CIS2571 – Intro to Java

Chapter16 → Event-Driven Programming

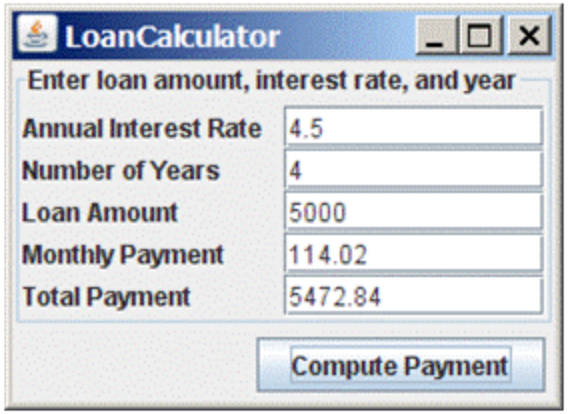
Information consolidated from a variety of textbook and online resources

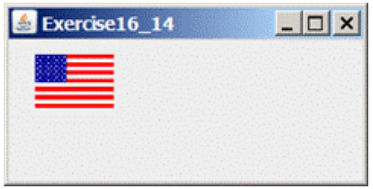
Topic Objectives

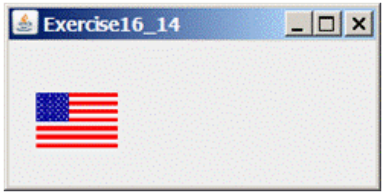
- Event-Driven Programming
- Actions, Events, and Listeners
- Delegation-Based Event Handling Model
- Internal Function of Source Component
  - Example: Control Circle
- Inner Classes
- Anonymous Class Listeners
- Alternative Ways of Defining Listener Classes
- Case Study: Loan Calculator
- Mouse Events
- Listener Interface Adapters
- Key Events and Key Listeners
- Animation Using Timer Class

Event-Driven Programming

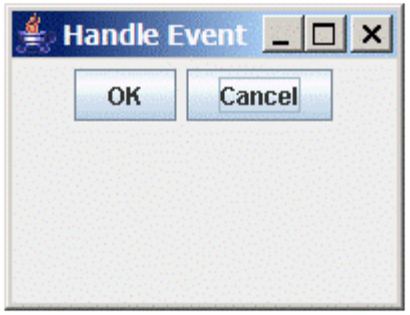
- In order for a GUI program to respond to user initiated actions (menu selection, button click, mouse movement, etc.), the GUI application must respond to **external** events
- Program code is executed when an event occurs

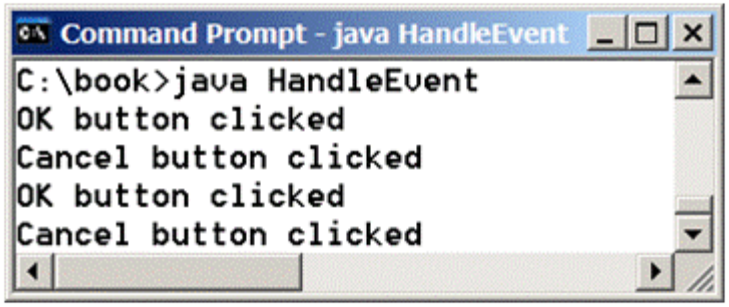




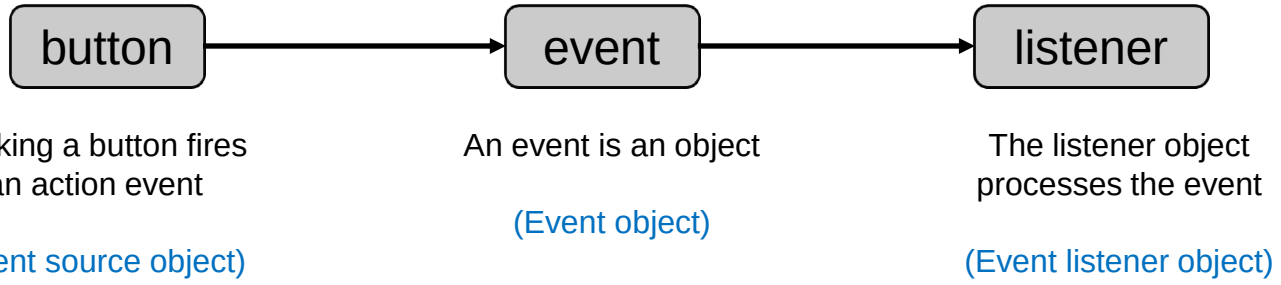


Event-Driven Programming





- Responding to button click
  - Create button object where action event originates
  - Create listener object to handle event action



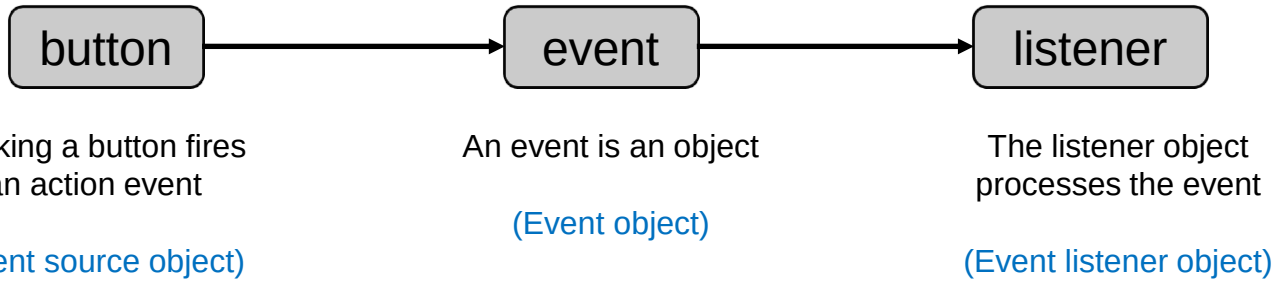
Event-Driven Programming

- The ActionListener Interface is used to respond to GUI action events
  - Class implementing interface **overrides** the following method

```
void actionPerformed(ActionEvent e);
```

  - Created object is **registered** with GUI component

```
void addActionListener(ActionListener l);
```



→Example

Event-Driven Programming

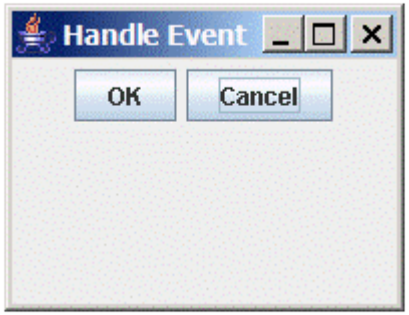
#1→ Create GUI Object

```
public class HandleEvent extends JFrame {
  public HandleEvent() {

    // Create two buttons
    JButton jbtOK = new JButton("OK");
    JButton jbtCancel = new JButton("Cancel);

    // Create panel to hold buttons
    JPanel panel = new JPanel();
    panel.add(jbtOK);
    panel.add(jbtCancel);

    add(panel); // Add panel to frame
    . . .
  }
  public static void main(String[] args) {
    . . .
  }
}
```

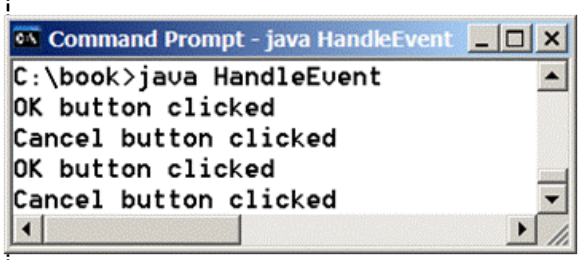


See 16.1 HandleEvent.java

Event-Driven Programming

#2→ Define Listener Class

```
public class HandleEvent extends JFrame {
  public HandleEvent() {
    . . .
  }
  public static void main(String[] args) {
    …
  }
}
class OKListenerClass implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    System.out.println("OK button clicked");
  }
}
class CancelListenerClass implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    System.out.println("Cancel button clicked");
  }
}
```



See 16.1 HandleEvent.java

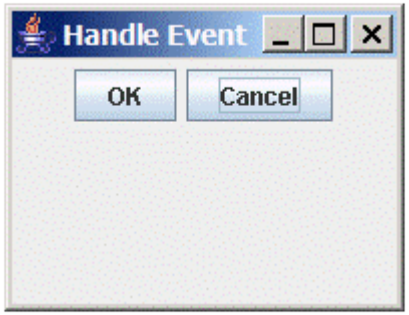
Event-Driven Programming

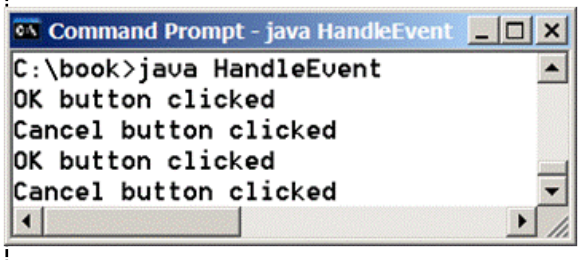
#3→ Create Listener Object and Register with GUI Object

```
public class HandleEvent extends JFrame {
  public HandleEvent() {
    . . .
    // Create listeners
    OKListenerClass listener1 = new OKListenerClass();
    CancelListenerClass listener2 = new
       CancelListenerClass();

    // Register listeners
    jbtOK.addActionListener(listener1);
    jbtCancel.addActionListener(listener2);
  }

  public static void main(String[] args) {
    …
  }
}
```

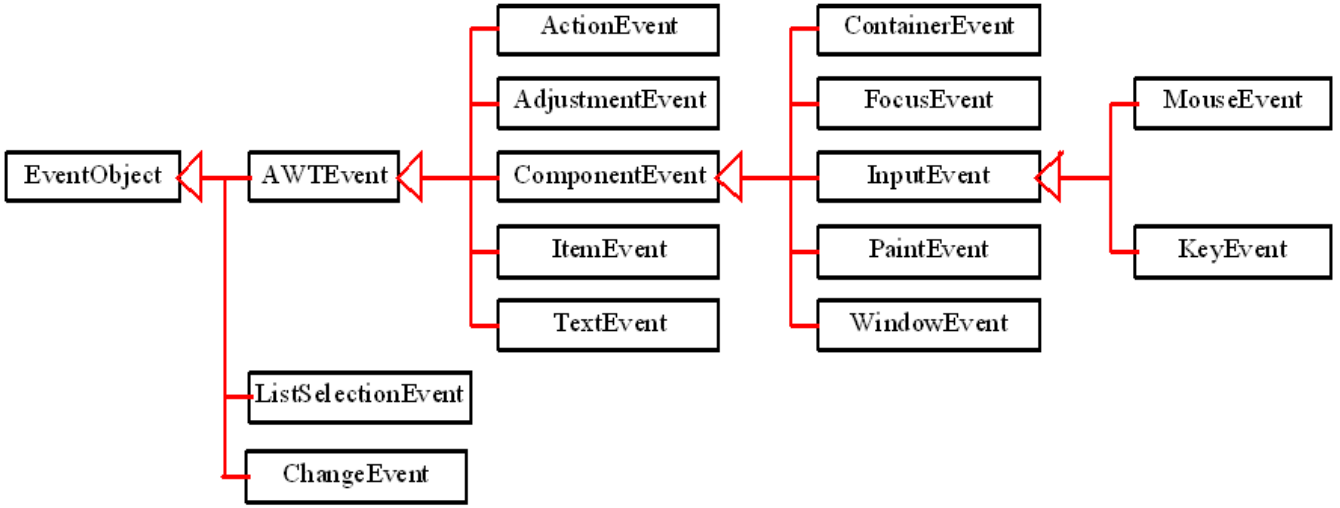




See 16.1 HandleEvent.java

Events

- Event is a signal to the program that something has happened
  - Triggered by
    - External user actions
      - Mouse movements, button clicks, keystrokes
    - Internal program activities
      - Timer
  - Program can choose to respond, or ignore, event



Actions and Events

- Event object contains properties pertinent to the event
- Can identify the source object of event through getSource() instance method of EventObject

User Action	Source Object	Event Type Fired
Click a button	JButton	ActionEvent
Press Enter in a text field	JTextField	ActionEvent
Select a new item	JComboBox	ActionEvent ItemEvent
Check or uncheck	JRadioButton	ActionEvent ItemEvent
Check or uncheck	JCheckBox	ActionEvent ItemEvent
Mouse pressed/ released/clicked/entered/ exited/moved/dragged	Component	MouseEvent
Key pressed/released/typed	Component	KeyEvent

See Table 16.1

Listeners and Event Handlers

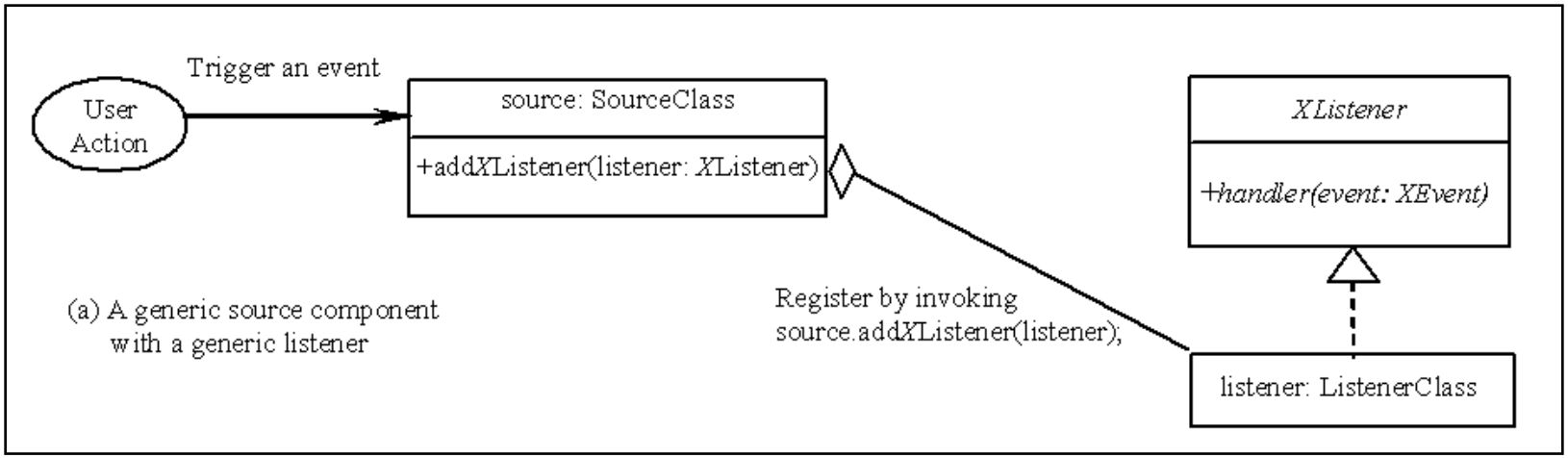
- Listener interface method must match user action and event type fired

Event Type Fired	Listener Interface	Listener Interface Methods
ActionEvent	ActionListener	actionPerformed(ActionEvent e)
ItemEvent	ItemListener	itemStateChanged(ItemEvent e)
MouseEvent	MouseListener	mousePressed(MouseEvent e)
		mouseReleased(MouseEvent e)
		mouseClicked(MouseEvent e)
		mouseEntered(MouseEvent e)
		mouseExited(MouseEvent e)
	MouseMotionListener	mouseMoved(MouseEvent e)
		mouseDragged(MouseEvent e)
KeyEvent	KeyListener	keyPressed(KeyEvent e)
		keyReleased(KeyEvent e)
		keyTyped(KeyEvent e)

See Table 16.1

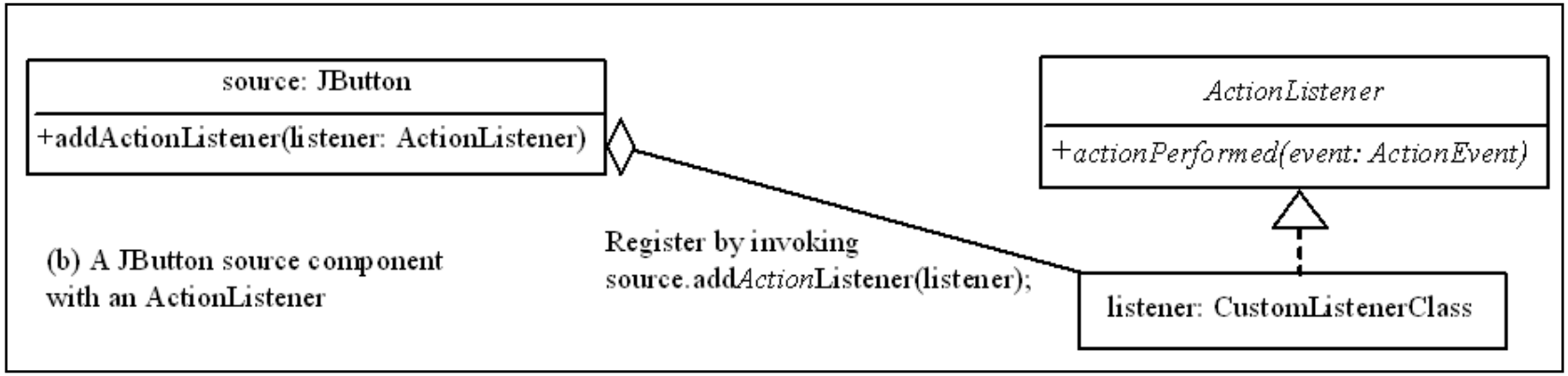
Delegation-Based Event Handling Model

- Delegation-Based Model
  - Source object fires event
  - Object interested in event handles it (*listener*)



Delegation-Based Event Handling Model

- Listener object must be
  - Instance of corresponding event-listener interface to ensure that listener has correct method for processing event
  - Registered by source object



Delegation Model Example

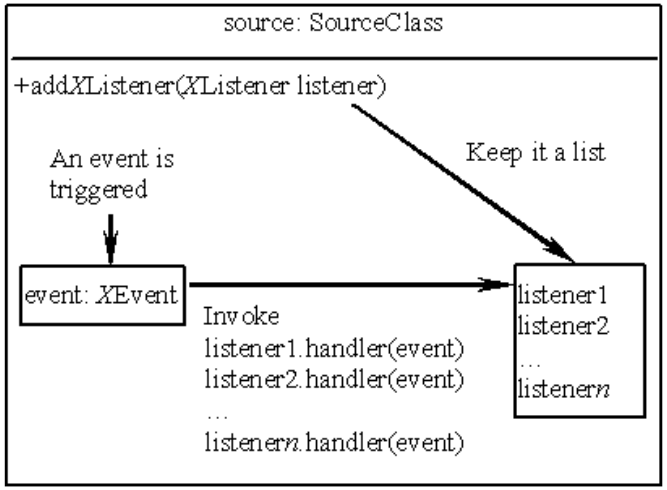
- Implementation steps:
  1. Create source object
  2. Create listener object
  3. Register listener

- Listing **16.1 HandleEvent.java** Example:

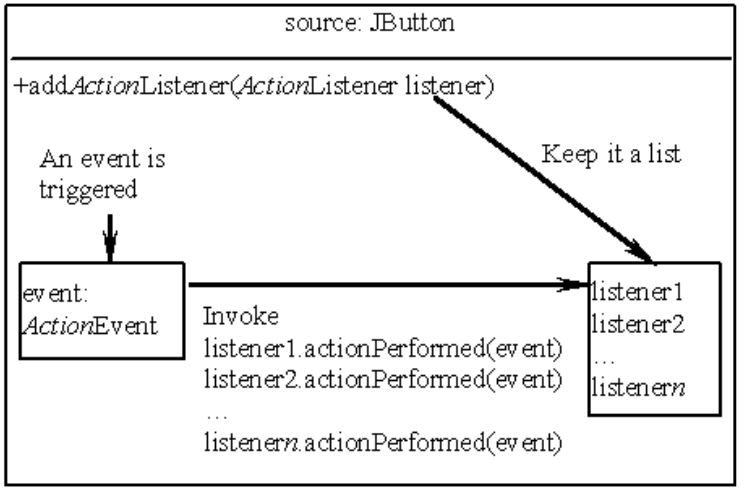
```
1. JButton jbtOK = new JButton("OK"); //line7
2. OKListenerClass listener1 = new
   OKListenerClass(); //line18
3. jbtOK.addActionListener(listener1); //line20
```

Internal Function of Source Component

- Source object may fire several types of events
  - Maintains a list of registered listeners for each event (through abstract JComponent inherited field)
  - Notifies listener by invoking handler for generated event



(a) Internal function of a generic source object

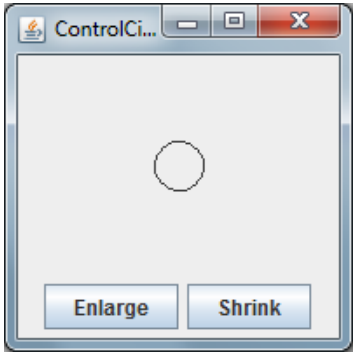


(b) Internal function of a JButton object

→Examples

Example: Control Circle

- Create two buttons to control size (*enlarge or shrink*) of circle




```
. . .
JButton jbtEnlarge = new JButton("Enlarge");
. . .
jbtEnlarge.addActionListener(new EnlargeListener());
. . .
canvas.enlarge();
. . .
radius++;
repaint();
```

See 16.3 ControlCircle.java

Inner Classes

- Java files can contain more than one class definition; however, only one class in the file can be public
  - Separate .class file created with name of other class

```
FirstClass.class
SecondClass.class
```

- Inner class, or nested class, is class defined within scope of another class
  - Typically done when inner class is only used by outer class
  - Combines dependent classes into a primary class
  - Reduces number of source (.java) files
  - Separate .class file created with combined name of outer and inner class

```
OuterClass.class
OuterClass$InnerClass.class
```

Inner classes

Multiple class definitions

```
public class Test {
  ...
}

class A {
  ...
}
```

(a)

```
public class Test {
  ...

  // Inner class
  public class A {
    ...
  }
}
```

(b)

Inner class definitions

```
// OuterClass.java: inner class demo
public class OuterClass {
  private int data;

  /** A method in the outer class */
  public void m() {
    // Do something
  }

  // An inner class
  class InnerClass {
    /** A method in the inner class */
    public void mi() {
      // Directly reference data and method
      // defined in its outer class
      data++;
      m();
    }
  }
}
```

(c)

Inner class definitions

Inner Classes

- Inner class can be used like regular class
  - Can reference data and methods in outer class
    - do not need to pass reference of outer class object
  - Can be defined with a visibility modifier subject to same rules as those applied to members of a class
    - public, protected, private
  - Can be defined static
    - Can be accessed using outer class name
    - Cannot access nonstatic members of outer class

Inner Classes

- Inner class can be used like regular class
  - Objects of inner class often created in outer class; however, you **can** create from another class
    - For **nonstatic** inner class → first create instance of outer, then create instance of inner

```
OuterClass outerObject = new OuterClass();
OuterClass.InnerClass innerObject =
  outerObject.new InnerClass();
```

    - For **static** inner class → create instance of inner

```
OuterClass.InnerClass innerObject = new
  OuterClass.InnerClass();
```

See TestInnerClass.java

Anonymous Class Listeners

- Listener class created for use by GUI component that will **not be shared** by other applications
  - Appropriate to define as **inner** class
- Anonymous inner class is **inner class without a name**
- Combines **defining** an inner class **and creating** an instance of class in one step

```
new SuperClassName/InterfaceName() {
  // Implement or override methods in superclass or interface
  // Other methods if necessary
}
```

→Examples

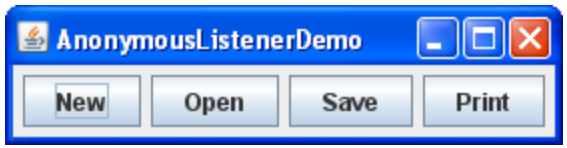
Anonymous Class Listeners

```
public ControlCircle() {
   . . .
   jbtEnlarge.addActionListener(
     new EnlargeListener());
}
class EnlargeListener
     implements ActionListener {
   @Override
   public void actionPerformed(ActionEvent e) {
     canvas.enlarge();
   }
}
```

```
public ControlCircle() {
   . . .
   jbtEnlarge.addActionListener(
     new ActionListener() {
       @Override
       public void
         actionPerformed(ActionEvent e) {
         canvas.enlarge();
       }
     });
}
```

Anonymous Class Listeners

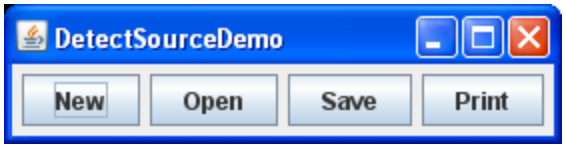
- Anonymous inner class is special kind of inner class
  - Must always extend a superclass **or** implement an interface, but it **cannot** have **explicit** extends or implements clause
  - Must implement **all** the abstract methods in the superclass or interface
  - Always uses no-arg constructor from superclass to create an instance
  - Compiled into class named **`OuterClassName$n.class`** where **`n`** is the **`n`**th occurrence of an anonymous inner class
  - Example creates anonymous class listeners for New, Open, Save, and Print buttons



See 16.4 AnonymousListenerDemo.java

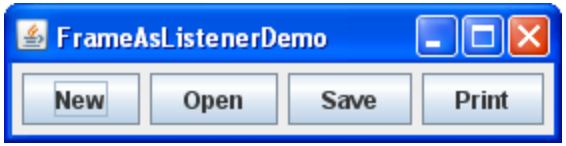
Alternative Ways of Defining Listener Classes

- Create **one** listener, register listener with all buttons and let listener detect the event **source** using getSource() method on passed ActionEvent object



See 16.4 DetectSourceDemo.java

- Custom frame class extends JFrame **and** implements ActionListener
  - Class is listener for action events
  - Register this as the listener

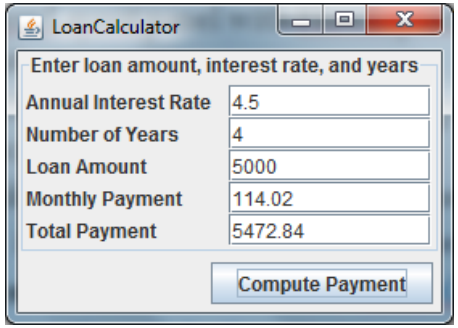


See 16.5 FrameAsListenerDemo.java

Case Study: Loan Calculator

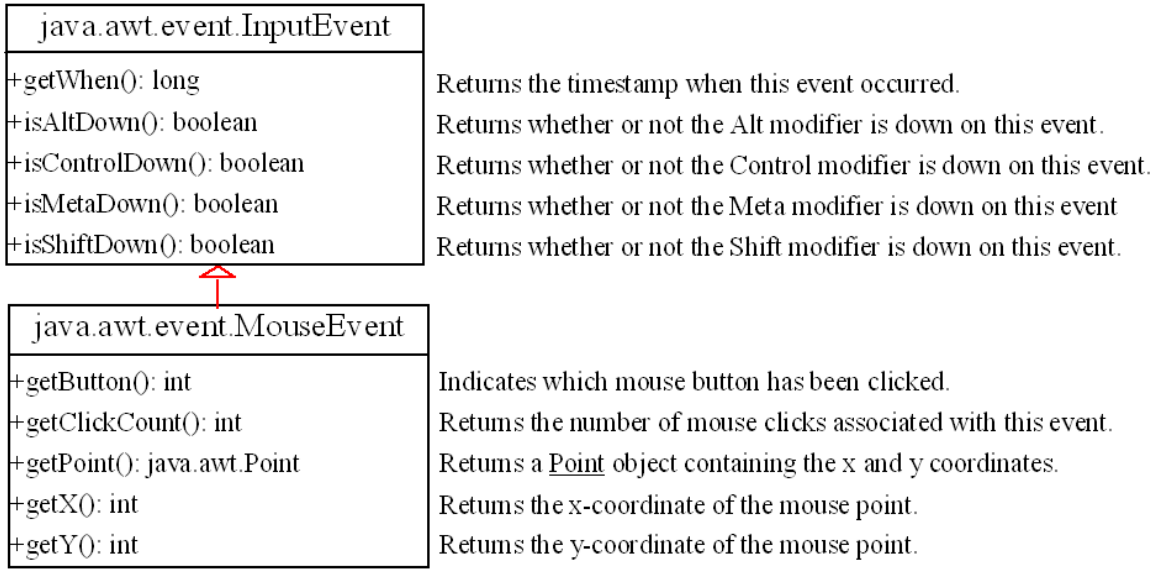
- Use **constructor** of LoanCalculator to create user interface:
  - GridLayout panel with 5 rows and 2 columns containing labels and JTextFields for input of loan data
  - FlowLayout panel with FlowLayout.RIGHT alignment containing button
  - Add both panels to BorderLayout panel with CENTER and SOUTH locations
  - Add ActionListener to process generated button event
    - Anonymous instance of separate inner class (ButtonListener)
- Use **static main** method of LoanCalculator to:
  - Create LoanCalculator instance
  - Set frame parameters and visibility

See 16.7 LoanCalculator.java
(requires 10.2 Loan.java)



Mouse Events

- Mouse event is fired whenever mouse button is
  - Pressed, Released, Clicked, Moved, or Dragged on a component
- MouseEvent object captures event
  - BUTTON1, BUTTON2, and BUTTON3 represent left, middle, and right buttons
  - Number of clicks associated with it
  - Location (x and y coordinates)



Mouse Events

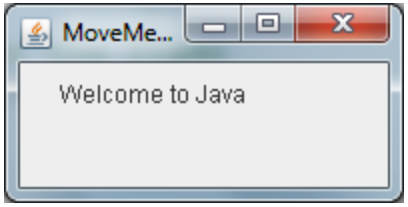
- java.awt.Point class represents a point on a component

```
Point(int x, int y)
```

- Java provides **two listener interfaces** to handle mouse events
  - MouseListener
    - listens for actions such as when the mouse is pressed, released, entered, exited, or clicked
  - MouseMotionListener
    - listens for actions such as dragging or moving the mouse

java.awt.event.MouseListener	
+mousePressed(e: MouseEvent): void	Invoked when the mouse button has been pressed on the source component.
+mouseReleased(e: MouseEvent): void	Invoked when the mouse button has been released on the source component.
+mouseClicked(e: MouseEvent): void	Invoked when the mouse button has been clicked (pressed and released) on the source component.
+mouseEntered(e: MouseEvent): void	Invoked when the mouse enters the source component.
+mouseExited(e: MouseEvent): void	Invoked when the mouse exits the source component.

java.awt.event.MouseMotionListener	
+mouseDragged(e: MouseEvent): void	Invoked when a mouse button is moved with a button pressed.
+mouseMoved(e: MouseEvent): void	Invoked when a mouse button is moved without a button pressed.



See 16.9 MoveMessageDemo.java

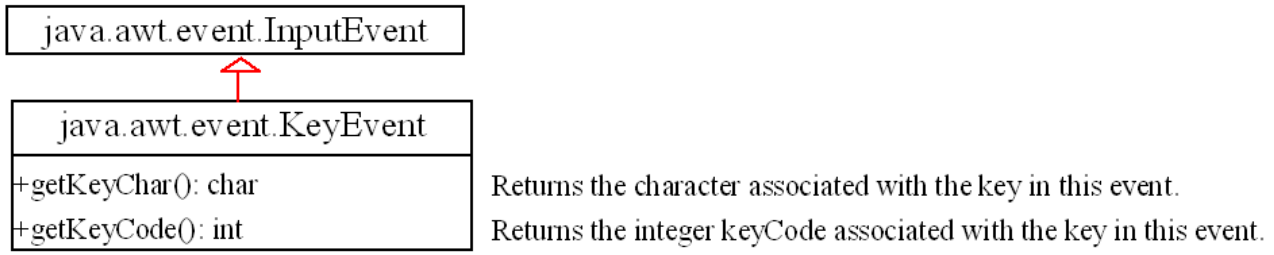
Listener Interface Adapters

- Listener interfaces are abstract
  - All methods must be implemented, even if they are not used
- Supporting adapters **classes** provide default implementations (empty body) for all the methods in listener **interface**
  - Only need to override desired method

Adapter	Interface
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
KeyAdapter	KeyListener
WindowAdapter	WindowListener

Key Events

- KeyEvent enable the use of the keys to control and perform actions or get input from the keyboard
  - Fired whenever key is pressed, released, or typed on a component
- Only **focused** components can receive KeyEvent
  - Set isFocusable property to true



Key Events

- Methods getKeyChar() or getKeyCode() returns values matching KeyEvent constants

Constant	Description
VK_HOME	The Home key
VK_END	The End key
VK_PGUP	The Page Up key
VK_PGDN	The Page Down key
VK_UP	The up-arrow key
VK_DOWN	The down-arrow key
VK_LEFT	The left-arrow key
VK_RIGHT	The right-arrow key
VK_ESCAPE	The Esc key
VK_TAB	The Tab key
VK_CONTROL	The Control key

Constant	Description
VK_SHIFT	The Shift key
VK_BACK_SPACE	The Backspace key
VK_CAPS_LOCK	The Caps Lock key
VK_NUM_LOCK	The Num Lock key
VK_ENTER	The Enter key
VK_UNDEFINED	The keyCode unknown
VK_F1 to VK_F12	The function keys from F1 to F12
VK_0 to VK_9	The number keys from 0 to 9
VK_A to VK_Z	The letter keys from A to Z

See Table 16.3

Key Listener

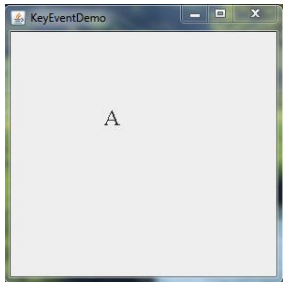
- KeyListener interface handles the following events

  `keyPressed(KeyEvent e)`
  - Called when a key is pressed

  `keyReleased(KeyEvent e)`
  - Called when a key is released.

  `keyTyped(KeyEvent e)`
  - Called when a key is pressed and then released

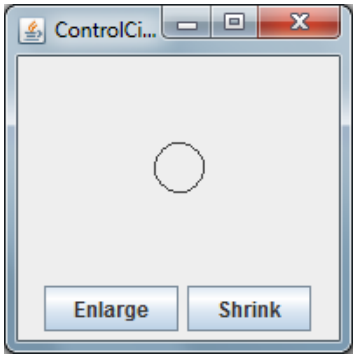
- Example: Display a user-input character. The user can also move the character up, down, left, and right using the arrow keys.



See 16.10 KeyEventDemo.java

Example Revisited: Control Circle

- Java application to control size (*enlarge or shrink*) of circle
  - Using Enlarge and Shrink buttons with ActionListeners
  - Using Left and Right Mouse buttons with MouseListeners
  - Using UP and DOWN arrow keys with KeyListeners

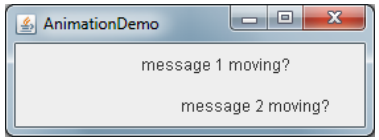


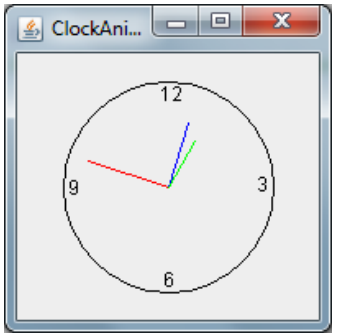
See 16.10 ControlCircleWithMouseAndKey.java

Animation Using Timer Class

- Some non-GUI components can fire events.
- The javax.swing.Timer class is a source component that fires an ActionEvent at a predefined rate
- Timer object serves as source of an ActionEvent
- Example: Timer class is used to control animations to display a moving message and repaint a clock with every elapsed second

javax.swing.Timer	
+Timer(delay: int, listener: ActionListener)	Creates a Timer with a specified delay in milliseconds and an ActionListener.
+addActionListener(listener: ActionListener): void	Adds an ActionListener to the timer.
+start(): void	Starts this timer.
+stop(): void	Stops this timer.
+setDelay(delay: int): void	Sets a new delay value for this timer.





See 16.11 AnimationDemo.java
See 16.12 ClockAnimation.java
(requires 13.10 StillClock.java)