Chapter 3G: Graphics

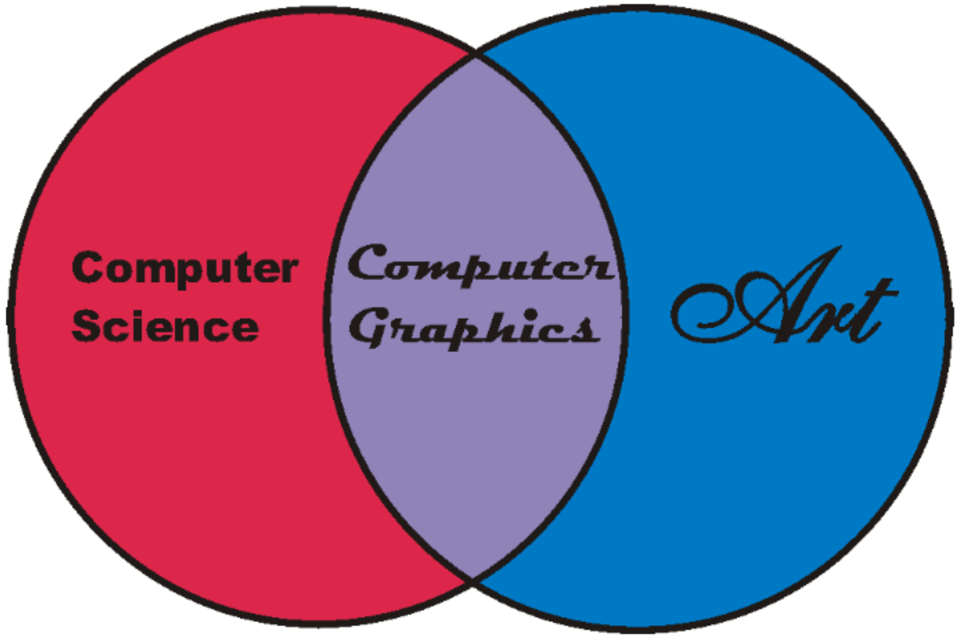
WEEK 4:

THE DrawingPanel CLASS AND Graphics CLASS

DRAWING LINES AND SHAPES AND TEXT

CHANGING COLORS AND FONTS

PROCEDURAL DECOMPOSITION WITH GRAPHICS



Introduction to Graphics

- Graphics are used for games, computer animations, modern GUIs (graphical user interfaces), and much more
- They also provide us with a good way to practice using parameters
- To create the graphical programs in this section, you'll need to include the following import declaration at the top of your program:

```
import java.awt.*;          // for graphics
```

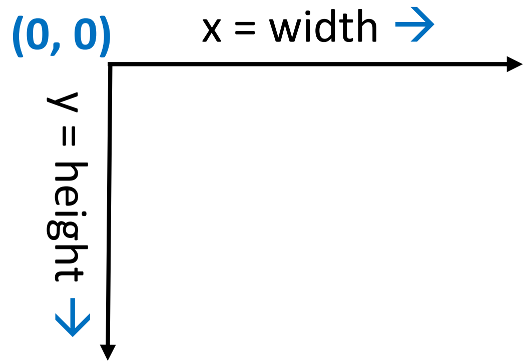
- We're going to use a custom class for this section called **DrawingPanel** (written by the authors of the textbook) – the **DrawingPanel.java** file is available for download on the class website and here: http://www.buildingjavaprograms.com/supplements2.shtml
- The DrawingPanel.java file will need to be placed **in the same file** as your programs

The DrawingPanel Object (your “canvas”)

- You can create a graphics window on your screen by constructing a DrawingPanel object using the following syntax:

DrawingPanel<name> = new DrawingPanel(<width>, <height>);

in pixels
(picture elements)



- DrawingPanel objects have two public methods:
  - **getGraphics()** -- returns a Graphics object that can be used to draw onto the panel
  - **setBackground(color)** -- sets the background color of the panel to a specified color (default = white)

The Graphics object (Your “paint brush”)

- The DrawingPanel class has a method called **getGraphics** that returns a Graphics object

```
Graphics g = <panel>.getGraphics();
```

- The Graphics object has many methods – these can be explained by looking in the Java API (Application Programming Interface) at the following link :
  http://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html

- Once you have a Graphics object, you can draw lines and shapes
- We’ll start with one of the simplest objects – a **line**:

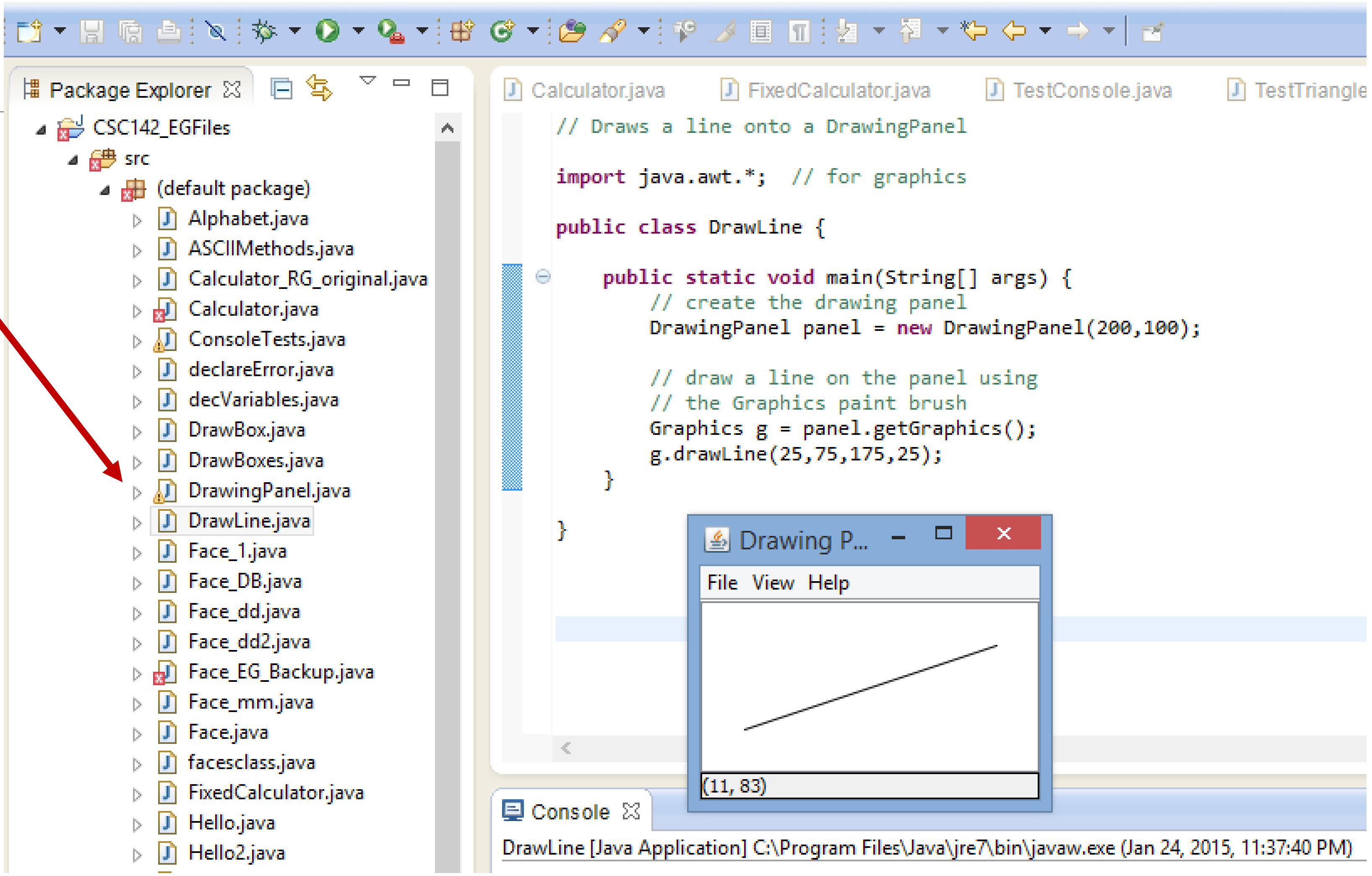
```
g.drawLine(<x1>, <y1>, <x2>, <y2>);
```

- The **drawLine method** draws a line on the panel from **point 1** (x1, y1) to **point 2** (x2, y2)

First Graphical Program

NOTE: The **DrawingPanel.java** file is in the same **src** folder as the java programs that will be **using** the class (and a lot of other java programs that won't **:-P**)

Instead of producing output in the console, the DrawLine program displays in a new graphics window



Package Explorer

- CSC142_EGFiles
  - src
    - (default package)
      - Alphabet.java
      - ASCIIMethods.java
      - Calculator_RG_original.java
      - Calculator.java
      - ConsoleTests.java
      - declareError.java
      - decVariables.java
      - DrawBox.java
      - DrawBoxes.java
      - DrawingPanel.java
      - DrawLine.java
      - Face_1.java
      - Face_DB.java
      - Face_dd.java
      - Face_dd2.java
      - Face_EG_Backup.java
      - Face_mm.java
      - Face.java
      - facesclass.java
      - FixedCalculator.java
      - Hello.java
      - Hello2.java

Calculator.java | FixedCalculator.java | TestConsole.java | TestTriangle

```
// Draws a line onto a DrawingPanel

import java.awt.*;  // for graphics

public class DrawLine {

    public static void main(String[] args) {
        // create the drawing panel
        DrawingPanel panel = new DrawingPanel(200,100);

        // draw a line on the panel using
        // the Graphics g paint brush
        Graphics g = panel.getGraphics();
        g.drawLine(25,75,175,25);
    }

}
```

```
DrawLine [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Jan 24, 2015, 11:37:40 PM)
```

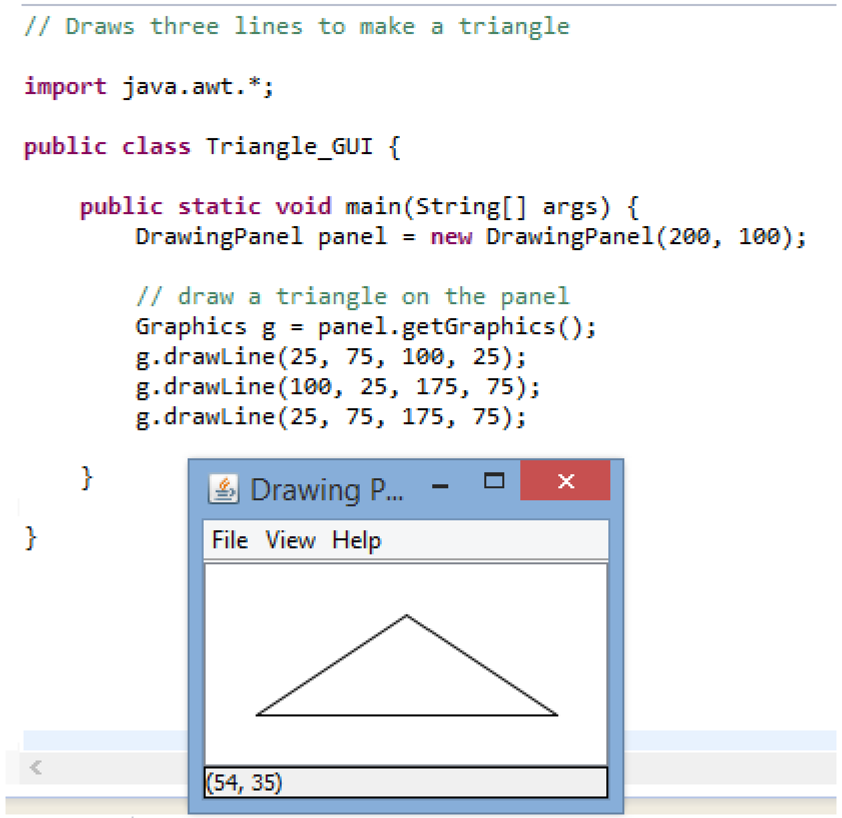
Useful Graphics Object Methods

Method	Description
drawLine(x1, y1, x2, y2)	Draws a line between the points (x1, y1) and (x2, y2)
drawOval(x, y, width, height)	Draws the outline of the largest oval that fits within the specified rectangle
drawRect(x, y, width, height)	Draws the outline of the specified rectangle
drawstring(message, x, y)	
fillOval(x, y, width, height)	Fills the largest oval that fits within the specified rectangle using the current color
fillRect(x, y, width, height)	Fills the specified rectangle with using the current color
setColor(color)	Sets the graphics context’s current color to the specified color
setFont(font)	Sets the graphics context’s current font to the specified font

NOTE: the changes made using setColor or setFont will **PERSIST** until they are changed via these methods again

A Slightly More Complicated Program

- Here, we specified 3 lines with coordinates chosen such that they form a triangle
- We could create other shapes this way, but the Graphics object also has methods that allow us to create particular shapes more easily
  - **g.drawRect(<x>, <y>, <width>, <height>)** causes a rectangle with the specified width and height ans with its upper left corner at (x,y) to be drawn
  - The **drawOval** method is closely related to the drawRect method, as it creates the largest oval that can fit within the rectangle that would be created by using the same values as arguments to the method



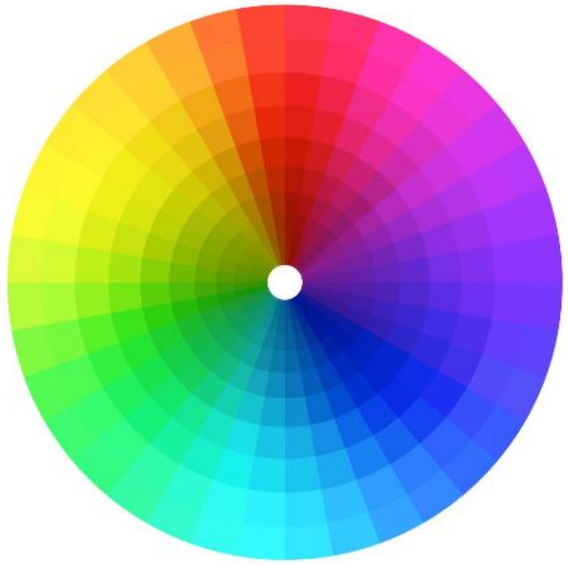
```java
// Draws three lines to make a triangle

import java.awt.*;

public class Triangle_GUI {

    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(200, 100);

        // draw a triangle on the panel
        Graphics g = panel.getGraphics();
        g.drawLine(25, 75, 100, 25);
        g.drawLine(100, 25, 175, 75);
        g.drawLine(25, 75, 175, 75);

    }

}
```

Colors

- So far, we’ve just been creating lines and shapes using their default colors (black and white)
- The standard Color class can be used to change the color of a Graphics object
- In addition, custom colors can be created using the RGB (red, green, blue) parameters
  - The syntax to create a custom color is **new Color(<red>, <green>, <blue>)**, where the color parameters are each an integer value in the range 0-255, inclusive

- **Color constants: Predefined colors of the Color class:**

Color.BLACK	Color.GREEN	Color.RED
Color.BLUE	Color.LIGHT_GRAY	Color.WHITE
Color.CYAN	Color.MAGENTA	Color.YELLOW
Color.DARK_GRAY	Color.ORANGE	
Color.GRAY	Color.PINK	



Changing Colors

- To change the color of a DrawingPanel object, use the syntax

  <panel>.setBackground(<color>);

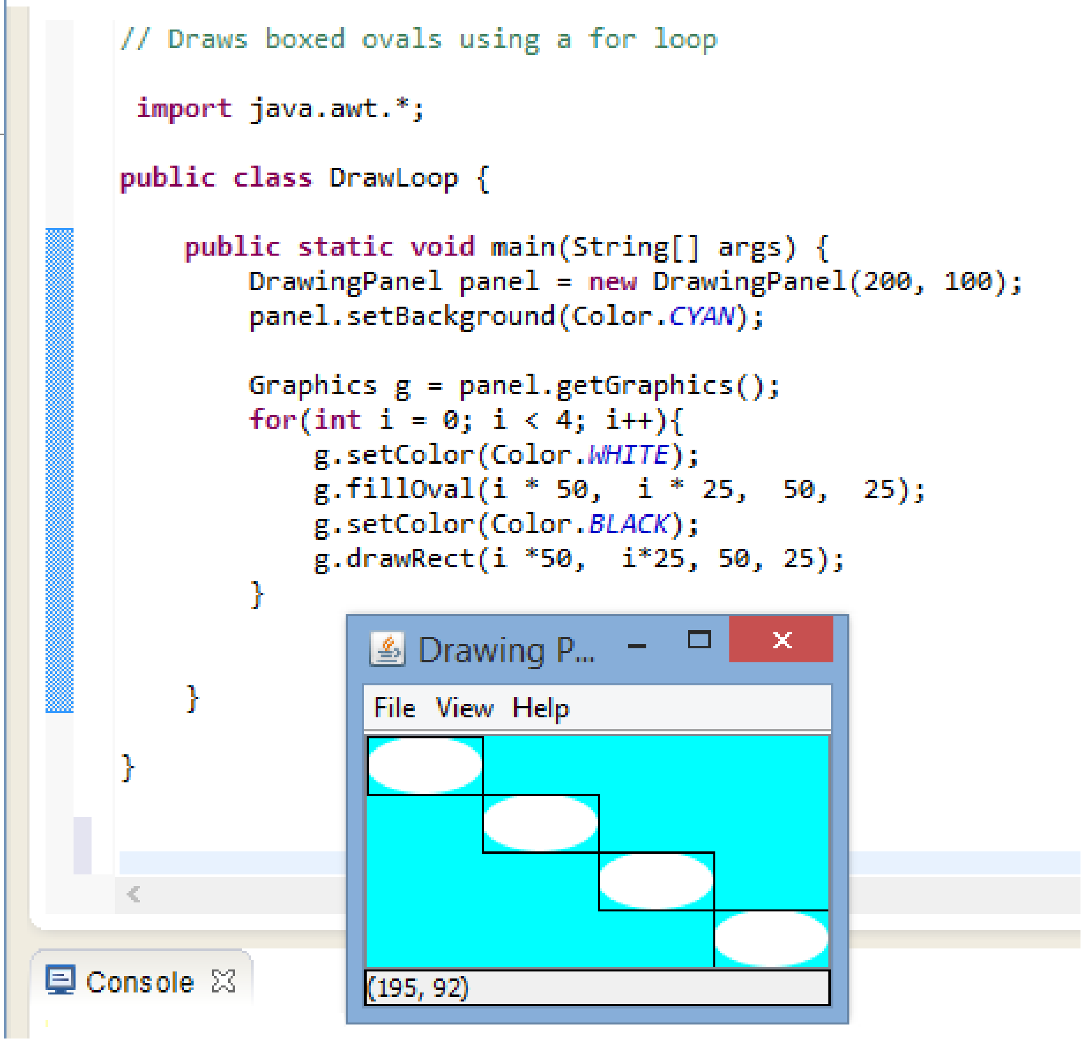
- To change the color of lines and shapes made using the draw or fill commands, use the syntax

  g.setColor(<color>)

- Calling the setColor command is like dipping your paint brush into a different color of paint
- Once you have used the setColor command, all subsequent commands will use that color, until another call to the setColor command is made to change the color again
- Consequently, the **ORDER** in which calls are made is critical
- **NOTE: Draw vs. Fill**
  - Use **draw** to create the outline of a shape using the currently set color
  - Use **fill** to create a solidly colored shape using the currently set color

Drawing with Loops

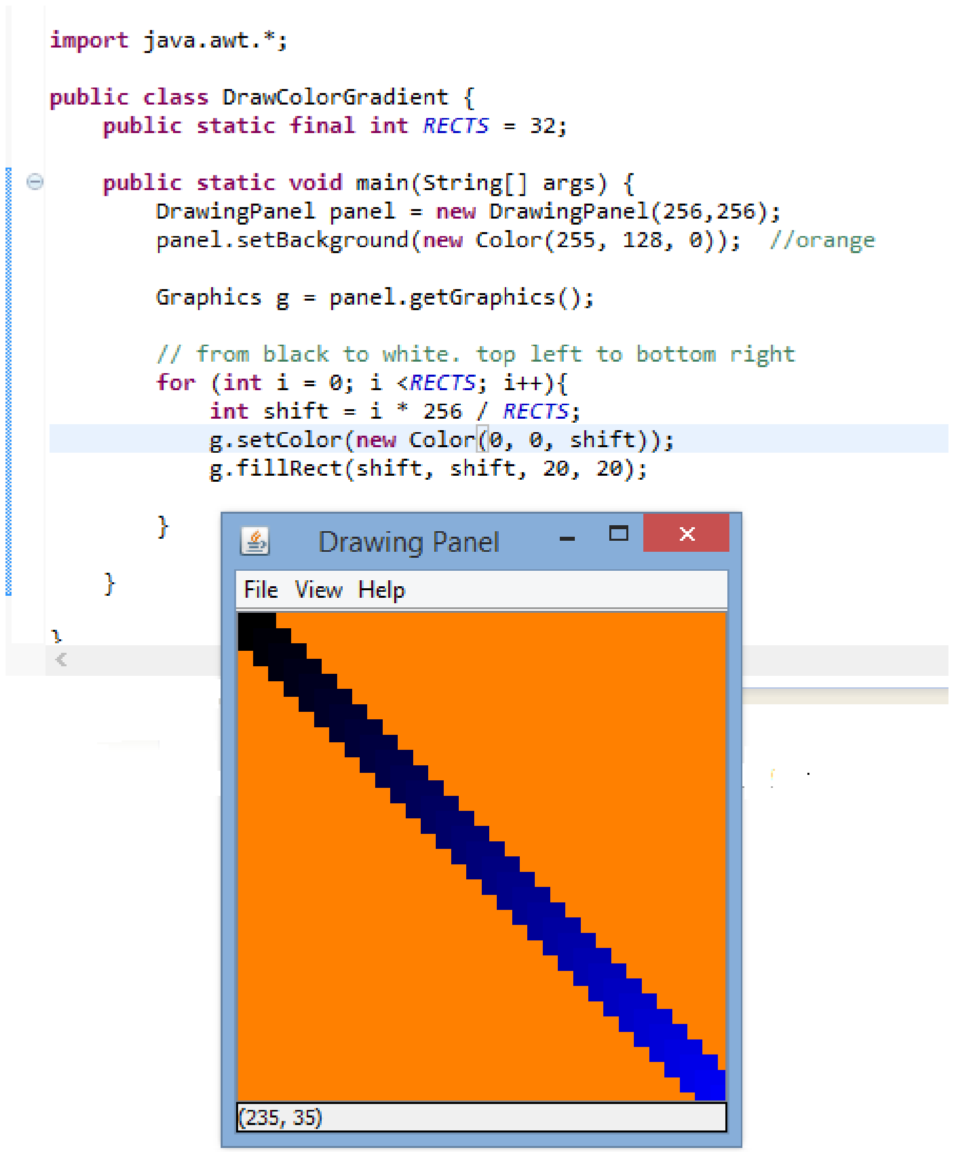
- When drawing shapes of different colors using a loop, BE CAREFUL about changing colors
- You need to make sure you “RESET” the color at the end of one loop iteration if you want the color to be different when the next iteration starts (i.e. change it back to the original color)
- If you forget the second setColor statement in this program, only the first oval would have a rectangle outlined in black – all the others would have WHITE outlines



```
// Draws boxed ovals using a for loop

 import java.awt.*;

public class DrawLoop {

    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(200, 100);
        panel.setBackground(Color.CYAN);

        Graphics g = panel.getGraphics();
        for(int i = 0; i < 4; i++){
            g.setColor(Color.WHITE);
            g.fillOval(i * 50,  i * 25,  50,  25);
            g.setColor(Color.BLACK);
            g.drawRect(i *50,  i*25, 50, 25);
        }

    }

}
```

Gradients

- I didn't like the black to white transition the book code created – and I don't think this is ALL that much better
- The first time I read this section, I skimmed the description, so I thought it was going to transition through more of a "rainbow" of colors – this result was find of dull and disappointing
- I haven't tried it myself, but it might be an interesting challenge – how might you modify the code to get more color in the final result (HINT: the highlighted line in the code is the important one… but actually figuring out how each value should change seems pretty tricky)



```
import java.awt.*;

public class DrawColorGradient {
    public static final int RECTS = 32;

    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(256,256);
        panel.setBackground(new Color(255, 128, 0));  //orange

        Graphics g = panel.getGraphics();

        // from black to white. top left to bottom right
        for (int i = 0; i <RECTS; i++){
            int shift = i * 256 / RECTS;
            g.setColor(new Color(0, 0, shift));
            g.fillRect(shift, shift, 20, 20);

        }
    }
}
```

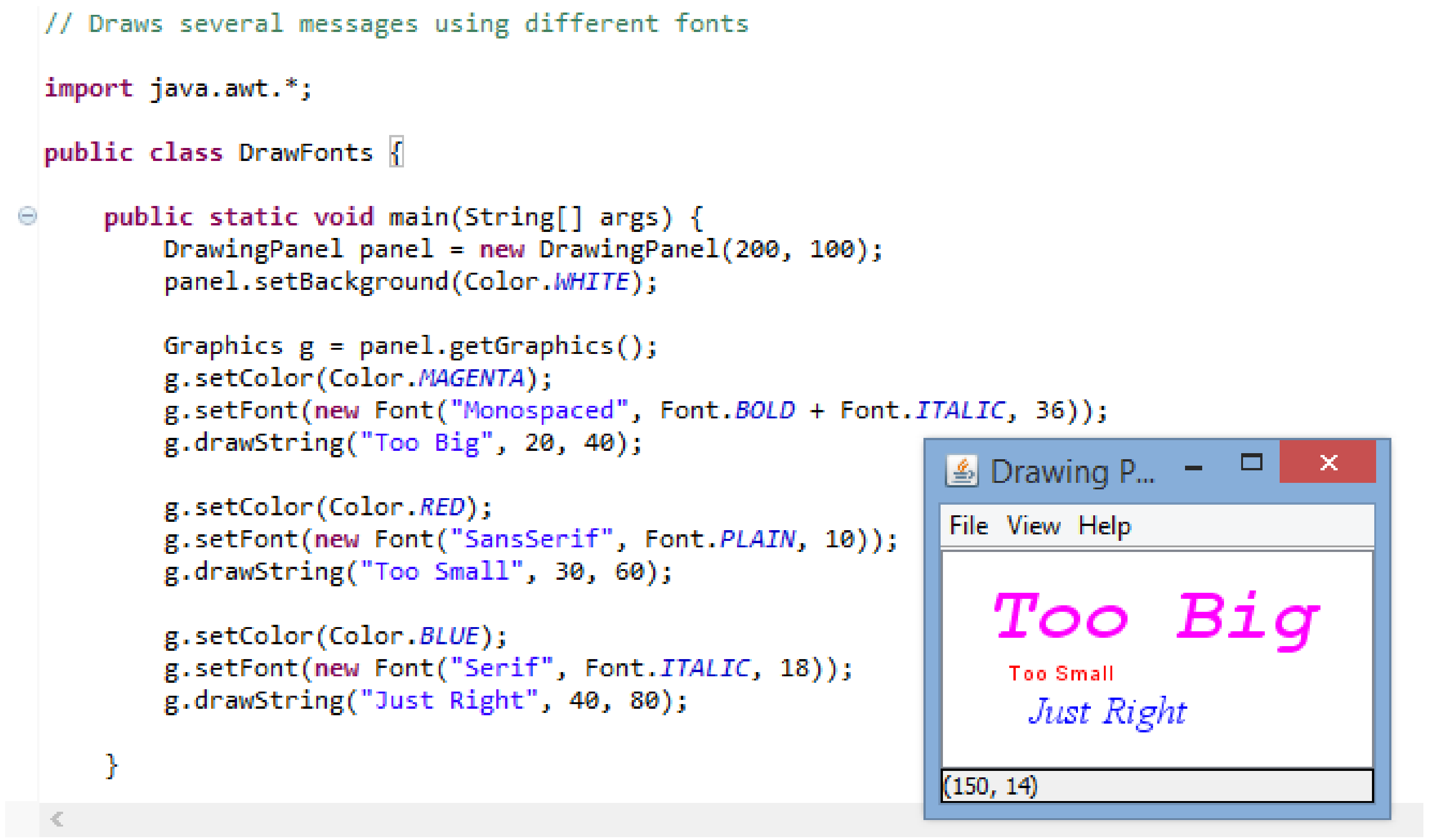
Texts and Fonts

- The **drawString** method of the Graphics object draws a given String with its lower-left corner at the coordinates (x, y)
- Here is the syntax to use: **g.drawString(<message>, <x>, <y>)**
- The **setFont** method changes the font (size and/or style) of the text
- the parameter sent to the **setFont** method is a **Font** object
- A Font object is constructed by passing 3 parameters:
  - The Font's name as a String
  - Its style (e.g. bold or italic)
  - Its size as an integer
- Here is the syntax to use: **new Font(<name>, <style>, <size>)**

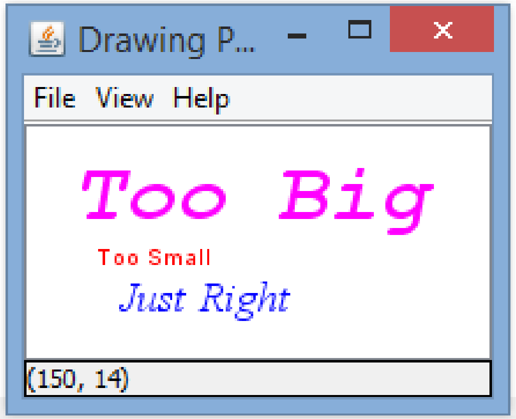
Texts and Fonts: Syntax

- The **drawString** method of the Graphics object draws a given String with its lower-left corner at the coordinates (x, y)
- Here is the syntax to use: **g.drawString(<message>, <x>, <y>)**
- The **setFont** method changes the font (size and/or style) of the text
- the parameter sent to the **setFont** method is a **Font** object
- A Font object is constructed by passing 3 parameters:
  - The Font's name as a String
  - Its style (e.g. bold or italic)
  - Its size as an integer
- Here is the syntax to use: **new Font(<name>, <style>, <size>)**

Texts and Fonts: Example



```
// Draws several messages using different fonts

import java.awt.*;

public class DrawFonts {

    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(200, 100);
        panel.setBackground(Color.WHITE);

        Graphics g = panel.getGraphics();
        g.setColor(Color.MAGENTA);
        g.setFont(new Font("Monospaced", Font.BOLD + Font.ITALIC, 36));
        g.drawString("Too Big", 20, 40);

        g.setColor(Color.RED);
        g.setFont(new Font("SansSerif", Font.PLAIN, 10));
        g.drawString("Too Small", 30, 60);

        g.setColor(Color.BLUE);
        g.setFont(new Font("Serif", Font.ITALIC, 18));
        g.drawString("Just Right", 40, 80);

    }
```



Too Big

Too Small

Just Right

(150, 14)

CONSTANTS of the Font Class:

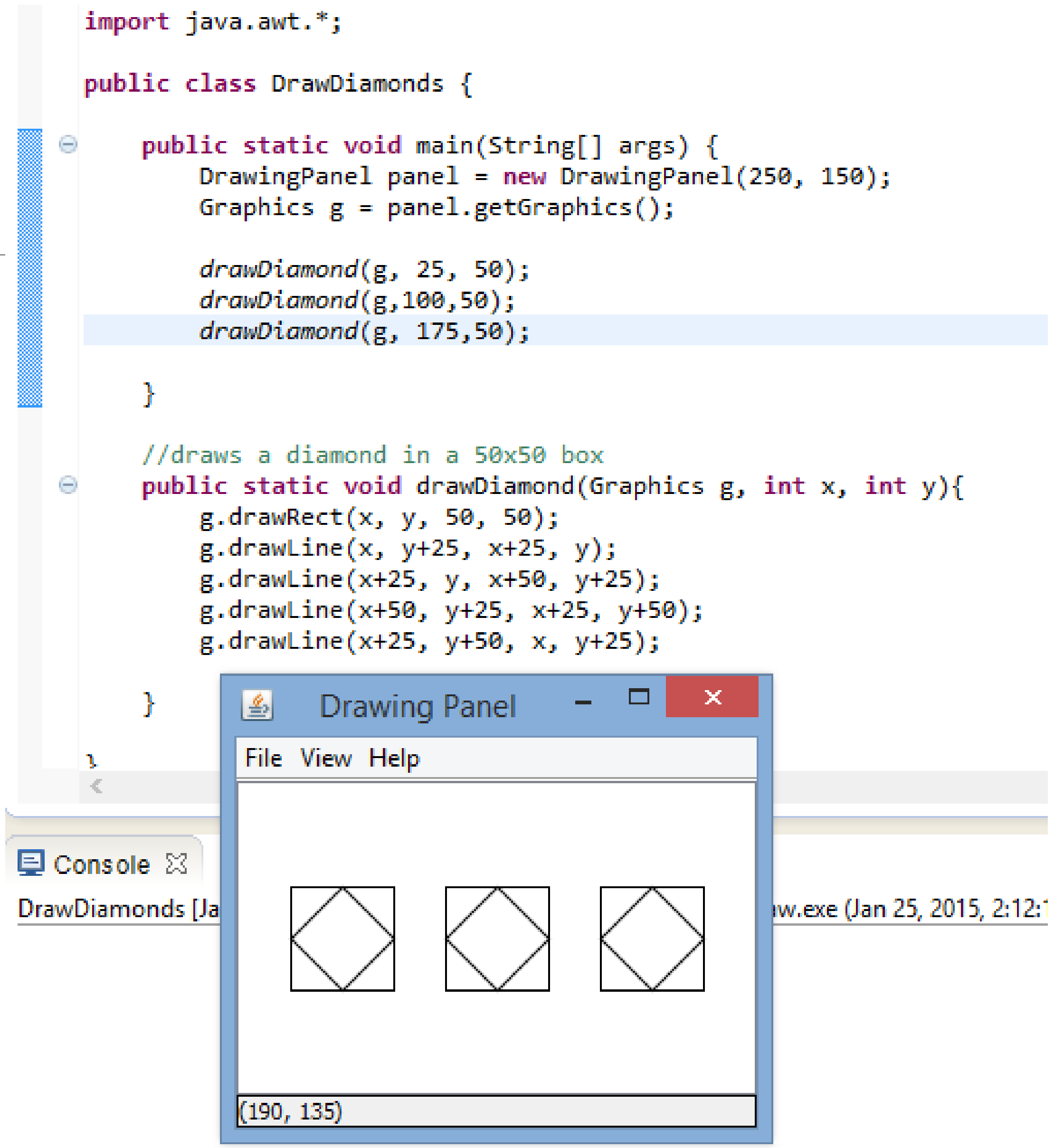
- Font.BOLD → **TEXT**
- Font.ITALIC → *TEXT*
- Font.BOLD + Font.ITALIC → ***TEXT***
- Font.Plain → TEXT

Common FONT NAMES:

- “Monospaced”
- “SansSerif”
- “Serif”

Procedural Decomposition with Graphics

- When writing complex drawing programs, organizing your code into several static methods can help structure your code and remove redundancy
- You will need to pass the Graphics object to each static method (as an argument)
- For example, to draw the following picture containing 3 identical diamond-in-a-box patterns, it would be useful to have a method to draw the diamond inside the box



```
import java.awt.*;

public class DrawDiamonds {

    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(250, 150);
        Graphics g = panel.getGraphics();

        drawDiamond(g, 25, 50);
        drawDiamond(g,100,50);
        drawDiamond(g, 175,50);

    }

    //draws a diamond in a 50x50 box
    public static void drawDiamond(Graphics g, int x, int y){
        g.drawRect(x, y, 50, 50);
        g.drawLine(x, y+25, x+25, y);
        g.drawLine(x+25, y, x+50, y+25);
        g.drawLine(x+50, y+25, x+25, y+50);
        g.drawLine(x+25, y+50, x, y+25);

    }
}
```

Concluding Comments

- **DrawingPanel:** A custom class provided by the authors to easily show a graphics window on the screen
- **Graphics object :** Object with many useful methods for drawing shapes and lines
- **Actual Parameter (argument):** A specific value or expression that appears inside parentheses in a method call
- **“Draw”:** construct a shape OUTLINE using the currently set color
- **“Fill”:** construct a shape that is COMPETELY FILLED IN with the currently set color
- **drawString:** A method that enable text to be written in the graphics window