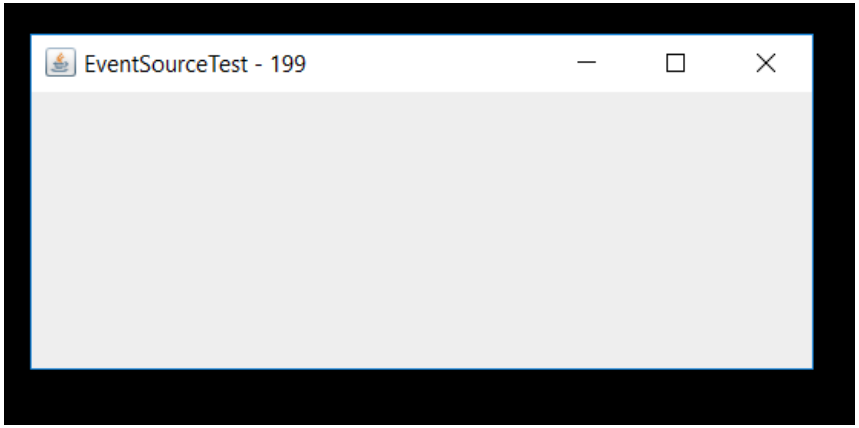
Programmazione III

Prof.ssa Liliana Ardissono
Dipartimento di Informatica
Università di Torino

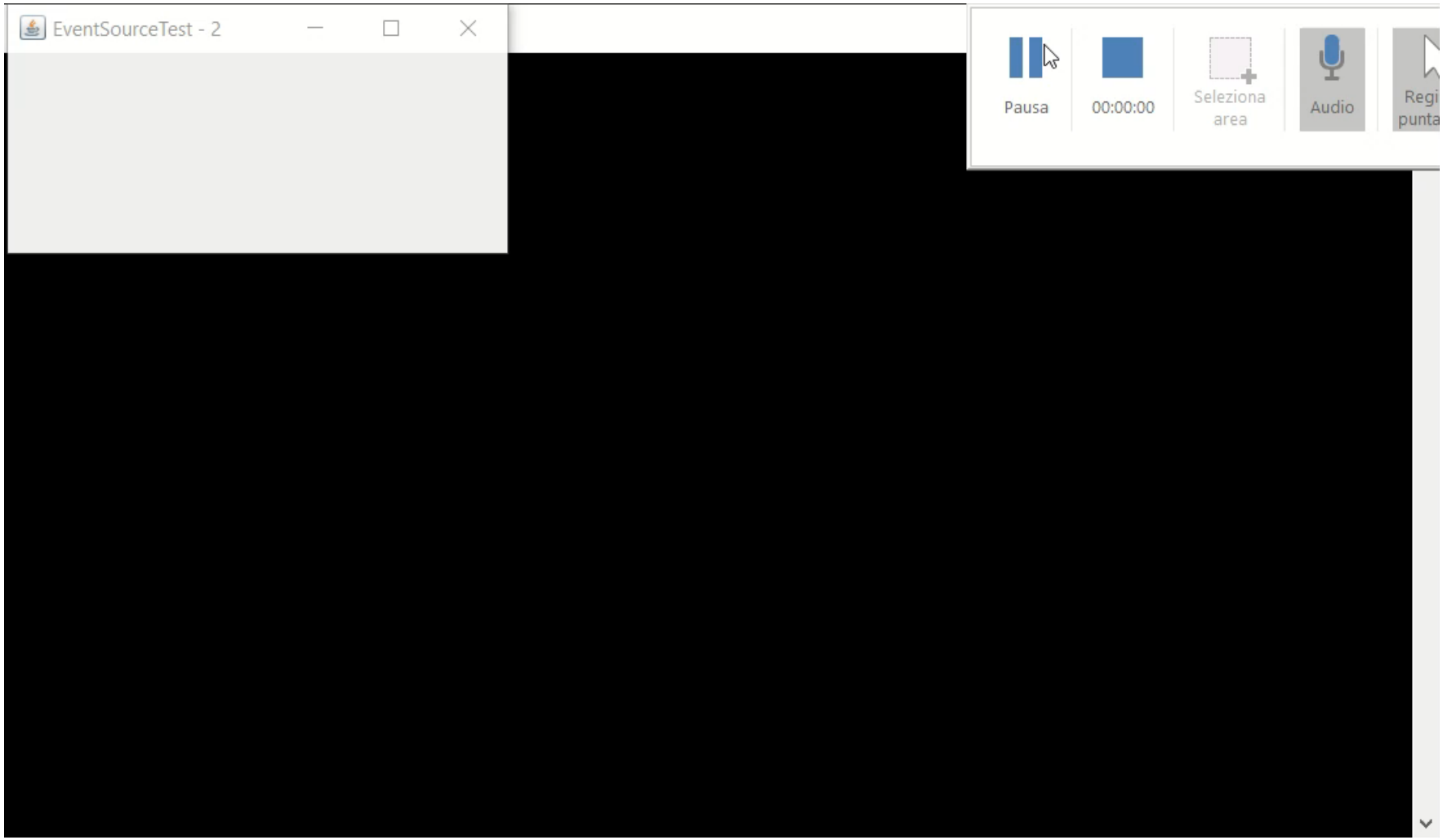
Interfacce Utente Grafiche (GUI) – parte 2
(basi con Java SWING)
Implementazione della gestione di eventi

Per capire meglio il meccanismo della gestione degli eventi vediamo un esempio **EventSourceTest** da Core Java che usa eventi diversi da quelli standard offerti dalle componenti grafiche → devono essere gestiti esplicitamente nel codice delle componenti grafiche.

Data una finestra, vogliamo contare quante volte il metodo **paintComponent()** è chiamato. Per farlo, ad ogni invocazione facciamo in modo che la finestra generi un evento e lo invii a un *listener*, che provvede ad aggiornare il titolo della finestra.

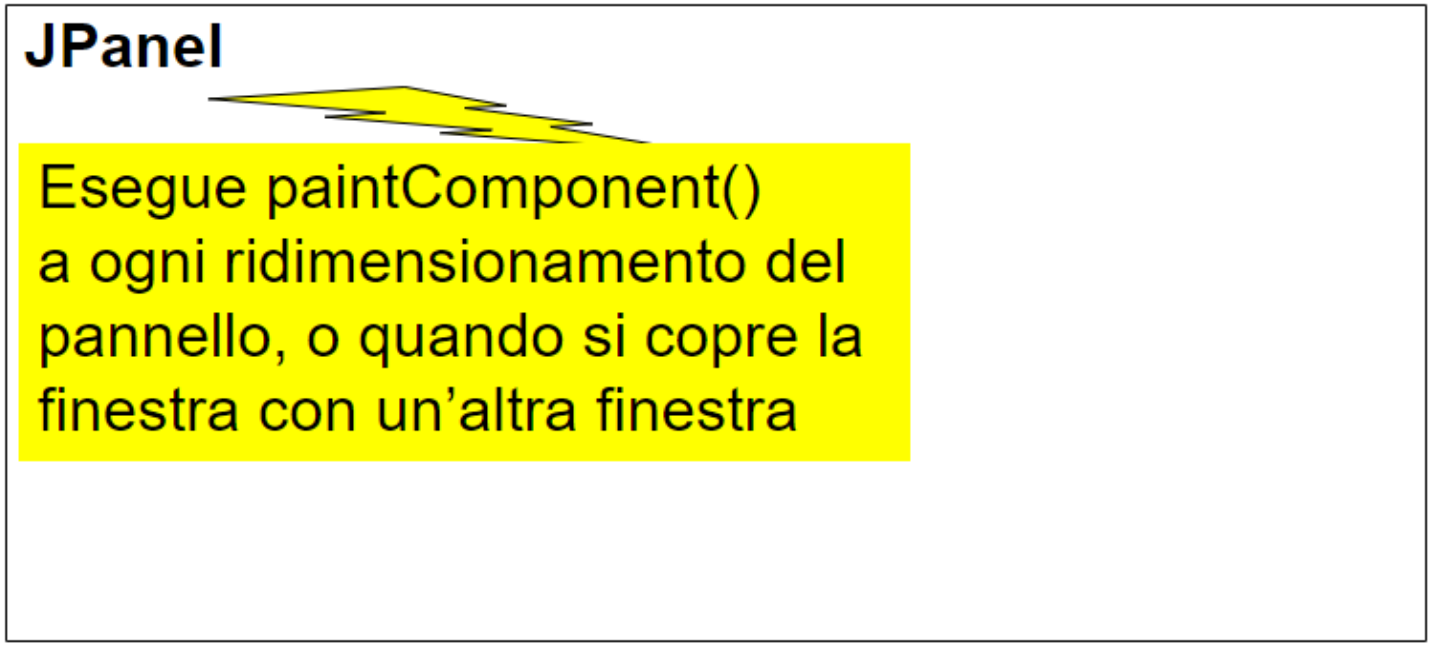


In questo esempio la finestra è stata ridimensionata e questo ha fatto eseguire paintComponent() 199 volte


EventSourceTest - 2
Pausa
00:00:00
Seleziona area
Audio

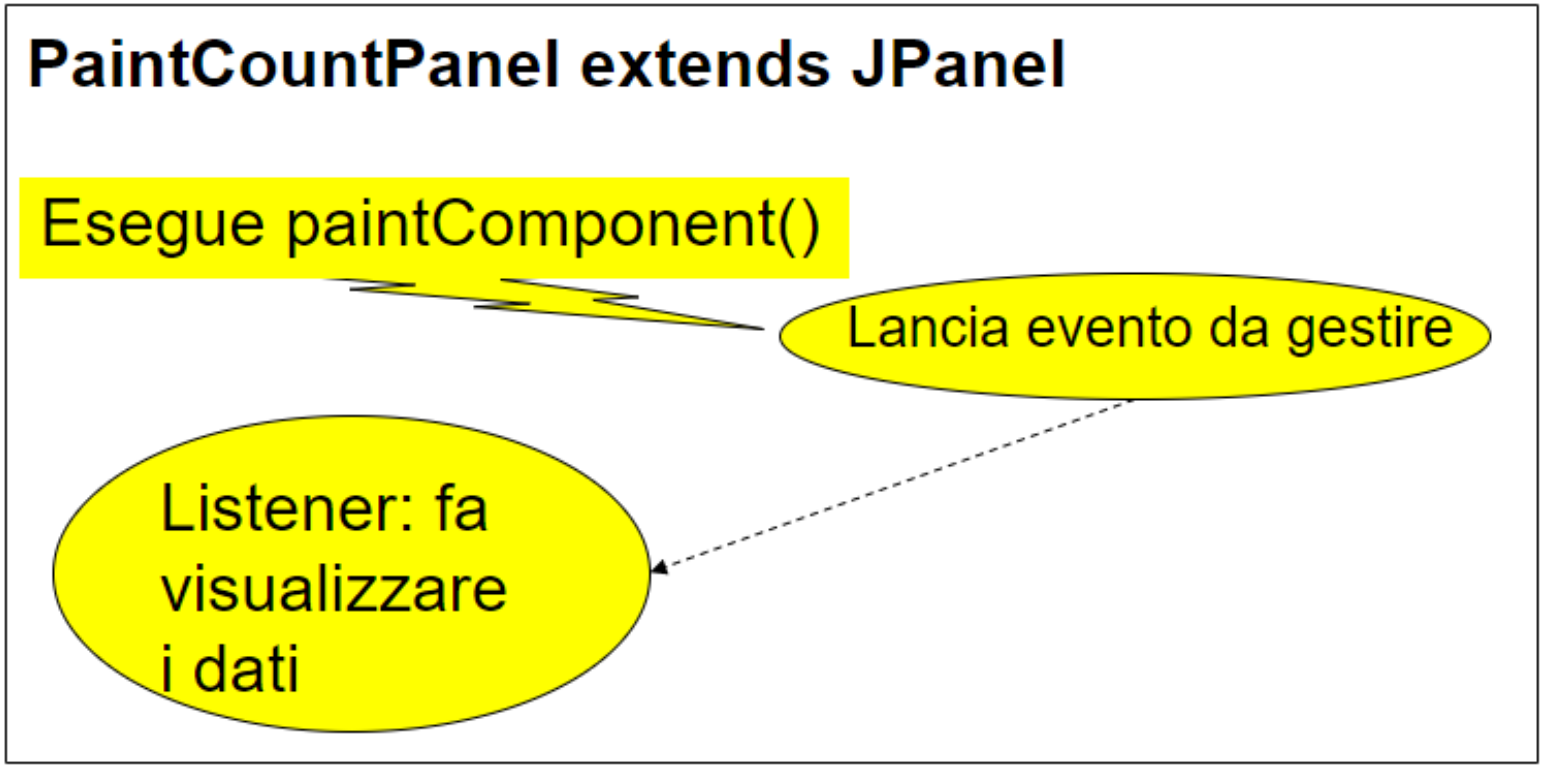
EventSourceTest - I

Il metodo **paintComponent()** è chiamato automaticamente ogni volta che una finestra deve essere ridisegnata: ad es. quando si cambia la dimensione, quando la finestra è coperta da un'altra.



EventSourceTest - II

Possiamo quindi utilizzare il metodo paintComponent() per far lanciare un evento a ogni sua invocazione



EventSourceTest - III

Definiamo la classe **PaintCountPanel** che è il pannello che genera gli eventi «non standard».

PaintCountPanel extends JPanel

Ci servono tre ingredienti:

- il tipo degli eventi generati dal **PaintCountPanel**
- l'interfaccia del *listener* per questo tipo di eventi
- i metodi per aggiungere (registrare) o togliere i *listener* al PaintCountPanel (che è la sorgente degli eventi)

E poi le modifiche del metodo di paintComponent() per far generare gli eventi.

EventSourceTest - IV

Come tipo di evento da lanciare scegliamo **PropertyChangeEvent, appartenente alla libreria degli eventi Java** (classe di eventi predefinita)

class PropertyChangeEvent - descrive gli eventi lanciati tutte le volte che un bean (qui il JPanel) cambia il valore di una sua property.

Costruttore:

PropertyChangeEvent(Object source,

String propertyName, Object oldValue, Object newValue)

- Source è la sorgente dell'evento
- oldValue il valore precedente della property
- newValue il valore attuale della property (dopo il cambiamento)

EventSourceTest - V

Il listener dei PropertyChangeEvent è il PropertyChangeListener (Interface)

Interface PropertyChangeListener {

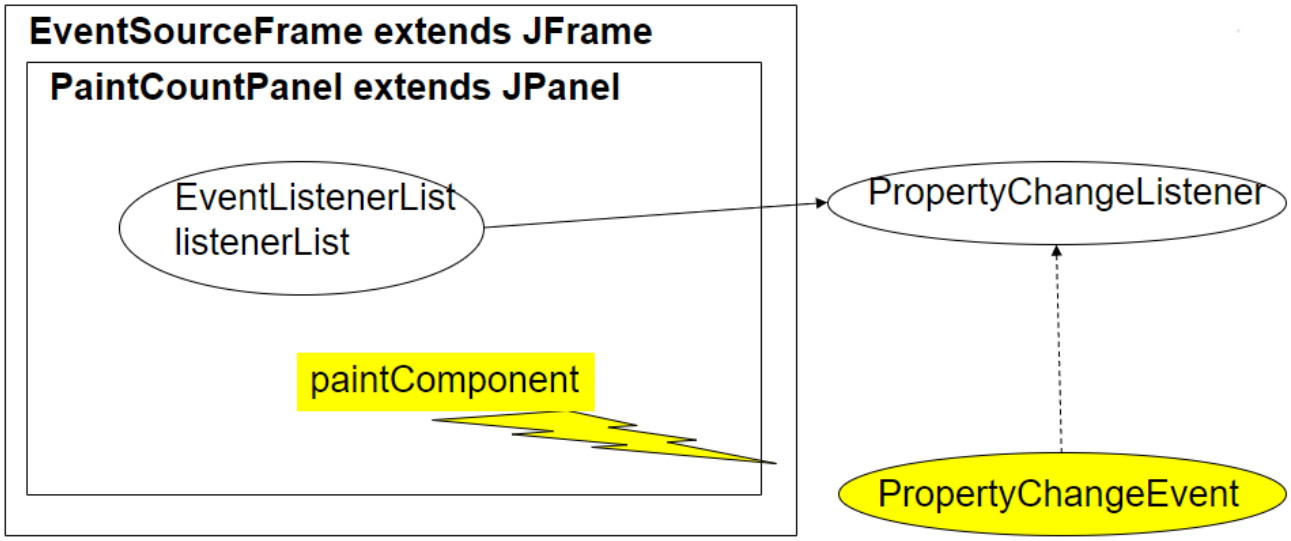
propertyChange(PropertyChangeEvent event)

}

Metodi per aggiungere e togliere listener al PaintCountPanel - occorre definire in PaintCountPanel i metodi:

addPropertyChangeListener(PropertyChangeListener l)

removePropertyChangeListener(PropertyChangeListener l)



Per gestire la lista dei *listener*, la classe **JPanel** ha un campo **listenerList** di tipo **EventListenerList**, ereditato da **JComponent**. La classe **EventListenerList**, fornita dalle librerie di Java, serve per tenere tutti i listener (di qualunque tipo) associati a una sorgente di eventi.

EventListenerList: metodi (da API Java)

public <T extends EventListener> T[] getListeners(Class<T> t):

restituisce un array di tutti i *listener* di tipo **t**

public <T extends EventListener> void add(Class<T> t, T l):

aggiunge il listener **l** di tipo **t** alla lista dei listeners

public <T extends EventListener> void remove(Class<T> t, T l):

toglie il listener **l** di tipo **t** dalla lista dei listeners

PaintCountPanel - I

Definiamo i metodi per aggiungere o togliere i *listener* ad un **PaintCountPanel**:

```
public void addPropertyChangeListener(PropertyChangeListener l) {
    listenerList.add(PropertyChangeListener.class, l);
}

public void removePropertyChangeListener(PropertyChangeListener l) {
    listenerList.remove(PropertyChangeListener.class, l);
}
```

Il listener è definito come classe anonima che viene registrata nel panel:

PropertyChangeEvent(Object source,
String propertyName, Object oldValue, Object newValue)

```
class EventSourceFrame extends JFrame {

  public EventSourceFrame() {
    setTitle("EventSourceTest");
    setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

    final PaintCountPanel panel = new PaintCountPanel();
    add(panel);

    panel.addPropertyChangeListener(new
        PropertyChangeListener() {
            public void propertyChange(PropertyChangeEvent ev){
                setTitle("EventSourceTest - " + ev.getNewValue());
            }
        });
}
```

PaintCountPanel – II

PaintCountPanel deve generare un evento ogni volta che si esegue paintComponent(), e inviare questo evento ai propri listener →dobbiamo ridefinire il metodo paintComponent(Graphics g)

```
class PaintCountPanel extends JPanel {
   private int paintCount; // conta il numero di «refresh» del panel
  @override
  public void paintComponent(Graphics g) {
    int oldPaintCount = paintCount;
    paintCount++;
    firePropertyChangeEvent(new PropertyChangeEvent(this,
                                "paintCount", oldPaintCount, paintCount));
    super.paintComponent(g);
  }
  public void addPropertyChangeListener(PropertyChangeListener l) {
          listenerList.add(PropertyChangeListener.class, l);
   }
  public void removePropertyChangeListener(PropertyChangeListener l) {
          listenerList.remove(PropertyChangeListener.class, l);
  }
  // continua…
```

PaintCountPanel – III

```
// continua…

public void firePropertyChangeEvent(PropertyChangeEvent event) {
  EventListener[] listeners =
        listenerList.getListeners(PropertyChangeListener.class);
  for (EventListener l : listeners) {
            ((PropertyChangeListener) l).propertyChange(event);
  }
} // end firePropertyChangeEvent
} // end PaintCountPanel
```

Ringraziamenti

Grazie al Prof. Emerito Alberto Martelli del Dipartimento di Informatica dell’Università di Torino per aver redatto la prima versione di queste slides.