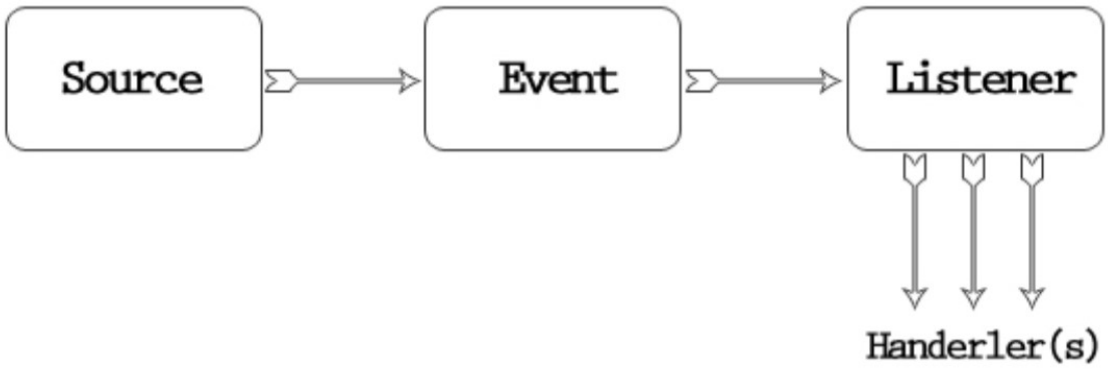
ITIS-LS "Francesco Giordani" Caserta

prof. Ennio Ranucci

a.s. 2020-2021

La programmazione orientata agli eventi



Event-Driving Programming Model

La programmazione basata su eventi è un paradigma di programmazione in cui il flusso del programma è determinato da eventi generati dall'utente (clic del mouse, pressioni dei tasti, output dei sensori o passaggio di messaggi da altri programmi o thread.

Invece di adottare lo stile: input/elaborazione/output il sistema esegue all'infinito un loop all'interno del quale vi sono istruzioni che verificano continuamente la comparsa di eventi ed eventualmente lanciare l'esecuzione del sottoprogramma che il programmatore ha scritto e collegato all'evento che è stato generato. Programmare "a eventi" vuol dire, quindi, definire le azioni con cui il sistema risponde al verificarsi di un certo evento.

Gli eventi a cui il programma deve reagire possono essere rilevati mediante polling eseguito all'interno di un loop di programma (*interrogazione ciclica*), oppure in risposta ad un Interrupt. In molte applicazioni si usa una combinazione di entrambe queste due tecniche.

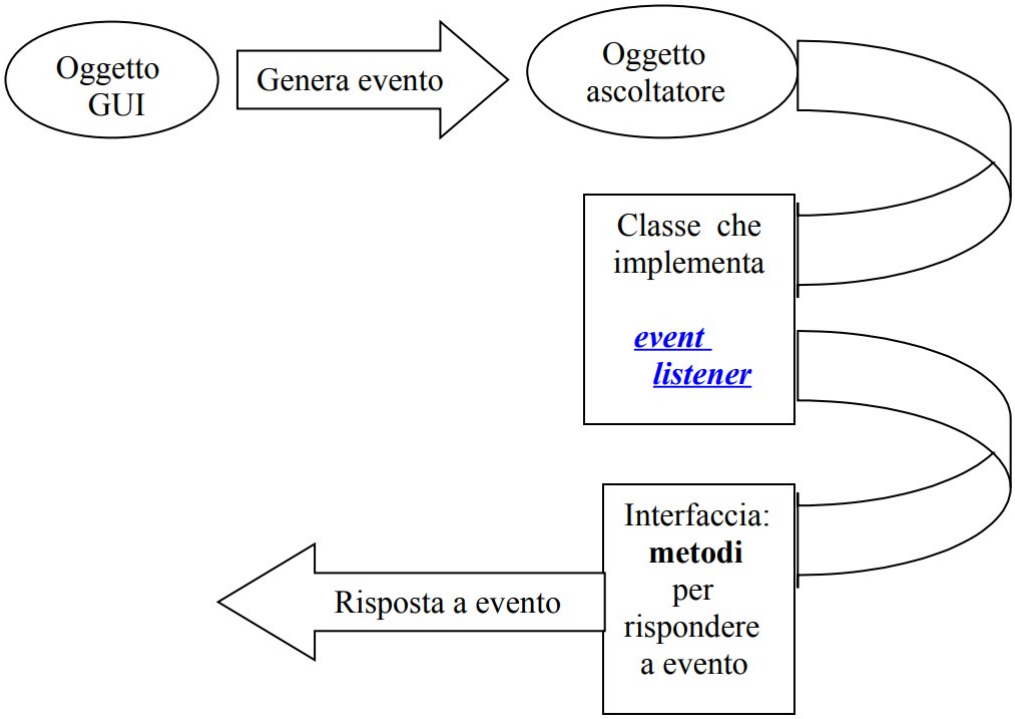
Il **polling** presenta lo svantaggio di sprecare preziose risorse di calcolo perché costretta a fare comunque un controllo su ciascuna sorgente di eventi. Le *interruzioni* sono di due tipi*:*
Interrupt hardware generati da dispositivi esterni alla CPU (periferiche), che hanno il compito di comunicare il verificarsi di eventi esterni, di solito dispositivi di Input/Output. Un interrupt hardware costringe il processore a memorizzare il suo stato di esecuzione fino all'arrivo dell'interrupt e ad iniziare l'esecuzione della subroutine (sottoprogramma) che esegue il compito richiesto dall'interrupt, terminato il quale il processore riprende l'esecuzione delle operazioni che stava precedentemente elaborando.
Interrupt software sono delle istruzioni assembly tipo *INT xx* o *SYSCALL*, che possono essere assimilate alle chiamate di sottoprogrammi, ma che sfruttano il meccanismo delle interruzioni per passare il controllo dal programma chiamante a quello chiamato, e viceversa; vengono utilizzati per accedere direttamente alle risorse del sistema operativo.

I programmi che utilizzano la programmazione a eventi (denominati spesso *"programmi event-driven"*) sono composti da sottoprogrammi chiamati gestori degli eventi (*event handlers*), che sono eseguiti in risposta agli eventi esterni, e da un dispatcher, che effettua materialmente la chiamata, spesso utilizzando una coda degli eventi che contiene l'elenco degli eventi verificatisi, ma non ancora "processati".

La **programmazione basata su eventi** è il paradigma dominante utilizzato nelle interfacce utente grafiche che sono centrate sull'esecuzione di determinate azioni in risposta all'input dell'utente.

Un **evento** è una condizione (hardware) segnalata. Un evento puo' essere generato dalla pressione di un tasto, dal click del mouse, ecc...

EVENTI JAVA



I passi per gestire gli eventi programmando in Java sono i seguenti:

1. Decidere quali eventi interessano l'applicazione. Tale fase si traduce nella scelta dell'interfaccia di ascolto da implementare;

2. Decidere qual è l'origine dell'evento: il componente o contenitore che può generare (notificare) l'evento e creare tale oggetto GUI

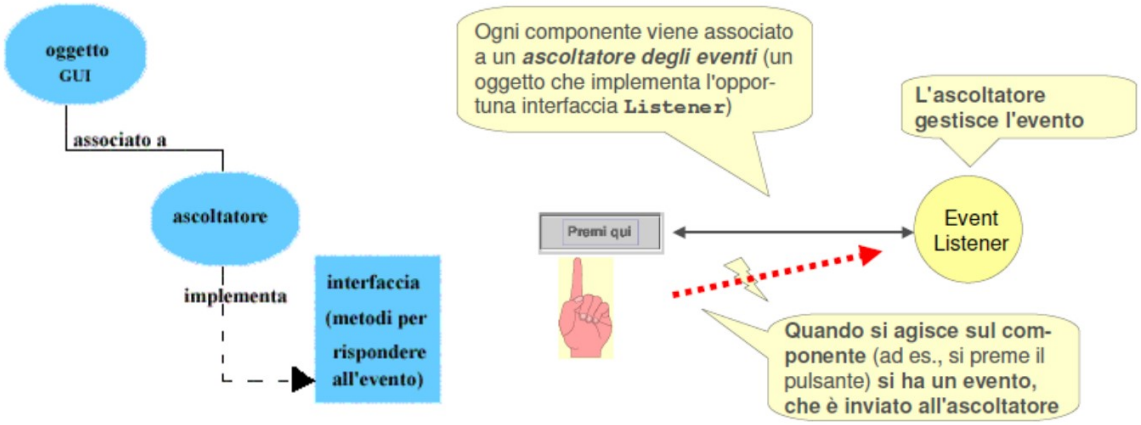
3. Creare una oggetto della classe di ascolto:

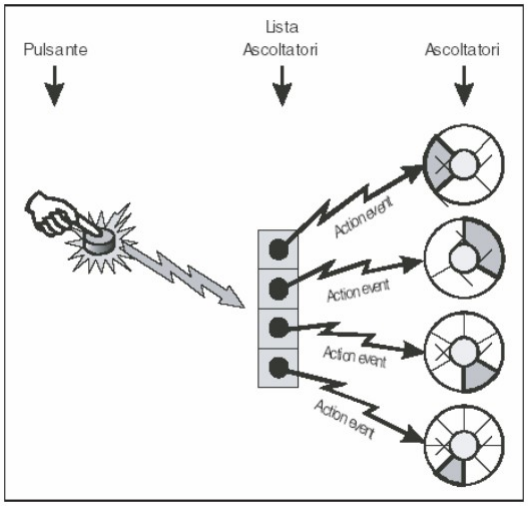
   - L'oggetto ascoltatore può essere ad esempio tutta l'applicazione (o l'applet) o una sua parte che implementa (implements) la classe predefinita d'ascolto di quell'evento che ne definisce l'interfaccia (metodi di risposta a quell'evento).

   - L'oggetto ascoltatore può essere creato come istanza di una classe di ascolto definita dall'utente che implementa (implements) la classe predefinita d'ascolto di quell'evento che ne definisce l'interfaccia (metodi di risposta a quell'evento)

4. Collegare all'origine dell'evento l'oggetto della classe d'ascolto di quell'evento cioè registrare l'ascoltatore

5. Implementare almeno un metodo dell'interfaccia per rispondere a quell'evento creando il codice di gestione che serve. Se sono definiti più metodi di risposta nell'interfaccia, è necessario scrivere le segnature dei metodi non usati oppure, in alternativa, creare l'oggetto ascoltatore come istanza di una classe che eredita (extends) dalla classe Adapter relativa a quell'evento





La gestione degli eventi grafici in Java segue il paradigma event delegation (conosciuto anche come event forwarding). Ogni oggetto grafico `e predisposto ad essere sollecitato in qualche modo dall'utente e ad ogni sollecitazione genera eventi che vengono inoltrati ad appositi ascoltatori, che reagiscono agli eventi secondo i desideri del programmatore. L'event delegation presenta il vantaggio di separare la sorgente degli eventi dal comportamento a essi associato: un componente non sa cosa avverrà al momento della sua sollecitazione: esso si limita a notificare ai propri ascoltatori che l'evento che essi attendevano è avvenuto, e questi provvederanno a produrre l'effetto desiderato.
Ogni componente può avere più ascoltatori per un determinato evento o per eventi differenti, come anche è possibile installare uno stesso ascoltatore su più componenti anche diversi a patto che entrambi possono essere sollecitati per generare l'evento. L'operazione di installare lo stesso ascoltatore su più controlli (che quindi si comporteranno nello stesso modo se sollecitati dallo stesso evento) è frequente: si pensi alle voci di un menu che vengono replicate su una toolbar per un più facile accesso ad alcune funzionalità frequenti.

La classe che desidera catturare gli eventi del mouse deve implementare l'interfaccia MouseListener definita nel package java.awt.event che è così definita:

```
public interface MouseListener{
  void mouseClicked(MouseEvent e);
  void mouseEntered(MouseEvent e);
  void mouseExited (MouseEvent e);
  void mousePressed(MouseEvent e);
  void mouseReleased(MouseEvent e);
}
```

dove MouseEvent `e la corrispondente classe evento il cui scopo è dare informazioni aggiuntive sull'evento del mouse.
Infatti, questa definisce due metodi che permettono di conoscere le coordinate del mouse allo scatenarsi dell'evento (i primi due) e un terzo metodo che permette di determinare quale bottone del mouse è stato premuto:

```
 int getX();
 int getY();
 int getModifiers()
```

Gli ascoltatori più importanti:
ActionListener Definisce 1 metodo per ricevere eventi-azione
ComponentListener Definisce 4 metodi per riconoscere quando un componente viene nascosto, spostato, mostrato o ridimensionato
FocusListener Definisce 2 metodi per riconoscere quando un componente ottiene o perde il focus
KeyListener Definisce 3 metodi per riconoscere quando viene premuto, rilasciato o battuto un tasto
MouseMotionListener Definisce 2 metodi per riconoscere quando il mouse è trascinato o spostato
MouseListener Definisce 5 metodi sopradescritti
TextListener Definisce 1 metodo per riconoscere quando cambia il valore di un campo testo
WindowListener Definisce 7 metodi per riconoscere quando una finestra viene attivata, chiusa, disattivata, ripristinata, ridotta a icona, ecc.

L'oggetto ActionEvent da informazioni aggiuntive sull'evento generato. Ad esempio, definisce il metodo Object getSource() che restituisce l'oggetto che ha generato l'evento. La maggior parte dei widget java sono predisposti per generare eventi azione. Ad esempio quando si clicca su un bottone, si preme INVIO in un campo di testo, si seleziona una voce di un menu oppure si seleziona/deseleziona una voce di una checkBox o un radioButton, viene generato un evento azione.

```
import java.awt.*;
import java.awt.event.*;
public class PulsanteConEvento

{

  Label lblSaluto;

  public PulsanteConEvento()

  {

    Frame frmF = new Frame("Pulsante con evento");

    lblSaluto = new Label("                                    ");

    Button btnPulsante = new Button("Saluta");

    frmF.setLayout(new FlowLayout()); //mette in colonna le componenti

    btnPulsante.addActionListener(new ascoltaPulsante());

    frmF.add(btnPulsante);

    frmF.add(lblSaluto);

    frmF.pack();

//gestore layout  a scorrimento: i componenti sono inseriti da sinistra verso destra con la loro
// la dimensione minima necessaria a disegnarlo interamente.

    frmF.setSize(600,500);
    frmF.setLocation(200,200);
    frmF.setVisible(true);

   }

   private class ascoltaPulsante implements ActionListener
//ascolta Pulsante è la classe ascoltatore che implementa ActionListener che è diversa da MouseListener

   {

     public void actionPerformed(ActionEvent e)

      {

        lblSaluto.setText("Buongiorno ragazzi");

      }

   }

} // fine classe PulsanteConEvento
```

Ogni programma deve creare un oggetto ascoltatore per ognuna delle classi cui è interessato. La classe di ascolto deve implementare almeno una interfaccia di ascolto (interfacce di tipo listener). Un ascoltatore di eventi è un oggetto di una classe di ascolto che implementa almeno una interfaccia di ascolto. Tali interfacce1 sono segmenti software predefiniti per gestire l'evento. Per creare un ascoltatore bisogna conoscere l'origine dell'evento che può essere ad esempio l'intera Applicazione o una sua parte. Per origine dell'evento si intende il contenitore o il componente che può generare l'evento (ad esempio un pulsante è l'origine dell'evento di azione tipo "clic su pulsante").