

Graphics

Object Orientated Programming in Java

Benjamin Kenwright

Outline

■ Essential Graphical Principals

- ▷ JFrame Window (Popup Windows)
- ▷ Extending JFrame
- ▷ Drawing from paintComponent
- ▷ Drawing images/text/graphics

■ Today's Practical

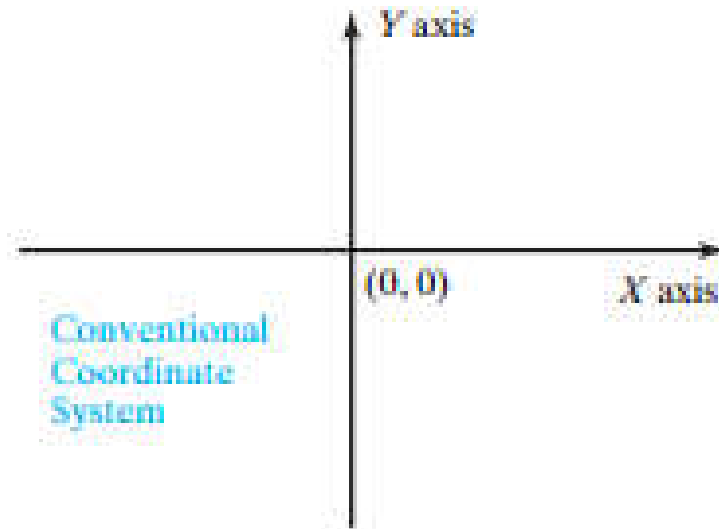
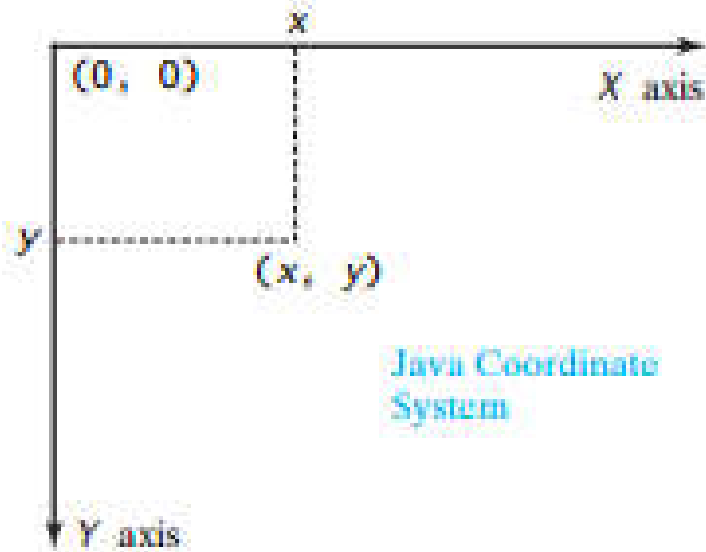
■ Review/Discussion

Graphics

- How do you draw `complex' diagrams?
- Customize/control what is drawn on our windows
 - ▷ Animated clocks, bar-charts, images,



Graphical Coordinate System



JFrame

■ Simple GUI Window

- ▷ Lets us perform simple drawing operations

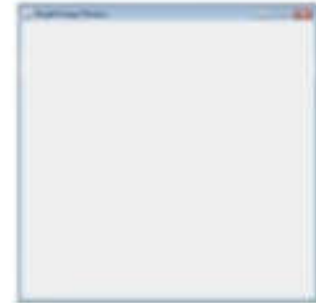
■ Basic Usage

- ▷ Instantiate the JFrame and assign text for the title bar
 - `JFrame frame = new JFrame('Title');`
- ▷ Specify size
 - `frame.setSize(width, height)`
- ▷ Designate that closing the window ends the program
 - `frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`
- ▷ Show the window
 - `frame.setVisible(true);`

Using JFrame

```
import javax.swing.*;

public class Popup1 {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Simple Popup Window");
        frame.setSize(500,500);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```



Extending JFrame

```
public class Popup2 extends JFrame {  
    public Popup2() {  
        super("Simple Popup Window");  
        setSize(500,500);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setVisible(true);  
    }  
  
    public static void main(String[] args) {  
        new Popup2();  
    }  
}
```



Basic Drawing

Make a subclass of JPanel

```
public class MyPanel extends JPanel
```

Override the paintComponent method

```
@Override
```

```
protected void paintComponent(Graphics g) { ... }
```

Call super.paintComponent

```
super.paintComponent(g);
```

- Does default behavior like handling opacity

Use the supplied Graphics object to draw

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

- (0,0) is *top* left corner, x goes to right, y goes *down*

Assign the panel as the content of the frame

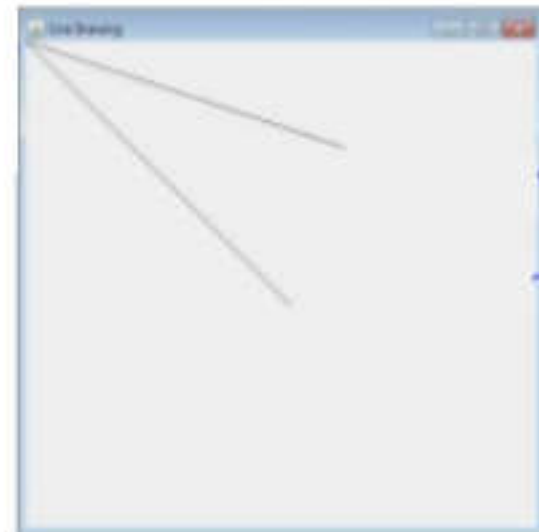
```
frame.setContentPane(new MyPanel());
```


Example JFrame

```
public class LineFrame extends JFrame {  
    public LineFrame() {  
        super("Line Drawing");  
        setContentPane(new LinePanel());  
        setSize(500,500);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setVisible(true);  
    }  
  
    public static void main(String[] args) {  
        new LineFrame();  
    }  
}
```

Example JPanel

```
public class LinePanel extends JPanel {  
    @Override  
    protected void paintComponent(Graphics g) {  
        super.paintComponent(g);  
        g.drawLine(0, 0, 250, 250);  
        g.drawLine(0, 0, 300, 100);  
    }  
}
```



Drawing Methods

Drawing lines and shapes

Use supplied Graphics object

`g.drawLine`, not `drawLine`

Understand coordinate system

(0,0) is top left corner, x goes to right, y goes down

Make outlines or solid shapes

`g.drawRect`: outline, `g.fillRect`: solid

Setting default features of JPanel

Do not use Graphics object

`setBackground`, `setForeground`, `setFont`, etc.

Usually called in constructor, not in `paintComponent`

Remember to call `super.paintComponent`

Background colors have no effect unless window is opaque (default) and you call `super.paintComponent`

Graphics Class Methods

drawString(string, left, bottom)

Draws a string in the current font and color with the *bottom left* corner of the string at the specified location

One of the few methods where the y coordinate refers to the bottom of shape, not the top. But y values are still with respect to the *top left* corner of the applet window

drawRect(left, top, width, height)

Draws the outline of a rectangle (1-pixel border) in the current color

fillRect(left, top, width, height)

Draws a solid rectangle in the current color

drawLine(x1, y1, x2, y2)

Draws a 1-pixel-thick line from (x1, y1) to (x2, y2)

Graphics Class Methods

drawOval, fillOval

Draws an outlined or solid oval, where the arguments describe a rectangle that bounds the oval

drawPolygon, fillPolygon

Draws an outlined or solid polygon whose points are defined by arrays or a Polygon (a class that stores a series of points). By default, polygon is closed; to make an open polygon use the drawPolyline method

drawImage

Draws an image

Image usually created with Toolkit.getDefaultToolkit().getImage – see upcoming example

Supports JPEG, GIF (including animated GIF), or PNG

Graphics Class Methods

setColor, getColor

Specifies the **foreground color** prior to drawing operation

By default, the graphics object receives the foreground color of the window

As set via setForeground from the constructor

Java has 16 predefined colors (Color.RED, Color.BLUE, etc.) or create your own color: new Color(r, g, b)

Changing the color of the Graphics object affects only the drawing that explicitly uses that Graphics object

To make permanent changes, call the *panel's* setForeground method

Question

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

public class MyPaint extends JFrame {
    public static void main(String[] args) {
        MyPaint frame = new MyPaint();
        frame.setTitle("MyPaint");
        frame.setContentPane(new MyPanel());
        frame.setSize(200, 100);
        frame.setLocationRelativeTo(null); // Center the frame
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}

class MyPanel extends JPanel
{
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawString("Hello", 0, 40);
        g.drawLine(0, 0, 50, 50);
    }
}
```

What would be the output of this program?

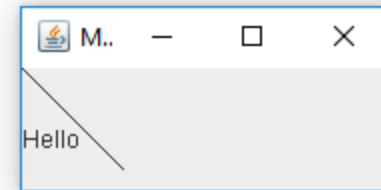
Sketch on paper what the output would look like

Answer

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

public class MyPaint extends JFrame {
    public static void main(String[] args) {
        MyPaint frame = new MyPaint();
        frame.setTitle("MyPaint");
        frame.setContentPane(new MyPanel());
        frame.setSize(200, 100);
        frame.setLocationRelativeTo(null); // Center the frame
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}

class MyPanel extends JPanel
{
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawString("Hello", 0, 40);
        g.drawLine(0, 0, 50, 50);
    }
}
```



```
javac MyPaint.java
java -cp . MyPaint
```


Example

```
public class ShapeFrame extends JFrame {  
    public ShapeFrame() {  
        super("Drawing Shapes");  
        setContentPane(new ShapePanel(Color.YELLOW));  
        setSize(500,500);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setVisible(true);  
    }  
}
```

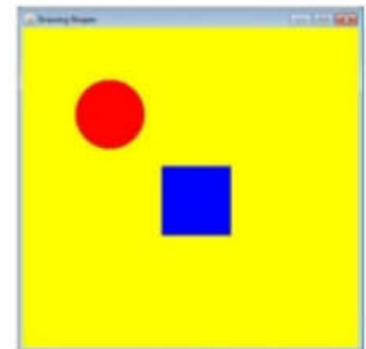
1

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

```
public class ShapePanel extends JPanel {  
    public ShapePanel(Color bgColor) {  
        setBackground(bgColor);  
    }  
}
```

2

```
@Override  
protected void paintComponent(Graphics g) {  
    super.paintComponent(g);  
    g.setColor(Color.RED);  
    g.fillOval(75, 75, 100, 100);  
    g.setColor(Color.BLUE);  
    g.fillRect(200, 200, 100, 100);  
}
```



Question

**What will the output
for the following program?**

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

public class MyPaint extends JFrame {
    public static void main(String[] args) {
        MyPaint frame = new MyPaint();
        frame.setTitle("MyPaint");
        frame.setContentPane(new MyPanel());
        frame.setSize(400, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}

class MyPanel extends JPanel {
    public MyPanel(){
        setBackground(Color.YELLOW);
    }
    protected void paintcomponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.RED);
        g.fillOval(75,75,100,100);
        g.setColor(Color.BLUE);
        g.fillRect(200,200,100,100);
    }
}
```

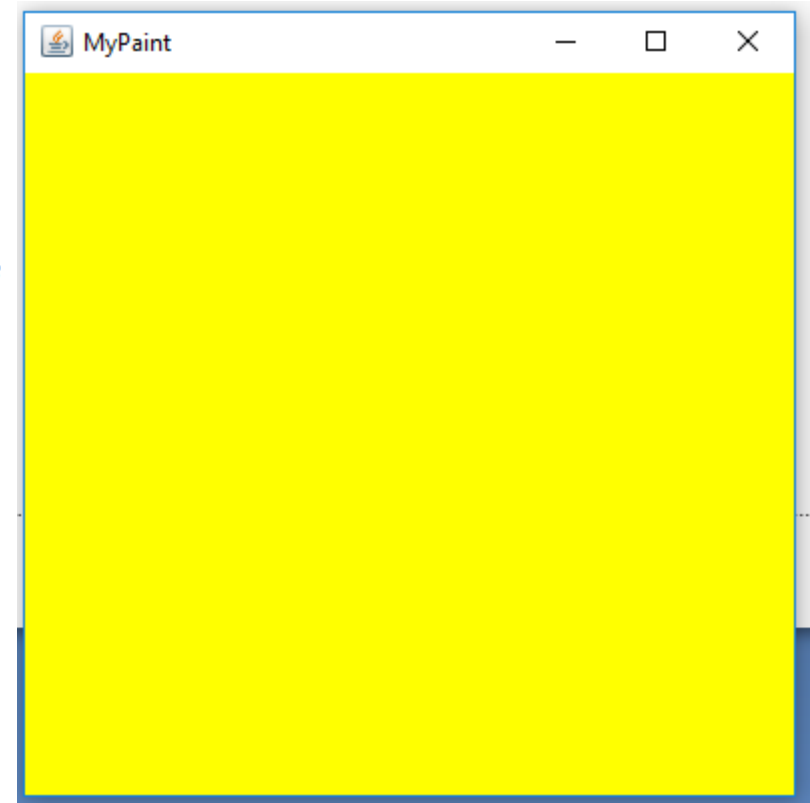
Answer

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

public class MyPaint extends JFrame {
    public static void main(String[] args) {
        MyPaint frame = new MyPaint();
        frame.setTitle("MyPaint");
        frame.setContentPane(new MyPanel());
        frame.setSize(400, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}

class MyPanel extends JPanel {
    public MyPanel() {
        setBackground(Color.YELLOW);
    }

    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.RED);
        g.fillOval(75, 75, 100, 100);
        g.setColor(Color.BLUE);
        g.fillRect(200, 200, 100, 100);
    }
}
```



Fixing Problems

Fix 1: Use @Override

@Override

```
protected void paintcomponent(Graphics g) { ... }
```

Code will fail at compile time, not work oddly at run time

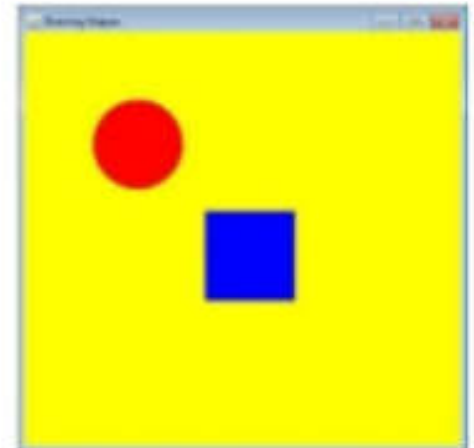
"The method paintcomponent(Graphics) of type ShapePanel must override or implement a supertype method"

Fix 2: call it paintComponent

@Override

```
protected void paintComponent(Graphics g) {  
    ...  
}
```

@Override has no effect other than documentation on *correct* code, but with *incorrect* code it helps catch errors at compile time



Drawing Images

Basic syntax

Define instance variable of type Image

```
private Image image;
```

Use Toolkit to get the image (usually in constructor)

```
image = Toolkit.getDefaultToolkit().getImage(...)
```

You can supply either a path relative to root of the code, or a URL

Resource can be JPEG, GIF, or PNG

Draw the image in paintComponent

```
g.drawImage(image, left, top, windowImageDrawnOn);
```

Special notes

getImage returns immediately; the image is downloaded in a background thread

If image is incomplete when paintComponent is called, paintComponent is automatically called again later

If you use a URL for the image location, you need a try/catch block

Images

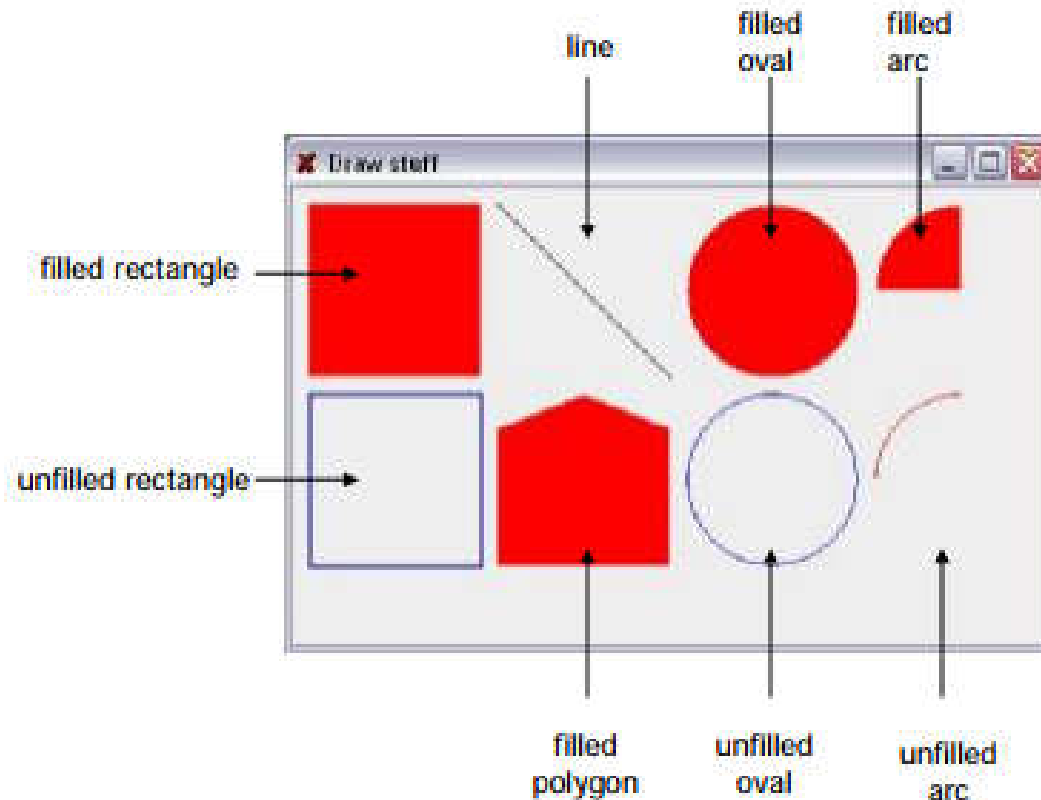
```
public class ImageFrame1 extends JFrame {  
    public ImageFrame1(String relativePath) {  
        super("Image Drawing");  
        setContentPane(new ImagePanell1(relativePath));  
        setSize(400,400);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setVisible(true);  
    }  
  
    public static void main(String[] args) {  
        new ImageFrame1("images/Java-Man.gif");  
    }  
}
```


Review Basic Graphic Methods

Method Name	Description
<code>g.drawLine(x1,y1,x2,y2);</code>	Line between points (x1,y1) and (x2,y2)
<code>g.drawOval(x,y,width,height);</code>	Outline largest oval that fits in a box of size width*height with top-left at (x,y)
<code>g.drawRect(x,y,width,height);</code>	Outline rectangle of size width*height with top-left at (x,y)
<code>g.drawString(text,x,y);</code>	text with bottom-left at (x,y)
<code>g.fillOval(x,y,width,height);</code>	Fill largest oval that fits in a box of size width*height with top-left at (x,y)
<code>g.fillRect(x,y,width,height);</code>	Fill rectangle of size width*height with top-left at (x,y)
<code>g.setColor(Color);</code>	Set <code>Graphics</code> to paint any following shapes in the given color.

Question

- Write down the graphics methods you'd use to draw the following output:



Question

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

public class MyPaint extends JFrame {
    public static void main(String[] args) {
        MyPaint frame = new MyPaint();
        frame.setTitle("MyPaint");
        frame.setContentPane(new JPanel());
        frame.setSize(400, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}

class JPanel extends JPanel {
    Color[] colors = new Color[]{Color.YELLOW, Color.RED};
    int colorIndex = 0;

    public JPanel() {
        addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent me) {
                colorIndex = (colorIndex+1)%2;
                repaint();
            }
        });
    }

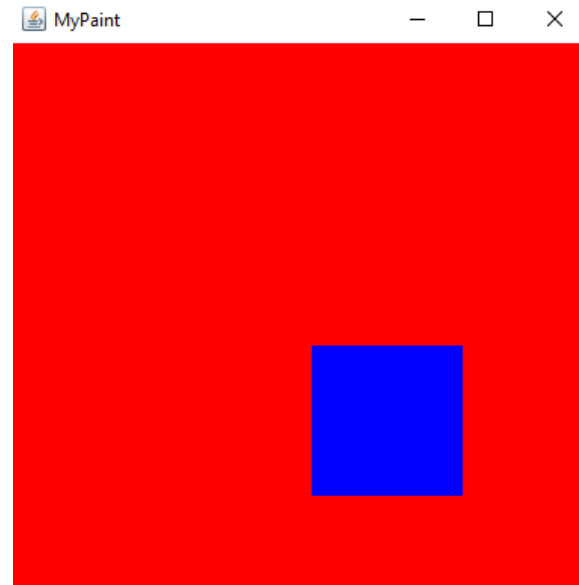
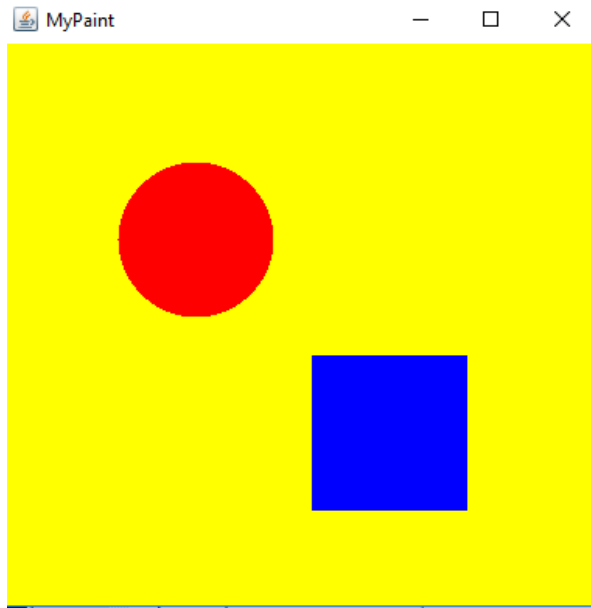
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        setBackground(colors[colorIndex]);
        g.setColor(Color.RED);
        g.fillOval(75, 75, 100, 100);
        g.setColor(Color.BLUE);
        g.fillRect(200, 200, 100, 100);
    }
}
```

**What would the following
program output?**

javac MyPaint.java
java -cp . MyPaint

Answer

■ Left mouse button is pressed



Force Redraw

- To notify Java to redraw our window (i.e., call `paintComponent`)
 - ▷ call `'repaint()'`
- For example, add new graphics each time the mouse button is pressed

Review

- Graphics can be drawn using a class which extends **JPanel**
- Swing will call the **paintComponent** method to draw:
 - ▷ `protected void paintComponent(Graphics g);`
- There are a variety of drawing methods:
 - ▷ `drawLine(int x1, int y1, int x2, int y2);`
 - ▷ `drawRect(int x, int y, int w, int h);`
 - ▷ `drawOval(int x, int y, int w, int h);`
 - ▷ `drawPolygon(int[] xpoints, int[] ypoints, int npoints);`

Other Graphics Capabilities

■ Lots of GUI Controls

- ▷ Buttons, sliders, checkboxes, tables, etc
 - which you can override and improve

■ Rich 2D drawing

- ▷ Line properties, translucent drawing, rotating, scaling images, coordinate transforms, ...

■ Animation

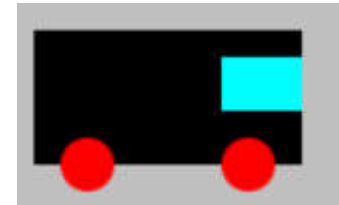
- ▷ Double-buffering, combining drawing with threads, ..

Examples

```
setBackground(Color.LIGHT_GRAY);  
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.GREEN);  
Polygon poly = new Polygon();  
poly.addPoint(10, 90);  
poly.addPoint(50, 10);  
poly.addPoint(90, 90);  
g.fillPolygon(poly);
```



Experiment

- Only scratched the surface of Graphics
 - ▷ Very simple windows and graphics
- Combine the graphics with GUI components
 - ▷ Buttons/menus
- Custom 'interfaces' components
 - ▷ e.g., Override default draw method for a button to create a new improved button look
- Events
 - ▷ Mouse button presses (drawing program)
 - ▷ Timing (animations)

Challenge

- Create a simple vector drawing program in Java
- Clear/load/save drawings
- Select buttons for different operations
 - ▷ line, circle, color, ...
- Detect mouse button presses
 - ▷ draw lines, drag shapes, pick objects

Summary

- Overview Essential Graphical Principles
 - ▷ JFrame, JPanel, paintComponent, @Override (Checking)
- Hands-On/Practical
- Today is about Graphics with Java
 - ▷ Drawing

This Week

- Read Associated Chapters

- ▷ Comprehensive Edition (Additional Chapters)

- Review Slides

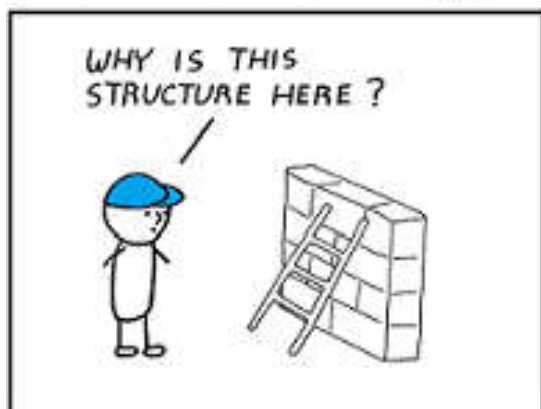
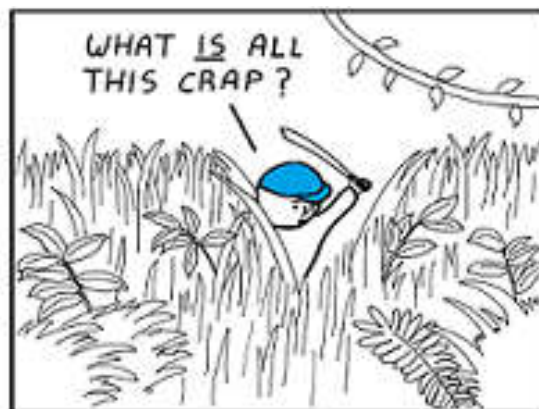
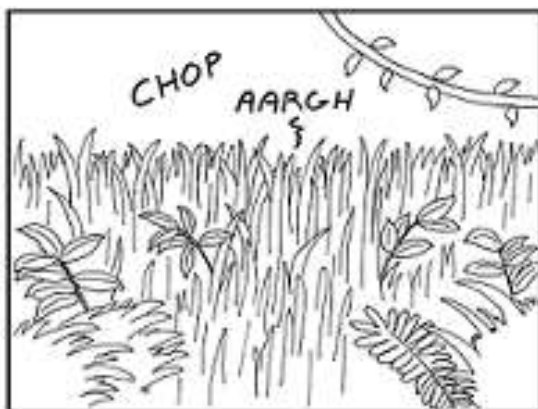
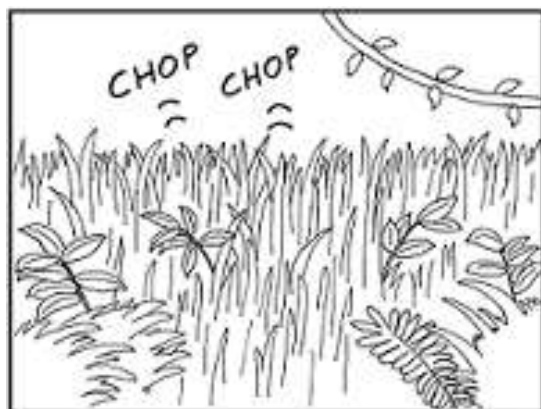
- Java Exercises

- Online Quizzes

Exercises

- Chapter exercises 15.1-15.3
- Comment your code
- Remember to zip the files together
 - ▷ The name of the .zip file should be your student number
 - ▷ (e.g., 39293923923.zip)

Questions/Discussion



I hate reading
other people's code.