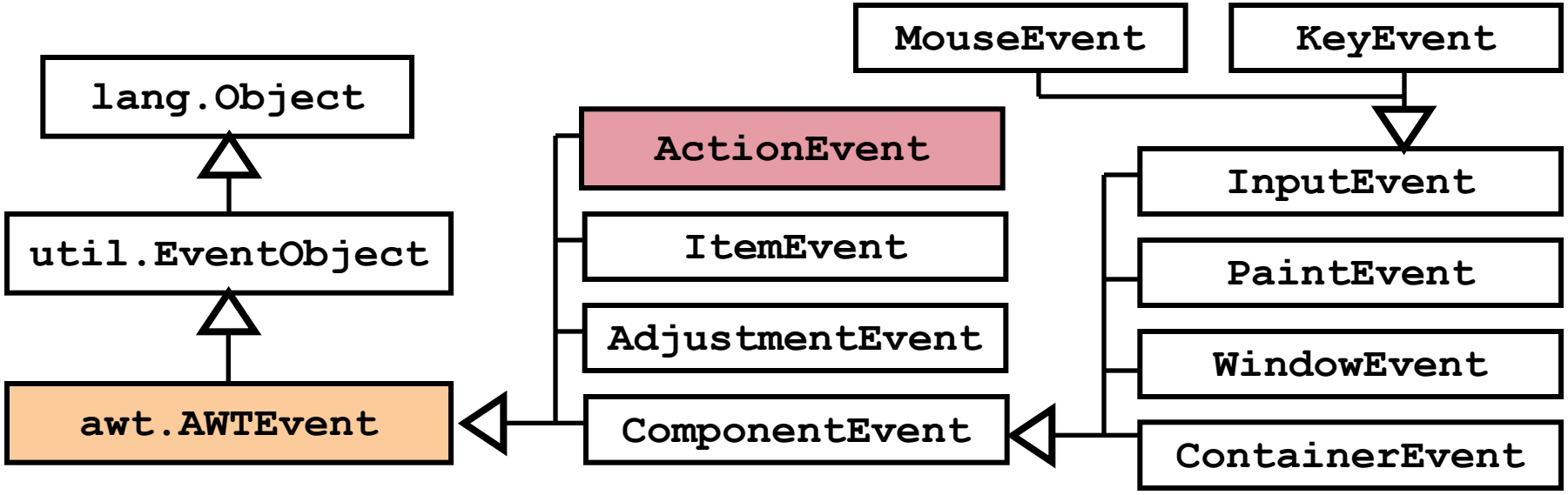
Interazioni col mondo: eventi

- Le interazioni di una GUI con il mondo esterno sono segnalate tramite **eventi**: click del mouse, inserimento di testo, selezione/attivazione...
  - Gli eventi che memorizzano queste informazioni sono oggetti e discendono dalla classe **`AWTEvent`**



Interazioni col mondo: eventi

- Gli oggetti rilevanti per un evento sono:
  - L'evento stesso (che è un oggetto; per esempio potrà essere un'istanza di **`ActionEvent`**)
  - La sorgente dell'evento (un componente GUI)
  - Il destinatario dell'evento: un oggetto *listener* (= ascoltatore, o percettore) dell'evento
- Un ascoltatore attende il verificarsi di certe tipologie di eventi (anche più d'uno) e prevede un metodo da eseguire se capitano
  - Va: definito, creato e **associato a una sorgente**.

Interazioni col mondo: eventi

- Solitamente le classi di eventi sono **`XyzEvent`** e gli ascoltatori sono istanze di **`XyzListener`**.
- Le listener sono interfacce tipicamente definite in **`java.awt.event`**, **`java.awt.beans`** e **`javax.swing.event`**.
  - Eventi e ascoltatori sono più generali che non relativi alle sole GUI. Quindi **`EventObject`** e **`EventListener`**, basi delle gerarchie, sono in **`java.util`** invece che **`awt`** o **`swing`**.
  - Le listener sono interfacce, quindi vanno **implementate** in una classe concreta.

Interazioni col mondo: eventi

- Le classi che creano eventi hanno anche metodi **`add/removeXyzListener()`** per associare quell'evento a un listener.
- Le interfacce listener hanno un metodo con parametro evento che dice cosa fare se capita.
- Per sapere l'oggetto che ha generato l'evento esiste il metodo generale **`getSource`.**
  - Ci sono metodi più specifici, ma non sono omogenei (esempio: **`ActionEvent`** ha il metodo **`getActionCommand`**, di tipo **`String`**)

Interazioni col mondo: eventi

- La classificazione di eventi, loro possibili lanciatori e possibili ascoltatori è complessa.
- Alcuni esempi:

Evento	Listener	Metodo	Lanciatore
`ActionEvent`	`ActionListener`	`actionPerformed`	`JButton, JMenuItem JList, JTextField`
`MouseEvent`	`MouseListener`	`mouse-: Clicked, Entered, Exited, Pressed, Released`	`JComponent`
	`MouseMotion Listener`	`mouse-: Dragged, Moved`	
`WindowEvent`	`WindowListener`	`window-: Closing, Closed, Activated`	`JWindow`

Interazioni col mondo: eventi

Evento	Listener	Metodo	Lanciatore
`ComponentEvent`	`Component Listener`	`component-: Resized, Moved, Hidden, Shown`	`JComponent`
`ContainerEvent`	`Container Listener`	`componentAdded, componentRemoved`	`JContainer`
`KeyEvent`	`KeyListener`	`key-: Pressed, Released, Typed`	`JComponent`
`ItemEvent`	`ItemListener`	`itemStateChanged`	`JCheckbox, JChoice, JList`

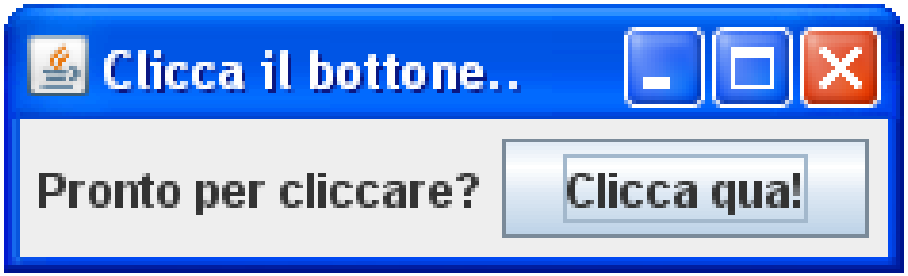
- La corrispondenza non è univoca: più oggetti possono lanciare lo stesso evento, e uno stesso evento (vedi **`MouseEvent`**) può avere più listener.

Gestione degli eventi

- Per gestire un evento occorre:
  - collegare al componente che lo genera uno o più percettori degli eventi di tipo opportuno
  - implementare un gestore degli eventi (non necessariamente distinto) per ogni percettore
- Un gestore degli eventi (event handler) è un metodo che viene automaticamente invocato dal listener come reazione all'evento.
  - Ogni interfaccia listener specifica uno o più handler, che **devono** essere definiti nella classe che implementa il listener

Gestione degli eventi

- Un esempio classico è un bottone da premere.
  - Il lanciatore di eventi è un'istanza di **`JButton`**.
  - L'evento innescato è istanza di **`ActionEvent`**.
  - L'ascoltatore è un'istanza di **`ActionListener`**.
  - Il metodo invocato è **`actionPerformed`**.



Gestione degli eventi

- Quando il bottone viene premuto:
  - la JVM genera un evento di tipo **`ActionEvent`**
  - per ogni oggetto che sia stato posti come ascoltatore dell'evento, esegue il suo metodo **`actionPerformed`** (lo ha poiché implementa **`ActionListener`**), con l'evento come parametro.

  - Proviamo a implementare il contatore...

Esempio: clicca bottone

```
import javax.swing.*;
import java.awt.event.*;
class Ascoltatore implements ActionListener
{ private CliccaBott cb;
  public Ascoltatore(CliccaBott cb)
  { this.cb = cb; }
  public void actionPerformed(ActionEvent e)
  { cb.conteggia(); }
}
public class CliccaBott extends JFrame
{ public CliccaBott() { ... }
  public void inizializzaGUI { ... }
  public void conteggia() { ... }
  public static void main(String[] args)
  { new CliccaBott(); }
}
```

Esempio: clicca bottone

```
public class CliccaBott extends JFrame
{ private JButton ilBottone; private JLabel etic;
  private int contatore;
  public CliccaBott()
  { super(); this.contatore=0;
    ...  this.setVisible(true);
  }
  private void inizializzaGUI()
  { ilBottone = new JButton("Clicca qua!");
    etic = new JLabel("Pronto per cliccare?");
    //aggiungi etic e ilBottone al pannello etc.
    ActionListener asc = new Ascoltatore(this);
    ilBottone.addActionListener(asc);
  }
  public void conteggia()
  { contatore++;
    etic.setText("Hai fatto "+contatore+" click");
  }
```

Innesco di eventi

- Le classi che innescano un evento **`XyzEvent`** di solito hanno a disposizione metodi protetti **`fireXyz()`** che creano un oggetto evento **`Xyz`** e lo **notificano** ai rispettivi ascoltatori.
  - Gli eventi possono essere percepiti da più listener
  - Ogni listener ascoltare quante sorgenti vuole
  - Le istanze di **`JComponent`** (o sottoclassi) mantengono un array **`EventListenerList`** (non è una hash table!) i cui elementi sono coppie (**`XyzListener.class`**, **`unXyzListener`**) ovvero: (classe, nome dell'istanza)

Innesco di eventi

- Aggiunte/rimozioni da **`EventListenerList`** implicano la creazione di un nuovo array mediante il metodo **`System.arrayCopy()`**
- L'array è più veloce di un tipo Collection, poiché la lista va scandita ad ogni evento
  - Quando si trova una classe compatibile, si notifica l'evento all'istanza (l'elemento subito dopo)
- Per coerenza tra thread multipli, i metodi di aggiunta/rimozione da **`EventListenerList`** **sincronizzano** l'accesso all'array

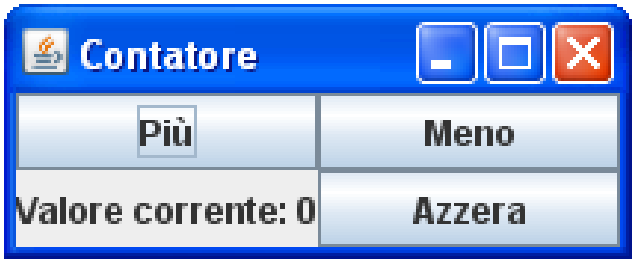
Gestione di più sorgenti

- Si possono gestire due sorgenti di eventi:
  - o definendo un ascoltatore per ognuna di esse
  - o definendo un singolo ascoltatore. Questi però dovrà distinguere tra gli eventi di una e dell'altra. Per farlo, userà il metodo **`getSource`** sull'evento.

```
public void actionPerformed(ActionEvent e)
{ if (e.getSource() == bottoneUno)
  { //fa' qualcosa }
  else if (e.getSource() == bottoneDue)
  { //fa' qualcos'altro}
  else ...
}
```

Esercizi suggeriti

- Creare contatori a **più bottoni**.



- Creare convertitori (per esempio da gradi Celsius a Fahrenheit, da dollari a euro, da metri a yarde, etc.): vanno usate una o più caselle di testo (**`TextField`**) dove l'utente inserisce input

Temporizzatori

- Nel package **`javax.swing`** esiste una classe **`Timer`** che serve a generare una sequenza di eventi a intervalli fissati, che vengono passati a un ascoltatore designato.

```
ActionListener asc = ... ;
int deltaT = 1000;
Timer t = new Timer(deltaT, asc);
t.start();
```

in millisecondi

fa partire un **nuovo thread**

- Si possono generare animazioni
- **Esercizio**: creare un display di un orologio.

Temporizzatori

- C'è una classe **`Timer`** un po' più generale nel package **`java.util`**. Quella di **`javax.swing`** è più indicata per le applicazioni grafiche leggere perché usa un **thread condiviso**.
- Attenzione: l'ascoltatore associato lancerà dei metodi **`actionPerformed`** per fare qualcosa.
  - Questi vengono eseguiti in un altro thread (**worker thread**) rispetto a quello che lancia gli eventi
  - Il worker thread deve essere leggero o la GUI diventa poco reattiva.

Gli eventi del mouse

- Qualunque **`JComponent`** può farsi ascoltare da **`MouseListener`** e **`MouseMotionListener`**
  - Per ascoltare pulsanti e movimenti del mouse
- Sono due interfacce che vanno implementate.
  - **`MouseListener`** ha **`mouseClicked`**, **`-Entered`**, **`-Exited`**, **`-Pressed`**, **`-Released`**
  - **`MouseMotionListener`** ha **`mouseDragged`**, **`mouseMoved`**
- Di solito non si vuole però implementarli tutti, ma solo uno o due.

Gli eventi del mouse

- Pertanto Java mette a disposizione le classi **`MouseAdapter`** e **`MouseMotionAdapter`**
  - Hanno un'implementazione concreta di tutti quei metodi (i primi cinque e i secondi due)
  - L'implementazione è non fare alcuna azione
- In questo modo, un ascoltatore del mouse può essere definito come estensione di queste classi adattatrici.
  - A questo punto si può fare ovverride solo dei metodi che devono fare qualcosa di specifico.