Drawing pictures with Java

JFrame: the basic Java window

- The swing package contains classes, objects and methods that can be used to create a consistent cross-platform graphical user interface
- The basic window component in this package is the JFrame
- Previously, we saw an example in which a JFrame object was included in a class so that a JOptionPane object had a parent window; that example is repeated on the next slide

Example

```java
import javax.swing.*;

class Example2 {
    public static void main(String[] args) {
        JFrame myWindow;
        myWindow = new JFrame();
        myWindow.setSize(300, 200);
        myWindow.setVisible(true);
        JOptionPane.showMessageDialog(myWindow,
                                       "It's my window");
        JOptionPane.showMessageDialog(null,
                                       "and I'll cry 'n if I want to");
    }
}
```
A new way: the window is the program

- In the previous example, we saw that JFrame objects come with member methods, such as setSize and setVisible
- We will see examples of other member methods throughout this lecture
- For many GUI programs, having a JFrame component isn’t enough to provide all the functionality we need
- In such a case, we can employ a mechanism called inheritance to create a program in which the class we define is a kind of window

Inheritance

- Inheritance is a powerful OOP tool; we will barely scratch the surface of its utility
- We can loosely define inheritance as the ability of one class to take on all of the characteristics of another class
  - the class that receives (or inherits) is called the child or subclass
  - the class that provides is called the parent or superclass
- We will confine our discussion of inheritance (for now) to the use of inherited characteristics by subclasses

Inheritance in Java

- We can declare our intention to inherit characteristics from another class by including the keyword extends and the name of the parent class in our class heading; example:
  ```java
  public class Picture extends JFrame {
  ```
- This heading indicates that class Picture has all the characteristics (and can use all of the member methods) of class JFrame
- A simple example follows
Simple window example

```java
import javax.swing.*;
public class SimpWin extends JFrame {
    public static void main(String[] args) {
        SimpWin sw = new SimpWin();
        sw.setSize(300, 200);
        sw.setTitle("Simple window");
        sw.setLocation(150, 150);
        sw.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        sw.setAlwaysOnTop(true);
        sw.setVisible(true);
    }
}
```

Notes on SimpWin example

- Because of the inheritance mechanism, we can say that a SimpWin object is a JFrame object, just as we can say that, for example, a Labrador Retriever is a dog
- SimpWin is just a specific type of JFrame
- The first line in the main method is worth noting: we are creating an instance of the class we're defining
- We have to do this because of the nature of main – it is a static method, and, therefore, exists independent of any SimpWin object – so we have to give it an object to work with
- The rest of the code in main() consists of instructions that set up the window to be displayed – this sort of functionality usually appears in a class's constructor, as in the next example

Example, part 2

```java
import javax.swing.*;
public class SimpWin2 extends JFrame {
    public SimpWin2 () {
        setSize(300, 200);
        setTitle("Simple window, part deux");
        setLocation(150, 150);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setAlwaysOnTop(true);
        setVisible(true);
    }
    public static void main(String[] args) {
        SimpWin2 sw = new SimpWin2();
    }
}
```
Notes on second part of example

- The second example is just like the first, except:
  - we have explicitly defined the constructor, which contains all of the code that describes the object to be created
  - we have limited main() to a simple call to the constructor; main() exists just as a tester method for the rest of the class
- Note the method call syntax within the constructor – we call the methods without an apparent calling object or class
- In fact, the calling object is the object being created by the constructor

A more interesting window

- We can now create a blank window, but we want to be able to draw pictures
- The JFrame class includes a method (inherited by any of its child classes) called paint(), which enables us to use the window as a blank canvas on which we can place shapes and colors
- The shapes and colors are controlled by an instance of a class named Graphics, which is passed to paint() as an argument

Drawing pictures with Java

- Java’s Graphics class contains methods for drawing lines and simple geometric figures (e.g. rectangles, ellipses)
- Figures can be hollow or filled with color
- Drawings are created within the context of an X-Y coordinate system illustrated on the next slide
Java screen coordinate system

Java coordinate system and graphics objects
- As the previous slide illustrates, the origin point (0,0) is in the upper lefthand corner of the screen (or Container, such as a JFrame)
- The X axis is horizontal and the Y axis is vertical; all values are positive
- A figure, such as a rectangle or ellipse, has a bounding rectangle that indicates the borders in which it is drawn in the space
- The upper left corner of this position and the size of the figure are specified in its constructor

Swing objects and the paint() method
- Most Swing components include a paint() method, which is used to draw the component on the screen
- This method is inherited from JComponent, an ancestor class for most of the Swing objects
- As an inherited method, we have been able to use it (invisibly, since none of our code has called it directly) without modifying, or overriding the original version
- In order to draw our own pictures, we will need to provide a new paint() definition, overriding the original
The paint() method

- The paint method for an object is automatically invoked when the object is made visible; there is almost never an explicit call to the method
- The paint() method has a single parameter of type Graphics, typically named g
- We use g to invoke the methods that draw pictures
- The Graphics class is defined in the awt package, which means we need another import statement if we want to add a paint method to our code

Example

```java
import javax.swing.*;
import java.awt.*;
public class BlueSquare extends JFrame {
    public BlueSquare () {
        setSize(140,160);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public void paint (Graphics g) {
        g.setColor(Color.BLUE);
        g.fillRect(20,40,100,100);
    }
    public static void main (String[] args) {
        BlueSquare b = new BlueSquare();
    }
}
```

Example

- The code from the previous slide (along with the constructor and main method) produces this result:

- The 100 x 100 pixel square is placed within a 140 x 160 window with its upper left corner at position (20, 40)
Graphics drawing methods

- Shape drawing methods include:
  - drawRect – rectangle outline
  - drawOval – ellipse outline
  - fillRect – filled rectangle
  - fillOval – filled ellipse
- Each takes four int arguments
  - 1st and 2nd: x,y coordinates of bounding rectangle’s upper left corner
  - 3rd and 4th: width and length of figure

Graphics drawing methods

- drawRoundRect and fillRoundRect: rectangles with rounded corners
  - First 4 arguments to these are the same as for the rectangle methods
  - 2 additional int arguments specify the width and height of the corner arcs

Graphics drawing methods

- draw3DRect and fill3DRect: rectangles with slightly raised edges
  - First 4 arguments same as other rectangles
  - Last argument is boolean
    - If true, the foreground color has the raised appearance with a small background color edge
    - If false, the background color is the raised part, with a small foreground color edge
Graphics drawing methods

- `drawLine`: draws a line between two points specified by 4 int arguments:
  - 1st and 3rd arguments are the x coordinates
  - 2nd and 4th are the y coordinates

Graphics drawing methods

- `drawArc` and `fillArc`: draw partial ellipses within bounding rectangles; each takes 6 int arguments:
  - 1st and 2nd: xy position of upper left corner of bounding rectangle
  - 3rd and 4th: width and height of bounding rectangle
  - 5th: start of drawing arc (number between 0 and 359)
  - 6th: sweep of drawing arc (number of degrees of arc sweep)
    - A positive sweep value draws arc in counterclockwise direction
    - A negative sweep value draws arc in clockwise direction

Specifying an Arc (Part 1 of 2)
The Color class

- As we have seen, we can draw shapes using Graphics methods
- To specify the color with which to draw, we use the setColor method, which takes an argument of type Color
- The color class contains both predefined colors and the ability to construct our own colors

Predefined Colors

- The Color class contains constants named for each of its predefined colors; they include Color.BLUE, Color.RED, Color.YELLOW, Color.WHITE, Color.BLACK, Color.CYAN, Color.GREEN, Color.MAGENTA, Color.ORANGE, Color.PINK and three shades of GRAY
- You can create your own Color objects using these constants; for example:
  ```java
  Color lg = new Color(Color.LIGHT_GRAY);
  ```
More code examples ...

• FinalPic.java
• Sampexam.java
• Shapes.java

Beyond the basic Crayola® set

• You are not limited to the predefined colors; another version of the Color constructor takes 3 int arguments, each of which must be a value between 0 and 255
• The first argument represents the amount of red in the color; the other two represent green and blue, respectively

One more example

• JFrameEx4.java