AWT AND SWING

Today

- Last time we looked at some basic principles for designing GUIs.
- This time we will continue the discussion of GUIS by beginning to look at how we can code them using Java.
- To do that we will take a lightening tour through some of the components that Java offers in its interface packages.
- This tour will continue next week.

AWT and Swing

- Early versions of Java introduced some basic support for building GUIs
- Abstract Window Toolkit

  AWT

  `java.awt.*`
- Swing builds on (expands) AWT.

  `javax.swing.*`
- We'll look at using bits of both packages to get what we want.

- Doing this will involve lots of the ideas that we saw in the material on basic Java:
  - Extending classes
  - Implementing interfaces
  - Polymorphism
- In fact we have already seen a bunch of these things in the homeworks already.

HelloButton

- The next few slides refer to the `HelloButton` program that you can download from the class website.

- `JFrame` creates the window that you see.
  - `JFrame` is from Swing
- The button title (the text that appears on the button) is `Hello Button`.
- A `Container` is a component, a thing for putting things in.

- Though a `JFrame` is itself a container, we don't put things directly in a `JFrame`.
  - Rather we put things in a special container that is part of every `JFrame`.
- This is the container that we access through `getContentPane()`.
- Looking at the remainder of the code:
  - `add()` adds things to the `ContentPane`
  - `pack()` makes the window just big enough to hold all of its components.
- To set the window to a specific size, we would use something like:

```
setSize(300, 200)
```

Event Handling

- Programs with a GUI are unlike other programs
  - Don't terminate when `main()` has run.
- Using Swing generates a separate *thread*.
  - That thread is then in an infinite loop.
- (We will conver threads in more detail later in the course.)
- The Swing thread responds to *events*

  Event-driven programming
- In general, GUI programming is event-driven programming

  (As are other forms of programming such as programming robots.)

- All Swing components are *event sources*
  - Things that can generate events from user input.
- In order to respond to events, you need to tell the event generator what object to notify.
- Use:

  `add`Something`Listener().`

  where Something is some event type.
- This is a *delegation* model — some object is delegated the job of handling some events.

- In the case of `JButton`, a button press is a source of a

  `java.awt.event.ActionEvent`

- To specify the listener for an `ActionEvent`, we need an object that implements the:

  ```
  java.awt.event.ActionListener
  ```

  interface.
- That means it needs a:

  ```
  actionPerformed()
  ```

  method.
- You connect the `ActionListener` to the event generator using:

  ```
  addActionListener()
  ```

- Now, let’s look at an example.

HelloGoodBye

- The next few slides refer to the `HelloGoodBye` program that you can download from the class website.

- Not this hello/goodbye:

YOU SAY YES I SAY NO YOU SAY STOP
AND I SAY GO GO GO OH OH NO
YOU SAY GOODBYE AND I SAY HELLO
HELLO HELLO I DON'T KNOW WHY YOU SAY GOODBYE I SAY HELLO
WHY AND I SAY OH NO!
1980
2001

- The key thing here is the setting up of a listener:

  ```
  GoodBye listener = new GoodBye();
  ```

  and connecting it to the button:

  - `hello.addActionListener(listener);`

- Recall that `hello` is the reference to the button.

Text Input

- Swapping the button for a textfield allows us to get input from the user.
- An example is the `TextInput` program that you can download from the class website.
- We'll discuss this program for the next few pages.

- Note the way that the event is handled here.
- Whereas with the button press we knew what we had to do

  Since only one action was associated with the button.

  here we need to do something that is specific to what was entered.
- So we manipulate the event, `e`
- The `getSource()` method returns a reference to the event.

  Since it is generic, we have to cast it to the right type.
- The listener then uses `getText()` to extract the text.

- If we want to input numbers, we can use the same code, but add something along the lines of:

```
double value =
Double.parseDouble(text.trim());
```

- This uses the `Double` object's method `parseDouble` to extract a number from a string representation.
- Note the use of `trim()` to remove whitespace which will mess with the parsing.

Text Output

- To write to a `textField` you can use the method `SetText()`

Multiple components

- So far we have had only one interactive component in an interface.
  - Clearly rather limiting.
- Easy to add more.
- These next few slides refer to the `MultiComp` program that you can download from the class website.

- Adding components is not the hard part, we just add them as before:

```
JTextField input =
   new JTextField("Edit this and hit <return>");
JButton goodbye = new JButton("Quit!");

pane.add(input);
pane.add(goodbye);
```

- In an attempt to be clever I decided to handle all the events in a single listener class `MultiListen`:

```
MultiListen listener =
   new MultiListen(input, goodbye);

input.addActionListener(listener);
goodbye.addActionListener(listener);
```

- Since there are two kinds of event that get delivered to the listener, it has to figure out which it is.
- We do this by giving the listener references to the components.

```
public MultiListen(JTextField field,
      JButton button){
   this.button = button;
   this.field = field;
}
```

- These can be used to disambiguate which component is causing the event

```
if (button == e.getSource()){
    System.out.println("Goodbye!");
    System.exit(0);
}
else{
    field = (JTextField)e.getSource();
    String text = field.getText();
    System.out.println(text);
}
```

Layout Managers

- Since `MultiComp` has several different components, we have to worry about how where they will appear.
  - Layout
- Rather we don't worry about it too much, we hand off the problem to a *layout manager*.
- In `MultiComp` we used the `java.awt.GridLayout` manager.
  - Here you set the number of rows and columns in the grid.
- You can also use the `java.awt.FlowLayout` manager, using the call:

```
pane.setLayout(new FlowLayout());
```

Simple drawing

- `StarBurst`, which you can download from the class webpage, is an example of simple drawing in Java.
- This, like all graphics, works by adding `Graphics` objects to a container.

- To understand drawing, you first have to understand how Java organises spaces on which you can draw:
- Window picture
- The inner rectangle would be drawn with:

```
g.drawrect(50, 20, 100, 40)
```

- Usually we draw things by creating an object that extends `JComponent`, and we add that to our GUI.
- The `StarBurst` project defines such an object and then places it in a frame.
- The file `Star.java` defines the object, and `StarBurst.java` puts it in a `JFrame`.

- `StarBurst.java` looks very familiar, with the only new lines being:

  ```
  Star star = new Star();
  pane.add(star);
  ```

- Note that we have no listener since this is not interactive.

- In `Star.java` we have a `paint` method which does all the work:

```
public void paint(Graphics g){
  double x1, x2, y1, y2;
  for (double angle = 0; angle < Math.PI;
    angle = angle +(Math.PI/16)){
    x1 = Math.cos(angle) * RADIUS + RADIUS;
    y1 = Math.sin(angle) * RADIUS + RADIUS;
    x2 = Math.cos(angle + Math.PI) * RADIUS + RADIUS;
    y2 = Math.sin(angle + Math.PI) * RADIUS + RADIUS;
    g.drawLine((int)x1, (int)y1, (int)x2, (int)y2);
  }
}
```

- `paint()` is called when the containing `JFrame` is told to `show()` itself.

- We also need to tell the layout manager how large to make the window.
- We do this by defining two functions that different layout managers call:

```
public Dimension getMinimumSize(){
  return new Dimension(2 * RADIUS, 2 * RADIUS);
}

public Dimension getPreferredSize(){
  return new Dimension(2 * RADIUS, 2 * RADIUS);
}
```

More with the mouse

- So far all we can do with the mouse is to click on a button.
  - More interesting things are possible.
- Our last example
  - `SimplePaint`, which, as ever, you can get from the class webpage

  allows the user to draw using the mouse.
- This time we have three classes:
  - `SimplePaint`
  - `DrawingCanvas`
  - `PaintListener`

- `SimplePaint` is much like the other top-level classes we have looked at:

```
DrawingCanvas canvas = new DrawingCanvas();
PaintListener listener = new PaintListener();
canvas.addMouseMotionListener(listener);
```

- The key difference is that the listener is listening to a different type of event.

- `DrawingCanvas` just sets the size of the area that we can draw on:

```
private static final int SIZE = 500;

public Dimension getMinimumSize(){
  return new Dimension(SIZE, SIZE);
}

public Dimension getPreferredSize(){
  return new Dimension(SIZE, SIZE);
}
```

- `PaintListener` does all the work:

```
class PaintListener implements MouseMotionListener{

public void mouseDragged(MouseEvent e){
  DrawingCanvas canvas = (DrawingCanvas)e.getSource();
  Graphics g = canvas.getGraphics();

  g.fillOval(e.getX() - RADIUS,
    e.getY() - RADIUS, DIAMETER, DIAMETER);
}
```

- The `mouseDragged` method is called whenever the mouse is over the canvas and is moved while the button is down.
- Here it draws a small circle.

- Note that a `MouseMotionListener` also has to define a `mouseMoved()` method.
- Here it does nothing:

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

- `mouseMoved()` is called when the mouse is over the canvas and the button is up.

- Note that `PaintListener` does not use any Swing components.
  - Entirely AWT.

Summary

- This lecture looked very quickly at a number of interface components in Java:
  - Buttons
  - Reading from text fields
  - Writing to text fields
  - Listeners for multiple components
  - Simple drawing
  - Mouse events