Lecture Fourteen – Graphics

- Introduction
- paint() method
- paintComponent() method
- Graphics class
- MyPicture program
- MyPicture code
- MyPicture walkthrough
- Final notes
- Drill

Read Chapter 13
Introduction

- In addition to providing us with easy ways to create simple GUIs, Java also provides us with simple ways to create graphics.

- This lesson will demonstrate how we can add color, change font, and draw very basic shapes in our programs.
paint() method

- Recall our last lesson where we discussed how to create simple applets.

- We used **HowdyApplet** as an example – its purpose was to simply draw the words “Howdy Ags” to our applet’s screen.

- We did this by overriding the **paint()** method which told the applet how it needed to draw itself.
paintComponent() method

- Similarly, we can specify how a particular component (like a JPanel object) needs to be painted.

- While applets use the `paint()` method, components use a method called `paintComponent()` to draw graphics to themselves.

- Note that both of these methods are called automatically and should never be called by us. If we need to force the program to draw itself we should call the `repaint()` method.
Both the `paint()` method of `JApplet` and the `paintComponent()` method of `JPanel` take a single argument of type `Graphics`.

This `Graphics` object is used to paint fonts, colors, and shapes to the screen.

You can think of this object as our painting toolbox that allows us to do operations like `setColor()`, `drawString()`, `fillRect()`, etc.
MyPicture program

- We will now take a look at a program that draws a simple picture to the screen.

- It will be an application (not an applet) that will paint a picture to a J Panel.

- Since we are painting to a J Panel and not an applet, we will need to use the paintComponent() method only.

- Type the following code in and get it to work before we discuss the details.
MyPicture program

- Here is what the program will look like once we have it working.

![Image of a house with a 'This is my picture!' message]
import javax.swing.*;       // for our basic swing components
import java.awt.*;          // for our 2D objects
import java.awt.event.*;    // for ActionListener
import java.awt.geom.*;     // for our basic swing components

public class MyPicture extends JFrame implements ActionListener
{
    // declare our components
    private JPanel buttonPanel;
    private JButton yesButton;
    private JButton noButton;
    private JLabel responseLabel;
    private MyPicturePanel myPanel;

    public static void main(String[] args)
    {
        MyPicture p = new MyPicture();
        p.setVisible(true);
        p.setVisible(true);
    }
}
public MyPicture()
{
    // application properties
    super("My First Java Picture");
    setSize(400, 400);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // create components
    myPanel = new MyPicturePanel();
    buttonPanel = new JPanel();
    yesButton = new JButton("Yes");
    yesButton.setBackground(Color.yellow);
    yesButton.setForeground(Color.green);
    noButton = new JButton("No");
    noButton.setBackground(Color.yellow);
    noButton.setForeground(Color.red);
    responseLabel = new JLabel("Do you like my picture?");
    Font sansSerifFont = new Font("SansSerif", Font.PLAIN, 22);
    responseLabel.setFont(sansSerifFont);
    responseLabel.setForeground(Color.blue);
buttonPanel.add(responseLabel);
buttonPanel.add(yesButton);
buttonPanel.add(noButton);

// create container and add components
Container window = getContentPane();
BorderLayout layout = new BorderLayout();
window.add(myPanel, BorderLayout.CENTER);
window.add(buttonPanel, BorderLayout.SOUTH);

// add listeners to our buttons
yesButton.addActionListener(this);
noButton.addActionListener(this);
}
class MyPicturePanel extends JPanel {

    public void paintComponent(Graphics g) {
        // fill in a cyan background
        g.setColor(Color.cyan);
        g.fillRect(0, 0, getSize().width, getSize().height);

        // create a title
        Font myFont = new Font("Serif", Font.BOLD + Font.ITALIC, 28);
        g.setFont(myFont);
        g.setColor(Color.red);
        g.drawString("This is my picture!", 75, 30);
// draw a thick line for grass
Graphics2D g2 = (Graphics2D) g;
BasicStroke thickStroke = new BasicStroke(50);
Stroke defaultStroke = g2.getStroke();
g2.setStroke(thickStroke);
Line2D.Float grass = new Line2D.Float(0f, 310f, 400f, 310f);
Color myGreen = new Color(0, 64, 0);
g2.setColor(myGreen);
g2.draw(grass);
g2.setStroke(defaultStroke);

// draw house using a filled rectangle
Rectangle2D.Float house = new Rectangle2D.Float(200f, 135f, 150f, 150f);
g2.setColor(Color.red);
g2.fill(house);
// draw door on house using a round rectangle
RoundRect2D.Float door = new RoundRect2D.Float(265f, 235f, 30f, 50f, 10f, 10f);
g2.setColor(Color.black);
g2.fill(door);

// draw the sun using an ellipse
Ellipse2D.Float sun = new Ellipse2D.Float(50f, 75f, 75f, 75f);
g2.setColor(Color.yellow);
g2.fill(sun);

// draw a face inside our sun using ellipses and arcs
Ellipse2D.Float eye1 = new Ellipse2D.Float(65f, 90f, 15f, 15f);
g2.setColor(Color.black);
g2.draw(eye1);
Ellipse2D.Float eye2 = new Ellipse2D.Float(95f, 90f, 15f, 15f);
g2.draw(eye2);
Arc2D.Float smile = new Arc2D.Float(65f, 110f, 45f, 25f, 180f, 180f, Arc2D.Float.OPEN);
g2.draw(smile);
// draw triangle for roof
GeneralPath roof = new GeneralPath();
roof.moveTo(200f, 135f);
roof.lineTo(275f, 70f);
roof.lineTo(350f, 135f);
roof.closePath();
g2.setColor(Color.black);
g2.fill(roof);
}

public void actionPerformed(ActionEvent e)
{
    if (e.getSource() == yesButton)
    {
        responseLabel.setText("Why Thank You!");
    }
    else if (e.getSource() == noButton)
    {
        responseLabel.setText("That's Too Bad!");
    }
}
MyPicture walkthrough

- Most of the code (with the exception of the MyPicturePanel class and some minor additions in our MyPicture constructor) should all seem familiar to you from previous lessons so we will not go over those details.

- Note that we need to import java.awt.geom.* in order to use our 2D graphics objects.

- The first place where we added a small change was after we created our yesButton and noButton in our MyPicture constructor.
Here we used `setBackground()` and `setForeground()` in order to change the color of the button and its text. These methods are inherited from `JComponent` so many other common components (like `JLabel`, `JPanel`, `JComboBox`, etc) have these methods as well.

- Both methods takes an argument of type `Color`.

- The `Color` class contains various default colors that you can use in the form of constants (like we did here). Consult the API for a full list of available color constants.
Font sansSerifFont = new Font("SansSerif", Font.PLAIN, 22);
responseLabel.setFont(sansSerifFont);

- Later in our constructor, we created a new Font and used the setFont() method (again, inherited from JComponent) to change the font for one of our labels.

- The constructor for Font takes a String for the name of the font, followed by an int for the style, and an int for the size.
Font sansSerifFont = new Font("SansSerif", Font.PLAIN, 22);
responseLabel.setFont(sansSerifFont);

- The name of the font can be any font name (Arial, SansSerif, Tahoma, etc.), but it is recommended that you use either SansSerif, Serif, or Monospaced since these are platform-independent fonts (meaning all systems will have these fonts).

- The style is usually specified by one of three Font constants: Font.PLAIN, Font.BOLD, or Font.ITALIC. You can create a combination of two by using the ‘+’ sign between constants. For example, Font.BOLD+Font.ITALIC would be bold italic.
class MyPicturePanel extends JPanel
{
    public void paintComponent(Graphics g)
    {
        // Now it’s time for the interesting stuff – creating our own panel that contains a picture.
        // We start by extending JPanel, which gives us all of its methods and capabilities (including the capability of being drawn own).
        // We override the paintComponent() method in order to specify how we want our panel drawn.
        // We will use the Graphics object g as our painting toolbox.
    }
}
MyPicture walkthrough

// fill in a cyan background
    g.setColor(Color.cyan);
    g.fillRect(0, 0, getSize().width, getSize().height);

- The first thing we do is paint our background.
- We use `setColor()` to change the current color being used by our toolkit.
- We then call `fillRect()` which doesn’t really “set” the background, but rather draws a rectangle that is the same size as the panel.
- When drawing in Java, newer objects are drawn on top of older objects. Therefore, we should always draw our background rectangle first.
MyPicture walkthrough

// fill in a cyan background

```java
g.setColor(Color.cyan);
g.fillRect(0, 0, getSize().width, getSize().height);
```

- **fillRect()** takes 4 arguments: a starting (x, y) coordinate pair, a width, and a height.

- In this case, we start at coordinate (0, 0) – the top-left corner of the panel – and use `getSize().width` and `getSize().height` (which is the width/height of the panel) as our background size.
MyPicture walkthrough

// draw a thick line for grass
Graphics2D g2 = (Graphics2D) g;
BasicStroke thickStroke = new BasicStroke(50);
Stroke defaultStroke = g2.getStroke();
g2.setStroke(thickStroke);

- Now we want to draw some grass. We could have used a rectangle to do this, but for the sake of example, we will just use a really thick line.

- Since we are about to draw more advanced 2D shapes, we need a more advanced toolkit. `Graphics2D` is a subclass of `Graphics` that includes this extra functionality and allows us to draw more advanced graphics.

- In order for our current toolkit to get these methods we must first cast it to a `Graphics2D` object – its new name will now be `g2`.
MyPicture walkthrough

// draw a thick line for grass
Graphics2D g2 = (Graphics2D) g;
BasicStroke thickStroke = new BasicStroke(50);
Stroke defaultStroke = g2.getStroke();
g2.setStroke(thickStroke);

- Since we want to draw a thick line, we need to change the stroke for our toolkit by calling the setStroke() method.

- We first create a BasicStroke object (a subclass of Stroke) whose constructor takes an integer for the line width (in pixels).

- However, since we have other objects to draw (and we don’t want to draw them with a line stroke of width 50) we save the current stroke in a variable named defaultStroke. We call the getStroke() method which returns the current stroke for the toolkit to do this.
Line2D.Float grass = new Line2D.Float(0f, 310f, 400f, 310f);
Color myGreen = new Color(0, 64, 0);
g2.setColor(myGreen);
g2.draw(grass);
g2.setStroke(defaultStroke);

- Now that we have our stroke set, we need to draw our line.

- First, we need to actually create a Line2D object. There are two different classes you can use to create a Line2D – Line2D.Float or Line2D.Double.

- Since we chose to use Line2D.Float, our constructor will take floating point numbers as arguments.

- The four arguments for the constructor consist of the starting XY coordinate and the ending XY coordinate. Here we draw a line from (0, 310) to (400, 310).
MyPicture walkthrough

```java
Line2D.Float grass = new Line2D.Float(0f, 310f, 400f, 310f);
Color myGreen = new Color(0, 64, 0);
g2.setColor(myGreen);
g2.draw(grass);
g2.setStroke(defaultStroke);
```

- Next we decide that to create our own color instead of using one of the default color constants provided in the Color class.

- The constructor for `Color` takes 3 values – a red value, a green value, and a blue value (RGB).

- These values can range from 0 to 255. The higher the number the more of that hue that is in that color. When all values are zero you have black, and when all values are 255 you have white.

- We then set the color and draw the line using the `draw()` method. Finally, we set our stroke back to the original stroke.
The next item on our list is to draw the house. We will use a `Rectangle2D` object to do this (we could have also used `fillRect()` like we did for the background).

The constructor for `Rectangle2D.Float` takes a starting `XY` coordinate (where the top-left corner of the rectangle starts), followed by a width and height.
// draw house using a filled rectangle
Rectangle2D.Float house = new Rectangle2D.Float(200f, 135f, 150f, 150f);
g2.setColor(Color.red);
g2.fill(house);

- Here we create a rectangle that has its top-left corner at (200, 135) and is 150x150 pixels in size.
- We then set the color to red and call the `fill()` method.
- `fill()` will actually draw and fill in the entire rectangle instead of just drawing an outline (like `draw()` does).
// draw door on house using a round rectangle
RoundRectangle2D.Float door = new RoundRectangle2D.Float(265f, 235f, 30f, 50f, 10f, 10f);
g2.setColor(Color.black);
g2.fill(door);

- Now we draw a door using a RoundedRectangle2D object.
- We construct this object just like a Rectangle2D object except we pass two other arguments that specify the distance from the corners where rounding should begin (in XY directions).
- In this case, we start rounding our rectangle 10 pixels away from the corners in both the X and Y direction.
// draw the sun using an ellipse
Ellipse2D.Float sun = new Ellipse2D.Float(50f, 75f, 75f, 75f);
g2.setColor(Color.yellow);
g2.fill(sun);

- Here we draw and fill a yellow sun using another 2D graphics object – **Ellipse2D**.

- Its constructor takes the same arguments as the **Rectangle2D** as well.

- Note that the XY coordinates specified indicate the top-left corner of the rectangle that bounds the ellipse, not the center of the ellipse.
Next we want to draw a face inside our sun.

We start by drawing two more ellipses for the eyes. We used the `draw()` method here to only draw an outline of the eyes.

Then we used an `Arc2D` object to draw the smile.
Arc2D.Float smile = new Arc2D.Float(65f, 110f, 45f, 25f, 180f, 180f, Arc2D.Float.OPEN);
g2.draw(smile);

- The constructor for `Arc2D.Float` takes a starting XY coordinate, a width, a height, a starting degree point, a size (in degrees), and a type.

- The type can be either `PIE`, `CHORD`, or `OPEN`. See page 157 of your textbook for a visual example of each.

- In our example, we have an arc that starts at (65, 110) and its bounding rectangle is 45x25 pixels in size. It starts at the $180^\circ$ point on a circle and has a size of $180^\circ$ (or half a circle) and is of type `OPEN`.
Our final task is to draw the roof. There is no triangle object that we can use, but we can use the `GeneralPath` object which is used to draw any general polygon.

- We start by calling its default constructor.

- Next we specify a starting XY coordinate using the `moveTo()` method.
MyPicture walkthrough

// draw triangle for roof
GeneralPath roof = new GeneralPath();
roof.moveTo(200f, 135f);
roof.lineTo(275f, 70f);
roof.lineTo(350f, 135f);
roof.closePath();
g2.setColor(Color.black);
g2.fill(roof);

- We then specify other XY coordinate points using `lineTo()`.

- Finally, we call the `closePath()` which connects the last point with the first in order to complete our triangle.

- We then can display our triangle by calling the `fill()` command.
Final Notes

- The hardest part of drawing 2D graphics in Java (for most people) is determining the XY coordinates and sizes for shapes.

- Some people may find it easier to draw pictures on graph paper and translate those coordinates to their Java program.

- Other people may find simple trial-and-error strategies work better.

- Either way, don’t let graphics frustrate you (it’s supposed to be fun!) – you will get better at it with time (just like programming).
Drill 14

1. Modify the MyPicture program that we used as an example and make some of the following changes.
   - Instead of using outlines for the face in the sun, make the shapes solid.
   - Add some windows to the house.
   - Change the color of the house by creating your own custom color.
   - Add a chimney to the roof.
   - Add any other changes that you want.

2. Create your own picture. Try to use all of the objects we learned about in our lesson. Be creative! Send us something worthwhile and we will post it on the web site - be sure to include a note that it is alright to post it.