Java Graphics

Java has two libraries for creating GUIs (graphical users interfaces): `awt` and `swing`. The Swing toolkit is newer, and richer. We'll be using both. Your programs will start by importing `awt` and `swing` classes as follows:

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

A GUI application consists of individual components you can interact with: buttons and menus and labels, text fields, drawing areas, icons. All these are called *components*.

To appear on screen, every GUI component must be part of a containment hierarchy. A containment hierarchy is a tree of components that has a top-level container as its root.

1. WRITING AN APPLICATION THAT HANDLES A WINDOW

Swing provides three container classes `JFrame, JApplet, JDialog` (defined in `javax.swing.*`) which allow the programmer to create and handle windows.

Today we'll learn about a `JFrame`. A `JFrame` is an object that can handle a window on screen, which has a toolbar and a border, can handle mouse events, colors, buttons, etc. It has a canvas on which it can draw things.

A class that needs to do graphics and pop up a window on the screen will *inherit* from `JFrame` (or one of the other containers listed above). When a class B inherits from a class A, this means that it implicitly contains *all* the instance variables and methods defined in class A (see the handout on inheritance).

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

public class xxx extends JFrame {



}
```

Suppose we want to write a class that will pop up a window. Its skeleton will be like this:

```
/*
 * GSkeleton: This is the skeleton of a graphics class
 * @author Laura Toma
 * @version jan 2008
*/
import javax.swing.*
import java.awt.*
```

```
public class GSkeleton extends JFrame {

    // instance variables


    public GSkeleton() {
        super("My first graphics window");
        setSize(400, 400);

        //...whatever else is needed in the constructor

        //exit the program when the window is closed
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //shows this component/window
        setVisible(true);
       }

};
```

2. HANDLING THE MOUSE

To pick up mouse events a class has to *implement* `MouseInputListener` (which extends `MouseListener` and `MouseMotionListener`). This is an *interface* —- a list of methods that must be implemented. When a class promises to implement an interface this basically means that it makes a contract to implement the set of methods specified in the interface. See the handout on interfaces.

In this case, the mouse handling methods specified in `MouseInputListener` are the following.

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

public class xxx extends JFrame implements MouseInputListener {

   ...

   public void mousePressed(MouseEvent e);

   public void mouseDragged(MouseEvent e);

   public void mouseReleased(MouseEvent e);

   public void mouseClicked(MouseEvent e);
```

```
   public void mouseEntered(MouseEvent e);

   public void mouseExited(MouseEvent e);

   public void mouseMoved(MouseEvent e);
}
```

Note that in order to handle mouse events you need to import `java.swing.event.*` and `java.awt.event.*`. Note that these are different than the ones you import to work with windows.

3. A JAVA CLASS THAT HANDLES A WINDOW AND MOUSE

Suppose we want to write a class that will pop up a window and handle some mouse events. Its skeleton will be like this:

```
/*
 * GSkeleton: This is the skeleton of a graphics class
 * @author Laura Toma
 * @version jan 2008
*/

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


public class GSkeleton extends JFrame implements MouseInputListener {

    // instance variables


    public GSkeleton() {
        super("My first graphics window");
        setSize(400, 400);

        //...whatever else is needed in the constructor

        //exit the program when the window is closed
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //shows this component/window
        setVisible(true);

        //this class will "listen" to the mouse, that is it will receive mouse events;
        //for this it must implement a set of pre-defined functions;
        //every mouse event  will call the corresponding method in this class
```

```
        addMouseMotionListener(this);
        addMouseListener(this);
    }

    //this method is called when mouse is pressed
    public void mousePressed(MouseEvent e) {
System.out.println("mouse pressed at (" + e.getX() + "," e.getY() + ")";
    }

    //this method is called when the mouse is dragged
    public void mouseDragged(MouseEvent e) {
        System.out.println("mouse dragged to" + e.getX() + "," e.getY() + ")";
    }

    public void mouseReleased(MouseEvent e) {}

    public void mouseClicked(MouseEvent e) {}

    public void mouseEntered(MouseEvent e) {}

    public void mouseExited(MouseEvent e) {}

    public void mouseMoved(MouseEvent e) {}

    public static void main (String args[]) {
GSkeleton  mywin = new GSkeleton();
    }

};
```

4. DRAWING IN A WINDOW

In order to draw in the window, one needs to grab the canvas of the `JFrame`, which is of type `Graphics`:

```
Graphics g = this.getGraphics();
```

Note: `this` refers to the current object.

Check the documentation for the methods supported by `Graphics`. Here are some of them. They must be called on a Graphics object.

—drawLine(Point p1, Point p2)

—drawImage(...)

—drawOval(), drawPolygon(), drawRect, filArc(), fillOval, etc

—getColor(), setColor(), getFont(), setFont() etc

The Java coordinate (0,0) is in the upper left corner.

To put an object on a canvas:

```
Graphics g = this.getGraphics();
g.drawLine(..)
```

5. IN-CLASS PROGRAMMING

The goal for the very first Java program is to write an application that lets you scribble on a canvas in the usua way: when pressing the mouse you want to start drawing, then keep the mouse pressed and drag it around while the movement is shown/drawn on the canvas, until the mouse is released.

You will have a single class that will do all the work. In addition to the skeleton above, it will need one or a few instance variables. What are these variables? What would you like to happen when you press the mouse? what about when you drag the mouse? When/where does the drawing actually happen? What happens when you release the mouse?

So, which mouse methods will you implement, and which ones will you leave empty?

Additional Reading

—Bailey Appendix B

—Bailey chapter 1, 2

—BlueJ tutorial