# Software Construction & Development

WEEK 03

## Event-Driven Programming

## What is Event-Driven Programming?

It is a programming paradigm in which the flow of a program is driven by events.

# Event Handling

CHAPTER # 24

# The Delegation Event Model

It defines standard and consistent mechanisms to generate and process events.

The Concept:

A **source** generates an **event** and sends it to one or more **listeners**

## Event

**An event is an object that describes a state change in a source.**

An event can be generated as a consequence of a person interacting with the elements in a graphical user interface.

- Pressing a button, entering a character via the keyboard, etc.

An event may also be generated when:

- A timer expires
- A counter exceeds a value
- A software or hardware failure occurs
- An operation is completed

# Event Sources

**A source is an object that generates an event.**

- Sources may generate more than one type of event.
- A source must register listeners (why…?)

A listener register method looks like:

```
public void addTypeListener (TypeListener el )
```

| | |
|---|---|
| Type | = name of event |
| el | = reference of the event listener |

## Event Sources

A source must also provide a method that allows a listener to unregister an interest in a specific type of event.

A listener unregister method looks like:

```
public void removeTypeListener(TypeListener el )
```

| | |
|---|---|
| Type | = name of event |
| el | = reference of the event listener |

## Multicasting & Unicasting an Event

**Multicasting:**

When an event occurs, **all** registered listeners are notified and receive a copy of the event object.

**Unicasting:**

Some sources may allow **only one** listener to register. When an event occurs, the registered listener is notified.

These methods throw an exception when more than one listener try to connect to it.

# Event Listeners

**A listener is an object that is notified when an event occurs.**

It has two requirements:

- Must be registered with one or more listeners
- Must implement methods to receive and process

An event handler must return quickly, and must not maintain control for an extended period. (Why?)

## Event Classes

**The classes that represent events are at the core of Java’s event handling mechanism.**

- **EventObject** is a superclass of all events.
- **AWTEvent** is a superclass of all AWT events that are handled by the delegation event model.

| Event Class | Description |
|---|---|
| ActionEvent | Generated when a button is pressed, a list item is double-clicked, or a menu item is selected. |
| AdjustmentEvent | Generated when a scroll bar is manipulated. |
| ComponentEvent | Generated when a component is hidden, moved, resized, or becomes visible. |
| ContainerEvent | Generated when a component is added to or removed from a container. |
| FocusEvent | Generated when a component gains or loses keyboard focus. |
| InputEvent | Abstract superclass for all component input event classes. |
| ItemEvent | Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected. |
| KeyEvent | Generated when input is received from the keyboard. |
| MouseEvent | Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component. |
| MouseWheelEvent | Generated when the mouse wheel is moved. |
| TextEvent | Generated when the value of a text area or text field is changed. |
| WindowEvent | Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit. |

**Table 24-1** Commonly Used Event Classes in **java.awt.event**

## Event Classes

**The ActionEvent Class**

An ActionEvent is generated when a button is pressed, a list item is double-clicked, or a menu item is selected.

**The AdjustmentEvent Class**

An AdjustmentEvent is generated by a scroll bar.

**The ComponentEvent Class**

A ComponentEvent is generated when the size, position, or visibility of a component is changed.

**The ContainerEvent Class**

A ContainerEvent is generated when a component is added to or removed from a container.

# Event Classes

## The FocusEvent Class

A FocusEvent is generated when a component gains or loses input focus.

## The InputEvent Class

The abstract class InputEvent is a subclass of **ComponentEvent** and is the superclass for component input events. Its subclasses are **KeyEvent** and **MouseEvent**.

## The ItemEvent Class

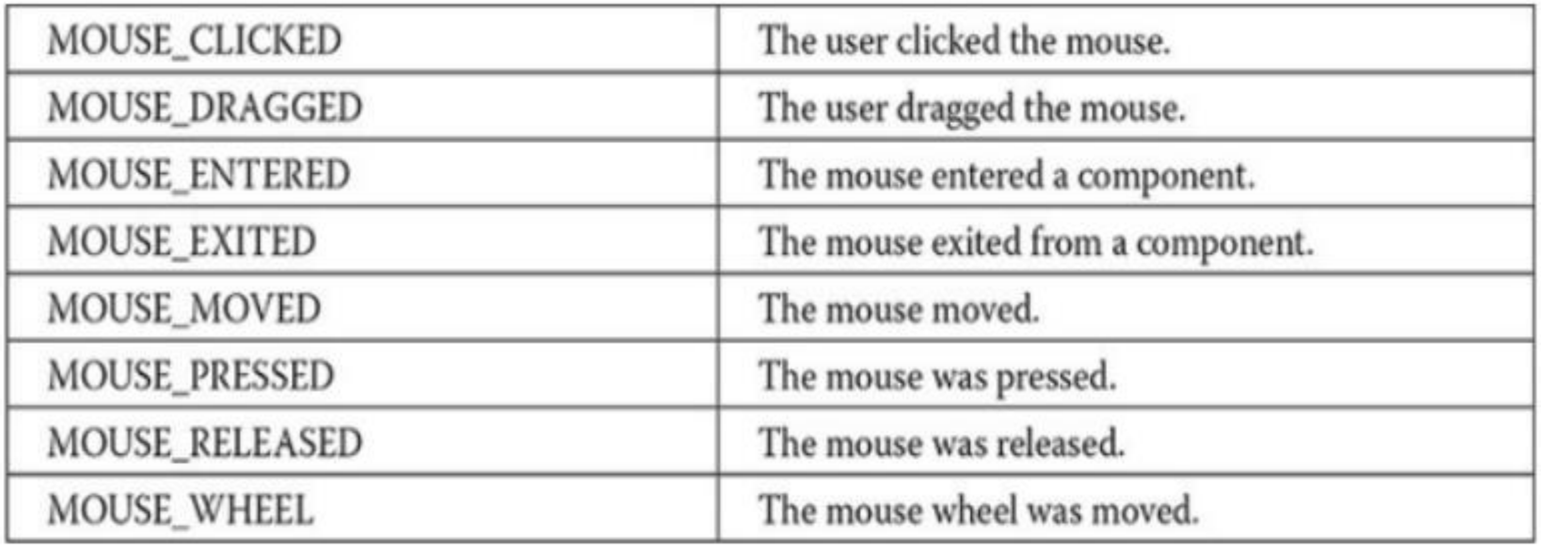
An ItemEvent is generated when a check box or a list item is clicked or when a checkable menu item is selected or deselected.

## Event Classes

**The KeyEvent Class**

A KeyEvent is generated when keyboard input occurs. (KEY_PRESSED, KEY_RELEASED, KEY_TYPED)

**The MouseEvent Class**



| | |
|---|---|
| MOUSE_CLICKED | The user clicked the mouse. |
| MOUSE_DRAGGED | The user dragged the mouse. |
| MOUSE_ENTERED | The mouse entered a component. |
| MOUSE_EXITED | The mouse exited from a component. |
| MOUSE_MOVED | The mouse moved. |
| MOUSE_PRESSED | The mouse was pressed. |
| MOUSE_RELEASED | The mouse was released. |
| MOUSE_WHEEL | The mouse wheel was moved. |

# Event Classes

The MouseWheelEvent Class

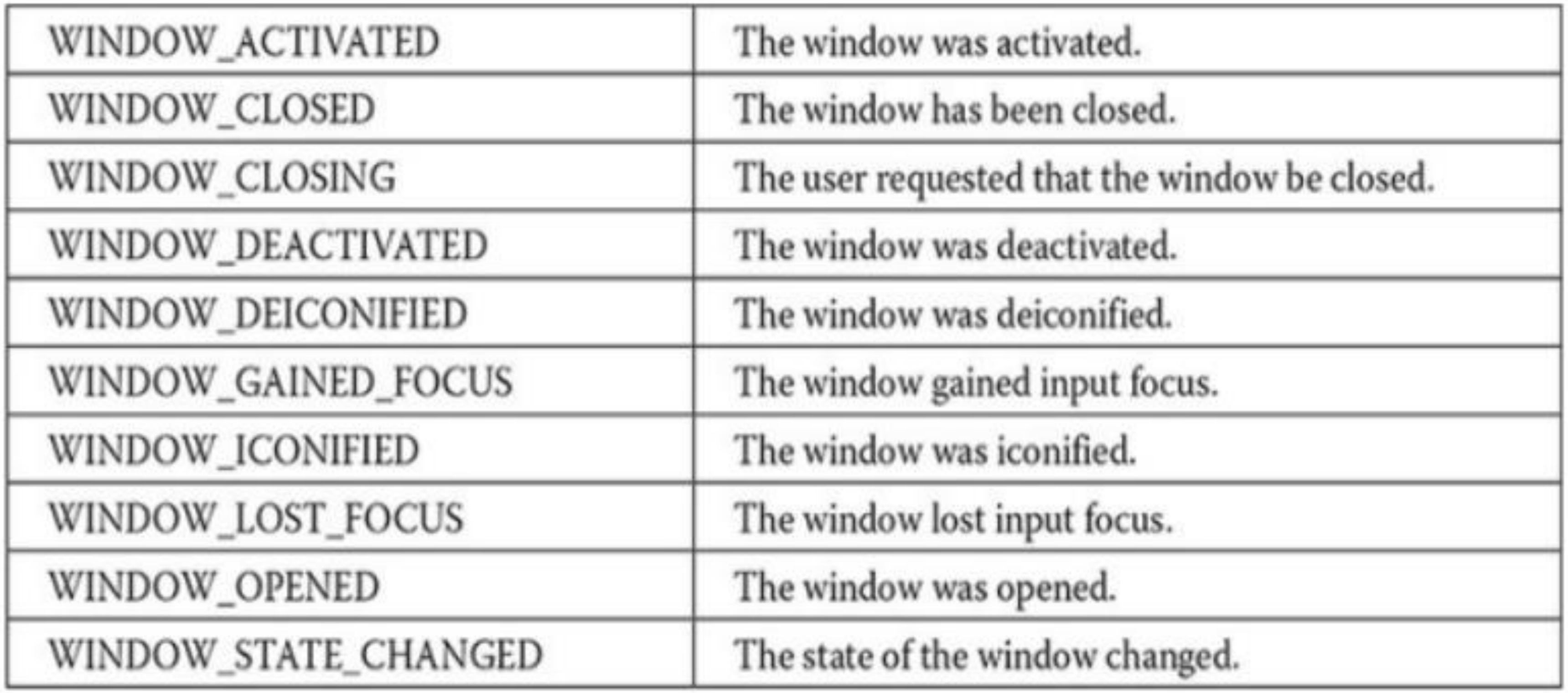
The MouseWheelEvent class encapsulates a mouse wheel event. It is a subclass of **MouseEvent**.

The TextEvent Class

Instances of this class describe text events. These are generated by text fields and text areas when characters are entered by a user or program.

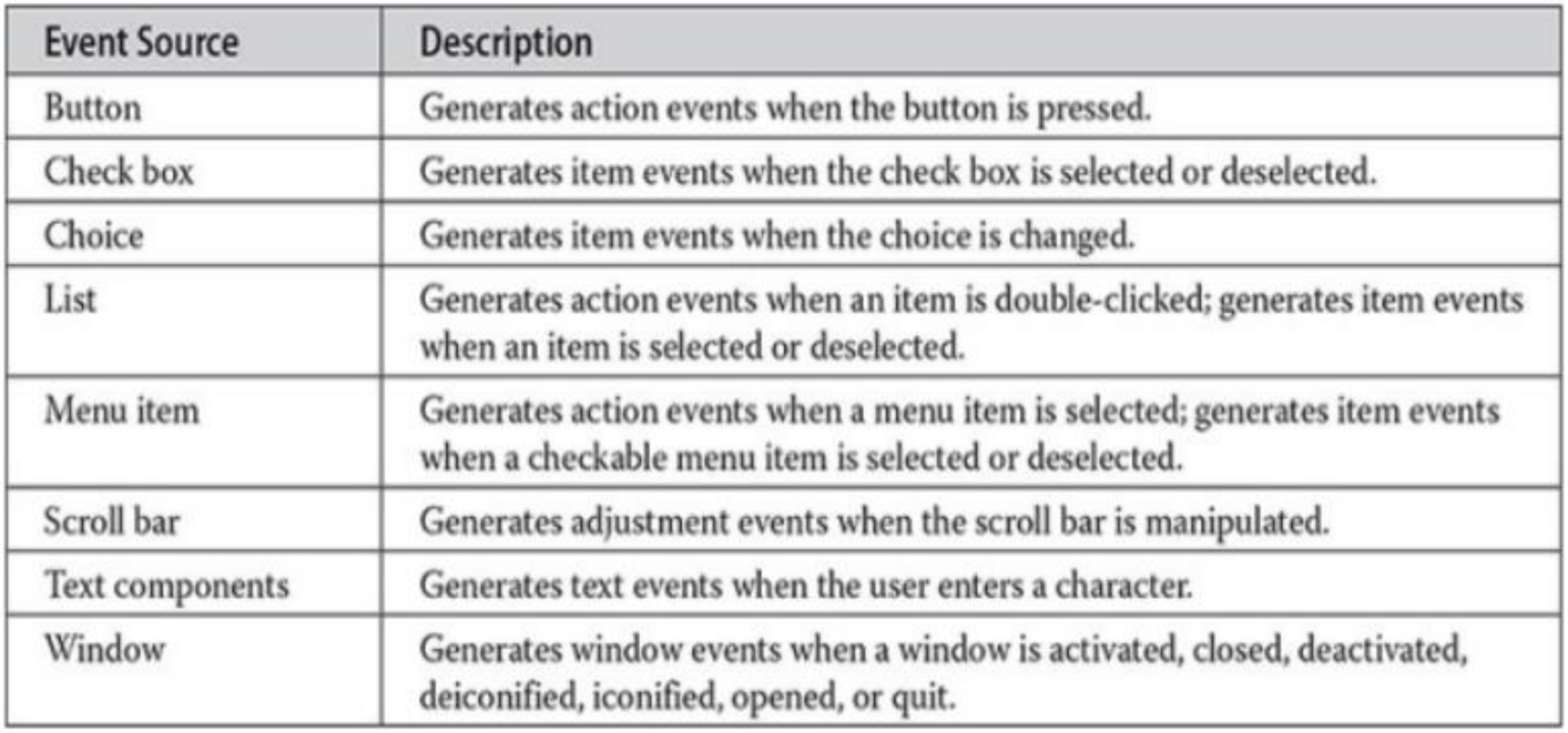
## Event Classes

**The WindowEvent Class**



| | |
|---|---|
| WINDOW_ACTIVATED | The window was activated. |
| WINDOW_CLOSED | The window has been closed. |
| WINDOW_CLOSING | The user requested that the window be closed. |
| WINDOW_DEACTIVATED | The window was deactivated. |
| WINDOW_DEICONIFIED | The window was deiconified. |
| WINDOW_GAINED_FOCUS | The window gained input focus. |
| WINDOW_ICONIFIED | The window was iconified. |
| WINDOW_LOST_FOCUS | The window lost input focus. |
| WINDOW_OPENED | The window was opened. |
| WINDOW_STATE_CHANGED | The state of the window changed. |

## Sources of Events



| Event Source | Description |
|---|---|
| Button | Generates action events when the button is pressed. |
| Check box | Generates item events when the check box is selected or deselected. |
| Choice | Generates item events when the choice is changed. |
| List | Generates action events when an item is double-clicked; generates item events when an item is selected or deselected. |
| Menu item | Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected. |
| Scroll bar | Generates adjustment events when the scroll bar is manipulated. |
| Text components | Generates text events when the user enters a character. |
| Window | Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit. |

**Table 24-2** Event Source Examples

## Event Listener Interfaces

Listeners are created by implementing one or more of the interfaces defined by the `java.awt.event` package.

When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.

| Interface | Description |
|---|---|
| ActionListener | Defines one method to receive action events. |
| AdjustmentListener | Defines one method to receive adjustment events. |
| ComponentListener | Defines four methods to recognize when a component is hidden, moved, resized, or shown. |
| ContainerListener | Defines two methods to recognize when a component is added to or removed from a container. |
| FocusListener | Defines two methods to recognize when a component gains or loses keyboard focus. |
| ItemListener | Defines one method to recognize when the state of an item changes. |
| KeyListener | Defines three methods to recognize when a key is pressed, released, or typed. |
| MouseListener | Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released. |
| MouseMotionListener | Defines two methods to recognize when the mouse is dragged or moved. |
| MouseWheelListener | Defines one method to recognize when the mouse wheel is moved. |
| TextListener | Defines one method to recognize when a text value changes. |
| WindowFocusListener | Defines two methods to recognize when a window gains or loses input focus. |
| WindowListener | Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit. |

**Table 24-3** Commonly Used Event Listener Interfaces

## Examples

MOUSE EVENTS & KEYBOARD EVENTS

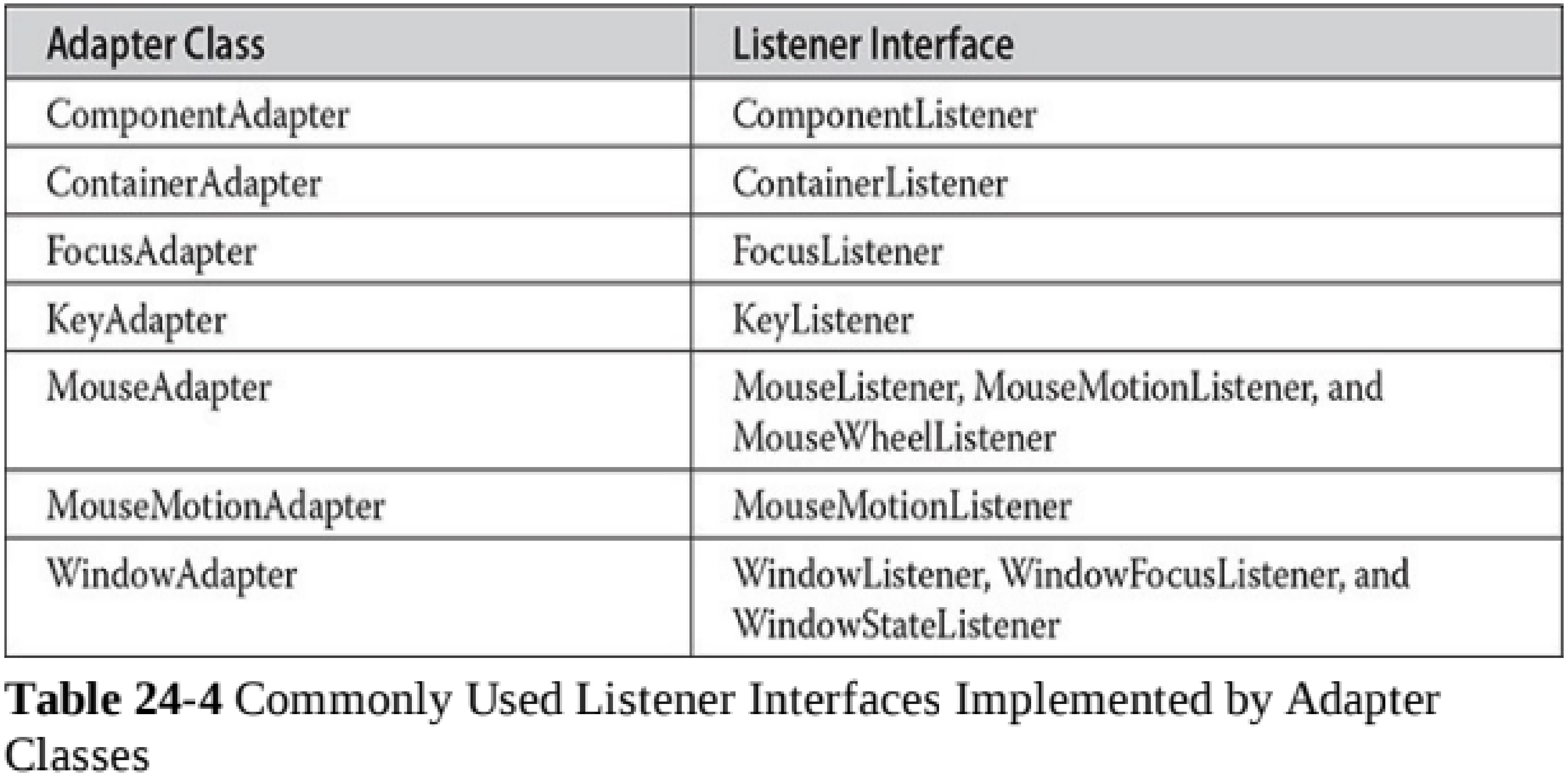
## Adapter Classes

**An adapter class provides an empty implementation of all methods in an event listener interface.**

**Why would we want Adapter Classes?**

Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.

## Adapter Classes



| Adapter Class | Listener Interface |
|---|---|
| ComponentAdapter | ComponentListener |
| ContainerAdapter | ContainerListener |
| FocusAdapter | FocusListener |
| KeyAdapter | KeyListener |
| MouseAdapter | MouseListener, MouseMotionListener, and MouseWheelListener |
| MouseMotionAdapter | MouseMotionListener |
| WindowAdapter | WindowListener, WindowFocusListener, and WindowStateListener |

**Table 24-4** Commonly Used Listener Interfaces Implemented by Adapter Classes

## Example

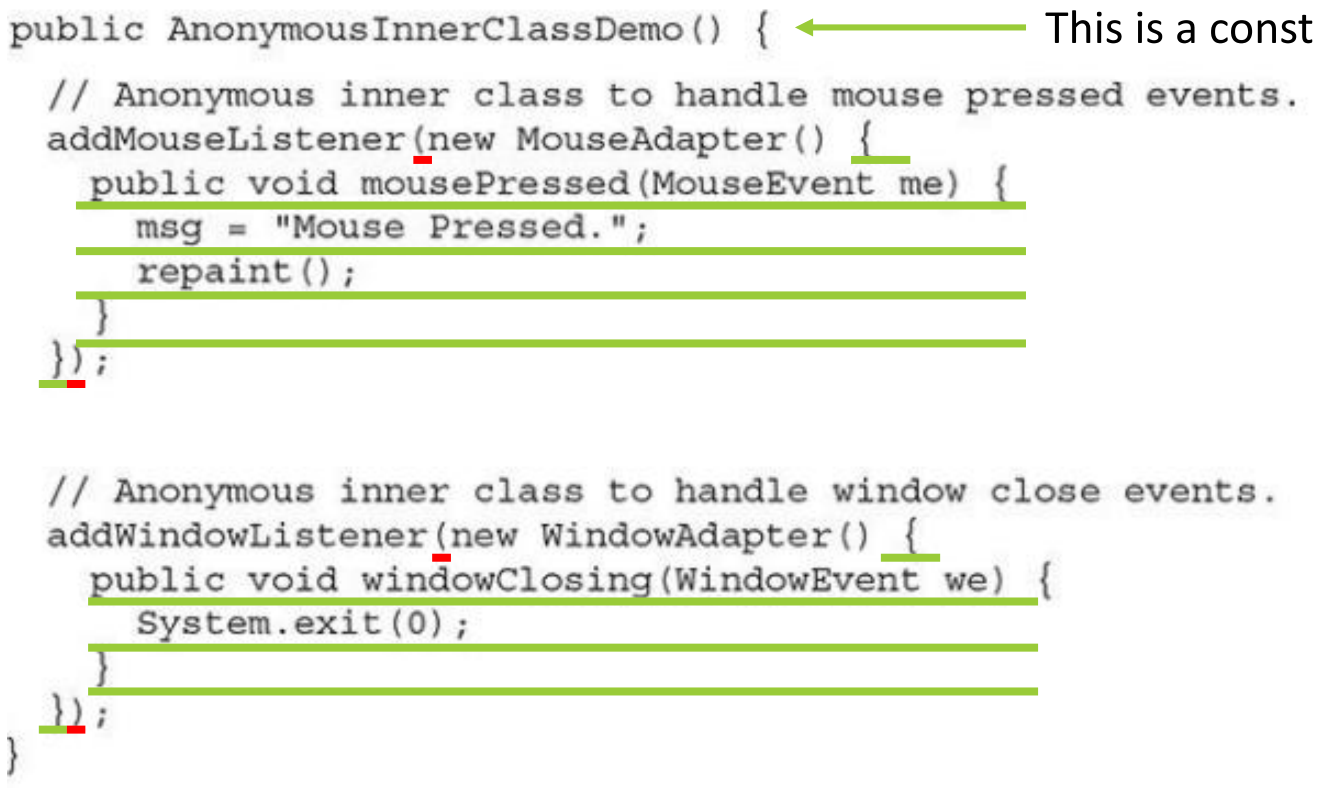
ADAPTER CLASSES

## Inner Classes

A class which is defined within another class, or within an expression.

# Anonymous Inner Classses

An anonymous inner class is an inner class that is not assigned a name



This is a constructor

```
public AnonymousInnerClassDemo() {

  // Anonymous inner class to handle mouse pressed events.
  addMouseListener(new MouseAdapter() {
    public void mousePressed(MouseEvent me) {
      msg = "Mouse Pressed.";
      repaint();
    }
  });


  // Anonymous inner class to handle window close events.
  addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent we) {
      System.exit(0);
    }
  });
}
```

Fin.