Intro to Programming II

GUIs

Chris Brooks

Department of Computer Science
University of San Francisco
13-0: Synchronous vs Asynchronous input

The programs you’ve built so far (lexer and parser) are examples of synchronous input.
- You prompt for input, then read input with a Scanner.

Programs with a graphical user interface (GUI) typically require asynchronous input
- A user can provide input at any time.

This requires a different model of programming.
A GUI consists of:
- Components
- Events
- Listeners
Components generate events (usually in response to user input)
Listeners wait for and handle these events
Typically by invoking a method.
Open eclipse and create a new project called 'Class-project'

Choose New-Other-GUI Forms-Swing-JFrame

Give the subclass the name ExampleJFrame

- A JFrame is an example of a *top-level container*
- Other components are added inside the JFrame
Choose ‘Absolute Layout’ and then add three buttons and a text field.
Give each button a different label.
Look at the code that Jigloo generates.
Use the color wheel to change each button’s background color.
When a user provides input to a component, an event is generated.

- For example, when the mouse is pressed or released.

Select button1, then choose 'MouseListener-mouse released' under the 'Events' tag.

Select 'handler method'

Look at the code Jigloo generates.
Now, we need to fix the event handler to do something interesting.

Let’s place the button’s label in the text field.

```java
private void button1MouseReleased(MouseEvent evt) {
    System.out.println("button1.mouseReleased, event=", evt);
    output.setText(button1.getLabel());
}
```

Add similar event handlers for button2 and button3.
Now, let’s add a JList.

JLists use a DefaultComboBoxModel to control access to their data.

Let’s add an Event handler to change the JList’s contents if the return key is pressed.

Choose Select the textfield, then choose KeyListener-KeyTyped under ‘Events’.

Take a look at the generated code.
13-8: Handling keyboard events

We need to look at the event and find out what key was pressed.

```java
private void outputKeyTyped(KeyEvent evt) {
    System.out.println("output.keyTyped, event=" + evt);
    if (evt.getKeyChar() == 'n') {
        DefaultComboBoxModel m = (DefaultComboBoxModel)thisList.getModel();
        m.addElement(output.getText());
    }
}
```

The DefaultComboBoxModel controls access to the list contents.
What’s this model stuff about?

A common technique for GUI design (and OO design more generally) is called *model-view-controller*.

A GUI should be separated into pieces:

- the model controls the data itself
- The view controls how the data is displayed.
- The controller governs how the data is accessed and changed.
The Absolute manager is nice, but limited.
  - Try resizing your app.

If we want to resize, we must pick a different layout manager.
Flow layout places components left to right as possible.

When one row is filled, the next is started.

Switch your JFrame to Flow layout, then resize the components.

How can we get our buttons to line up vertically?

Add subpanels and place the buttons in them.

Add two JPanels and use the tree on the right to place the components in them.

Use hgap and vgap under the 'layout' menu to space components.
13-12: Border Layout

- Flow Layout is OK, but resizing may not do what you’d expect.
- Border layout breaks a container into North, South, East, West, Center.
- Change the JFrame to Border Layout, and the JList to Flow. Change the layout for panel1 to be Border.
- Set panel1 to ’West’, and panel2 to ’East’
- Try resizing now.
Border Layout is nice for subpanels, but awkward for components.

Grid Layout lets you break the Frame into rectangular subsections.

Components fill left-to-right and top-to-bottom.

Remove the panels and place the components in a grid.

The components are still not a nice size, but we can add new subpanels.
13-14: Exercise: building a simple calculator

- Remove the list box and add buttons for numbers.
- Add buttons for operators.
- To begin:
  - When a number is pressed, it should show up in the text box.
13-15: Exercise: building a simple calculator

6 Add keys for plus, minus, and equals.
6 Add instance variables for operand1, operand2, operator.
6 When plus, minus, or equals is pressed, we must:
   △ Do a calculation
   △ Store the result
   △ Display it in the text box.
private void plusButtonMouseReleased(MouseEvent evt) {
    System.out.println("plusButton.mouseReleased, event=" + evt);
    operand1 = operand2;
    operand2 = new Double(resultField.getText()).doubleValue();
    operator = '+';
    double result = compute();
    operand2 = result;
    resultField.setText(Double.toString(result));
    resultDisplayed = true;
}
Add a method to compute:

```java
private double compute() {
    System.out.println("op1: " + operand1 + " op2: " + operand2);
    if (operator == '+') {
        return operand1 + operand2;
    } else if (operator == '-') {
        return operand1 - operand2;
    } else if (operator == '*') {
        return operand1 * operand2;
    } else {
        return operand1 / operand2;
    }
}
```