Inheritance in Action: Drawing

CS110: Introduction to Computer Science

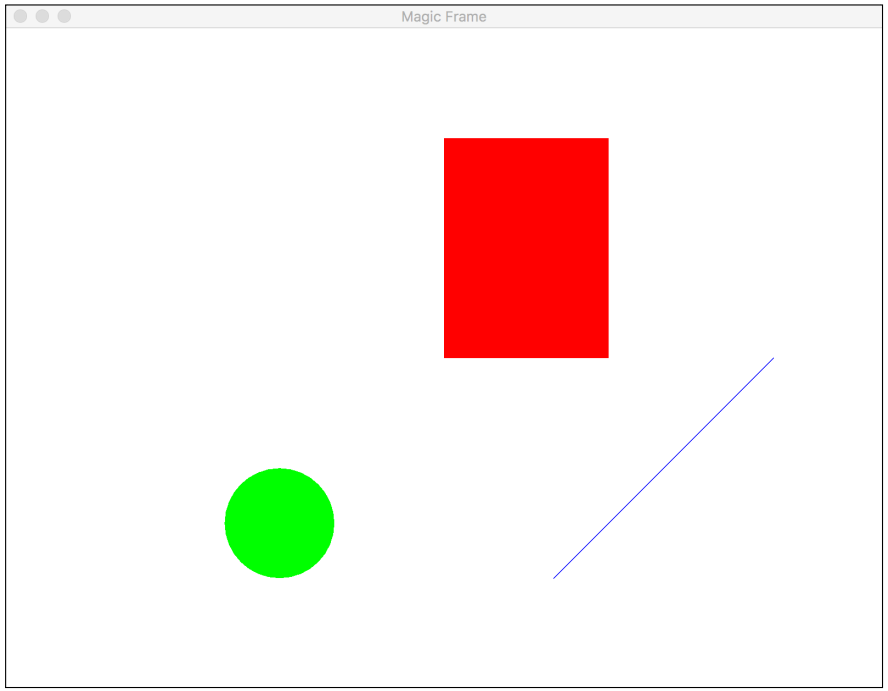
Finally...

- Here is what you came to learn.How to REALLY draw.
- Buckle up, Java's gonna make us jump through a LOT of hoops.

What do we see?

- Yes...Rectangle, Circle, Line
- They all live in a **panel**.
- The panel lives in a **frame**.

- We create this in the order: Frame, Panel, and Shapes



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

public class Starter {

    public static void main(String[] args){
        JFrame frame = new JFrame("The Frame We See");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);









    }
}
```

Make a JFrame

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

public class Starter {

    public static void main(String[] args){
        JFrame frame = new JFrame("The Frame We See");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

        JPanel mainPanel = new JPanel();
        mainPanel.setPreferredSize(new Dimension(800,600));
        mainPanel.setBackground(Color.blue);
        frame.getContentPane().add(mainPanel);
        frame.pack();









    }
}
```

Make a `JPanel` inside a `JFrame`

Make a JPanel inside a JPanel

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

public class Starter {

    public static void main(String[] args){
        JFrame frame = new JFrame("The Frame We See");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

        JPanel mainPanel = new JPanel();
        mainPanel.setPreferredSize(new Dimension(800,600));
        mainPanel.setBackground(Color.blue);
        frame.getContentPane().add(mainPanel);
        frame.pack();

        JPanel secondPanel = new JPanel();
        secondPanel.setPreferredSize(new Dimension(100,600));
        secondPanel.setBackground(Color.red);
        mainPanel.add(secondPanel);




    }
}
```

Make a JPanel inside a JPanel

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

public class Starter {

    public static void main(String[] args){
        JFrame frame = new JFrame("The Frame We See");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

        JPanel mainPanel = new JPanel();
        mainPanel.setPreferredSize(new Dimension(800,600));
        mainPanel.setBackground(Color.blue);
        frame.getContentPane().add(mainPanel);
        frame.pack();

        JPanel secondPanel = new JPanel();
        secondPanel.setPreferredSize(new Dimension(100,600));
        secondPanel.setBackground(Color.red);
        mainPanel.add(secondPanel);

        JPanel thirdPanel = new JPanel();
        thirdPanel.setPreferredSize(new Dimension(650,600));
        thirdPanel.setBackground(Color.green);
        mainPanel.add(thirdPanel);
    }
}
```

Drawing

- How do we draw?
- This is where it becomes complicated.
- In order to draw, we have to make our own class by extending `JPanel`.

paintComponent NOT paintComponents

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

public class DrawingPanel extends JPanel {

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

        g.fillRect(200,300,20,30);
    }
}
```

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

public class Starter {

    public static void main(String[] args){
        JFrame frame = new JFrame("The Frame We See");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

        JPanel mainPanel = new JPanel();
        mainPanel.setPreferredSize(new Dimension(800,600));
        mainPanel.setBackground(Color.blue);
        frame.getContentPane().add(mainPanel);
        frame.pack();

        DrawingPanel dPanel = new DrawingPanel();
        dPanel.setPreferredSize(new Dimension(600,600));
        dPanel.setBackground(Color.white);
        mainPanel.add(dPanel);

    }
}
```

Changing Color?

- Set the color, than draw

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

public class DrawingPanel extends JPanel {

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

        g.setColor(Color.red);
        g.fillRect(200,300,20,30);
    }
}
```

Lots and Lots of things to draw

- The Graphics type has tons of things that we can draw.
  - Ovals, lines, string, rounded rectangle
  - Shapes can all be filled or unfilled
    - Little different than Magic.java
    - drawRect() is just the outline
    - fillRect() is what you are use to.
- **Let's use autocomplete to see what we can do.**

Animation

- How do we animate?
- We could use Magic.sleep(), but we're about to kill magic.
- So, let's use the real stuff.
- `Timer`
- We're going to implement `ActionListener` in `DrawingPanel`

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

public class DrawingPanel extends JPanel implements ActionListener {

    int x, y;

    public DrawingPanel(int x, int y){
        this.x = x;
        this.y = y;
    }
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        g.setColor(Color.red);
        g.fillRect(x,y,20,30);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        x+=1;
        y+=1;
        this.repaint();
    }
}
```

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

public class Starter {

    public static void main(String[] args){
        JFrame frame = new JFrame("The Frame We See");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

        JPanel mainPanel = new JPanel();
        mainPanel.setPreferredSize(new Dimension(800,600));
        mainPanel.setBackground(Color.blue);
        frame.getContentPane().add(mainPanel);
        frame.pack();

        DrawingPanel dPanel = new DrawingPanel(20,30);
        dPanel.setPreferredSize(new Dimension(600,600));
        dPanel.setBackground(Color.white);
        mainPanel.add(dPanel);

        Timer t = new Timer(20,dPanel);
        t.start();
    }
}
```

Update the main too.

Now Let's Make It Better with Objects

- What we've done now works well with one rectangle
- But, what if we thought about this as objects.
- What do we have here?
- Rather than storing the "x" and "y" in the panel directly, Let's make a `Rectangle` object.

```java
import java.awt.*;

public class Rectangle {
    private int x,y,width,height;

    public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }

    public void moveBy(int deltaX,int deltaY){
        this.x +=deltaX;
        this.y +=deltaY;
    }

    public void draw(Graphics g){
        g.setColor(Color.red);
        g.fillRect(x,y,width,height);
    }
}
```

Gotta update DrawingPanel

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

public class DrawingPanel extends JPanel implements ActionListener {

    Rectangle r;

    public DrawingPanel(int x, int y){
        r = new Rectangle(x,y,20,30);
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        r.draw(g);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        r.moveBy(1,1);
        this.repaint();
    }
}
```

Why?

- Makes it easier to do things like this:
- Let's make 100 Rectangles!

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

public class DrawingPanel extends JPanel implements ActionListener {
    Rectangle[] rectangles;
    public DrawingPanel(int numberOfRectangles){
        rectangles = new Rectangle[numberOfRectangles];
        for(int i = 0;i < rectangles.length; i++) {
            rectangles[i] = new Rectangle(i * 25, i * 25, 20, 30);
        }
    }
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        for(Rectangle rect: rectangles) {
            rect.draw(g);
        }
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        for(Rectangle rect: rectangles) {
            rect.moveBy(1,1);
        }
        this.repaint();
    }
}
```

Let's make this even Cooler

- Let's have it bounce off walls

```
public class Rectangle {
    private int x,y,width,height, panelWidth,panelHeight;
    private boolean isXIncreasing, isYIncreasing;

    public Rectangle(int x, int y, int width, int height,
                     int panelWidth, int panelHeight) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.panelWidth = panelWidth;
        this.panelHeight = panelHeight;
        this.isXIncreasing = true;
        this.isYIncreasing = true;
    }

    public void moveBy(int deltaX,int deltaY){
        if (isXIncreasing && x + width + deltaX > panelWidth ){
            isXIncreasing = false;
        }
        if (!isXIncreasing && x - deltaX < 0){
            isXIncreasing = true;
        }

        if (isYIncreasing && y + height + deltaX > panelHeight){
            isYIncreasing = false;
        }
        if (!isYIncreasing && y - deltaX < 0){
            isYIncreasing = true;
        }
        if(isXIncreasing) {
            this.x += deltaX;
        } else {
            this.x -= deltaX;
        }
        if(isYIncreasing) {
            this.y += deltaY;
        } else {
            this.y -= deltaY;
        }

    }

    public void draw(Graphics g){
        g.setColor(Color.red);
        g.fillRect(x,y,width,height);
    }
}
```

Now!

- Now that we have that. Let's make 100 of them and make them start randomly

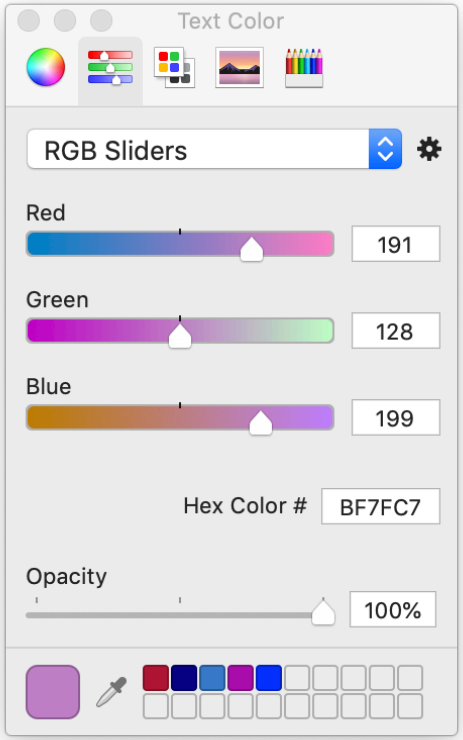
Color!

- We can use more than the built in colors.
- We can crate our own color object using the values for **red**, **green**, **blue**, and **alpha** (aka, *transparency*).

```
int red = 250;
int green = 65;
int blue = 99;
int alpha = 255;
Color c;
c = new Color(red,green,blue,alpha);
```

RGBA

- Each RGBA color is marked between 0 and 255.
- 0 is the absence of the color 255 is the color in full.
- 0 for alpha is completely transparent, 255 for alpha is fully opaque.
- 0,0,0 is black.
- 255,255,255 is white.



Mouse

- What fun is drawing if you can use the mouse.
- If we want to use the mouse, we can implement two additional interfaces: `MouseListener` and `MouseMotionListener`.
- `MouseListener` is for detecting clicks
- `MouseMotionListener` is for detecting mouse movement.
- Let's discuss these in turn.

MouseListener

- We'll implement MouseListener in the panel.
- This requires us to make the methods:
  - `public void mouseClicked(MouseEvent e)`
  - `public void mousePressed(MouseEvent e)`
  - `public void mouseReleased(MouseEvent e)`
  - `public void mouseEntered(MouseEvent e)`
  - `public void mouseExited(MouseEvent e)`
- Don't worry about `mouseEntered()` and `mouseExited()`, they won't matter for what we do
- The most important is `mouseClicked()`

One more thing

- Before we can use the mouse, we have to tell the panel to "listen to itself"
- So, in the constructor for the panel add the line:

```
this.addMouseListener(this);
```

Lets do this

- Let's now add some code to `mouseClicked()`.
- Now, let's use some of the data we get from `MouseEvent`.

`MouseMotionListener`

- The interface `MouseMotionListener` can be used to detect mouse motion and dragging.
- Two methods to implement
  - `public void mouseDragged(MouseEvent e)`
  - `public void mouseMoved(MouseEvent e)`
- Also, we need this to the panel constructor
  - `this.addMouseMotionListener(this);`