Lecture 24: Event Programming; Applets

CS 170, Section 000
1 December 2009
Lecture Plan

- Events (essentials, Ch. 11, 15)

- **Applets** (Ch. 17)
  - Life of an applet
  - Graphical User Interfaces (GUI) in Java
  - Programming GUI Events

- Homework 8
Review: What is an interface?

• Interface is to specify **behavior** for objects
  – For example, we can specify that the objects are comparable, edible, cloneable using appropriate interfaces such as `Comparable`, `Edible`, and `Cloneable`

• You can create an instance from a class **that implements an interface**

• You can use an interface as a data type for a variable, as the result of casting, and so on.
Example 1: The Cloneable Interfaces

- A class that implements the `Cloneable` interface declares that its objects can be copied using the `clone()` method defined in the `Object` class.

```java
package java.lang;

public interface Cloneable {
}
```
Implementing Cloneable Interface

• Declare a custom class that implements the Cloneable interface

```java
public class House implements Cloneable{
    private int id;
    private double area;
    private java.util.Date whenBuilt;
    ...
    public Object clone() { //overrides Object’s clone() method
        try {
            return super.clone(); //Object’s clone implemen.
        } catch (CloneNotSupportedException ex) {
            return null;
        }
    }
    ...
}
```

House.java
Shallow vs. Deep Copy

House house1 = new House(1, 1750.50, new Date("1/1/2009"));
House house2 = (House) house1.clone();
house1.area = 3000; house1.whenBuilt = new Date("1/1/1900");

house2.area = ? house2.whenBuilt=?
Event Programming

• Event programming
  – the flow of the program is determined by user actions (mouse clicks, key presses) or messages from other programs.

• Components
  – Event sources: user interface components or other sources that generate the events
  – Events: user actions or other events
  – Event listener: reactions on events
Events

• An *event* can be defined as a type of signal to the program that something has happened.

• The event is generated by external user actions such as mouse movements, mouse clicks, and keystrokes, or by the operating system or program activities, such as a timer.
The Role of Event Listeners

- Role of an event listener is similar to Fred and George Weasley’s “Extendable Ears” from Harry Potter

- Suppose that you wanted to use these magical listeners to detect events like mouse clicks.

- All you need to do is send those ears into the room so they keep you informed on anything that goes on there, making it possible for you to respond.
Event Types

- Java events come in many different types including:
  - Mouse events, which occur when the user moves or clicks the mouse
  - Keyboard events, which occur when the user types on the keyboard
  - Action events, which occur in response to user-interface actions

- Each event type is associated with a set of methods that specify how listeners should respond. These methods are defined in a listener interface for each event type.

- As an example, one of the methods in the mouse listener interface is `mouseClicked`. As you would expect, Java calls that method when you click the mouse.

- Listener methods like `mouseClicked` take a parameter that contains more information about the event. In the case of `mouseClicked`, the argument is a `MouseEvent` indicating the location at which the click occurred.
Basic Steps in Event Programming

• Define event listener
  – implements an interface `ActionListener` which contains a method `actionPerformed()` for processing the event

```java
class OKListenerClass implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("OK button clicked");
    }
}
```

• Register event listener with event sources

```java
JButton jbtOK = new JButton("OK");
...
OKListenerClass listener1 = new OKListenerClass();
jbtOK.addActionListener(listener1);
```
Taste of Event-Driven Programming

- The example displays two buttons in the frame. A message is displayed on the console when a button is clicked.
- It’s all about the interfaces!

- HandleEvent.java
Event Classes
## Selected User Actions

<table>
<thead>
<tr>
<th>User Action</th>
<th>Source Object</th>
<th>Event Type Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click a button</td>
<td>JButton</td>
<td>ActionEvent</td>
</tr>
<tr>
<td>Click a check box</td>
<td>JCheckBox</td>
<td>ItemEvent, ActionEvent</td>
</tr>
<tr>
<td>Click a radio button</td>
<td>JRadioButton</td>
<td>ItemEvent, ActionEvent</td>
</tr>
<tr>
<td>Press return on a text field</td>
<td>JTextField</td>
<td>ActionEvent</td>
</tr>
<tr>
<td>Select a new item</td>
<td>JComboBox</td>
<td>ItemEvent, ActionEvent</td>
</tr>
<tr>
<td>Window opened, closed, etc.</td>
<td>Window</td>
<td>WindowEvent</td>
</tr>
<tr>
<td>Mouse pressed, released, etc.</td>
<td>Component</td>
<td>MouseEvent</td>
</tr>
<tr>
<td>Key released, pressed, etc.</td>
<td>Component</td>
<td>KeyEvent</td>
</tr>
</tbody>
</table>
The Delegation Model

Trigger an event

User Action

source: SourceClass
+addXListener(listener: XListener)

XListener
+handler(event: XEvent)

listener: ListenerClass

(a) A generic source component with a generic listener

Register by invoking
source.addXListener(listener);

(b) A JButton source component with an ActionListener

source: JButton
+addActionListener(listener: ActionListener)

ActionListener
+actionPerformed(event: ActionEvent)

listener: CustomListenerClass

Register by invoking
source.addActionListener(listener);
Internal Function of a Source Component

(a) Internal function of a generic source object

source: SourceClass

+addXListener(XListener listener)

An event is triggered

event: XEvent

Invoke

listener1.handler(event)

listener2.handler(event)

…

listenern.handler(event)

(b) Internal function of a JButton object

source: JButton

+addActionListener(ActionListener listener)

An event is triggered

event: ActionEvent

Invoke

listener1.actionPerformed(event)

listener2.actionPerformed(event)

…

listenern.actionPerformed(event)
## Selected Event Handlers

<table>
<thead>
<tr>
<th>Event Class</th>
<th>Listener Interface</th>
<th>Listener Methods (Handlers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionEvent</td>
<td>ActionListener</td>
<td>actionPerformed(ActionEvent)</td>
</tr>
<tr>
<td>ItemEvent</td>
<td>ItemListener</td>
<td>itemStateChanged(ItemEvent)</td>
</tr>
<tr>
<td>WindowEvent</td>
<td>WindowListener</td>
<td>windowClosing(WindowEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowOpened(WindowEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowIconified(WindowEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowDeiconified(WindowEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowClosed(WindowEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowActivated(WindowEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>windowDeactivated(WindowEvent)</td>
</tr>
<tr>
<td>ContainerEvent</td>
<td>ContainerListener</td>
<td>componentAdded(ContainerEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>componentRemoved(ContainerEvent)</td>
</tr>
<tr>
<td>MouseEvent</td>
<td>MouseListener</td>
<td>mousePressed(MouseEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mouseReleased(MouseEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mouseClicked(MouseEvent)</td>
</tr>
<tr>
<td>KeyEvent</td>
<td>KeyListener</td>
<td>keyPressed(KeyEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>keyReleased(KeyEvent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>keyTyped(KeyEvent)</td>
</tr>
</tbody>
</table>
Listener class as **Inner Classes**

- A listener class is designed specifically to create a listener object for a GUI component (e.g., a button).
- It is appropriate to define the listener class inside the frame class as an inner class.
Inner Classes

• An *inner class*, or nested class, is a class defined within the scope of another class
  – Defined inside a class but outside its methods
  – Defined inside a method

• Inner classes can make programs simple and concise.
• Compiler turns an inner class into a regular class file \texttt{OuterClassName$InnerClassName.class}.

• An inner class can reference the data and methods defined in the outer class in which it nests
Example: Defining Listener Class as an Inner Class

- A simple event demo program with an OK button
  - SimpleEventDemo.java

- The event demo program using inner class
  - SimpleEventDemoInnerClass.java
The Timer Class

• Some non-GUI components can fire events. The Timer class is a source component that fires an ActionEvent at a predefined rate.

• The Timer class can be used to control animations.

<table>
<thead>
<tr>
<th>javax.swing.Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Timer(delay: int, listener: ActionListener)</td>
</tr>
<tr>
<td>+addActionListener(listener: ActionListener): void</td>
</tr>
<tr>
<td>+start(): void</td>
</tr>
<tr>
<td>+stop(): void</td>
</tr>
<tr>
<td>+setDelay(delay: int): void</td>
</tr>
</tbody>
</table>

Creates a Timer with a specified delay in milliseconds and an ActionListener.
Adds an ActionListener to the timer.
Starts this timer.
Stops this timer.
Sets a new delay value for this timer.
How to Use Timer Class?

- Define an listener class that implements the ActionListener

```java
class MyListener implements ActionListener {
    void actionPerformed(ActionEvent event) {
        // This action will be executed at each timer event
    }
}
```

- Add listener to timer and start the timer

```java
MyListener listener = new MyListener();
Timer t = new Timer(interval, listener);
t.start();
```
Example: Timer Class

• Count down example – count down every second
  – TimerTester.java

• Shape mover example – move a box to a different position every second
  – TimerTester2.java
Review question

• Which of the following statements are true?

A. You can use the addActionListener method in the Timer class to add a listener.
B. You can specify a delay in the Timer constructor.
C. You must always specify a listener when creating a Timer object.
D. When a timer is created, it is automatically started.
When Web Became Fun: Applets

• Back in 1996, the Web was a boring place

• Then, Java (and Applets) came along
  – Othello: http://www.voges.info/marvin/
    • Can be improved... for extra Credit (homework 9)
public class MyApplet extends java.applet.Applet {

/** The no-arg constructor is called by the browser when the Web page containing this applet is initially loaded, or reloaded */
public MyApplet() {
...
}

/** Called by the browser after the applet is loaded */
public void init() {
...
}

/** Called by the browser after the init() method, or every time the Web page is visited */
public void start() {
...
}

/** Called by the browser when the page containing this applet becomes inactive */
public void stop() {
...
}

/** Called by the browser when the Web browser exits */
public void destroy() {
...
}

/** Other methods if necessary... */
}
The Applet Class, cont.

When the applet is loaded, the Web browser creates an instance of the applet by invoking the applet's no-arg constructor. The browser uses the init, start, stop, and destroy methods to control the applet. By default, these methods do nothing. To perform specific functions, they need to be modified in the user's applet so that the browser can call your code properly.
Browser Calling Applet Methods

- JVM loads the applet class
- Browser creates the applet
- Browser invokes init()
- Browser invokes start()
- Browser invokes stop()
- Browser invokes destroy()

- Loaded
- Created
- Initialized
- Started
- Stopped
- Destroyed
The `init()` Method

Invoked when the applet is first loaded and again if the applet is reloaded.

A subclass of `Applet` should override this method if the subclass has an initialization to perform. The functions usually implemented in this method include creating new threads, loading images, setting up user-interface components, and getting string parameter values from the `<applet>` tag in the HTML page.
The \texttt{start()} Method

Invoked after the \texttt{init()} method is executed; also called whenever the applet becomes active again after a period of inactivity (for example, when the user returns to the page containing the applet after surfing other Web pages).

A subclass of \texttt{Applet} overrides this method if it has any operation that needs to be performed whenever the Web page containing the applet is visited. An applet with animation, for example, might use the \texttt{start} method to resume animation.
The `stop()` Method

The opposite of the `start()` method, which is called when the user moves back to the page containing the applet; the `stop()` method is invoked when the user moves off the page.

A subclass of `Applet` overrides this method if it has any operation that needs to be performed each time the Web page containing the applet is no longer visible. When the user leaves the page, any threads the applet has started but not completed will continue to run. You should override the `stop` method to suspend the running threads so that the applet does not take up system resources when it is inactive.
The `destroy()` Method

Invoked when the browser exits normally to inform the applet that it is no longer needed and that it should release any resources it has allocated.

A subclass of `Applet` overrides this method if it has any operation that needs to be performed before it is destroyed. Usually, you won't need to override this method unless you wish to release specific resources, such as threads that the applet created.
The JApplet Class

To use Swing components in Java applets, it is necessary to create a Java applet that extends `javax.swing.JApplet`, which is a subclass of `java.applet.Applet`.

`JApplet` inherits all the methods from the `Applet` class. In addition, it provides support for laying out Swing components.
First Simple Applet

// WelcomeApplet.java: Applet for displaying a message
import javax.swing.*;
public class WelcomeApplet extends JApplet {
   /** Initialize the applet */
   public void init() {
      add(new JLabel("Welcome to Java", JLabel.CENTER));
   }
}

// WelcomeApplet.java: Applet for displaying a message
import javax.swing.*;
public class WelcomeApplet extends JApplet {
   /** Initialize the applet */
   public WelcomeApplet() {
      add(new JLabel("Welcome to Java", JLabel.CENTER));
   }
}
First Simple Applet

```html
<html>
<head>
<title>Welcome Java Applet</title>
</head>
<body>
<applet
code = "WelcomeApplet.class"
width = 350
height = 200>

</applet>
</body>
</html>
```
Writing Applets

- Always extends the JApplet class, which is a subclass of Applet for Swing components.

- Override init(), start(), stop(), and destroy() if necessary. By default, these methods are empty.

- Add your own methods and data if necessary.

- Applets are always embedded in an HTML page.
The `<applet>` HTML Tag

```html
<applet
    code=classfilename.class
    width=applet_viewing_width_in_pixels
    height=applet_viewing_height_in_pixels
    [archive=archivefile]
    [codebase=applet_url]
    [vspace=vertical_margin]
    [hspace=horizontal_margin]
    [align=applet_alignment]
    [alt=alternative_text]
>
<param name=param_name1
    value=param_value1>
</applet>
```
Passing Parameters to Applets

```xml
<applet
    code = "DisplayMessage.class"
    width = 200
    height = 50>
<param name=MESSAGE value="Welcome to Java">
<param name=X value=20>
<param name=Y value=20>
alt="You must have a Java-enabled browser to view the applet"
</applet>
```
Example: Passing Parameters to Java Applets

Objective: Display a message at a specified location. The message and the location \((x, y)\) are obtained from the HTML source.

![Applet Viewer: DisplayMessage.class](Image)

Welcome to Java

Applet started.

DisplayMessage.java
Applications vs. Applets

• Similarities
  – Since JFrame and JApplet both are subclasses of the Container class, all the user interface components, layout managers, and event-handling features are the same for both classes.

• Differences
  – Applications are invoked from the static main method by the Java interpreter, and applets are run by the Web browser. The Web browser creates an instance of the applet using the applet’s no-arg constructor and controls and executes the applet through the init, start, stop, and destroy methods.
  – Applets have security restrictions
  – Web browser creates graphical environment for applets, GUI applications are placed in a frame.
Security Restrictions on Applets

- Applets are not allowed to read from, or write to, the file system of the computer viewing the applets.
- Applets are not allowed to run any programs on the browser’s computer.
- Applets are not allowed to establish connections between the user’s computer and another computer except with the server where the applets are stored.
Conversions Between Applications and Applets

• Conversions between applications and applets are simple and easy.

• You can always convert an applet into an application.

• You can convert an application to an applet as long as security restrictions are not violated.
Example: Running a Program as an Applet and as an Application

- Objective: Modify `DisplayMessage` to enable it to run both as an applet and as an application.

`DisplayMessageApp`
Case Study: TicTacToe

- **Token used in the cell (default: ' ')**
- Returns the token in the cell.
- Sets a new token in the cell.
- Paints the token in the cell.
- Handles a mouse click on the cell.

**Cell**

- `-token: char`
- `+getToken(): char`
- `+setToken(token: char): void`
- `#paintComponent(g: Graphics): void`
- `+mouseClicked(e: MouseEvent): void`
Case Study: TicTacToe, cont.

TicTacToe

- whoseTurn: char
  - Indicates which player has the turn, initially 'X'.
- cell: Cell[][]
  - A 3 by 3, two dimensional array for cells.
- jlblStatus: JLabel
  - A label to display game status.

+ TicTacToe()
+ isFull(): boolean
+ isWon(token: char): boolean

Constructs the TicTacToe user interface.
Returns true if all cells are filled.
Returns true if a player with the specified token has won.
Locating Resource for Applets

Due to security restrictions, applets cannot access local files. How can an applet load resource files for image and audio?
The java.net.URL class can be used to identify files (image, audio, text, etc.) on the Internet. In general, a URL (Uniform Resource Locator) is a pointer to a “resource” on the World Wide Web on a local machine or a remote host. A resource can be something as simple as a file or a directory.
Displaying Image

Write a program that displays an image from /image/us.gif in the class directory on a panel.

DisplayImageWithURL

Run
HW8: Game of Life

• A mathematical game invented by mathematician John Conway in 1970

• Game rules
  – A dead cell with exactly three live neighbors becomes a live cell (birth).
  – A live cell with two or three live neighbors stays alive (survival).
  – In all other cases, a cell dies or remains dead (overcrowding or loneliness).

• Resources and demos
  – http://www.ibiblio.org/lifepatterns/

• Implementation
  – Use timer class for animation
  – Use two dimensional arrays for cell updates, then drawImage!
  – Extra credit: Draw individual cells
    • Alternatively: draw Image!