Event Driven Programming

1. Objectives 2
2. Definitions 2
3. Event-Driven Style of Programming 2
4. Event Polling Model 3
5. Java's Event Delegation Model 5
6. How to Implement an Event Handler? 7

1. Objectives

- The user is in charge instead of the program: He or she can perform tasks in any order.

2. Definitions

- Events can be triggered by human interaction or alternatively a software or hardware process causing an event to be triggered.

- An event is a hardware condition that is signaled
- Common events are from input devices:
    - mouse events - mouse up, mouse down, mouse move
    - keyboard events - key up, key down
    - network events - packet arrival
    - window events - window resized, window dragged, focus lost
- Special hardware detects and signals these events. Each event has additional information. For example a mouse movement event has an x and y coordinate.

3. Event-Driven Style of Programming

- In event-driven programming, there is essentially no normal flow-of-control! Only the event handlers exist.

- Flow-of-control in the program is entirely driven by responding to events.
- Program is driven by responding to events
- Two main models:
    - Polling:
        - Has the event happened yet?
        - Are we there yet? Are we there yet? Etc.
    - Delegating:
        - Here is what I want you to do when this event occurs.

4. Event Polling Model

- How does it work?
    - Loop: Polling tends to push all event-handling code into one location (inside the big loop
    - The resulting interactions within the big loop tend to be complex.
    - In addition, polling requires a program to sit in a loop, consuming CPU cycles, while waiting for the user to do something -- a serious waste of a valuable resource.

- The Event Loop pseudo-code:

```
loop
    wait for an event to arrive
    get the event details
```

do something based on the event
until a stop event occurs

- Examples: Windows and Palm OS

- **The Basic Palm Event Loop**

  The basic Palm event loop looks like this:

```
EventType event;
do {
   EvtGetEvent( &event, evtWaitForever );
   // dispatch the event here
} while( event.eType != appStopEvent ); // until told to stop
```

Where EvtGetEvent is a Palm OS API:

- ✓ It returns the next event for the application by copying it into the **EventType** structure passed in as the first parameter.

- ✓ The second parameter is a timeout value, which tells the operating system how long the application wants to wait before an event occurs:

  evtWaitForever constant: Wait indefinitely
  Number of ticks: Pass in the number of **ticks** (hundredths of a second) to wait.

5. Java's Event Delegation Model

- Supersedes Java's original containment and inheritance based event handling.
- This model is more efficient since not all events are handled by your program.
- Three kinds of things (with various corresponding Java classes):

1. Events:
    - Representation of things that happen, derived from the superclass EventObject.
    - They occur in response to an action by the user within the Graphical User Interface (GUI).
    - When an event occurs, the source of the event notifies listeners.
2. Event sources:
    - Objects that can generate events
    - It only notifies listeners that have been registered to receive the event

    - Examples:

Event Source	How triggered	Event Generated
Button	Button pressed	Action events
Checkbox	Checkbox selected or deselected	Item events

- **Multicasting of the event**: When an event occurs, the source notifies all the listeners that have registered for that event. The notification includes the details about the event, for example, the name of the source that generated the event, the key that was pressed or the location where the mouse was clicked.

- Example of events:

Event Classes	Description
ActionEvent	Button pressed, list item selected, or item menu selected
TextEvent	Value or text field or text area changed.
KeyEvent	Key is pressed or released or typed
MouseEvent	Mouse clicked, pressed, released, dragged, moved, enters, or exits a compment.
FocusEvent	Component gains or loses focus(keyboard).

3. Event listeners (handlers):
   - Are responsible to recognize and process events. Objects that want to respond to events.
   - A listener is an object that is notified by the source, if registered, when an event occurs.
   - Register their interest in certain events with event sources.
   - Listeners usually sit back and waits for an event to happen.

6. How to Implement an Event Handler?

Every event handler requires three bits of code:

1. In the declaration for the event handler class, code that specifies that the class either implements a listener interface or extends a class that implements a listener interface. For example:

```
public class MyButtonHandler implements ActionListener {
        // class members ...
        // constructor …
          ……




        }
```

2. Code that registers an instance of the event handler class as a listener upon one or more components. For example:

```
public class MyButtonHandler implements ActionListener {
        // class members ...
        // constructor …
          …

        //someComponent.addActionListener(instanceOfMyClass);
        button.addActionListener(this);
          …

        }
```

3. Code that implements the methods in the listener interface. For example:

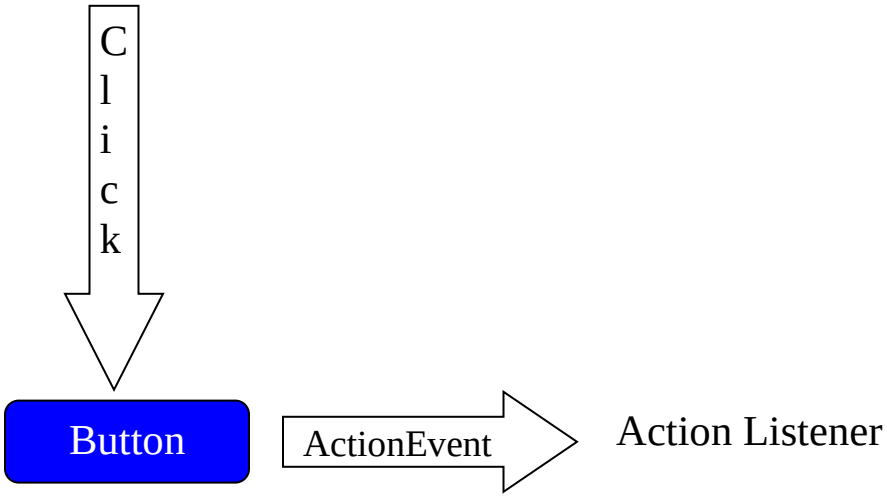
```
public class MyButtonHandler implements ActionListener {
        // class members ...
        // constructor …
        …

        //someComponent.addActionListener(instanceOfMyClass);
        button.addActionListener(this);

        …

        public void actionPerformed(ActionEvent e) {
            // handle the event
            // code that reacts to the action...
        }

}
```



When the user clicks a button, the button's action listeners are notified.

↯ *Some Event Classes*

- **KeyEvent**: for keyboard input
- **MouseEvent**: for all sorts of mouse events: press, release, move, drag,...
- **ActionEvent**: for GUI actions like clicking on a button, selecting a menu item,...

...

↯ *Some Event Listener Interfaces*

- **MouseListener**:
  - **Methods**:

    void mouseClicked(MouseEvent me)
    void mouseEntered(MouseEvent me)
    void mouseExited(MouseEvent me)
    void mousePressed(MouseEvent me)
    void mouseReleased(MouseEvent me)

  - **Registered** with some event source by calling:

    void addMouseListener(MouseListener ml)

- **MouseMotionListener**
  - **Methods**:

    void mouseDragged(MouseEvent me)
    void mouseMoved(MouseEvent me)

  - **Registered** with some event source by calling:

void addMouseMotionListener(MouseMotionListener mml)

- **KeyListener**

  - **Methods**:

    void keyPressed(KeyEvent *ke*)
    void keyReleased(KeyEvent *ke*)
    void keyTyped(KeyEvent *ke*)

  - **Registered** with some event source by calling

    void addKeyListener(KeyListener *kl*)

- **ActionListener**

  - **Methods**:

    void actionPerformed(ActionEvent ae)

  - Registered with some event source by calling:

    void addActionListener(ActionListener al)