CS 325 - Class 17

- Today
  - Introduction to Java's Swing library
  - Frames
  - Basic event handling
- Announcements
  - Continue working on Project 3

AWT

- The first version of Java included a graphics system called **AWT** (**A**bstract **W**indow **T**oolkit)
  - Each AWT object is represented and drawn by the host operating system's analogue for that object
  - So, creating an AWT dialog box simply creates a native dialog box that is presented to the user
- But Windows, UNIX, MacOS dialog boxes look very different from each other
  - Difficult to create platform-independent applications with platform-independent user interfaces

Swing

- Java2 introduced the **Swing** toolset
  - Layered on top of AWT
  - Swing essentially works by creating a blank canvas, and then painting all user interface elements inside that window
  - Because it only depends on the operating system for canvas creation, the majority of an application can be designed to look the same on any platform
- Swing is implemented by calling AWT methods to draw using the operating system's graphic primitives

Packages

- AWT classes are in **java.awt** and its subpackages

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

- Swing classes are in the package **javax.swing** and its subpackages

```
import javax.swing.*;
```

- In some cases the division between Swing and AWT is blurry

The next few weeks…

- We will learn some Swing (and also some AWT) by studying example Java programs
- A complete description of all packages, classes, methods, etc. can be found at **http://java.sun.com/javase/6/docs/api/**

JFrame

- What most people consider a "window" is referred to in Swing as a JFrame (javax.swing.JFrame)
- JFrames use Frames (java.awt.Frame), which in turn call the operating system's native windowing methods
- The following code creates a window and displays it

Example: creating a JFrame

```
import javax.swing.*;

public class ExampleJFrame {
   public static void main(String[ ] args) {
       JFrame frame = new JFrame("My Frame");
       frame.setSize(200,200);
       frame. setVisible(true);
    }
}
```


Class Exercise

- Download from **http://cs.ua.edu/325/Summer2007/examples/ExampleJFrame.java**
- Compile and run
- What happens when you click on the close button (X)?
  - The program continues running, and you're not returned to the command prompt
  - This is intentional, because an application can own several windows (for instance, a word processor with multiple open documents), and closing just one of them should not quit the entire application
  - Click on command window, then press control-C

Improved JFrame program

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

public class ExampleJFrame2 {
   public static void main(String[ ] args) {
   JFrame frame = new JFrame("My Frame 2");
   frame.addWindowListener
              (new MyWindowListener( ));
   frame.setSize(200,200);
   frame.setVisible(true);
   }
}
```


Better JFrame program (cont.)

```
class MyWindowListener extends WindowAdapter {

   public void windowClosing (WindowEvent e) {
        System.exit(0);
   }

}
```


Things to note

- Our new class MyWindowListener is a subclass of the class WindowAdapter, which is defined in the imported package java.awt.event
- We added a *listener* on the JFrame
  - Listeners are frequently used in event-driven programs such as a GUI applications, where the events are things like mouse clicks and key presses
  - Just register a listener, and then when the event occurs, a method of the listener is automatically called

Class Exercise

- Download from **http://cs.ua.edu/325/Summer2007/examples/ExampleJFrame2.java**
- Compile and run
- What happens when you click on the close button (X)?
- Note that the compiler creates two class files: **ExampleJFrame2.class** and **MyWindowListener.class**

Shorter JFrame program

```
public class ExampleJFrame3 {
   public static void main(String[ ] args) {
   JFrame frame = new JFrame("My Frame 3");
   frame.addWindowListener(new WindowAdapter () {
        public void windowClosing (WindowEvent e) {
           System.exit(0);
        }
    });        // compiler creates an "anonymous" class
   frame.setSize(200,200);
   frame.setVisible(true);
   }
}
```


Class Exercise

- Download from **http://cs.ua.edu/325/Summer2007/examples/ExampleJFrame3.java**
- Compile and run
- What happens when you click on the close button (X)?
- Note that the compiler creates two class files: **ExampleJFrame3.class** and **ExampleJFrame3$1.class**
  - **ExampleJFrame3$1.class** refers to the "anonymous" class that was automatically created

Shortest JFrame program

```
public class ExampleJFrame4 {
   public static void main(String[ ] args) {
   JFrame frame = new JFrame("My Frame 4");
   frame.setDefaultCloseOperation
              (JFrame.EXIT_ON_CLOSE);

   frame.setSize(200,200);
   frame.setVisible(true);
   }
}
```


Class Exercise

- Download from **http://cs.ua.edu/325/Summer2007/examples/ExampleJFrame4.java**
- Compile and run
- Make sure it works the same way as **ExampleJFrame3.java**

Recognizing key events

- Pressing a key
- Releasing a key
- Typing a key (usually the preferred way to capture character input)
  - Some characters are produced by a single key press ('a')
  - Other characters require a series of key presses (Shift + 'a' => 'A')

Key event program

```
public class ExampleKeyPress {
   public static void main(String[ ] args) {
   JFrame frame = new JFrame("Key Press Frame");

   frame.addWindowListener(new WindowAdapter ( ) {
        public void windowClosing (WindowEvent e) {
           System.exit(0);
        }
    });
// continued next slide
```

Key event program (cont.)

```
    frame.addKeyListener( new KeyAdapter( ) {
          public void keyPressed(KeyEvent e) {
             System.out.println("PRESSED " + e);
          }
          public void keyReleased(KeyEvent e) {
             System.out.println("RELEASED " + e);
          }
          public void keyTyped(KeyEvent e) {
             System.out.println("TYPED " + e);
          }
       } );
    frame.setSize(400,200);
    frame.setVisible(true);
   }
}
```


Things to note

- Whenever an event occurs, Swing "pushes" that event along to our event handlers
- Everything works fine provided that the "focus" remains on our JFrame
  - When focus is not on the JFrame, events do not get handled by the program
  - More complex applications with multiple windows or subwindows must carefully manage where the focus is at all times

Class Exercise

- Download from **http://cs.ua.edu/325/Summer2007/examples/ExampleKeyPress.java**
- Compile and run
- When does each kind of key event get generated?
  - Try typing 'a' (see output on command window)
  - Try pressing and releasing Shift
  - Try typing 'A'
  - Try pressing and holding 'a'

Simpler key event program

```
public class ExampleKeyPress2 {
   public static void main(String[ ] args) {
   …
   frame.addKeyListener( new KeyAdapter( ) {
        public void keyTyped(KeyEvent e) {
           System.out.println("TYPED " + e);
        }
    } );        // ignore keyPressed and keyReleased
   …
}
```


Class Exercise

- Download from **http://cs.ua.edu/325/Summer2007/examples/ExampleKeyPress2.java**
- Compile and run
- When does each kind of key event get generated?
  - Try typing 'a'
  - Try typing 'A'
  - Try typing control-A