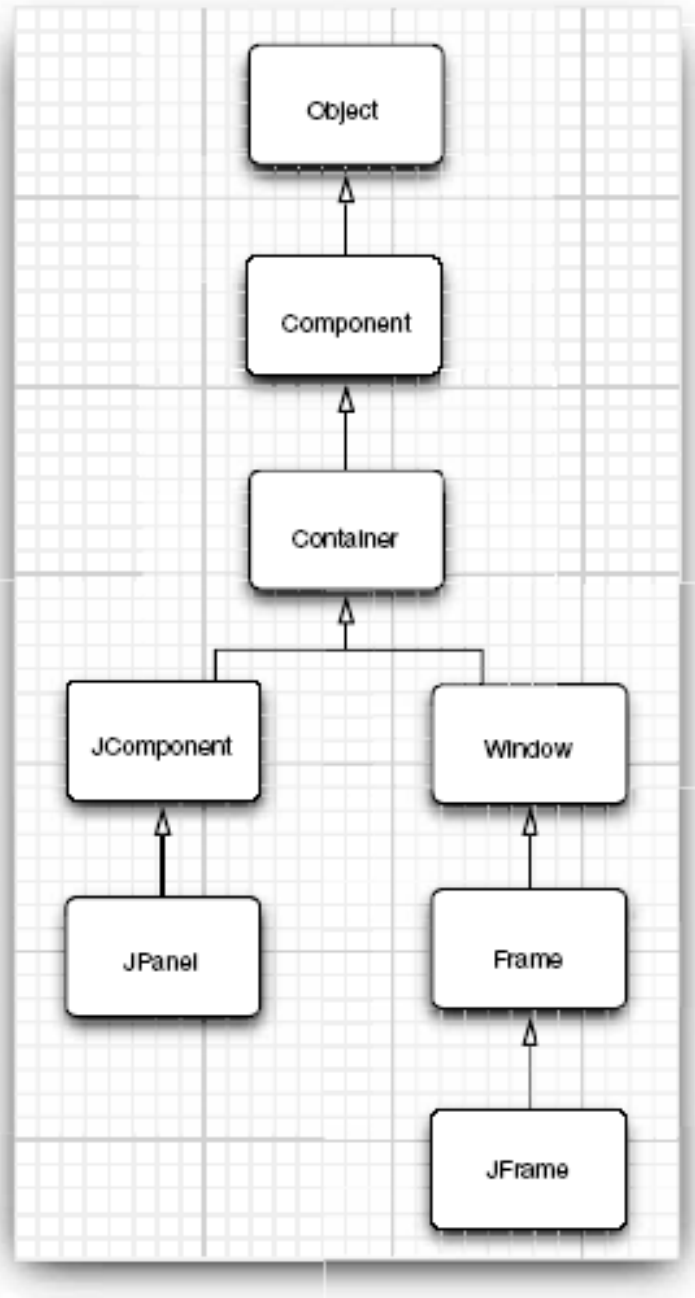
Programmazione Orientata agli Oggetti

Elementi di programmazione dell'Interfaccia Utente Grafica (GUI) utilizzando Java AWT/Swing

Libero Nigro

Introduzione

- AWT – A Windowing Toolkit, è stato il primo framework di Java per lo sviluppo di GUI.
- AWT delega molto al Sistema Operativo (S.O.) sottostante (es. MS Windows) per l'ottenimento ed il tracciamento degli elementi della GUI (finestre, pulsanti, menu, etc.).
- Swing è nato come esperimento di minimizzare le dipendenze dal S.O.
- Solo la finestra è chiesta al S.O. Il tracciamento e la gestione di ogni altro elemento di GUI è responsabilità del programmatore Swing che cosi può assicurare un look uniforme alle applicazioni anche cambiando S.O.
- Il modello di gestione degli eventi di Swing è comunque quello precedentemente definito da AWT: **Event Delegation Model.**
- AWT e Swing sono esempi di **framework.**
- Un framework è una collezione di classi già pronte per l'uso. Se richiesto, tali classi possono essere specializzate via inheritance prima di essere utilizzate.
- Accanto alla collezione di classi, un framework definisce anche un proprio **flusso di eventi**. È possibile far partecipare i propri oggetti specializzazione di classi del framework, al flusso degli eventi.
- Evoluzione: JavaFX come nuovo framework per costruire GUI.



Gerarchia base finestre

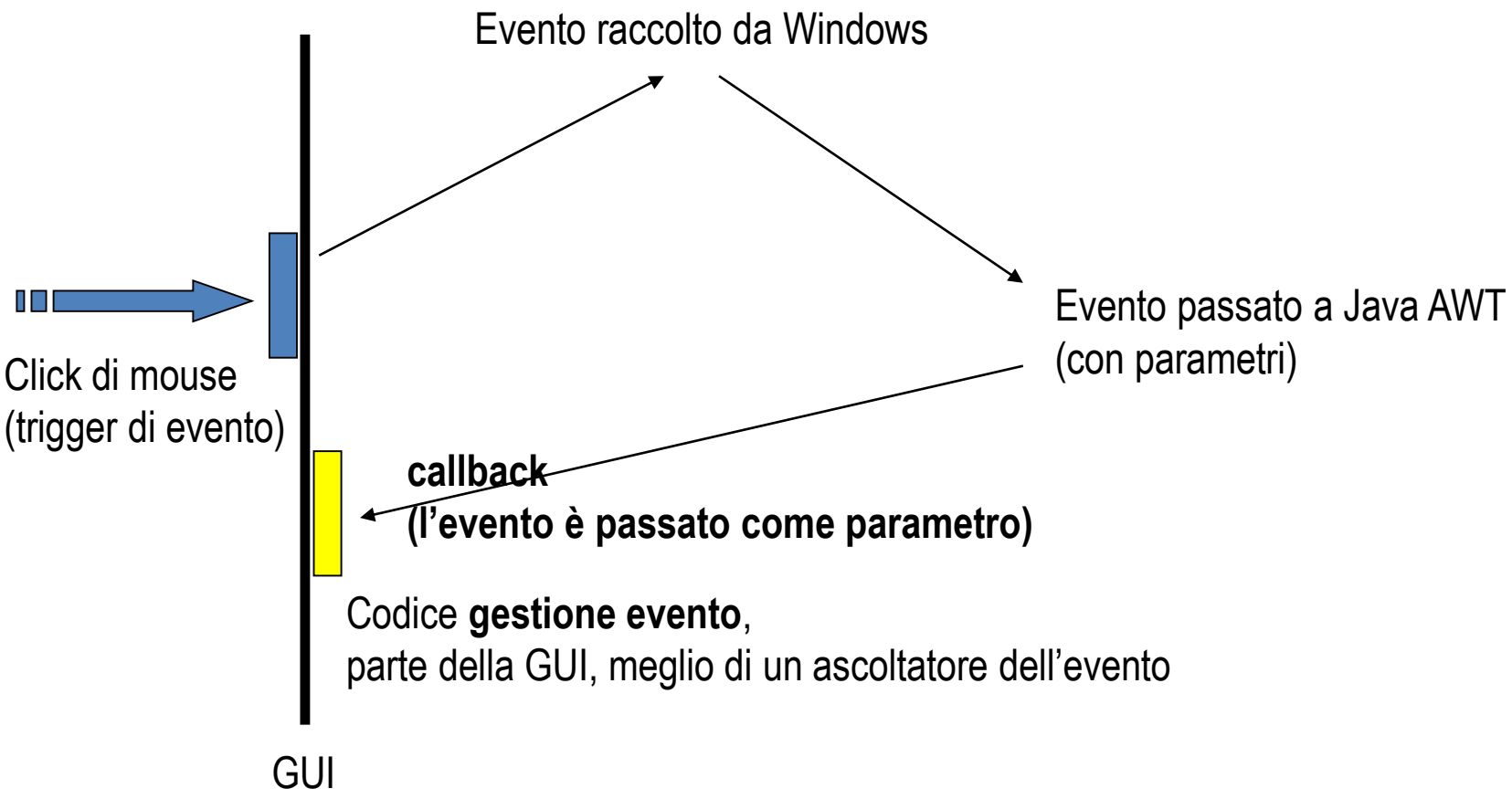
Una GUI è una gerarchia di componenti
Si distinguono contenitori e componenti

Come caso particolare, un componente innestato in un contenitore, può essere a sua volta un contenitore etc.

Frame e pannelli

- Due contenitori chiave sono **JFrame** e **JPanel.**
- Componenti elementari sono JLabel, JTextField, JButton, JRadioButton, etc.
- Le classi JFrame e JPanel si possono utilizzare direttamente per istanziazione o, molto spesso, possono venire estese e poi utilizzate.
- Una JFrame assume normalmente il ruolo di contenitore base degli elementi della GUI di un'applicazione.
- Un JPanel è un componente parte di una JFrame. Esso è utile per la visualizzazione grafica (tracciamento di figure, immagini, …). Un caso particolare di JPanel è JApplet, ossia le mini applicazioni agganciabili alle pagine web.

Event Delegation Model



GUI: Programmazione Pilotata dagli Eventi (Event Driven Programming)

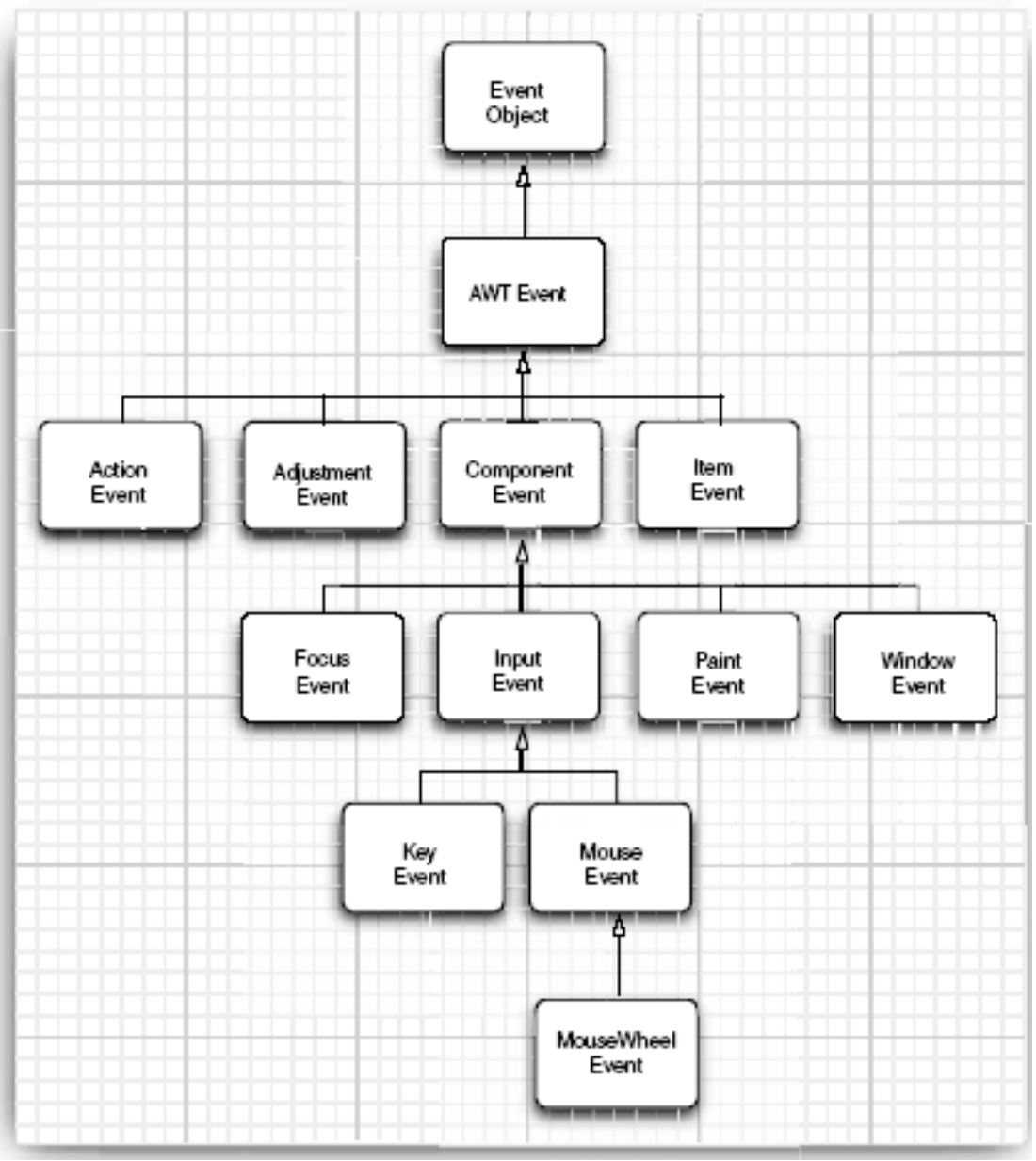
- Il cuore della programmazione di una GUI consiste nel predisporre il codice di gestione degli eventi scatenati sulla interfaccia.
- Quando verrà eseguito tale codice ? Quando l'utente genera l'evento es. con un click di mouse sulla GUI.
- È importante notare che i codici di gestione degli eventi (codici delle callback) definiscono uno ***schema di esecuzione non deterministico***: non sappiamo in che ordine verranno generati gli eventi possibili.
- Il programmatore deve garantire che quale che sia l'ordine il comportamento è corretto.
- In realtà, gli eventi vengono dapprima bufferizzati in una coda di eventi (EventQueue) di AWT/Swing e quindi processati a cura di un thread fondamentale: **Event Dispatch Thread** (EDT) (thread controllore degli eventi).
- Non sempre è vero che dopo aver scatenato un evento esso verrà *subito* processato. Tutto dipende dal fatto se esistono o meno altri eventi sulla coda e ultimamente l'ordine di processamento (elaborazione) dipende dalla particolare politica adottata da EDT.

Classificazione (parziale) degli eventi

- Eventi legati alle finestre (**WindowEvent**)
- Eventi azione (**ActionEvent**)
- Eventi di mouse (**MouseEvent**)
- Eventi legati alla keyboard (**KeyEvent**)

- Quando nasce un evento, viene generato un oggetto della classe di appartenenza ed inizializzato con dati (parametri) caratteristici dell'evento.
- Ad esempio, un evento di click di mouse si accompagna alle coordinate x,y del punto di click, un evento azione si accompagna agli attributi della sua sorgente (il tipo di elemento di interfaccia su cui è nato l'evento) ed al suo contenuto (si pensi ad un campo di testo, in cui l'utente digita una certa stringa).

Gerarchia delle classi di eventi di AWT



Interfacce Ascoltatori di Eventi (Event Listener)

- WindowListener
- ActionListener
- MouseListener
- MouseMotionListener
- …

Tali interfacce introducono uno o più metodi callback che ricevono l'evento scatenante (munito di parametri) come argomento.

Interface	Methods	Parameter/ Accessors	Events Generated By
ActionListener	actionPerformed	ActionEvent • getActionCommand • getModifiers	AbstractButton JComboBox JTextField Timer
AdjustmentListener	adjustmentValueChanged	AdjustmentEvent • getAdjustable • getAdjustmentType • getValue	JScrollbar
ItemListener	itemStateChanged	ItemEvent • getItem • getItemSelectable • getStateChange	AbstractButton JComboBox
FocusListener	focusGained focusLost	FocusEvent • isTemporary	Component
KeyListener	keyPressed keyReleased keyTyped	KeyEvent • getKeyChar • getKeyCode • getKeyModifiersText • getKeyText • isActionKey	Component

MouseListener	mousePressed mouseReleased mouseEntered mouseExited mouseClicked	MouseEvent • getClickCount • getX • getY • getPoint • translatePoint	Component
MouseMotionListener	mouseDragged mouseMoved	MouseEvent	Component
MouseWheelListener	mouseWheelMoved	MouseWheelEvent • getWheelRotation • getScrollAmount	Component
WindowListener	windowClosing windowOpened windowIconified windowDeiconified windowClosed windowActivated windowDeactivated	WindowEvent • getWindow	Window
WindowFocusListener	windowGainedFocus windowLostFocus	WindowEvent • getOppositeWindow	Window
WindowStateListener	windowStateChanged	WindowEvent • getOldState • getNewState	Window

Adattatori

- Quando un'interfaccia listener introduce diversi metodi callback, per semplificare la vita del programmatore che potrebbe essere interessato solo a certi (pochi) metodi callback anziché a tutti, il framework AWT mette a disposizione anche **classi adapter** che "implementano" i metodi a vuoto, ossia il corpo è { }.
- Il programmatore potrebbe quindi estendere (anche "al volo") queste classi di adapter e interessarsi alla ridefinizione dei soli metodi di interesse, oppure usare delle lambda expression quando si hanno *interfacce funzionali.*
- Un esempio tipico è WindowListener che definisce sette (7) metodi callback:
  - windowClosing, windowClosed, windowIconified, windowDeiconified, windowOpened, windowActivated, windowDeactivated
- Adattatori disponibili:
  - WindowAdapter
  - MouseAdapter
  - MouseMotionAdapter
  - KeyAdapter
  - FocusAdapter

Progetto di un ascoltatore

- Nel progettare una classe ascoltatore di eventi, o si implementa (una o più) interfaccia listener (ossia si implementano TUTTI i metodi dell'interfaccia) o si estende, se esiste, una corrispondente classe adapter
- Dovendo gestire l'evento di chiusura di una frame (l'utente clicca sul pulsante **x** in alto a destra, o digita ALT-F4 o seleziona esci da un menu), che è catturato dalla callback windowClosing(WindowEvent e), è conveniente introdurre una classe (tipicamente inner) che estende WindowAdapter e si limita a ridefinire il metodo windowClosing(…) anziché implementare l'interfaccia WindowListener

Esempio
di JFrame

Codice per generare il frame

```
package poo.swing;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Finestra extends JFrame{
   public Finestra(){
      setTitle("Finestra Chiudibile");
      setSize(300,200);
      setLocation(50,200);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
}//Finestra
```

```
public class FinestraChiudibile{
   public static void main( String []args ){
      JFrame f=new Finestra();
      f.setVisible(true);
   }
}//FinestraChiudibile
```

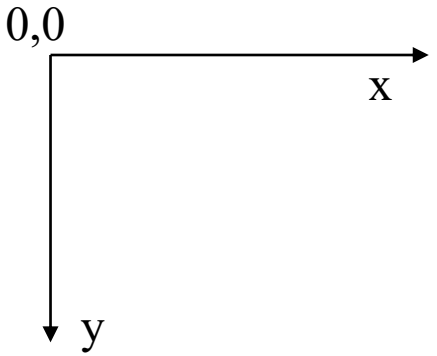
In questo modo, quando l'utente clicca sul pulsante X di chiusura o digita ALT-F4, si abbandona immediatamente l'applicazione (non sempre una cosa buona …)

Intercettazione e gestione dell'evento di chiusura installando un ascoltatore

```
public Finestra(){//costruttore di Finestra
    setTitle("Finestra Chiudibile");
    setSize(300,200);
    setLocation(50,200);
    setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    addWindowListener( new WindowAdapter(){
       public void windowClosing(WindowEvent e){//callback
          System.exit(0); //o, meglio, altra gestione come opportuno
       }
    } );
 }
```

Sistema di coordinate

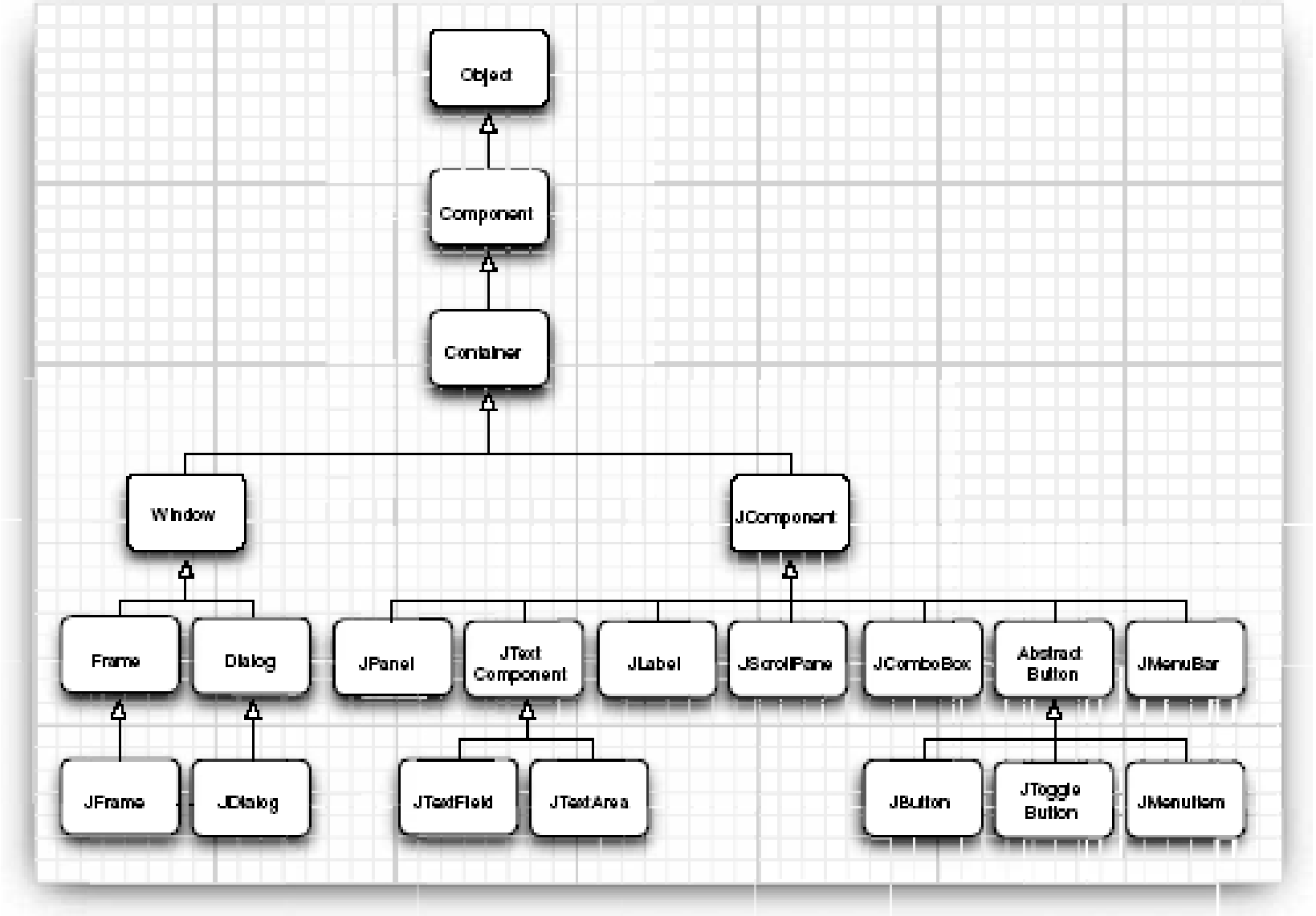
- setLocation definisce dove è collocata inizialmente la finestra sul video. Specifica le coordinate <x,y> (in pixel) del punto più in alto a sinistra della finestra
- Il sistema di coordinate, in questo caso, si riferisce all'intero video, con l'origine <0,0> posta nello spigolo più in alto a sinistra del video
- L'asse x va verso destra
- L'asse y va verso il basso



- Più in generale, quando si forniscono le coordinate di tracciamento di un oggetto es. su un pannello, il *sistema di coordinate* da adottare è quello *locale*:

l'origine <0,0> è lo spigolo in alto a sinistra del componente e non del video, asse x diretto verso destra, asse y diretto verso il basso

Gerarchia componenti di GUI



Inizializzazione di una JFrame

```
public class FinestraChiudibile{
   public static void main( String []args ){
      JFrame f=new Finestra();
      f.setVisible(true);
   }
}//FinestraChiudibile
```

Affidata al thread del main

```
public class FinestraChiudibile{
   public static void main( String []args ){
     EventQueue.invokeLater( new Runnable(){
        public void run(){
          JFrame f=new Finestra();
          f.setVisible(true);
        }
     });
   }//main
}//FinestraChiudibile
```

Affidata allo “Event Dispatch Thread” di Swing

Uso di JFrame e JPanel

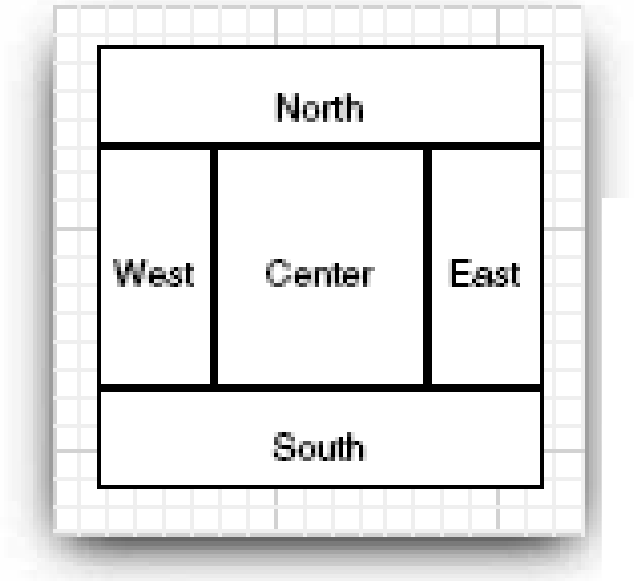
- Su un JFrame di norma si colloca una struttura di menu per controllare l'applicazione
- Su un JPanel (o JComponent) (es. JApplet) si tracciano o si visualizzano oggetti grafici o si inseriscono elementi di interfacciamento con l'utente come campi di testo (JTextField), bottoni (JButton) etc.
- È pratica comune quella di creare più pannelli da inserire in una JFrame che realizza la GUI e disporre nei singoli pannelli elementi di interfacciamento diversi da menu.
- Da una JFrame si può anche creare al volo una differente JFrame per gestire l'interazione con l'utente in una particolare situazione della GUI.

- Potendo inserire più elementi in un contenitore, si pone in generale il problema della loro disposizione. Esistono in proposito i **layout manager.**
- Su una JFrame è di default il **BorderLayout**, che consente di inserire gli elementi (es. pannelli) a NORTH, a SOUTH, a EAST, a WEST o nel CENTER, es.

```
frame.add( component, BorderLayout.SOUTH );
```

- Se si desidera, si può cambiare il default installando un differente layout manager
- Su un pannello è di default il **FlowLayout** secondo cui gli elementi vengono ad occupare posizioni successive delle "righe" del pannello, a partire dalla prima, a mano a mano che l'utente effettua gli inserimenti.

BorderLayout (Su una Jframe)



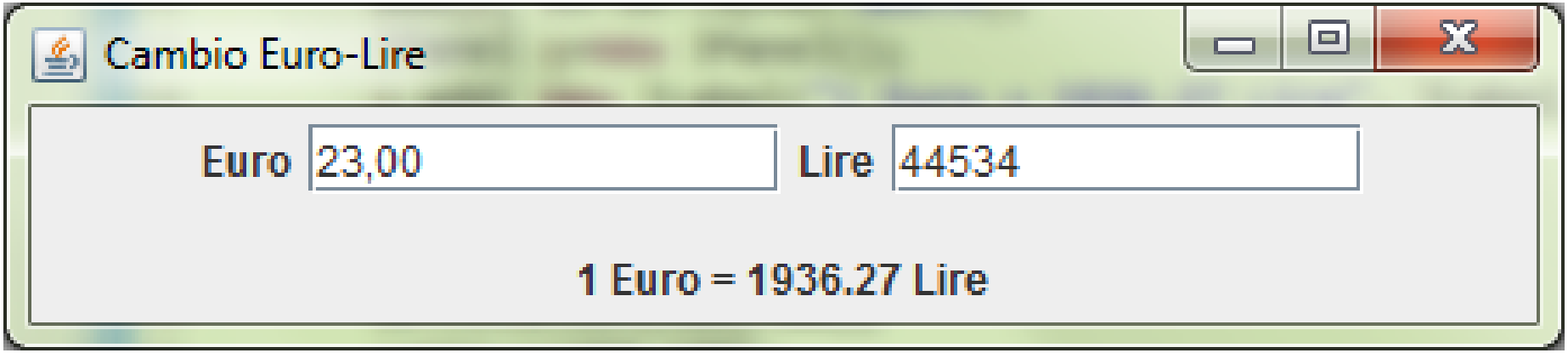
FlowLayout (Su un JPanel)

GridLayout

pannello.**setLayout**(new GridLayout(4,4)); //4 righe e 4 colonne
Gli elementi inseriti occuperanno ordinatamente le posizioni della griglia

pannello.**setLayout**(new GridLayout(4,4,3,3));
Gli ultimi due parametri stabiliscono i gap (in pixel) vert./orizz. tra i componenti

Finestra di Cambio Euro-Lire



Sono presenti due campi di testo (JTextField), ciascuno provvisto di label. La GUI è una JFrame con un pannello interno contenenti i due oggetti JTextField; un altro pannello contiene la scritta del cambio

Codice Java

```
package poo.swing;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;

public class Cambio{
   public static void main(String []args){//esempio
      FinestraCambio fc=new FinestraCambio();
      fc.setVisible(true);
   }
}//Cambio
```

FinestraCambio

```
class FinestraCambio extends JFrame implements ActionListener{
   private JTextField euro, lire;
   private final float EURO_LIRE=1936.27f;
   public FinestraCambio(){
      setTitle("Cambio Euro-Lire");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JPanel p=new JPanel();
      p.add( new JLabel("Euro", JLabel.RIGHT) );
      p.add( euro=new JTextField("",12) );
      p.add( new JLabel("Lire", JLabel.RIGHT) );
      p.add( lire=new JTextField("",12) );
      add(p, BorderLayout.NORTH );
      JPanel q=new JPanel();
      q.add( new JLabel("1 Euro = 1936.27 Lire", JLabel.RIGHT) );
      add( q, BorderLayout.SOUTH );
      euro.addActionListener( this );
      lire.addActionListener( this );
      setSize(450,100);
   }
```

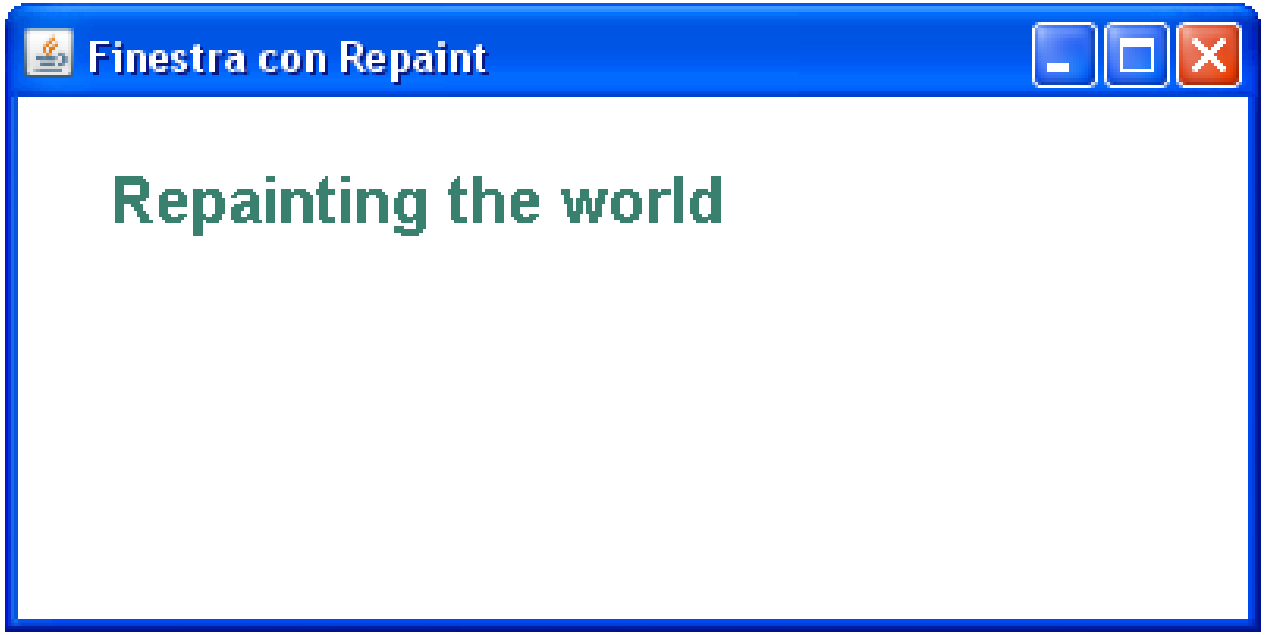
Per semplicità la finestra è anche l'ascoltatore degli eventi azione dei JTextField

FinestraCambio

```
public void actionPerformed( ActionEvent  evt ){//callback
    if( evt.getSource()==euro ){
        float e=Float.parseFloat( euro.getText() );
        euro.setText( String.format("%1.2f",e) );
        float lit=Math.round(e*EURO_LIRE);
        lire.setText( String.format("%1.0f",lit) );
    }
    else if( evt.getSource()==lire ){
        float eur=Float.parseFloat( lire.getText() )/EURO_LIRE;
        lire.setText( String.format("%1.0f", Float.parseFloat( lire.getText() ) ));
        euro.setText( String.format("%1.2f",eur) );
    }
  }
}//FinestraCambio
```

- L'applicazione è preparata a ricevere o lire o euro (*non determinismo*) e a rispondere mostrando sull'altro text field il valore corrispondente del cambio.
- Gli oggetti text field sono associati allo action listener rappresentato dalla stessa finestra JFrame che, in questo esempio, implementa la callback quando si scatena un evento azione su un campo di testo.
- In generale è **preferibile** programmare a parte la classe di ascoltatore, in modo da separare le istruzioni di risposta agli eventi da tutto il resto del codice.
- Su un JTextField si fa nascere un evento azione quando si digita una stringa di testo e si conclude (**importante**) con INVIO. Senza l'INVIO NON si trasmette il contenuto della stringa.
- Si nota, infine, che FinestraCambio, per semplicità, ignora situazioni di errore come: scrivere un ammontare in lire frazionario o negativo, usare un ammontare in euro negativo, usare quantità di denaro non numeriche. Una versione più robusta potrebbe segnalare l'errore sulla console e imbiancare i campi di testo. Consultare il codice disponibile al di fuori di queste slide.

Una Finestra con Repainting



Il programma mostra la stringa “Repainting the world” ad ogni occasione
In cui è richiesta la ri-visualizzazione. Randomaticamente si sceglie una
tra due posizioni per il drawing. Si utilizzano un font ed un colore particolari
La ri-visualizzazione avviene quando si passa da minimizzazione
a massimizzazione, re-sizing etc della finestra.

```
package poo.swing;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class FinestraRepaint extends JFrame{
   private Pannello p=null;
   private Font f=new Font("Helvetica", Font.BOLD, 20);
   private Color col=new Color( /*red*/57, /*green*/128, /*blu*/110 );

   public FinestraRepaint(){
      setTitle("Finestra con Repaint");
      setSize(400,200);
      setLocation(50,200);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      add( p=new Pannello() );
   }
```

```
class Pannello extends JPanel{ //esempio di JComponent
    public void paintComponent( Graphics g ){
      super.paintComponent( g ); //obbligatorio
      System.out.println("paintComponent chiamata");
      setBackground( Color.white );
      g.setFont( f );
      g.setColor( col );
      if( Math.random()<0.5 )
          g.drawString( "Repainting the world", 30, 40 );
      else
          g.drawString( "Repainting the world", 100, 100);
    }
 }//Pannello
```

```
public class FinestraConRepaint{
   public static void main( String []args ){
     EventQueue.invokeLater( new Runnable(){
          public void run(){
               JFrame f=new FinestraRepaint();
               f.setVisible(true);
          }//run
     });
   }//main

}//FinestraConRepaint
```

Altri componenti di GUI (lista parziale)

- **JTextArea**: generalizza JTextField. Mentre un JTextField è normalmente usato per l'input di una linea di testo, una JTextArea può essere impiegata per inserire un testo di più linee. Una JTextArea può essere equipaggiata con le barre di scorrimento, procedendo come segue:

  JTextArea textArea=new JTextArea(10, 50); //10 righe, 50 colonne–nominali o di partenza

  JScrollPane textAreaScrollable=new JScrollPane(textArea);
- Sia un JTextField che una JTextArea possono essere controllati per fornire solo dati di uscita, generati dal programma, non modificabili dall'utente come segue:

  textField o textArea.setEditable(false);
- Una porzione di testo (es. una linea provvista di fine linea) può essere aggiunta alla fine di una text area col metodo textArea.append(String testo).
- Il contenuto di tutta l'area può essere prelevato col metodo String paramString().
- È possibile inserire testo ad una certa posizione (riga) pos col metodo: insert(String testo, int pos). Si può rimpiazzare il testo tra le posizioni start ed end (end>=start) col metodo replacerange(String str, int start, int end).
- I cambiamenti al contenuto di una text area possono essere monitorati mediante un DocumentListener cui vengono trasmessi DocumentEvent (che recano informazioni tipo la posizione dove il cambiamento è intervenuto etc.). Si rimanda alle API di Java per dettagli.

- **Check boxes:** utilizzabili quando un numero limitato di scelte sono a disposizione dell'utente. La classe è **JCheckBox**, es.

  JCheckBox italiano=new JCheckBox("Italiano");

  JCheckBox inglese=new JCheckBox("Inglese");

  italiano.addActionListener(ascoltatore_eventi_azione); //si può passare una lambda

  inglese.addActionListener(ascoltatore_eventi_azione);
- La caratteristica dei check box è che possono essere "spuntati". La spunta (click di mouse) genera un evento azione che un opportuno ascoltatore di eventi azione può sentire e processare. Più check box possono essere spuntati contemporaneamente
- **Radio buttons** (classe **JRadioButton**). Rappresentano elementi di interfaccia simili ai check box. La differenza è che i radio buttons vanno costituiti in gruppi. In un gruppo la selezione di un radio button è esclusiva rispetto agli altri radio button del gruppo.
- ButtonGroup gruppo=new ButtonGroup();

  JRadioButton al=new JRadioButton("ArrayList", false);

  gruppo.add(al);

  JRadioButtol ll=new JRadioButton("LinkedList", true); //selezionato inizialmente

  gruppo.add(ll);

  …
- I r.b. di un gruppo possono essere "attaccati" ad uno stesso ActionListener o differenti action listener possono essere definiti uno per ogni r.b.

- Quando il numero di r.b. non è piccolo, essi possono occupare troppo spazio sul pannello. In questi casi è utile un **Combo box** che quando evocato (basta clickare sul componente che lo possiede) mostra una lista di scelte tra cui l'utente seleziona quella di interesse.
- La classe **JComboBox** consente di costruire oggetti combo box; il numero delle scelte può essere editabile dinamicamente (anticipando setEditable(true)).
- È possibile associare un combo box con un ascoltatore di eventi azione. Quando una scelta è fatta nel combo box, il metodo actionPerformed() consente di risalire all'oggetto combo box source che ha generato l'evento. Il metodo getSelectedItem() del combo box consente quindi di stabilire la scelta effettuata dall'utente.
- La classe **JSlider** consente di gestire slider che forniscono all'utente un "continuo" di valori selezionabili spostando lo slider col mouse:

  JSlider slider=new JSlider(min, max, valore_iniziale);

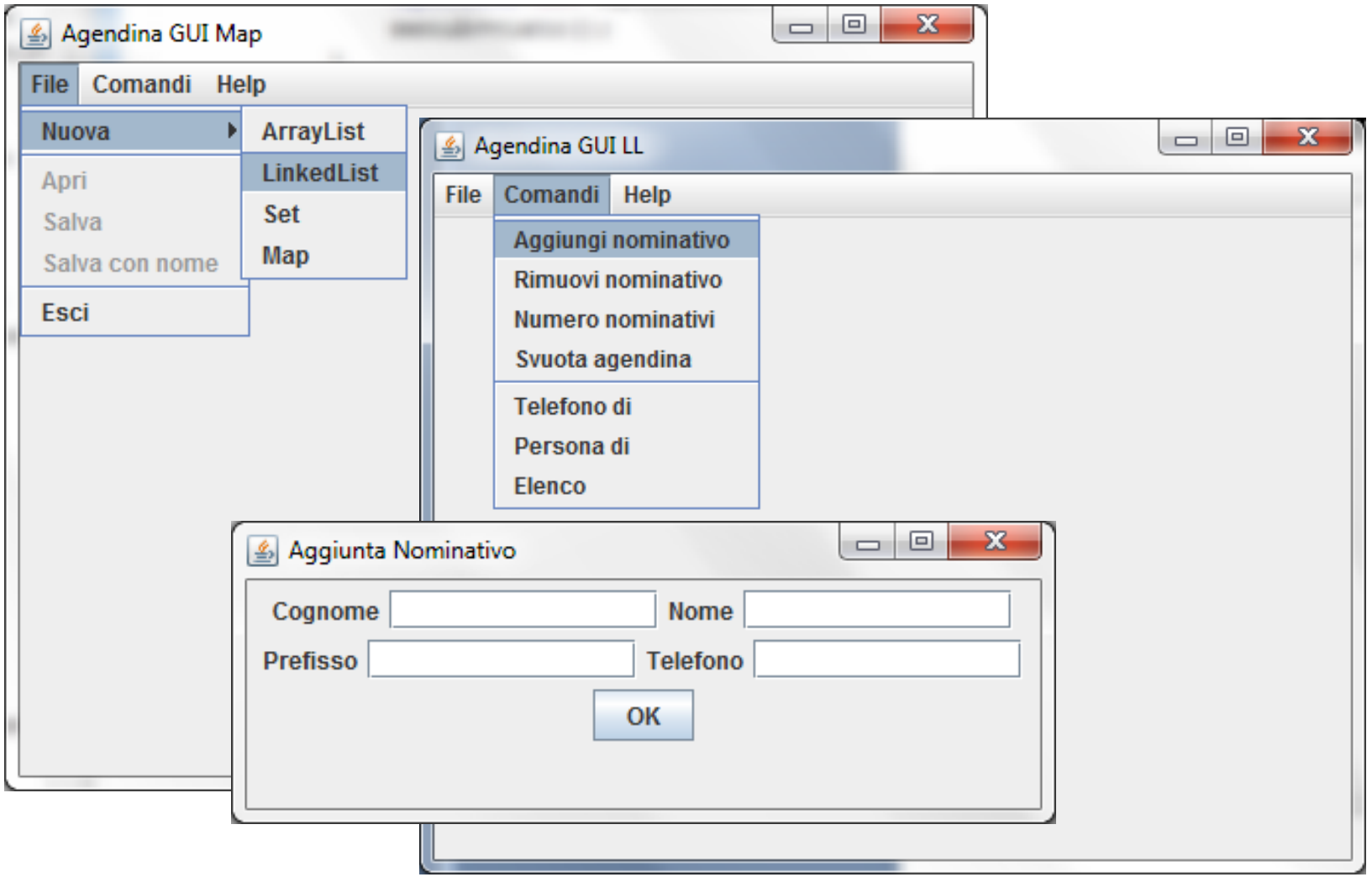
  quando il valore cambia, un ChangeEvent è generato che un ChangeListener è in grado di catturare e gestire. Il metodo getValue() sull'oggetto source slider consente di ottenere il valore.
- In realtà è possibile fissare dei tick di variazione (major e minor tick) sullo slider per cui il valore potrebbe essere "costretto" al tick più vicino (snap to ticks). Per i dettagli si rimanda alle API di Java.

- Sulle finestre (JFrame) si possono costituire barre di menù (oggetti di classe **JMenuBar**); ogni menù (oggetti di classe **JMenu**) è poi un classico menù a tendina, in cui le opzioni (oggetti di classe **JMenuItem**) possono essere scelte col mouse. Un menu item a sua volta può essere un (sotto) menù in modo da creare menù concatenati. È possibile introdurre dei separatori tra sotto insiemi di item di un menù
- Un menù item, quando scelto, genera un ActionEvent, per cui per il corretto funzionamento della GUI è necessario stabilire un ascoltatore di eventi azione (ActionListener) ed associarlo (metodo addActionListener) ai vari JMenuItem. Si può dire che i menù item sono in tutto simili a bottoni che compaiono solo quando si apre il corrispondente menù a tendina. Per esempi di strutture a menù si veda più avanti.
- Mentre un menù è aggiunto ad una menu bar ed occupa una posizione fissa sulla GUI, i pop-up menù (classe **JPopupMenu**) non sono agganciati ad una menù bar e possono essere evocati per la scelta ad es. cliccando col pulsante destro del mouse su una posizione del componente:

  JPopupMenu popup=new JPopupMenu();
  JMenuItem item1=new JMenuItem("Item1");item1.addActionListener(actListener);
  popup.add(item1); // e cosi via per altri item
  component.setComponentPopupMenu(popup); //aggancia il popup a component
  popup.show(parent_component, x, y);

  chiede di mostrare il popup sul parent_component alla posizione indicata del parent component.

Esempio di JFrame (AgendinaGUI.java di poo.agendina)



Cenni al codice Java

Di seguito si riporta una parte del costruttore della classe AgendinaGUI che illustra i dettagli di costituzione del menu File. Per maggiori informazioni si rimanda al codice.

```
package poo.agendina;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

class FinestraGUI extends JFrame{
    private File fileDiSalvataggio=null;
    private JMenuItem          tipoAL, tipoLL, tipoSet, tipoMap,
                               apri, salva, salvaConNome, esci, about,
                               aggiungiNominativo, rimuoviNominativo,
                               numeroNominativi, svuota,
                               telefonoDi, personaDi, elenco;
    private String titolo="Agendina GUI";
    private String impl=" Map "; //default
    …
```

```
public FinestraGUI(){//costruttore
        setTitle(titolo+impl);
        setDefaultCloseOperation( JFrame.DO_NOTHING_ON_CLOSE );
        addWindowListener( new WindowAdapter() {
              public void windowClosing(WindowEvent e){
                   if( consensoUscita() ) System.exit(0);
              }
         } );
    AscoltatoreEventiAzione listener=new AscoltatoreEventiAzione();
    //creazione barra dei menu'
    JMenuBar menuBar=new JMenuBar();
     this.setJMenuBar( menuBar );
        //creazione file menu
    JMenu fileMenu=new JMenu("File"); menuBar.add(fileMenu);
    //creazione voci del menu File
    JMenu tipoImpl=new JMenu("Nuova"); fileMenu.add(tipoImpl);
    tipoAL=new JMenuItem("ArrayList");
    tipoAL.addActionListener(listener); tipoImpl.add(tipoAL);
    tipoLL=new JMenuItem("LinkedList");
    tipoLL.addActionListener(listener); tipoImpl.add(tipoLL);
    tipoSet=new JMenuItem("Set"); tipoSet.addActionListener(listener);
    tipoImpl.add(tipoSet);
    tipoMap=new JMenuItem("Map"); tipoMap.addActionListener(listener);
    tipoImpl.add(tipoMap);
    fileMenu.addSeparator(); //aggiunge una linea di separazione
```

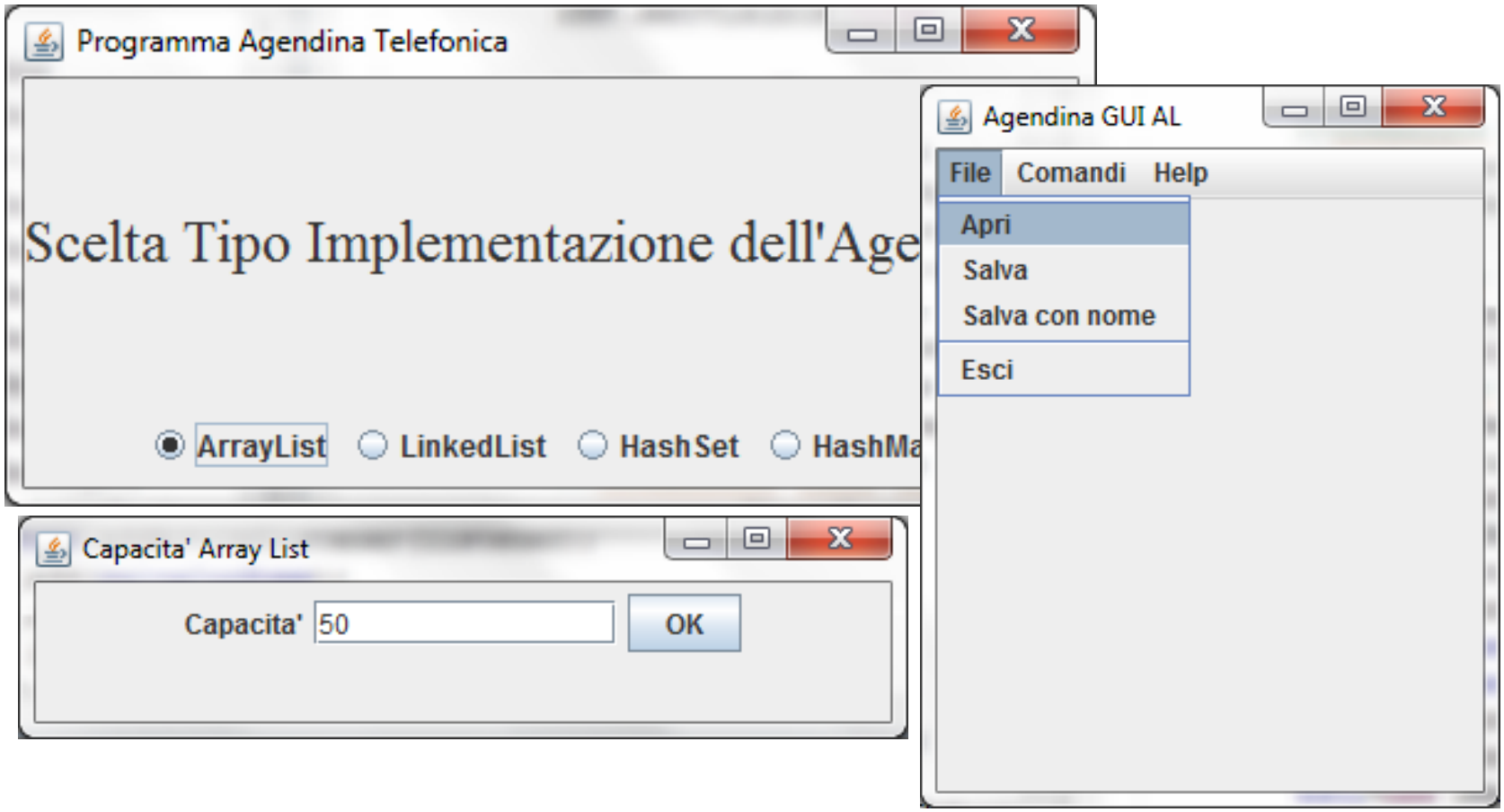
```
        apri=new JMenuItem("Apri");  apri.addActionListener(listener);
        fileMenu.add(apri);
        salva=new JMenuItem("Salva"); salva.addActionListener(listener);
        fileMenu.add(salva);
        salvaConNome=new JMenuItem("Salva con nome");
        salvaConNome.addActionListener(listener);
        fileMenu.add(salvaConNome);
        fileMenu.addSeparator();
        esci=new JMenuItem("Esci"); esci.addActionListener(listener);
        fileMenu.add(esci);
        //creazione menu Comandi
        …
        //creazione menu Help
        …
        pack();
        setLocation(200,200);
        setSize(500,400);
}//costruttore
```

In neretto sono indicate le azioni di “aggancio” ai menu item dell’action listener, qui unico e implementato mediante una inner class della classe finestra

pack components according to their preferred size

Le finestre per il settaggio dei campi dei vari comandi sono create e rese visibili quando strettamente necessario

Altro esempio
(AgendinaGUI2.java di poo.agendina)



Altre letture

- I concetti della programmazione in Java di GUI possono essere approfonditi su:
  - Cay Horstmann, G. Cornell, *Core Java*, Vol. I Fundamentals, Vol. II Advanced Features, 11th Edition, Pearson, Prentice Hall, 2018/2019