Announcements

- Project 2 due 11/01
- Project 3 posted in a few days
- Exam 2 next week

Material

- Java Graphical User Interfaces (GUI)
Simple GUI Input

- String input
- Use `showInputDialog()` of `JOptionPane` class

```java
import javax.swing.JOptionPane;

public class InputTest {
    public static void main(String[] args) {
        String input = JOptionPane.showInputDialog("How old are you?");
        if (input != null) {
            int age = Integer.parseInt(input);
            age++;
            JOptionPane.showMessageDialog(null, "Next year, you'll be " + age);
        }
        System.exit(0);
    }
}
```

null indicates the dialog has no parent window
• Message dialog icon
• Use overloaded showMessageDialog():

```java
import javax.swing.*;

public class InputTest {
    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null,                            //parent window
                                        "Kernel Crash, Bummer Man!",     //message
                                        "System Shutdown",               //dialog title
                                        JOptionPane.INFORMATION_MESSAGE, //msg type
                                        new ImageIcon("biohazard.gif"));  //icon

        System.exit(0);
    }
}
```
Icon Interface

- Icon type is an interface
- ImageIcon implements Icon interface

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int getIconHeight()</code></td>
<td>Returns the icon's height.</td>
</tr>
<tr>
<td><code>int getIconWidth()</code></td>
<td>Returns the icon's width.</td>
</tr>
<tr>
<td><code>void paintIcon(Component c, Graphics g, int x, int y)</code></td>
<td>Draw the icon at the specified location.</td>
</tr>
</tbody>
</table>
Anonymous Classes

• Typical in Graphical User Interfaces
  • Lots of interfaces to be implemented
  • Often declared as “inner classes”

• Lets see some examples
Anonymous Icon class

Anonymous inner class

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

public class InputTest
{
    public static void main(String[] args)
    {
        JOptionPane.showMessageDialog(
            null, //parent window
            "Kernel Crash, Bummer Man!", //message
            "System Shutdown", //dialog title
            JOptionPane.INFORMATION_MESSAGE, //msg type
            new Icon()
        {
            public int getIconWidth() { return size; }
            public int getIconHeight() { return size; }
        }
    }

    Instantiate anonymous object from an anonymous class
```
Anonymous Icon class

Continued

```java
public void paintIcon(Component c, Graphics g, int x, int y)
{
    Graphics2D g2 = (Graphics2D) g;
    Ellipse2D.Double ellipse =
        new Ellipse2D.Double(x, y, size, size);
    g2.setColor(Color.YELLOW);
    g2.fill(ellipse); //g2.draw(ellipse); for outline only
    Line2D.Double bang =
        new Line2D.Double(new Point2D.Double(25, 5),
            new Point2D.Double(25, 35));
    g2.setColor(Color.RED);
    g2.draw(bang);
    g2.draw(bangBang);
}
```

Graphics object passed by Java is a context for drawing. Actual object passed is a newer more powerful Graphics2D object.

Interface type cannot contain implementation, but an interface definition can.
Windows

• Frames - implement windows
• Frame window has decorations (e.g. title bar, close box, etc.)
• Frames are provided by the windowing system

```java
import javax.swing.*;
JFrame frame = new JFrame();
frame.pack(); //minimize frame size to hold component
//frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.show();
```

Must be set to terminate the spawned process.
Components

• Objects you add to a window and often provide some specialized behavior (like buttons)
• Construct components
  JButton helloButton = new JButton("Say Hello");
• Set content pane layout
  Container contentPane = frame.getContentPane();
  container.setLayout(new FlowLayout());
• Add components to content pane
  contentPane.add(helloButton);

Content pane is window area below title bar. Holds window components.

Content pane layout manager determines positioning of components. FlowLayout aligns components adjacently.
How to handle buttons

• What should happen when a button is pressed?

• How should we build code that does some action when the button is pressed?

• The Java designers did not know what each button would do

• Hence => User Interface Actions
User Interface Actions

- By default, program's buttons don't have any effect
- Attach listener object(s) to button
- A listener object belongs to class implementing the ActionListener interface type

```java
public interface ActionListener {
    int actionPerformed(ActionEvent event);
}
```

- Listeners are notified by JVM when button is clicked
Adding Listeners

- ActionListener objects are usually from anonymous classes:

  ```java
def helloButton.addActionListener(new ActionListener()
  {
      public void actionPerformed(ActionEvent event)
      {
          do something here
      }
  });
```

- More than just cutesy use of anonymous class
- Listener (inner class) can access variables in enclosing scope, particularly instance fields

Local variables of enclosing scope must be declared as final variables.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ActionTest {
    public static void main(String[] args) {
        JFrame frame = new JFrame();

        final int FIELD_WIDTH = 20;
        final JTextField textField = new JTextField(FIELD_WIDTH);
        textField.setText("Click a button!");

        JButton helloButton = new JButton("Say Hello");

        helloButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                textField.setText("Hello, World!");
            }
        });
    }
}
Horstmann: ActionTest.java

```java
JButton goodbyeButton = new JButton("Say Goodbye");

goodbyeButton.addActionListener(new ActionListener()
{
   public void actionPerformed(ActionEvent event)
   {
      textField.setText("Goodbye, World!");
   }
});

Container contentPane = frame.getContentPane();
contentPane.setLayout(new FlowLayout());
contentPane.add(helloButton);
contentPane.add(goodbyeButton);
contentPane.add(textField);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.pack();
frame.show();
```
Simple Factory Method

- Action Listeners with similar actions can be constructed by a factory method:

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

public class ActionTest2 {
    public static void main(String[] args) {
        JFrame frame = new JFrame();

        final int FIELD_WIDTH = 20;
        textField = new JTextField(FIELD_WIDTH);
        textField.setText("Click a button!");

        JButton helloButton = new JButton("Say Hello");
        helloButton.addActionListener(
            createGreetingButtonListener("Hello, World!"));

        JButton goodbyeButton = new JButton("Say Goodbye");
        goodbyeButton.addActionListener(createGreetingButtonListener("Goodbye, World!"));
    }
}
```

Factory (helper) method returns anonymous object of the anonymous class.
Factory Method - continued

- ActionListener factory construction

```java
goodbyeButton.addActionListener(
    createGreetingButtonListener("Goodbye, World!");

    Container contentPane = frame.getContentPane();
    contentPane.setLayout(new FlowLayout());

    contentPane.add(helloButton);
    contentPane.add(goodbyeButton);
    contentPane.add(textField);

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.show();
```
Factory Method - continued

- `ActionListener` factory construction method

  ```java
  public static ActionListener createGreetingButtonListener(final String message) {
    return new ActionListener() {
      public void actionPerformed(ActionEvent event) {
        textField.setText(message);
      }
    };
  }
  
  private static JTextField textField;
  ```

  The parameter to the Factory (helper) method must be declared as final to be able to access it in the anonymous class.
Layout Managers in java

• User interfaces are made up of components
• Components are placed in containers
• Container needs to arrange the components
• Swing doesn't use hard-coded pixel coordinates
• Advantages:
  Can switch "look and feel"
  Can internationalize strings
  Layout manager controls arrangement
Layout Managers

- **FlowLayout**: left to right, start new row when full
- **BoxLayout**: left to right or top to bottom (no new rows or columns)
- **BorderLayout**: 5 areas
  - North
  - East
  - Center
  - West
  - South
- **GridLayout**: rectangular grid, all components have same size
- **GridBagLayout**: complex, like HTML table
Layout examples

FlowLayout

BoxLayout (horizontal)

BorderLayout

GridLayout

GridBagLayout
BorderLayout Example

- BorderLayout Code

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

public class BorderLayoutEg {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        Container contentPane = frame.getContentPane();
        //contentPane.setLayout(new BorderLayout());

        Button centerBtn;
        Button northBtn;
        Button westBtn;
        Button eastBtn;
        Button southBtn;
        centerBtn = new Button("Center Button");
        northBtn = new Button("North Button");
        westBtn = new Button("West Button");
        eastBtn = new Button("East Button");
        southBtn = new Button("South Button");
    }
}
```

BorderLayout is the default layout manager of the content pane of every JFrame.
BorderLayout Example

• BorderLayout Code (continued)

```java
// always says where the component should be placed when adding
// Options are center, East, West, North and South
contentPane.add(centerBtn); //, "Center"
contentPane.add(northBtn, "North");
contentPane.add(westBtn, "West");
contentPane.add(eastBtn, "East");
contentPane.add(southBtn, "South");

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.pack();
frame.show();
```

The string add parameter determines the placement of the object, default is, "Center". Not all areas must be occupied.
Mouse related Listeners

• Two interfaces are mouse event handlers
  • MouseListener
    - public void mouseClicked(MouseEvent arg0)
    - public void mousePressed(MouseEvent arg0)
    - public void mouseReleased(MouseEvent arg0)
    - public void mouseEntered(MouseEvent arg0)
    - public void mouseExited(MouseEvent arg0)
  • MouseMotionListener
    - public void mouseDragged(MouseEvent arg0)
    - public void mouseMoved(MouseEvent arg0)

Listens for mouse clicks.
Listens for mouse movement. Expensive execution. Invoked every time the user moves the mouse.
Mouse events occur in a component only if mouse cursor is over the un-obscured part of the component's bounds when the action occurs.

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MouseEventClass</strong></td>
</tr>
<tr>
<td>int <strong>getButton</strong>()</td>
</tr>
<tr>
<td>Returns which, if any, of the mouse buttons has changed state.</td>
</tr>
<tr>
<td>int <strong>getClickCount</strong>()</td>
</tr>
<tr>
<td>Returns the number of mouse clicks associated with this event.</td>
</tr>
<tr>
<td>static String <strong>getMouseModifiersText</strong>(int modifiers)</td>
</tr>
<tr>
<td>Returns a String describing the modifier keys and mouse buttons that were down during the event, such as &quot;Shift&quot;, or &quot;Ctrl+Shift&quot;.</td>
</tr>
<tr>
<td>Point <strong>getPoint</strong>()</td>
</tr>
<tr>
<td>Returns the x,y position of the event relative to the source component.</td>
</tr>
<tr>
<td>int <strong>getX</strong>()</td>
</tr>
<tr>
<td>Returns the horizontal x position of the event relative to the source component.</td>
</tr>
<tr>
<td>int <strong>getY</strong>()</td>
</tr>
<tr>
<td>Returns the vertical y position of the event relative to the source component.</td>
</tr>
<tr>
<td>boolean <strong>isPopupTrigger</strong>()</td>
</tr>
<tr>
<td>Returns whether or not this mouse event is the popup menu trigger event for the platform.</td>
</tr>
<tr>
<td>String <strong>paramString</strong>()</td>
</tr>
<tr>
<td>Returns a parameter string identifying this event.</td>
</tr>
<tr>
<td>void <strong>translatePoint</strong>(int x, int y)</td>
</tr>
<tr>
<td>Translates the event's coordinates to a new position by adding specified x (horizontal) and y (vertical) offsets.</td>
</tr>
</tbody>
</table>
MouseAdapters

• What if you wanted to implement only the mouseClick part of a mouse listener?

• MouseAdapter looks like this

```java
public class MouseAdapter implements MouseListener {
    public void mouseClicked(MouseEvent event) {}
    public void mousePressed(MouseEvent event) {}
    public void mouseReleased(MouseEvent event) {}
    public void mouseEntered(MouseEvent event) {}
    public void mouseExited(MouseEvent event) {}
}
```

• Component adds the listener like this:

```java
addMouseListener(new MouseAdapter() {
    public void mousePressed(MouseEvent event) {
        mouse action goes here
    }
});
```

Analogous MouseMotionAdapter class implements MouseMotionListener with do nothing methods.

Anonymous object of new MouseAdapter class implicitly extends MouseAdapter. Only necessary methods are over-ridden.
Simple MouseEvent Example

- Bang! Icon -> BioHazard gif when clicked

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

public class BangBioHaz {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container contentPane = frame.getContentPane();
        contentPane.add(new WarnPanel());
        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
        frame.show();
    }

    private static final int FRAME_WIDTH = 200;
    private static final int FRAME_HEIGHT = 200;
}
```

```
BangShape Class

• BangShape constructor

```java
import java.awt.*;
import java.awt.geom.*;

public class BangShape
{
    /**
     * Constructs a Bang shape!
     * @param x the left of the bounding rectangle
     * @param y the top of the bounding rectangle
     * @param aWidth the width of the bounding rectangle
     */
    public BangShape(int x, int y, int aWidth)
    {
        this.x = x;
        this.y = y;
        width = aWidth;
    }
}
```
public void draw(Graphics2D g2) {
    Ellipse2D.Double ellipse =
        new Ellipse2D.Double(x, y, width, width);
    g2.setColor(Color.YELLOW);
    g2.fill(ellipse); //g2.draw(ellipse); for outline only
    Line2D.Double bang = new Line2D.Double( new Point2D.Double(width/2, 5),
        new Point2D.Double(width/2, width-15));
    g2.setColor(Color.RED);
    g2.draw(bang);
    Line2D.Double bangBang = new Line2D.Double( new Point2D.Double(width/2, width-10),
        new Point2D.Double(width/2, width-5));
    g2.setColor(Color.RED);
    g2.draw(bangBang);
}

• BangShape drawing
• BangShape helper & instance fields

```java
public boolean contains(Point2D p) {
    return x <= p.getX() && p.getX() <= x + width
            && y <= p.getY() && p.getY() <= y + width;
}

private int x;
private int y;
private int width;
```
WarnPanel Class

• WarnPanel inherits from JPanel

```java
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.util.*;
import java.awt.util.*;

public class WarnPanel extends JPanel {

    public WarnPanel() {
        final WarnPanel wPanel = this; //save for inner class
        bangImage = true; //initial Bang! display
        mousePoint = null;

        bang = new BangShape(0, 0, 50);
        bioIcon = new ImageIcon("biohazard.gif");

        //attach mouseListener
        addMouseListener(new MouseAdapter() {
            //MouseAdapter class implements MouseListener with do nothing methods.
        });
    }
}
```
public void mouseClicked(MouseEvent event) {
    mousePoint = event.getPoint();
    String strImage = null;
    strImage = (bangImage) ? "Bang!" : "Biohazard.";
    if (bangImage)
        if (bang.contains(mousePoint)) {
            bangImage = !bangImage;
            repaint();
        }
        else  JOptionPane.showMessageDialog(wPanel, "Click on " + strImage);
    else //biohazard displayed
        if (!bang.contains(mousePoint)) {
            bangImage = !bangImage;
            repaint();
        }
        else
            JOptionPane.showMessageDialog(wPanel, "Click on " + strImage);
}}//mouseAdapter

repaint() inherited from JPanel (which
inherited it from java.awt.Component). Results in a call to this component's
paint method.
WarnPanel Class

- WarnPanel drawing

```java
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    if (bangImage)
        bang.draw(g2);
    else
        bioIcon.paintIcon(this, g, 20, 20);
}
```

private BangShape bang;
private Point mousePoint;
private boolean bangImage;
private ImageIcon bioIcon;
```