A Tale of Two Companies

- As I noted in an earlier story, the modern user interface was developed at the Xerox Palo Alto Research Center (Xerox PARC) but commercialized at Apple.
- Xerox had tried to reach the commercial office market and had priced its Alto accordingly. By contrast, Apple recognized that their machine had to be affordable to individual users.

Creating a Simple GUI

- Most application programs today include a graphical user interface or GUI (pronounced gooey) consisting of buttons and other on-screen controls. Collectively, these controls are called interactors.
- Java defines many types of interactors, most of which are part of a collection called the Swing library, which is described in section 10.6. You create a GUI by constructing the Swing interactors you need and then arranging them appropriately in the program window.
- The text outlines two strategies for arranging interactors on the screen. The simple approach is to create a control strip along one or more edges of the window, as described on the next slide. You can, however, create more general GUIs by using Java’s layout managers, as described in section 10.7.

Creating a Control Strip

- When you create an instance of any Program subclass, Java divides the window area into five regions as follows:

```
+----------------+-----------------+-----------------+-----------------+-----------------+
| NORTH          | CENTER          | EAST            | SOUTH           |
| +/-            | +/-             | +/-             | +/-             |
|              |                 |                 |                 |
```

- The CENTER region is typically where the action takes place. A ConsoleProgram adds a console to the CENTER region, and a GraphicsProgram puts a GCanvas there.
- The other regions are visible only if you add an interactor to them. The examples in the text use the SOUTH region as a control strip containing a set of interactors, which are laid out from left to right in the order in which they were added.

Creating a GUI with a Single Button

The HitchhikerButton program on the next slide uses this vignette from Hitchhiker’s Guide to the Galaxy to illustrate the process of creating a GUI without focusing on the details. The code creates a single button and adds it to the SOUTH region. It then waits for the user to click the button, at which point the program responds by printing a simple message on the console.
The HitchhikerButton Program

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

/** This program puts up a button on the screen, which triggers a message inspired by Douglas Adams’s novel. 
 * 
 */
public class HitchhikerButton extends ConsoleProgram {
    /* Initializes the user-interface buttons */
    public void init() {
        add(new JButton("Red"), SOUTH);
        addActionListeners();
    } /* Responds to a button action */
    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("Red")) {
            println("Please do not press this button again.");
        }
    }
}
```

The Swing Interactor Hierarchy

The following diagram shows the Swing classes used in this text. With the exception of `IntField` and `DoubleField`, all of these classes live in the `javax.swing` package.

The JButton Class

- The most common interactor in GUI-based applications is an on-screen button, which is implemented in Swing by the class `JButton`. A `JButton` object looks something like

```
  JButton pushMeButton = new JButton("Push Me");
```

- The constructor for the `JButton` class is

```
new JButton(label)
```

where `label` is a string telling the user what the button does.

- The button shown earlier on this slide is therefore created by

```
JButton pushMeButton = new JButton("Push Me");
```

- When you click on a button, Java generates an action event, which in turn invokes a call to `actionPerformed` in any listeners that are waiting for action events.

The JTextField Class

Adding Features to DrawStarMap

- The text illustrates the various Swing interactors by adding new features to the `DrawStarMap` application. The first step is adding a Clear button that erases the screen.

```java
public void init() { 
    add(new JButton("Clear"), SOUTH);
    addActionListeners();
}
```

- Adding the button is accomplished in the `init` method:

```
button pushClearButton = new JButton("Clear");
```

- The response to the button appears in `actionPerformed`:

```
if (e.getActionCommand().equals("Clear")) { 
    removeAll();
}
```

Exercise: Interactive Stoptlight

Design and implement a `GStoplight` class that represents a compound object with three colored lights—red, yellow, and green—as in a traditional traffic signal. Once you have finished that, write a `GraphicsProgram` that creates a stoptlight and three buttons labeled Red, Yellow, and Green, as shown in the sample run below. Clicking on a button should send a message to the stoptlight to change its state accordingly.
The JToggleButton Class

• The JToggleButton class is another type of button that is similar to JButton but maintains an on/off state. On the screen, a JToggleButton looks just like a JButton except for the fact that it stays on after you release the mouse button.

• As its name suggests, a JToggleButton toggles back and forth between on and off when it is clicked. Clicking the first time turns it from off to on; clicking a second time turns it off.

• You can determine whether a JToggleButton is on by calling isSelected, which returns true if the button is on.

• The JToggleButton class itself is not used as much as two of its subclasses, JCheckBox and JRadioButton, which are described on the next two slides.

The JCheckBox Class

• The JCheckBox class is a subclass of JToggleButton and therefore inherits its behavior.

• In terms of its operation, a JCheckBox works exactly like an instance of its parent class. The only difference is in what the button looks like on the screen. In a JCheckBox, the button label appears to the right of a small square that either contains or does not contain a check mark, like this:

• Because a JCheckBox is a JToggleButton, you can call the isSelected method to determine its state.

• Like a JButton, a JCheckBox generates action events when it is clicked. Both of these classes inherit this behavior from AbstractButton, which is their common superclass.

The JRadioButton Class

• The JRadioButtton class also extends JToggleButton and behaves in much the same way. In this case, the button is displayed as a circle that is tinted and marked with a dot when it is selected, as follows:

• Radio buttons are ordinarily not used individually but instead as a set. If you create a ButtonGroup object and then add several radio buttons to it, the Swing libraries make sure that only one of those buttons is selected at a time.

• Grouped radio buttons are used to allow the user to choose among several mutually exclusive options. As an example, the text extends the DrawStarMap program to allow the user to choose the size of the star by selecting a radio button:

• Radio button

Small Medium Large

The JSlider Class

• In many applications, you want to let the user adjust a value over a wide range instead of selecting among a set of options.

• The Swing libraries include several different interactors that allow the user to adjust a parameter. The text uses the JSlider class, which appears on the screen like this:

• The user can adjust a JSlider by dragging the slider knob.

• The simplest form of the JSlider constructor looks like this:

new JSlider(min, max, value)

where min and max are integers giving the minimum and maximum values of the slider and value is the initial value.

• You can retrieve the current value by calling getValue.

The JLabel Class

• The interactors you display on the screen sometimes don’t provide the user with enough information. In such cases, it is useful to include JLabel objects, which appear as text strings in the user interface but do not respond to any events.

• As an example, if you wanted to label a slider so that it was clear it controlled size, you could use the following code to produce the control strip shown at the bottom of the screen:

• As an example, if you wanted to label a slider so that it was clear it controlled size, you could use the following code to produce the control strip shown at the bottom of the screen:

add(new JLabel("Small"), SOUTH);
add(drawSlider, SOUTH);
add(new JLabel("Large"), SOUTH);

The JComboBox Class

• In some applications, you may need to allow the user to chose among a set of options that would take up too much space on the screen if you listed them all. In such situations, you can use the JComboBox class, which lists the available options in a popup menu that goes away once the selection is made.

• A JComboBox used to select T-shirt sizes might look like this on the screen:

• From the user’s point of view, a JComboBox works like this:

– Depressing the mouse brings up a popup menu.
– Dragging the mouse selects from the different options.
– Releasing the mouse sets the state to the current option.

• Given that its purpose is to offer the user a choice of options, the JComboBox interactor is sometimes called a chooser.
Using the JComboBox Interactor

- The standard constructor for a JComboBox creates an empty
  interactor that contains no options; you then add the desired
  options by calling the addItem method for each one.
- The code to create the T-shirt size chooser looks like this:

```java
JComboBox sizeChooser = new JComboBox();
sizeChooser.addItem("Small");
sizeChooser.addItem("Medium");
sizeChooser.addItem("Large");
sizeChooser.addItem("X-Large");
sizeChooser.setEditable(false);
```

The last line prevents the user from typing in some other size.

- The items in a JComboBox need not be strings but can instead
  be any object. The label that appears in the popup menu is
  determined by applying the object’s toString method.
- The getSelectedItem and setSelectedItem methods allow
  you to determine and set which item is selected.

The JTextField Class

- Although Swing’s set of interactors usually make it possible
  for the user to control an application using only the mouse,
  there are nonetheless some situations in which keyboard input
  is necessary.
- You can accept keyboard input in a user interface by using
  the JTextField class, which provides the user with an area in
  which it is possible to enter a single line of text.
- The HelloGUI program on the next slide illustrates the use of
  the JTextField class in a ConsoleProgram that prints a
  greeting each time a name is entered in the text field.

```
HelloGUI
Name
Hello, world.
Hello, Eric.
```

```
import acm.program.*;
import java.awt.event.*;
import javax.swing.*;
/** This class displays a greeting whenever a name is entered */
public class HelloGUI extends ConsoleProgram {
public void init() {
    nameField = new JTextField(10);
    add(new JLabel("Name"), SOUTH);
    add(nameField, SOUTH);
    nameField.addActionListener(this);
}
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == nameField) {
        println("Hello, " + nameField.getText());
    }
}
/* Private instance variables */
private JTextField nameField;
}
```

Notes on the JTextField Class

- The constructor for the JTextField class has the form
  ```java
  new JTextField(columns)
  ```
  where columns is the number of text columns assigned to the
  field. The space often appears larger than one might expect,
  because Java reserves space for the widest characters.
- You can get and set the string entered in a JTextField by
  calling the getText and setText methods.
- A JTextField generates an action event if the user presses
  the ENTER key in the field. If you want your program to
  respond to that action event, you need to register the program
  as an action listener for the field. In the HelloGUI example,
  the action listener is enable by the statement
  ```java
  nameField.addActionListener(this);
  ```

Numeric Fields

- The acm.gui package includes two JTextField subclasses
  that simplify the process of reading numeric input within a
  graphical user interface. The IntField class interprets its
  text string as an int; the DoubleField class interprets the
  text string as a double.
- In addition to the usual operations on a JTextField, the
  IntField and DoubleField classes export getValue and
  setValue methods that get and set the numeric value of the
  field.
- Although it is beyond the scope of the text, the IntField and
  DoubleField classes support numeric formatting so that you
  can control the number of digits in the display. The methods
  that support this capability are described in the javadoc
  documentation for these classes.