Introductory Graphics in Java

Swing
JFrame objects
JPanel components

Easy GUI Interfaces: JOptionPane

import javax.swing;
public static void main (String[] args) {
  JOptionPane.showMessageDialog (null, "Hello World!");
} // end method main

import javax.swing;
public static void main (String[] args) {
  String filename = JOptionPane.showInputDialog (null, "Enter filename:");
  System.out.println(filename);
} // end method main

Swing library: JFrame

JFrame
ToolBar
MenuBar
JPanel ContentPane
JComponent
JComponent
JComponent
Position determined by:
- order in list
- layout type
Creating a window

- The JFrame class represents an object that you can display on the screen.
  - A JFrame holds a title bar, menu bar, and content pane.
  - Essentially JFrame is your "standard" abstract window object.
- You can "nest" components in each other. Syntax is like ArrayList.
  - Swing library classes exist for: buttons, menus, radio buttons, labels, text entry areas, check boxes, and other standard GUI widgets.

- After adding components, call methods to layout/display the window:

```
import javax.swing.*

JFrame frame = new JFrame(); // Step 1: Make a window frame
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel contentPane = (JPanel) frame.getContentPane();
JLabel label = new JLabel("Hello World!"); // Step 2: Make components
contentPane.add(label); // Step 3: Nest components
frame.setSize(300,300); // Step 4: Display frame
frame.setVisible(true);
```

Making your own component: extend JPanel

- To put your own images on the screen, you need to make a 2D graphics component (a paintable widget).
- The easiest way to do this is to extend the JPanel class and override its behavior with the specific behavior of your component.
- You only need to override paintComponent()!
- You will never call paintComponent yourself (although you can ask for a repaint()).

```
import java.awt.*; // Old library classes, you still need them
import javax.swing.*; // New library classes, Swing extends AWT

class MyComponent extends JPanel {
    public void paintComponent(Graphics g) {
        // your code to tell "g" how to "paint" your object goes here
    } // end method paintComponent
} // end class MyComponent
```

Painting a component with a graphics object

```
public void paintComponent(Graphics g) {
    g.setColor(Color.red); // Dip the paintbrush in red
    g.fillRect(20,50,100,100); // Paint a 100x100 rectangle @ (20,50)
} // end method paintComponent
```

```
public void paintComponent(Graphics g) {
    Image image = new ImageIcon("rose.jpg").getImage();
    g.drawImage(image,2,2,this); // end method
```

```
public void paintComponent(Graphics g) {
    g.setColor(Color.black); // set color black
    g.fillRect(0,0,this.getWidth(),this.getHeight()); // paint background
    int red = (int) (Math.random() * 255);
    int green = (int) (Math.random() * 255);
    int blue = (int) (Math.random() * 255);
    Color randomColor = new Color(red, green, blue);
    g.setColor(randomColor); // set RGB color randomly
    g.fillOval(70,70,100,100); // paint 100x100 oval @ (70,70)
} // end method paintComponent
```
**Graphics classes**

- The argument to `paintComponent` is type `java.awt.Graphics`.
- `g` IS-A Graphics
- In the Swing library, `g` is actually an instance of the subclass `javax.swing.Graphics2D` which extends `java.awt.Graphics`.
- Thus, the object `g` has access to Graphics2D methods, but you'll need to typecast to get access them!

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

public class BouncingBallPanel extends JPanel {
    int x = 100; int y = 100; int ballSize = 20; int run = 1; int rise = 3;

    public void paintComponent(Graphics g) {
        g.setColor(Color.black);
        g.fillRect(0,0,getWidth(),getHeight());
        g.setColor(Color.red);
        g.fillOval(x,y,ballSize,ballSize);  // method paintComponent (required)
    }

    public void move () {
        if (x < 0 || x > getWidth() - ballSize) {
            run = -run;
        }
        if (y < 0 || y > getHeight() - ballSize) {
            rise = -rise;
        }
        x += run;
        y += rise;
    }
}
```

### Simple animation: Bouncing Ball

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

public class Main {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300,300);
        frame.setVisible(true);
        BouncingBallPanel ballPanel = new BouncingBallPanel();
        frame.getContentPane().add(ballPanel);
        while (true) {
            pause();
            ballPanel.move();
            ballPanel.repaint();
        }
    }
}
```

### Simple animation: Test Class

```
import javax.swing.*;

public class Main {
    public static void pause () {
        try {
            Thread.sleep(5);  // wait 5ms
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main (String[] args) {
        // method main
    }
}
```

---

**Simple animation: Bouncing Ball**

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

public class BouncingBallPanel extends JPanel {
    int x = 100; int y = 100; int ballSize = 20; int run = 1; int rise = 3;

    public void paintComponent(Graphics g) {
        g.setColor(Color.black);
        g.fillRect(0,0,getWidth(),getHeight());
        g.setColor(Color.red);
        g.fillOval(x,y,ballSize,ballSize);  // method paintComponent (required)
    }

    public void move () {
        if (x < 0 || x > getWidth() - ballSize) {
            run = -run;
        }
        if (y < 0 || y > getHeight() - ballSize) {
            rise = -rise;
        }
        x += run;
        y += rise;
    }
}
```

### Simple animation: Test Class

```
import javax.swing.*;

public class Main {
    public static void pause () {
        try {
            Thread.sleep(5);  // wait 5ms
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main (String[] args) {
        // method main
    }
}
```
Example: Bouncing balls

- How can we add multiple balls?
  - How can we allow differentiation? (color, speed, initial position, etc)
- How can we detect collisions and have the balls respond appropriately?