 2;

       int radius = 60;

       g.setColor(Color.BLUE);
       g.drawArc(xCenter - radius, yCenter - radius,
                                         2*radius,2*radius, 0, 30);

       g.setColor(Color.RED);
       g.fillArc(xCenter - radius, yCenter - radius,
                                         2*radius,2*radius, 180, 30);
   }
}
```

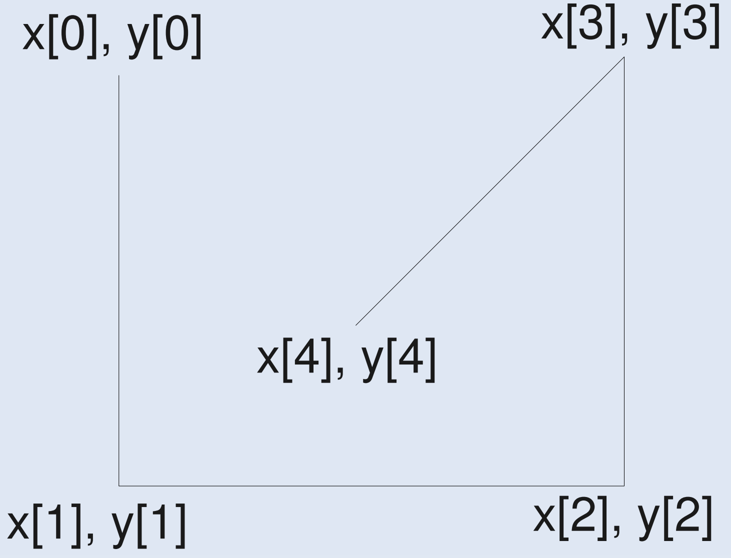
Drawing an arc

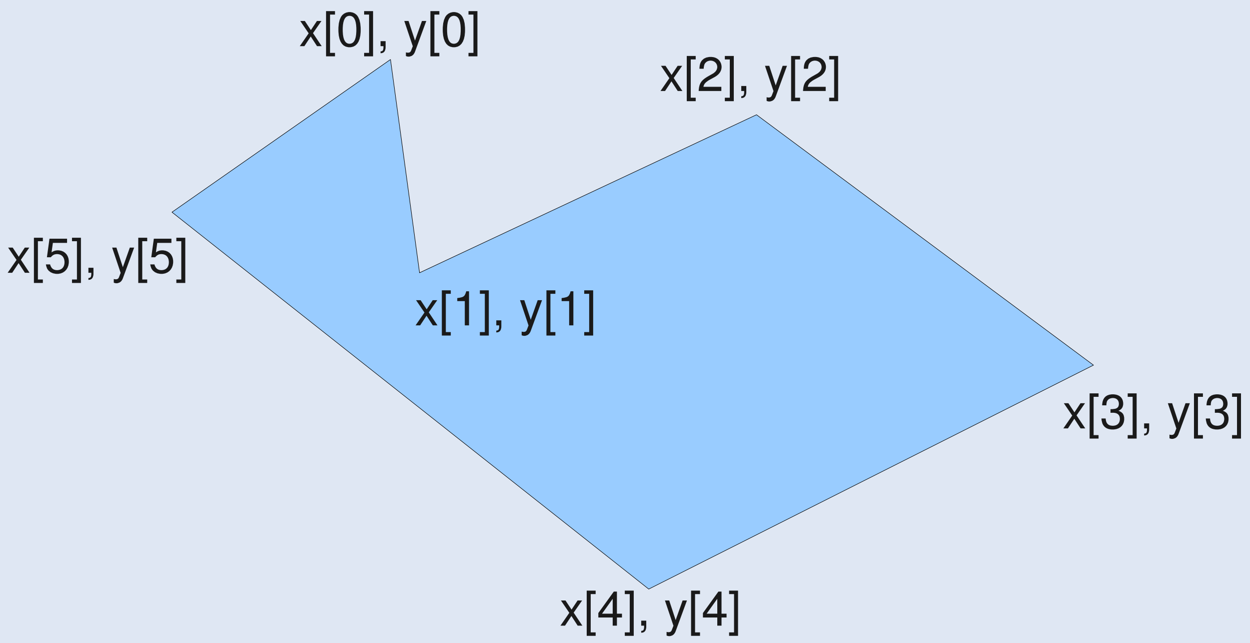


Drawing poly-shapes

- Drawing polyshapes in Java is different then the other shapes because it takes in two arrays of x and y coordinates.

```
drawPolygon(xPoints: int[], yPoints: int[], nPoints int)
fillPolygon(xPoints: int[], yPoints: int[], nPoints int)
drawPolyline(xPoints: int[], yPoints: int[], nPoints: int)
```

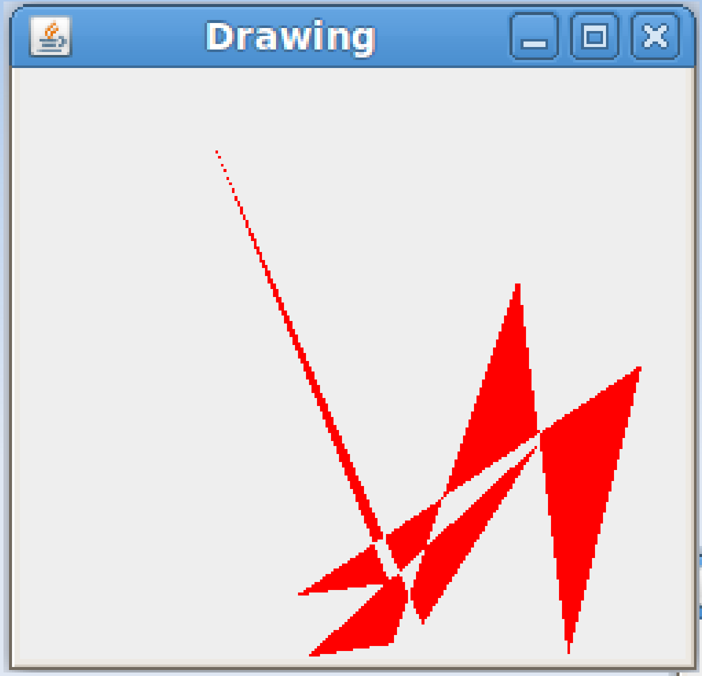




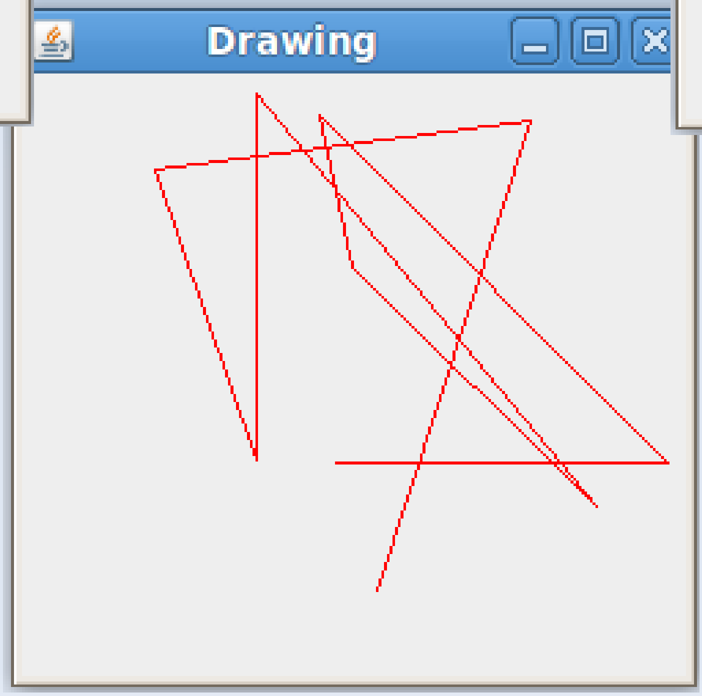
Drawing a poly-shape

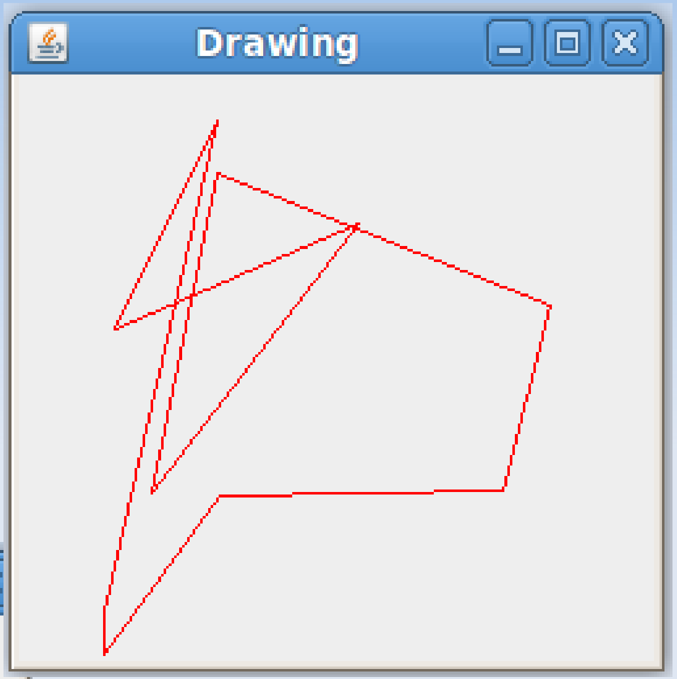
```
import javax.swing.*;
import java.util.*;
import java.awt.*;

public class DrawPanel extends JPanel{
    Random rng = new Random();
    int numberOfPoints = 10;
    int[] xPoints = new int[numberOfPoints];
    int[] yPoints = new int[numberOfPoints];

    protected void paintComponent(Graphics g){
        super.paintComponent(g);

        g.setColor(Color.RED);

        for (int i = 0; i < numberOfPoints; i++){
            xPoints[i] = rng.nextInt(getWidth());
            yPoints[i] = rng.nextInt(getHeight());
        }

        g.drawPolygon(xPoints, yPoints, numberOfPoints);
    }
}
```

Drawing a poly-shape



drawPolyline()





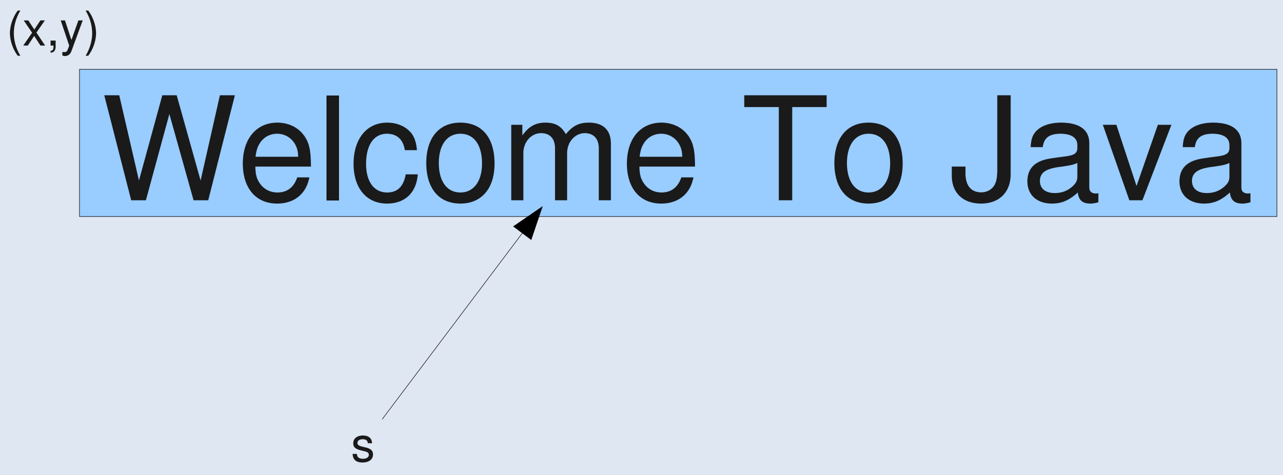
fillPolygon()

drawPolygon()

Drawing a string

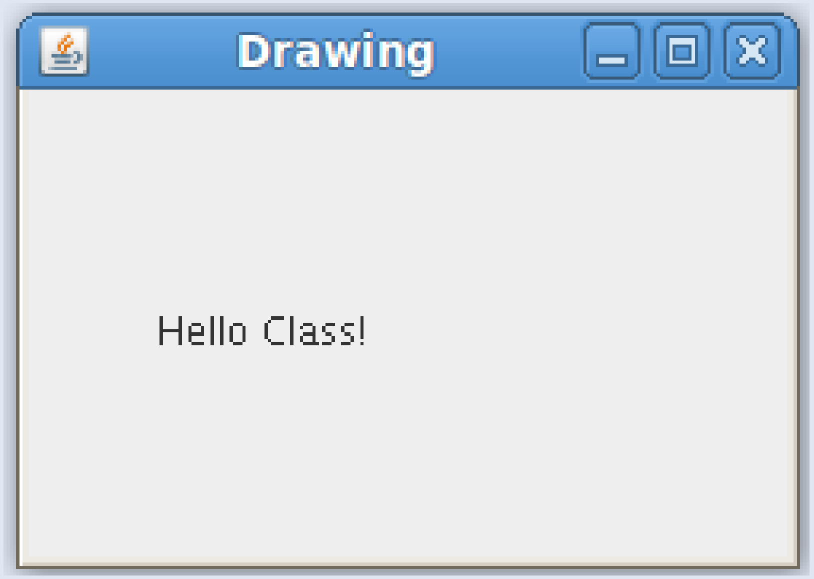
- You don't just have the ability to draw shapes, but also draw a string on the screen. This can be useful to display messages to the user.

```
drawString(s: String, x: int, y: int)
```



Drawing a string

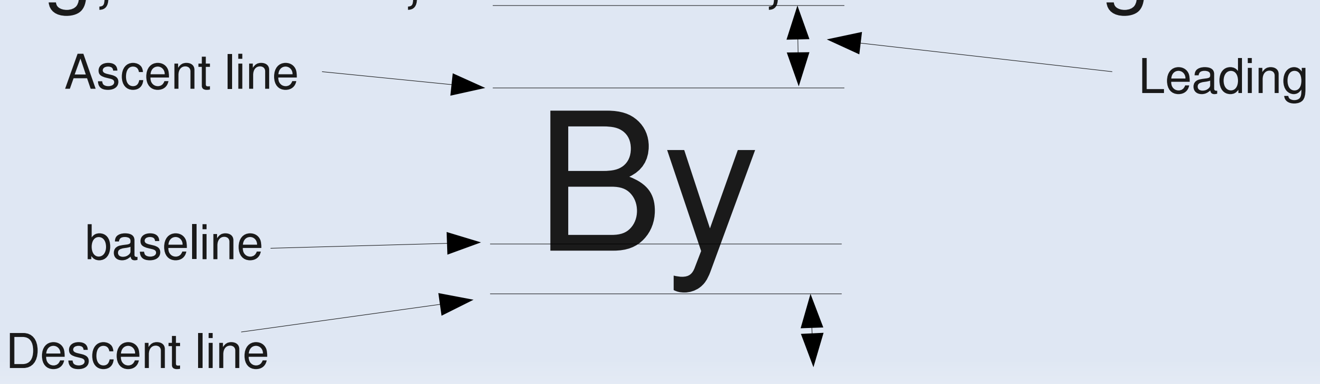
```
import javax.swing.*;
import java.util.*;
import java.awt.*;

public class DrawPanel extends JPanel{

   protected void paintComponent(Graphics g){
      super.paintComponent(g);

      g.drawString("Hello Class!", 40, 80);
   }
}
```



```
Hello Class!
```

Using FontMetrics

- Drawing regular strings is okay, but if you want to have more control over the placement of a string, or what it looks like, you need to use the FontMetrics class
- Each font has a FontMetrics object that describes that fonts meta data. Things like Leading, Ascent, Descent, and Height.



Using FontMetrics

- To get the FontMetrics of a specific font you can use the following methods defined in the Graphics class

```
getFontMetrics(Font font)
GetFontMetrics()
```

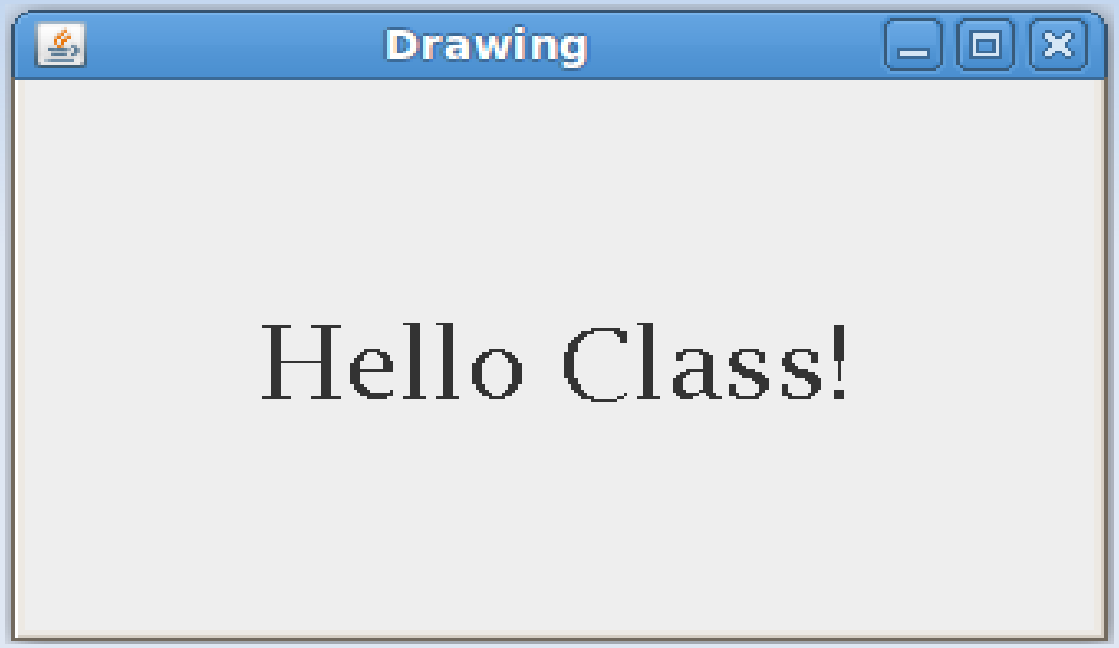
These will return the FontMetrics object for the specified font, and the current font respectively.

Using FontMetrics

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

public class DrawPanel extends JPanel{
    String message = "Hello Class!";

    protected void paintComponent(Graphics g){
        super.paintComponent(g);
        int xCenter = getWidth() / 2;
        int yCenter = getHeight() / 2;

        Font font = new Font("Serif", Font.PLAIN, 32);
        g.setFont(font);
        FontMetrics fm = g.getFontMetrics();
        int stringWidth = fm.stringWidth(message) / 2;
        int stringHeight = fm.getAscent() / 2;

        g.drawString(message, xCenter - stringWidth, yCenter +
                                                      stringHeight);
    }
}
```

Center the string on the screen

Using FontMetrics


Drawing

Hello Class!

Painting Images

- We learned how to create image icons and display them in labels and buttons.
- Now we will examine how to draw an image directly on the screen
- To draw an image, we need use the java.awt.Image class

```
ImageIcon icon = new ImageIcon("images/flag.png");
Image imageToDraw = icon.getImage();
```

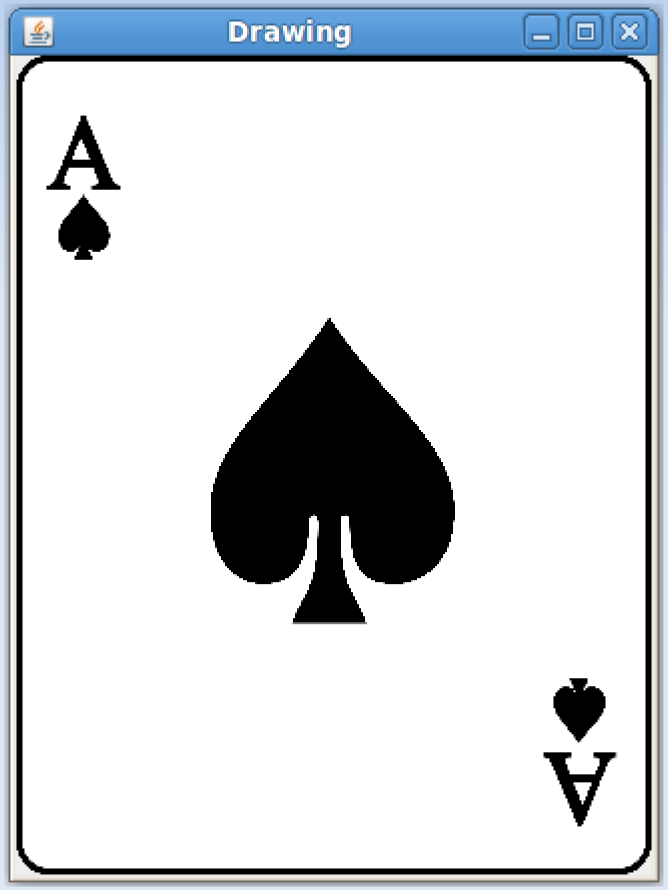
Painting Images

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

public class DrawPanel extends JPanel{
    ImageIcon icon = new ImageIcon("card.png");
    Image image = icon.getImage();

    protected void paintComponent(Graphics g){
        super.paintComponent(g);

        g.drawImage(image, 0,0,getWidth(), getHeight(), this);
    }
}
```

You can draw the image in the following ways

drawImage(i: image, x: int, y: int, bgcolor: Color, observer: ImageObserver)
drawImage(i: image, x: int, y: int, observer: ImageObserver)
drawImage(i: image, x: int, y: int, width: int, height: int, observer: ImageObserver)
drawImage(i: image, x: int, y: int. width: int, height: int, bgcolor: Color, observer: ImageObserver)

Painting Images



ImageObservers

- ImageObserver is an asynchronous update interface that receives notifications of image information a the image is constructed.
- The Component class implements ImageObserver, therefore every GUI component is an instance of ImageObserver.
- The ImageObserver will monitor the loading of a non local resource and call repaint() on the component as the image is loaded into memory

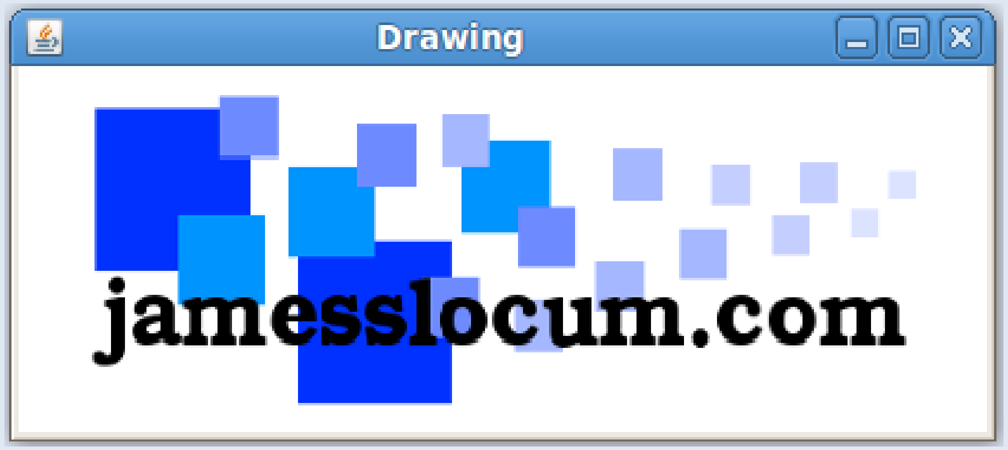
ImageObservers

- This is especially useful for images being loaded over the Internet.

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

public class DrawPanel extends JPanel{
   ImageIcon icon = null;
   Image image = null;

   public DrawPanel(){
      try{
         setBackground(Color.WHITE);
         icon = new ImageIcon(new URL("http", "www.jamesslocum.com",
                      80, "/images/banner/banner_top_scale.png"));
         image = icon.getImage();
      }catch (MalformedURLException e){
         e.printStackTrace();
      }
   }

   protected void paintComponent(Graphics g){
      super.paintComponent(g);

      g.drawImage(image, 0,0,getWidth(), getHeight(), this);
   }
}
```

ImageObserver

```
import javax.swing.*;

public class DrawFrame extends JFrame{

    public DrawFrame(){
        add(new DrawPanel());
    }

    public static void main(String[] args){
        JFrame frame = new DrawFrame();
        frame.setTitle("Drawing");
        frame.setSize(417,180);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```



ImageObserver

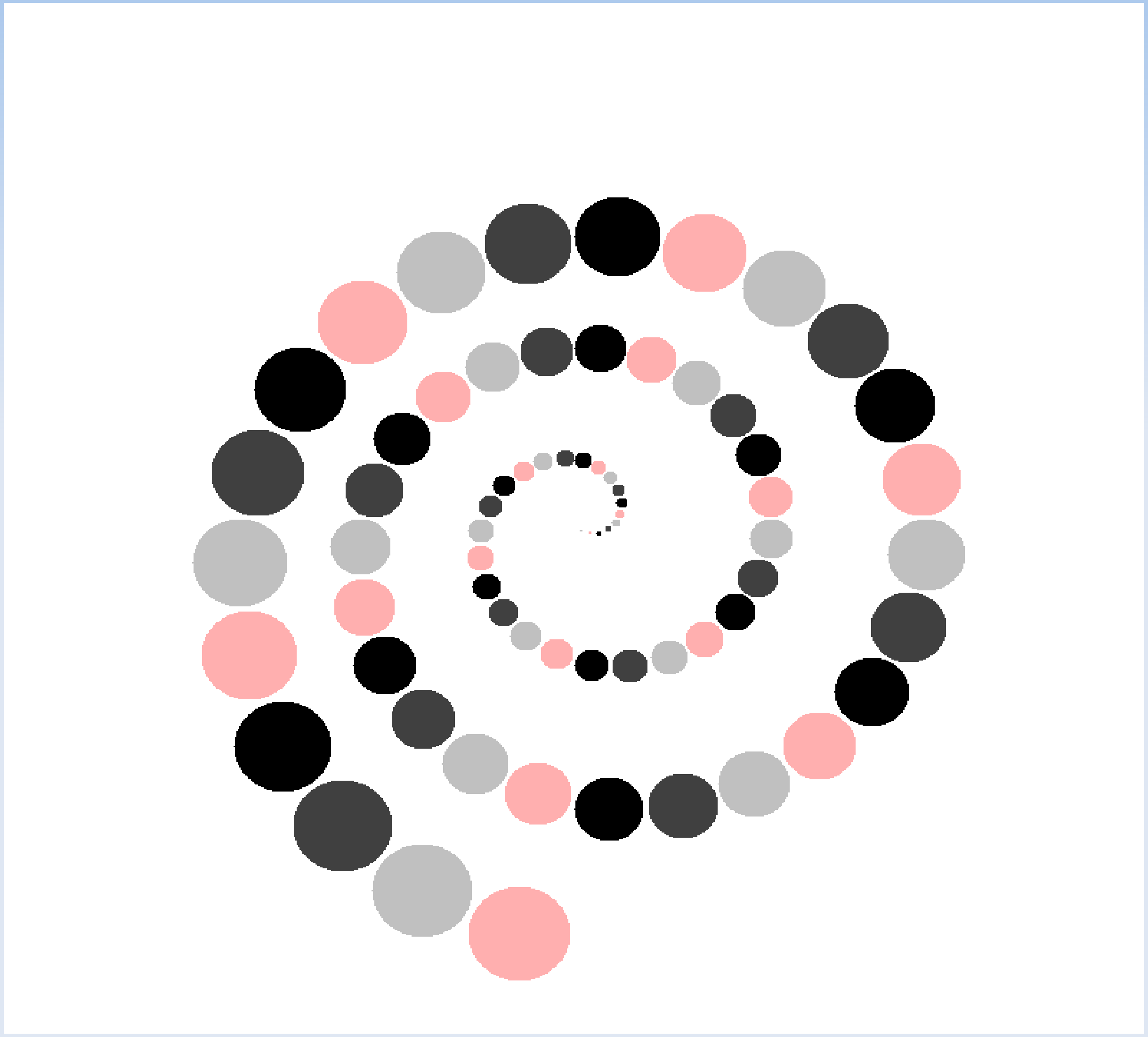
- If you run the previous program, you will see a 1 – 3 second pause depending on your Internet connection while it loads the image into memory.
- This pause is the paint method waiting for the ImageObserver to say the image has been loaded and is ready for painting.

Lab Assignment

- 14.9 Page 479 – Drawing Fans

  Homework:
- 14.11** Page 479 - Plotting the square Function
- 14.13** Page 480 – Plotting functions using abstract methods

Advanced Homework



Produce a spiral of circles using an algorithm. (Not a bunch of g.fillOvals())