Why learn GUls?
- Learn about event-driven programming techniques – perhaps the most-used version of inversion-of-control
- Practice learning and using a large, complex API
- A chance to see how it is designed and learn from it (design pattern usage, etc.)
- Caution: There is a ton of information for GUI programming – huge APIs
  - You won’t memorize it all; you will look things up as you need them
  - But you have to learn the fundamental concepts and general ideas

Don’t mistake…
- … how to build a GUI well with …
- … what is a good UI for people
- Just another version of “building the system right vs. building the right system”
- We’ll come back to some usability issues – much more related to “building the right system” later in the term

Java GUI History
- Abstract Windowing Toolkit (AWT): Sun’s initial effort to create a set of cross-platform GUI classes (JDK 1.0 - 1.1)
  - Maps general Java code to each operating system’s real GUI system
  - Limited to lowest common denominator; clunky to use
- Swing: A newer GUI library written from the ground up that allows much more powerful graphics and GUI construction (JDK 1.2+)
  - Paints GUI controls itself pixel-by-pixel rather than handing off to OS
  - Better features, better compatibility, better design
- Warning: Both exist in Java now; easy to get them mixed up; still have to use both in various places

GUI terminology
- window: A first-class citizen of the graphical desktop
  - Also called a top-level container
  - Ex: frame, dialog box, applet
- component: A GUI widget that resides in a window
  - Also called controls in many other languages
  - Ex: button, text box, label
- container: A logical grouping for storing components
  - Ex: panel, box
Swing inheritance hierarchy

- Component (AWT)
  - Window
    - JFrame (Swing)
    - JDialog
  - Container
    - JComponent (Swing)
      - JButton
      - JColorChooser
      - JFileChooser
      - JComboBox
      - JProgressbar
      - JScrollPane
      - JTree
      - JTextField

Component properties

- Each has a get/is accessor and a set modifier
  - Ex: getColor, setFont, setEnabled, isVisible

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>Color</td>
<td>background color behind component</td>
</tr>
<tr>
<td>border</td>
<td>Border</td>
<td>border line around component</td>
</tr>
<tr>
<td>enabled</td>
<td>boolean</td>
<td>whether it can be interacted with</td>
</tr>
<tr>
<td>focusable</td>
<td>boolean</td>
<td>whether key text can be typed on it</td>
</tr>
<tr>
<td>font</td>
<td>Font</td>
<td>font used for text in component</td>
</tr>
<tr>
<td>foreground</td>
<td>Color</td>
<td>foreground color of component</td>
</tr>
<tr>
<td>height, width</td>
<td>int</td>
<td>component's current size in pixels</td>
</tr>
<tr>
<td>visible</td>
<td>boolean</td>
<td>whether component can be seen</td>
</tr>
<tr>
<td>tooltip text</td>
<td>String</td>
<td>text shown when hovering mouse</td>
</tr>
<tr>
<td>size, minimum/maximum/preferred size</td>
<td>Dimension</td>
<td>various sizes, size limits, or desired sizes that the component may take</td>
</tr>
</tbody>
</table>

GUI example

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

public class GuiExample1 {  
  public static void main(String[] args) {  
    JFrame frame = new JFrame();  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    frame.setSize(new Dimension(300, 100));  
    frame.setTitle("A frame");  
    JButton button1 = new JButton();  
    button1.setText("I'm a button.");  
    button1.setBackground(Color.BLUE);  
    frame.add(button1);  
    JButton button2 = new JButton();  
    button2.setText("Click me!");  
    button2.setBackground(Color.RED);  
    frame.add(button2);  
    frame.setVisible(true);  
  }  
}
```

- We defined two buttons, but only one is visible. Why?
- What happens when we click button2?

It's tedious... and there is more ...

- Size and positioning
  - Preferred/minimum sizes, absolute/relative positioning
- Containers and layout
  - Flow layout – laying out components in a container
  - Border layout – NORTH, SOUTH, EAST, WEST, CENTER
  - Grid layout
- And more, lots more...
- ...

GUI control structure

- event: An object representing a user's interaction with a GUI component
- listener: An object responding to events
  - To handle an event, attach a listener to a component (such as a button)
  - The listener will be notified when the event occurs (such as a button click)
Event-driven programming

- A programming style where the overall flow of execution is dictated by events
- The program defines a set of listeners that wait for specific events
- As each event happens due to a user action, the program runs specific code
- The overall flow of execution is determined by the series of events that occur, not a pre-determined order
- The events invoke client code (through the listeners) without knowing which client code is invoked
  - The invokes relation (in part) no longer matches the names relation

Action events

- **action event**: An action on a GUI component
- The most common, general event type in Swing, caused by
  - button or menu clicks,
  - check box checking / unchecking,
  - pressing Enter in a text field, ...
- Represented by a class named `ActionEvent`
- Handled by objects that implement interface `ActionListener`

Implementing a listener

```java
public class name implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        code to handle the event;
    }
}
```

- Attaches the given listener to be notified of clicks and events that occur on this component
- `JButton` and other graphical components have this method

Event hierarchy

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

```
EventObject
- AWTEvent (AWT)
  - ActionEvent
  - TextEvent
  - ComponentEvent
    - FocusEvent
    - WindowEvent
  - InputEvent
    - KeyEvent
    - MouseEvent
-EventListener
  - AWTEventListener
  - ActionListener
  - TextListener
  - ComponentListener
  - FocusListener
  - WindowListener
  - KeyListener
  - MouseListener
```

Nested classes

- **nested class**: A class defined inside of another class
- Nested objects can access/modify the fields of their outer object
  - If necessary, can refer to outer object as `OuterClassName.this`
- Only the outer class can see the nested class or make objects of it
- Event listeners are often defined as nested classes inside a GUI

Instance of EnclosingClass

```
Instance of InnerClass
```

GUI event example

```java
public class MyGUI {
    private JFrame frame;
    private JButton stutter;
    private JTextField textField;
    public MyGUI() {
        stutter.addActionListener(new StutterListener());
    }
    public void actionPerformed(ActionEvent event) {
        String text = textField.getText();
        textField.setText(text + text);
    }
}
```
Mouse and keyboard events

- Low-level events — close to the hardware — to listen for and respond to mouse clicks/movements and keyboard entry/echoing

MouseListener interface

```java
public interface MouseListener {
    public void mouseClicked(MouseEvent event);
    public void mouseEntered(MouseEvent event);
    public void mouseExited(MouseEvent event);
    public void mousePressed(MouseEvent event);
    public void mouseReleased(MouseEvent event);
}
```

- Most AWT/Swing components have this method
```java
public void addMouseListener(MouseListener ml)
```

Implementing listener

```java
public class MyMouseListener implements MouseListener {
    public void mouseClicked(MouseEvent event) {} // Tidious to define the empty method for the events you are not interested in
    public void mouseEntered(MouseEvent event) {}
    public void mouseExited(MouseEvent event) {}
    public void mousePressed(MouseEvent event) {
        System.out.println("You pressed the button!");
    }
    public void mouseReleased(MouseEvent event) {} // elsewhere, myComponent.addMouseListener(new MyMouseListener());
}
```

Implementing listener

```java
public class MyMouseAdapter extends MouseAdapter {
    public void mousePressed(MouseEvent event) {
        Object source = event.getSource();
        if (source == button && event.getX() < 10) {
            JOptionPane.showMessageDialog(null, "You clicked the left edge!");
        }
    }
}
```

Adapter pattern to the rescue

- Provide an adapter class that connects to GUI components but exposes to us the interface we prefer — only a method or two
- event adapter: A class with empty implementations of all of a given listener interface’s methods
- Ex: MouseAdapter, KeyAdapter, FocusAdapter
- Ex: To extend MouseAdapter only override methods you want to implement
  - Don’t have to type in empty methods for the ones you don’t want!

An abstract event adapter

```java
public abstract class MouseAdapter implements MouseListener {
    public void mouseClicked(MouseEvent event) {} // (from java.awt.event)
    public void mouseEntered(MouseEvent event) {}
    public void mouseExited(MouseEvent event) {}
    public void mousePressed(MouseEvent event) {} // An empty implementation of all MouseListener methods
    public void mouseReleased(MouseEvent event) {}
    // Classes can extend MouseAdapter rather than implementing MouseListener
    // client gets the complete mouse listener interface it wants
    // implementer gets to write just the few mouse methods they want
    public class MyMouseAdapter extends MouseAdapter {
        public void mousePressed(MouseEvent event) {
            System.out.println("You pressed the button!");
        }
    }
    // elsewhere, myComponent.addMouseListener(new MyMouseAdapter());
}
```

Using MouseEvent

```java
public class MyMouseAdapter extends MouseAdapter {
    public void mousePressed(MouseEvent event) {
        Object source = event.getSource();
        if (source == button && event.getClickCount() < 10) {
            JOptionPane.showMessageDialog(null, "You clicked the left edge!");
        }
    }
```
Mouse input listener

- The `MouseInputListener` interface and the `MouseInputAdapter` class ease the development of an object that listens to mouse clicks, movement, and/or wheel events.

```java
public class MyMouseInputAdapter extends MouseInputAdapter {
    public void mousePressed(MouseEvent event) {
        System.out.println("Mouse was pressed");
    }
    public void mouseDragged(MouseEvent event) {
        Point p = event.getPoint();
        System.out.println("Mouse is at " + p);
    }
}

MyMouseInputAdapter adapter = new MyMouseInputAdapter();
myPanel.addMouseListener(adapter);
myPanel.addMouseMotionListener(adapter);
```

Similar for keyboard events

```java
public interface KeyListener {
    public void keyPressed(KeyEvent event);
    public void keyReleased(KeyEvent event);
    public void keyTyped(KeyEvent event);
}

// what key code was pressed? (one for almost every key)
public static final int VK_A, VK_B, ..., VK_Z,
VK_0, ..., VK_9, VK_F1, ..., VK_F10, VK_UP, VK_LEFT, ...
// Were any modifier keys held down?
public static final int CTRL_MASK, ALT_MASK, SHIFT_MASK
public char getKeyChar(){ // use VK_* with this
    public Object getSource(){ // use *_MASK with this

    }
}
```

Focus: current target of keyboard input

- If a component doesn’t have the focus, it will not receive events
- By default, most components don’t receive focus
  - Buttons, text fields, and some others default to on
- `JComponent` methods for focus
  - public void setFocusable(boolean b)
  - public void requestFocus()
  - Asks for this component to be given the current keyboard focus
- `FocusListener` (focus gained or lost), `focusAdapter`, also available

Other events

- Window events (closed, opened, iconified, ...)
- Change events (state changed in a `JSlider`, ...)
- Component events (component hidden, resized, shown, ...)
- `JList`/`JTree` select events
- Document events (for text fields)
- ...

Next steps

- Assignment 3: due Sunday October 30, 11:59PM
- Lectures
  - W (Midterm review, including example questions)
- Upcoming: Friday 10/28, in class midterm – open book, open note, closed neighbor, closed electronic devices