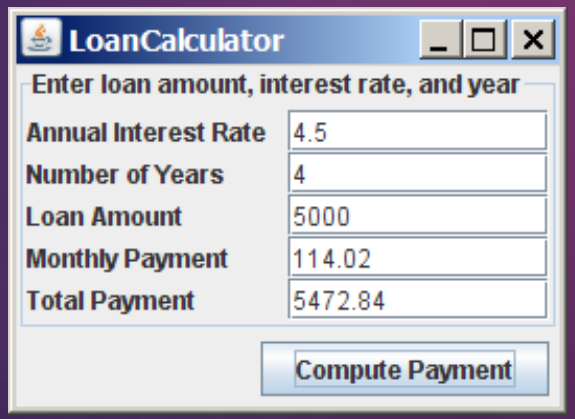
Event-Driven Programming

Motivations

Suppose you wish to write a GUI program that lets the user enter the loan amount, annual interest rate, and number of years, and click the *Compute Loan* button to obtain the monthly payment and total payment. How do you accomplish the task? You have to use event-driven programming to write the code to respond to the button-clicking event.



LoanCalculator

Run

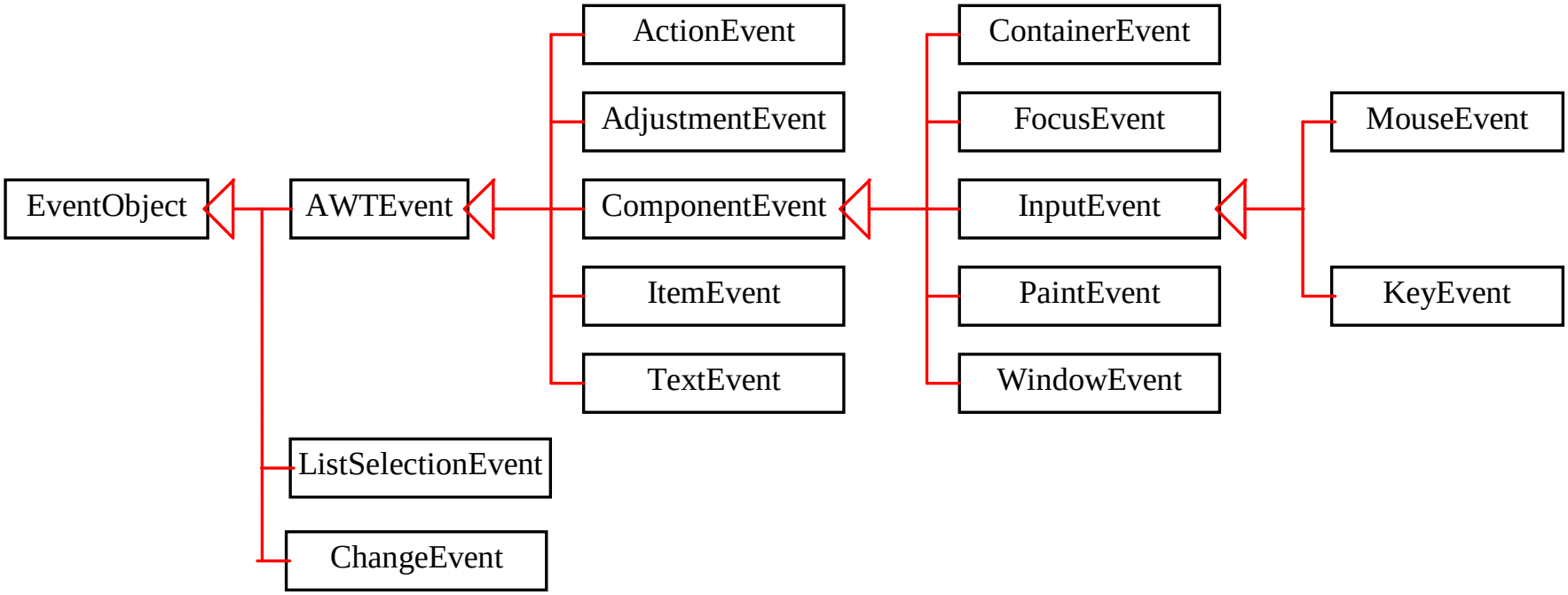
Procedural vs. Event-Driven Programming

- *Procedural programming* is executed in procedural order.
- In event-driven programming, code is executed upon activation of 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, such as a timer.

Event Classes



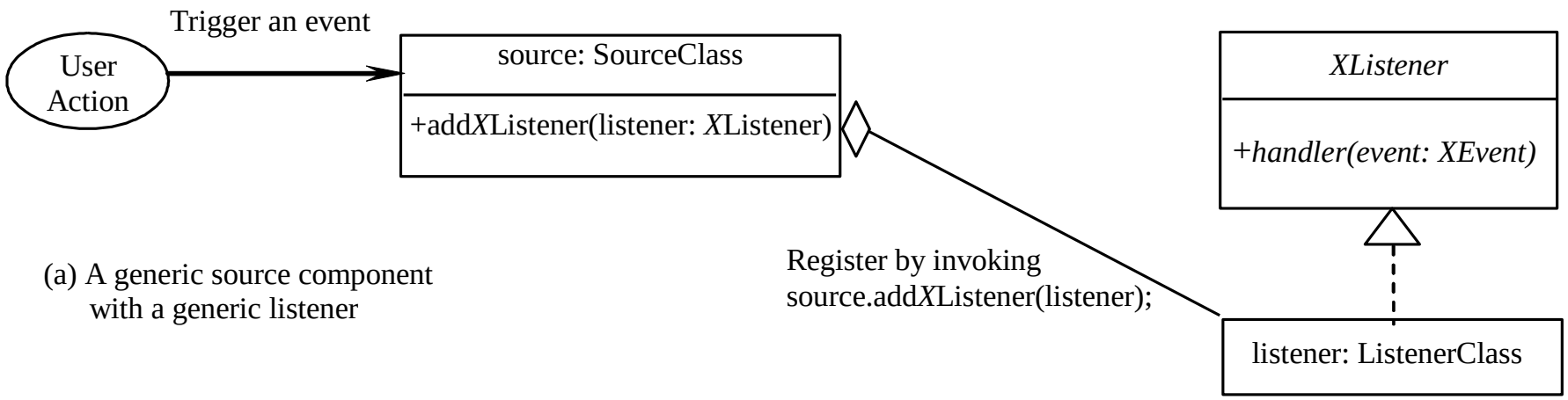
Event Information

An event object contains whatever properties are pertinent to the event. You can identify the source object of the event using the getSource() instance method in the EventObject class. The subclasses of EventObject deal with special types of events, such as button actions, window events, component events, mouse movements, and keystrokes. Table 15.1 lists external user actions, source objects, and event types generated.

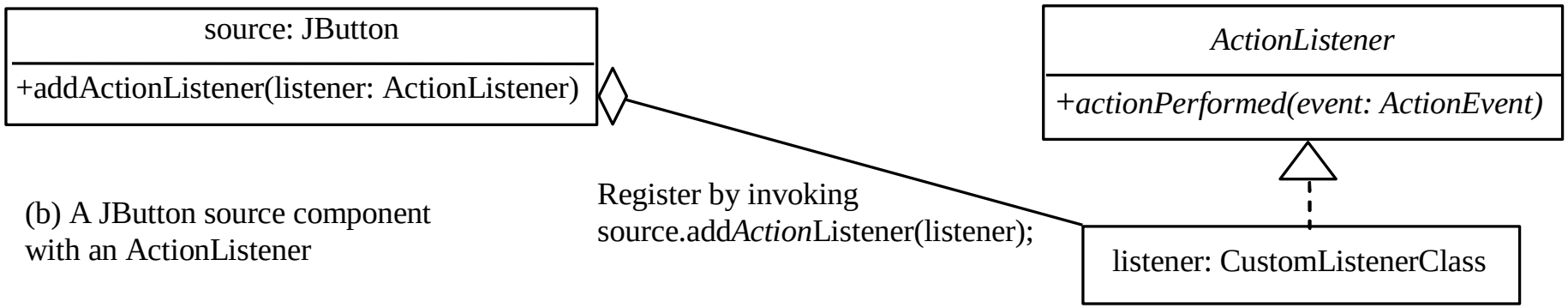
Selected User Actions

User Action	Source Object	Event Type Generated
Click a button	`JButton`	`ActionEvent`
Click a check box	`JCheckBox`	`ItemEvent`, `ActionEvent`
Click a radio button	`JRadioButton`	`ItemEvent`, `ActionEvent`
Press return on a text field	`JTextField`	`ActionEvent`
Select a new item	`JComboBox`	`ItemEvent`, `ActionEvent`
Window opened, closed, etc.	`Window`	`WindowEvent`
Mouse pressed, released, etc.	`Component`	`MouseEvent`
Key released, pressed, etc.	`Component`	`KeyEvent`

The Delegation Model



(a) A generic source component with a generic listener



(b) A JButton source component with an ActionListener

The Delegation Model: Example


```
JButton jbt = new JButton("OK");
ActionListener listener = new OKListener();
jbt.addActionListener(listener);
```

Selected Event Handlers

Event Class	Listener Interface	Listener Methods (Handlers)
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
WindowEvent	WindowListener	windowClosing(WindowEvent)
		windowOpened(WindowEvent)
		windowIconified(WindowEvent)
		windowDeiconified(WindowEvent)
		windowClosed(WindowEvent)
		windowActivated(WindowEvent)
		windowDeactivated(WindowEvent)
ContainerEvent	ContainerListener	componentAdded(ContainerEvent)
		componentRemoved(ContainerEvent)
MouseEvent	MouseListener	mousePressed(MouseEvent)
		mouseReleased(MouseEvent)
		mouseClicked(MouseEvent)
		mouseExited(MouseEvent)
		mouseEntered(MouseEvent)
KeyEvent	KeyListener	keyPressed(KeyEvent)
		keyReleased(KeyEvent)
		keyTypeed(KeyEvent)

MouseEvent

java.awt.event.InputEvent	
+getWhen(): long	Returns the timestamp when this event occurred.
+isAltDown(): boolean	Returns whether or not the Alt modifier is down on this event.
+isControlDown(): boolean	Returns whether or not the Control modifier is down on this event.
+isMetaDown(): boolean	Returns whether or not the Meta modifier is down on this event
+isShiftDown(): boolean	Returns whether or not the Shift modifier is down on this event.

java.awt.event.MouseEvent	
+getButton(): int	Indicates which mouse button has been clicked.
+getClickCount(): int	Returns the number of mouse clicks associated with this event.
+getPoint(): java.awt.Point	Returns a Point object containing the x and y coordinates.
+getX(): int	Returns the x-coordinate of the mouse point.
+getY(): int	Returns the y-coordinate of the mouse point.

Java Event Handling Example

- Courtesy: https://www.javatpoint.com/event-handling-in-java

- Three ways to implement
  - Within class
  - Using another class
  - Using anonymous class

Within Class


```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){

//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);

//register listener
b.addActionListener(this);//passing current instance
```

```
//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}
}
```

Using outer class


```
import java.awt.*;
import java.awt.event.*;
class AEvent2 extends Frame{
TextField tf;
AEvent2(){
//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
//register listener
Outer o=new Outer(this);
```

```
b.addActionListener(o);//passing outer class instance
//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent2();
}
}
```

Using outer class (contd.)


```
import java.awt.event.*;
class Outer implements ActionListener{
AEvent2 obj;
Outer(AEvent2 obj){
this.obj=obj;
}
public void actionPerformed(ActionEvent e){
obj.tf.setText("welcome");
}
}
```

Using anonymous class


```
import java.awt.*;
import java.awt.event.*;
class AEvent3 extends Frame{
TextField tf;
AEvent3(){
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(50,120,80,30);
```

```
b.addActionListener(new ActionListener(){
public void actionPerformed(){
tf.setText("hello");
}
});
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent3();
}
}
```

Handling Mouse Events

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

Handling Mouse Events

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.

Handling Keyboard Events

To process a keyboard event, use the following handlers in the `KeyListener` interface:

- `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.

The `KeyEvent` Class

- Methods:

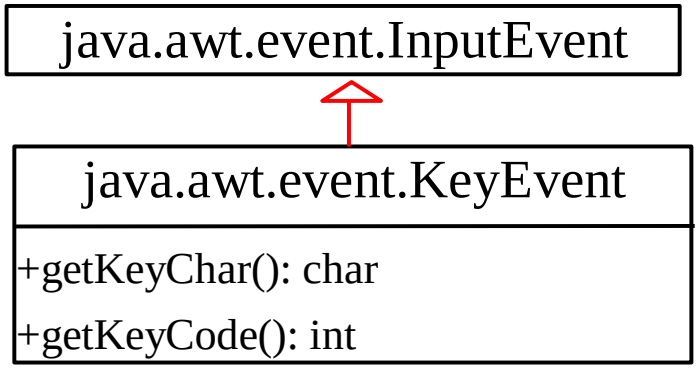
  `getKeyChar()` method

  `getKeyCode()` method

- Keys:

Home	`VK_HOME`
End	`VK_END`
Page Up	`VK_PGUP`
Page Down	`VK_PGDN`
etc...	

The `KeyEvent` Class, cont.



Returns the character associated with the key in this event.

Returns the integer keyCode associated with the key in this event.