Event Handling
part 3

Handling Window Events Example Revisited

Window Exit Handler Example Reveisted

```
// Event Handlers for WindowListener
   public void windowActivated (WindowEvent we) {    }
   public void windowClosed (WindowEvent we) {  }
   public void windowClosing (WindowEvent we) {
      JOptionPane.showMessageDialog(null, “Good Bye”);
      System.exit(0)
   }
   public void windowDeactivated (WindowEvent we) {   }
   public void windowDeiconified (WindowEvent we) {   }
   public void windowIconified (WindowEvent we) {   }
   public void windowOpened (WindowEvent we) {   }
```

Last Code Example

- Problem
  - Interested in windowClosing method only
  - But have to provide definitions of all the methods, Why ?
  - Because a class implementing an interface has to provide definitions of all methods present in that interface.
- Solution
  - Use Adapter classes

Adapter Classes

Adapter Classes

- For listener interfaces containing more than one event handling methods, jdk defines adapter classes. Examples are
  - For WindowListener → WindowAdapter
  - For MouseMotionListener → MouseMotionAdapter
  - and many more

- Adapter classes provides definitions for all the methods (empty bodies) of their corresponding Listener interface

- It means that WindowAdapter class implements WindowListener interface.

Adapter Classes

```
public interface MouseMotionListener {

   public void mouseDragged (MouseEvent me);

   public void mouseMoved (MouseEvent me);

}
```

```
public class MouseMotionAdapter impelments
                                        MouseMotionListener {

   public void mouseDragged (MouseEvent me) { }

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

How to use Adapter Classes

- Previously handler class need to implement interface

```
public class EventsEx implements
                MouseMotionListener { …. }
```

- Therefore it has to provide definitions of all the methods inside that interface

How to use Adapter Classes

- Now our handler class will inherit from adapter class
  - public class EventsEx extends MouseMotionAdapter { …. }
  - Due to inheritance, all the methods of the adapter class will be available inside our handler class
  - Since adapter classes has already provided definitions with empty bodies.
  - We do not have to provide implementations of all the methods again
  - We only need to override our method of interest.

Adapter Classes

Listener	Adapter Class (If Any)	Registration Method
ActionListener		addActionListener
AdjustmentListener		addAdjustmentListener
ComponentListener	ComponentAdapter	addComponentListener
ContainerListener	ContainerAdapter	addContainerListener
FocusListener	FocusAdapter	addFocusListener
ItemListener		addItemListener
KeyListener	KeyAdapter	addKeyListener
MouseListener	MouseAdapter	addMouseListener
MouseMotionListener	MouseMotionAdapter	addMouseMotionListener
TextListener		addTextListener
WindowListener	WindowAdapter	addWindowListener

Example Code: Adapter Classes Modification of EventsEx

```
// File EventsEx.java, Code listed in Handout section 13.1
……..

public class EventsEx extends WindowAdapter {

    JFrame f;
    JLabel coord;

    public void initGUI () {
        …….  // set layouts

        f.addWindowListener(this);
       …….
    }//end initGUI
```

Example Code: Adpater Classes

Modification of EventsEx.java

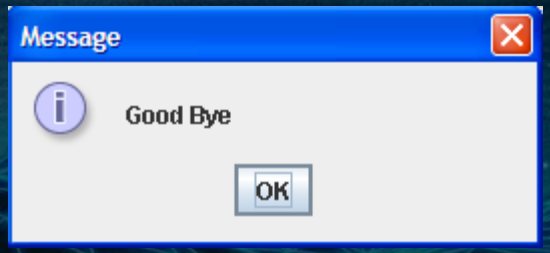
```
// Event Handler for WindowListener
   public void windowClosing (WindowEvent we) {
      JOptionPane.showMessageDialog(null, “Good Bye”);
      System.exit(0)
   }

  public static void main (String args[ ]){
        EventsEx ex = new EventsEx();
   }
} // end class
```

Example Code: Window Exit Handler

Modification of EventsEx.java

Output

When user closes the window,
Message would be displayed



After pressing Ok button
Program will exit

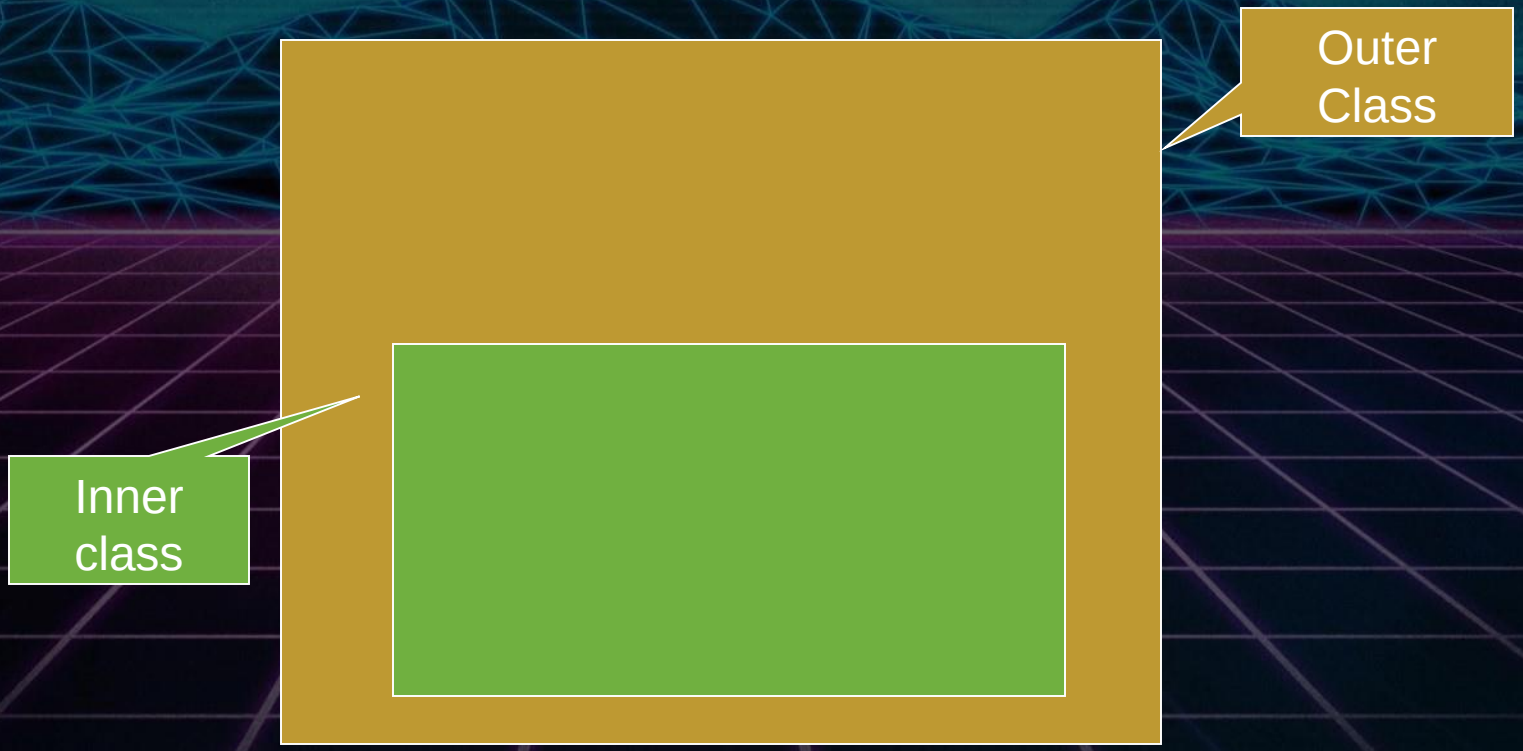
Last Code Example

- We have inherited from WindowAdapter
- What if we want to use MouseMotionAdpater as well ? or what if our class already inherits form some other class ?
- **Problem**
  - But, Java allows single inheritance
- **Solution**
  - Use Inner Classes

Inner Classes

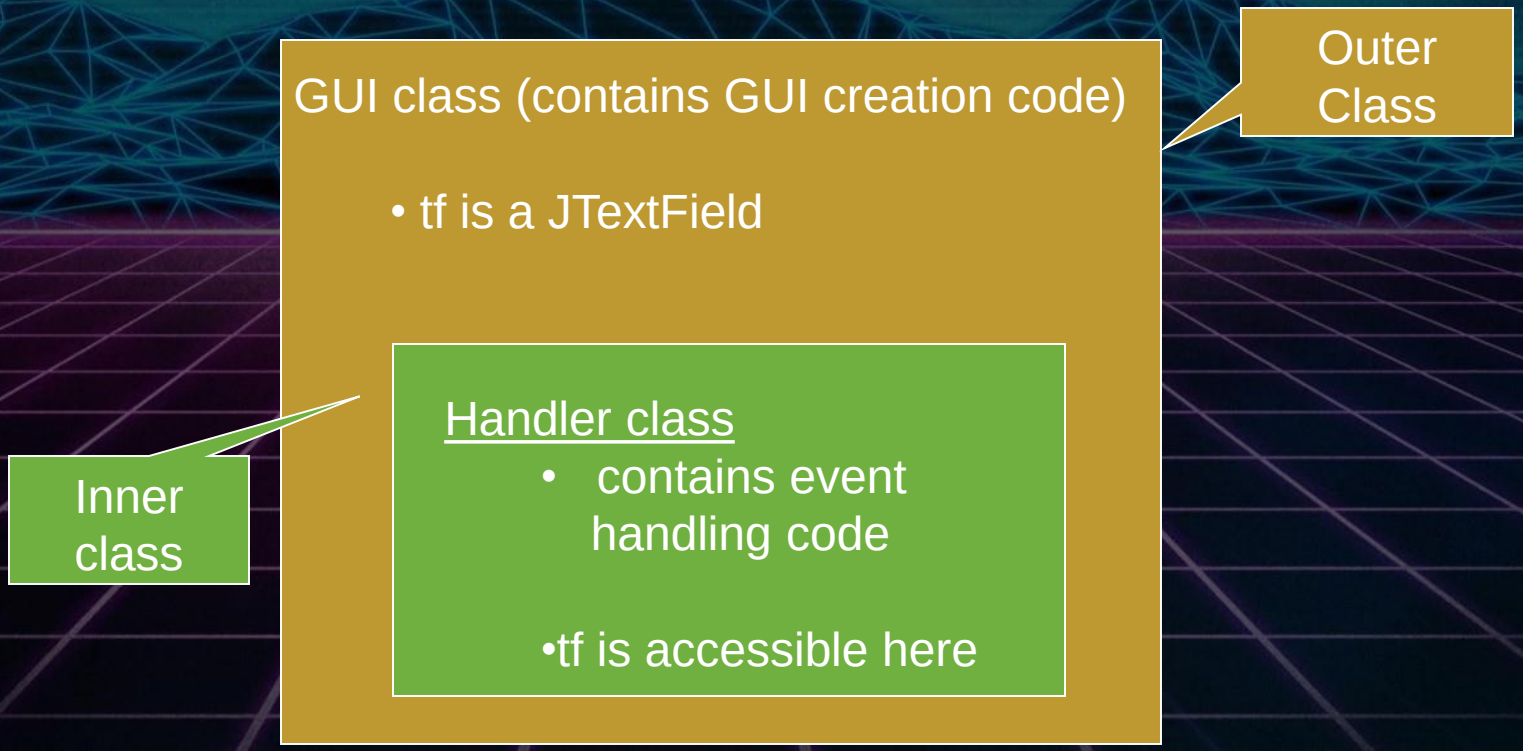
Inner Classes

- A class defined inside another class



Inner Classes

- Inner class can access the instance variables and members of outer class



Inner Classes

- It can have constructors, instance variables and methods, just like a regular class

- Generally used as a private utility class which does not need to be seen by others classes

Inner Classes

Handling Window Events

Example Code: Inner Classes Handling Window Events

```
// File EventsEx.java, Code listed in Handout section 13.2
……..
public class EventsEx {

    JFrame f;
    JLabel coord;


   //inner class
  private class WindowHandler extends WindowAdapter {

     // Event Handler for WindowListener
     public void windowClosing (WindowEvent we) {
         JOptionPane.showMessageDialog(null, “Good Bye”);
         System.exit(0)
     }
 } // end of WindowHandler
```

Example Code: Inner Classes Handling Window Events

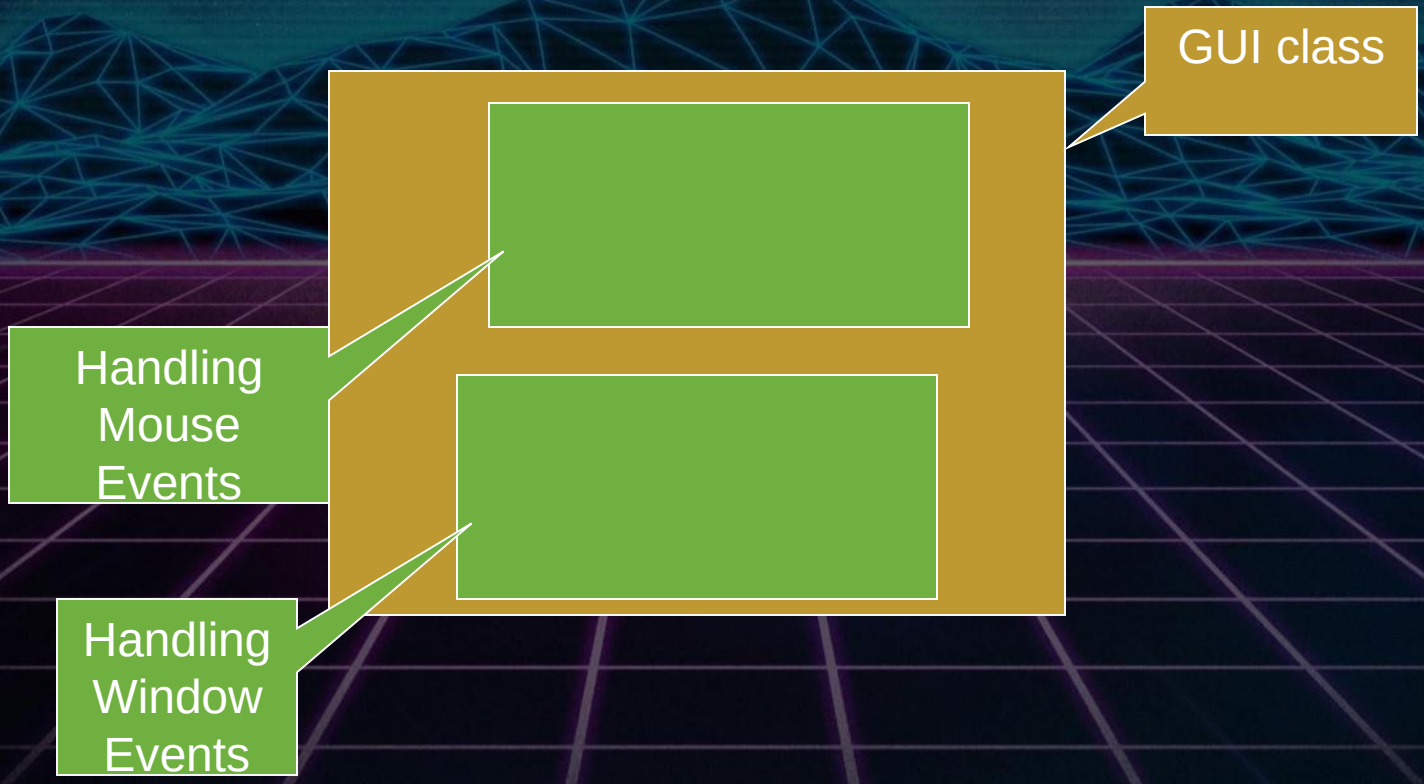
```
public void initGUI () {
        …….  // set layouts

        WindowHandler handler = new Window Handler ();
        f.addWindowListener(handler);
       …….
    }//end initGUI

     // main method

} // end of EventsEx class
```

Inner Classes

Handling Window & Mouse Motion Events

Inner Classes

- Handling Windows & Mouse motion events



Example Code: Inner Classes Handling Window & MouseMotion Events

```
// File EventsEx.java, Code listed in Handout section 13.3
……..
public class EventsEx {

    JFrame f;
    JLabel coord;


   //inner class
  private class WindowHandler extends WindowAdapter {

     // Event Handler for WindowListener
     public void windowClosing (WindowEvent we) {
         JOptionPane.showMessageDialog(null, “Good Bye”);
         System.exit(0)
     }
 } // end of WindowHandler
```

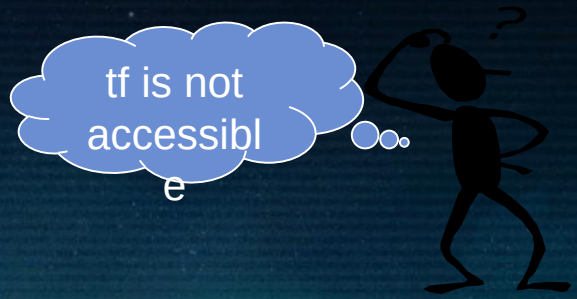
Example Code: Inner Classes Handling Window Events

```
 //inner class
private class MouseHandler extends MouseMotionAdapter {

   // Event Handler for mouse motion events
   public void mouseMoved (MouseEvent me) {
      int x = me.getX();
      int y = me.getY();
      coord.setText("Moved at [" + x + "," + y + "]" );
   }
} // end of MouseHandler
```

Example Code: Inner Classes Handling Window Events

```
public void initGUI () {
        …….  // set layouts

        WindowHandler wHandler = new WindowHandler ();
        f.addWindowListener(wHandler);

       MouseHandler mHandler = new MouseHandler();
       f.addMouseMotionListener(mHandler);

        …….
    }//end initGUI

     //main method

} // end EventsExclass
```

Why Not Outer Class?



Event Generator class

```
.....
.....

JFrame frame;
JTextField tf;
.....
.....
```

Action Event Handler

```
..........
String s = tf.getText();


...........
```

Window Event Handler

Anonymous Inner Classes

Anonymous Inner Classes

- Has no name
- Same as inner class in capabilities
- much shorter
- difficult to understand

Named vs. Anonymous Objects

- Named
  - String s = “hello”;

    System.out.println(s);
  - “hello” has a named reference *s*.

- Anonymous
  - System.out.println(“hello”);

Example Code: Anonymous Classes

```
// File EventsEx.java, Code listed in Handout section 13.4
……..


public class EventsEx {

    JFrame f;
    JLabel coord;
```

Example Code: Anonymous Classes

```
public void initGUI () {
        …….  // set layouts

     //anonymous inner class
    f.addWindowListner (new WindowAdapter ( ) {

    public void windowClosing (WindowEvent we) {
        JOptionPane.showMessageDialog(null, “Good Bye”);
        System.exit(0)
    }
  }
  );
         …….
}//end initGUI
```

Summary of Handling Events Approahces

Approaches for Handling Events

1. By implementing Interfaces
2. By extending from Adapter classes

To implement the above two techniques we can use

- Same class
  - (putting event handler & generator in one class)
- Separate class
  - Outer class
    - Putting event handlers & generator in two different classes
  - Inner classes
  - Anonymous Inner classes