Interfacce grafiche (GUI)

AWT e Swing

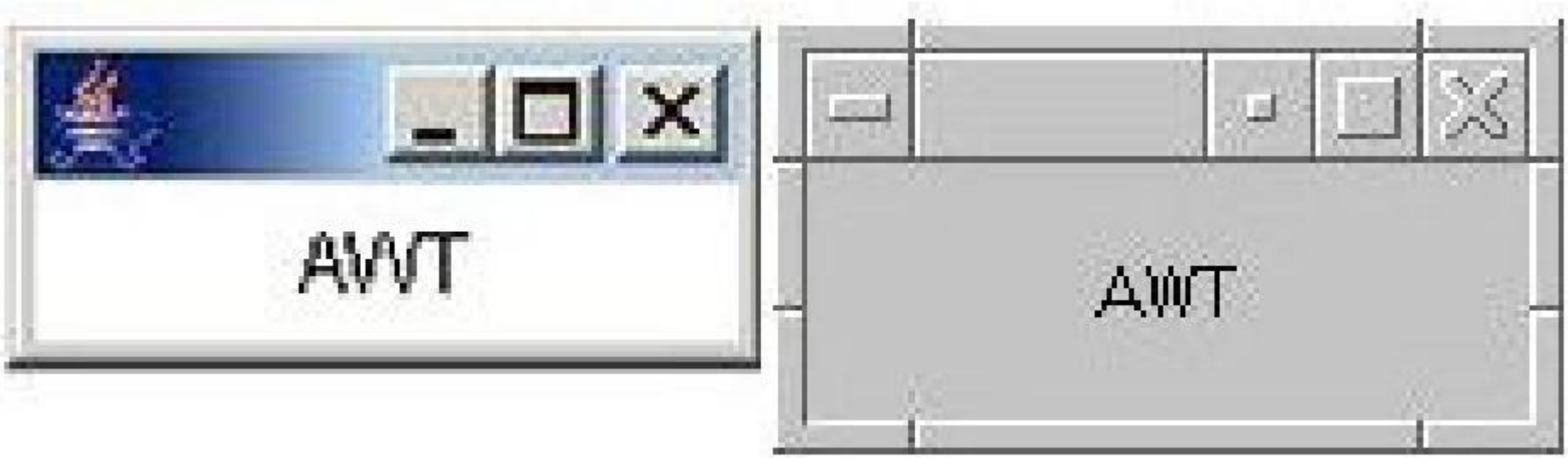
In questo lezione vedremo come Java permette di creare programmi con una interfaccia grafica.

Quando si dota un programma di una GUI, cambia completamente il ciclo di vita del programma, infatti, mentre tutte le applicazioni sviluppate fino ad adesso, duravano il "tempo di eseguire un main()", adesso una volta che viene visualizzata una GUI, la JVM fa partire un nuovo thread che si chiama "AWT thread", mantiene sullo schermo la GUI stessa, cattura eventuali eventi su di essa.

Quindi un'applicazione che fa uso di GUI, una volta lanciata, rimane in attesa dell'input dell'utente e termina solo in base a un determinato input.

Abstract Window Toolkit (AWT)

AWT è una libreria per creare interfacce grafiche utente che sfruttano componenti dipendenti dal sistema operativo, e ciò significa che eseguendo la stessa applicazione grafica su sistemi operativi differenti, lo stile dei componenti grafici (“Look & Feel”) sarà imposto dal sistema operativo.



Il codice per generare la GUI precedente è il seguente:

```
import java.awt.*;
public class AWTGUI {
public static void main(String[] args) {
Frame frame = new Frame();
Label l = new Label("AWT", Label.CENTER);
frame.add(l);
frame.pack();
frame.setVisible(true);
}
}
```

La libreria AWT offre una serie abbastanza ampia di classi ed interfacce
Per la creazione di GUI.

È possibile utilizzare bottoni, checkbox, liste, combo box (classe Choice), label, radio button (utilizzando checkbox raggruppati mediante la classe CheckboxGroup), aree e campi di testo, scrollbar, finestre di dialogo (classe Dialog), finestre per navigare sul file system (classe FileDialog).

Per esempio il seguente codice crea un'area di testo con testo iniziale "Java AWT", 4 righe, 10 colonne e con la caratteristica di andare da capo automaticamente.

```
import java.awt.*;
public class AWTGUI {
public static void main(String[] args) {
Frame frame = new Frame();
TextArea ta = new TextArea("Java AWT",
4,10,TextArea.SCROLLBARS_VERTICAL_ONLY);
frame.add(ta);
frame.pack();
frame.setVisible(true);}}
```

È molto semplice creare anche menu personalizzati tramite le classi MenuBar, Menu, MenuItem, CheckboxMenuItem ed eventualmente MenuShortcut per utilizzarli direttamente con la tastiera mediante le cosiddette “scorciatoie”.

```
Frame f = new Frame("MenuBar");
MenuBar mb = new MenuBar();
Menu m1 = new Menu("File");
Menu m2 = new Menu("Edit");
Menu m3 = new Menu("Help");
mb.add(m1);mb.add(m2);
MenuItem mi1 = new MenuItem("New");
MenuItem mi2 = new MenuItem("Open");
MenuItem mi3 = new MenuItem("Save");
MenuItem mi4 = new MenuItem("Quit");
m1.add(mi1);m1.add(mi2);m1.add(mi3);
m1.addSeparator();m1.add(mi4);
mb.setHelpMenu(m3);f.setMenuBar(mb);
```

La classe Toolkit ci permette di accedere a varie caratteristiche grafiche e non grafiche, del sistema si cui ci troviamo, tra cui:

- getScreenSize() restituisce la dimensione del nostro schermo
- getPrintJob() che offre il supporto per la stampa .

Per ottenere un oggetto Toolkit possiamo utilizzare il metodo getDefaultToolkit()

```
Toolkit toolkit = Toolkit.getDefaultToolkit();
```

AWT ci offre anche la possibilità di utilizzare i font di sistema o personalizzati, tramite la classe Font ad esempio quando deve stampare un file, EJE utilizza il seguente Font:

```
Font font = new Font("Monospaced", Font.BOLD, 14);
```

Con AWT è anche possibile disegnare, infatti ogni Component può essere esteso e si può overridare il metodo paint(Graphics g), che disegna il componente stesso.

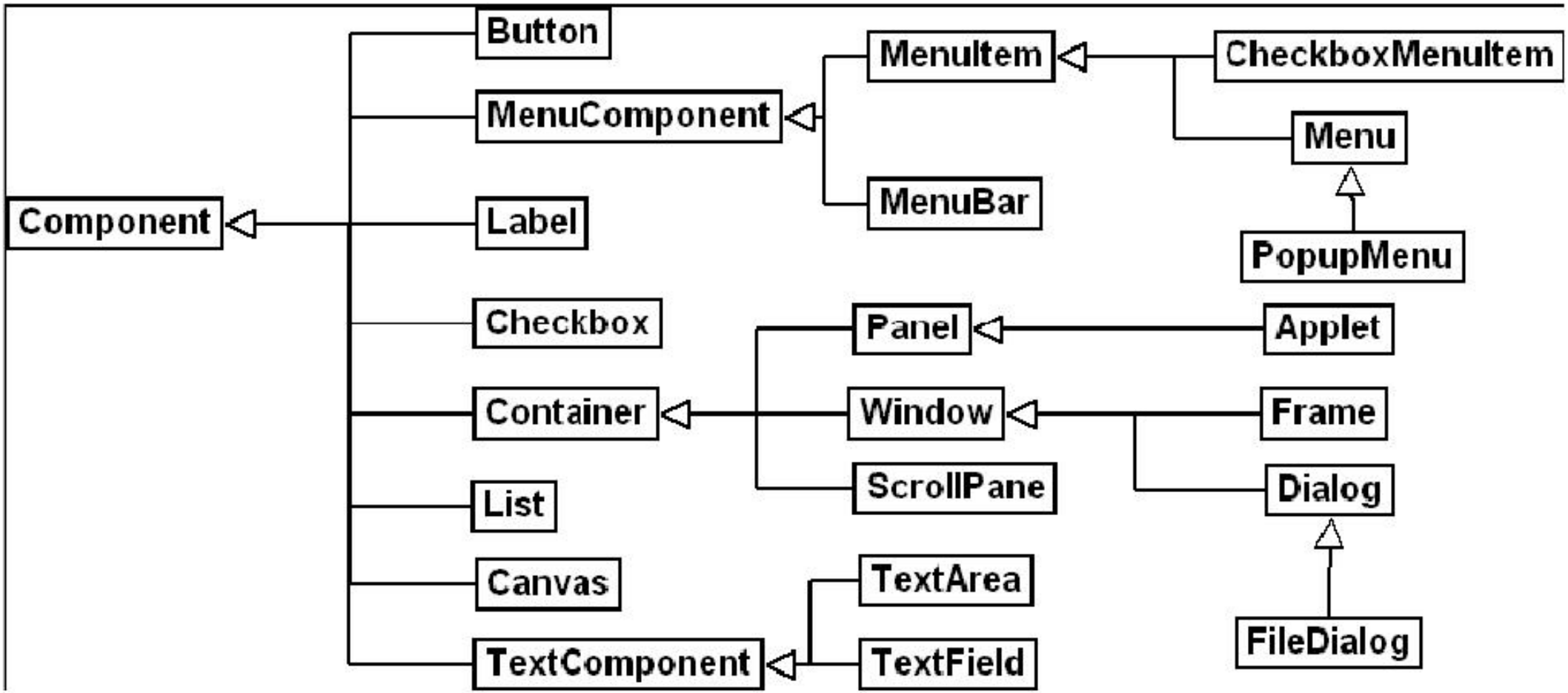
In particolare, la classe Canvas (in italiano “tela”), è stata creata per disegnare linee ovali, rettangoli, poligoni etc, ad esempio:

```
public class MyCanvas extends Canvas {
public void paint(Graphics g) {
g.drawString("java",10,10);
g.setColor(Color.red);
g.drawLine(10,5, 35,5);
}}
```

Con la classe Color si possono anche creare colori ad hoc con lo standard RGB:

```
Color c = new Color (255, 10 ,110 );
```

Riepilogando queste sono le classi AWT



La classe astratta Component astrae il concetto di componente generico e le classi **Button, Label, Checkbox, List** e **Canvas** sono sottoclassi dirette di Component.

La classe **Container**, sottoclasse di Component, definisce il concetto di componente grafico contenente altri componenti, essa non è una classe astratta, e solitamente si utilizzano le sue sottoclassi Frame e Panel.

Le applicazioni grafiche Java si basano sempre su di un "top level container", quindi è sempre necessario istanziare un top level container che di solito è un Frame.

Il metodo chiave delle classi container è il metodo **add(Component c)**, che permette di aggiungere altri componenti ma anche altri container.

La posizione di un componente aggiunto a un container dipende **"layout manager"** e i principali layout manager sono :

1. FlowLayout
2. BorderLayout
3. GridLayout
4. GridBagLayout
5. CardLayout

FlowLayout è il layout manager di default di Panel, e dispone i componenti aggiunti in un flusso ordinato che va da sinistra a destra con un allineamento centrato verso l'alto.

Ad esempio il codice seguente

```
import java.awt.*;
public class AWTGUI {
public static void main(String[] args) {
Frame f = new Frame("FlowLayout");
Panel p = new Panel();
Button button1 = new Button("Java");
Button button2 = new Button("Windows");
Button button3 = new Button("Motif");
p.add(button1);p.add(button2);p.add(button3);
f.add(p);f.pack();
f.setVisible(true);}}
```

Genera questo output

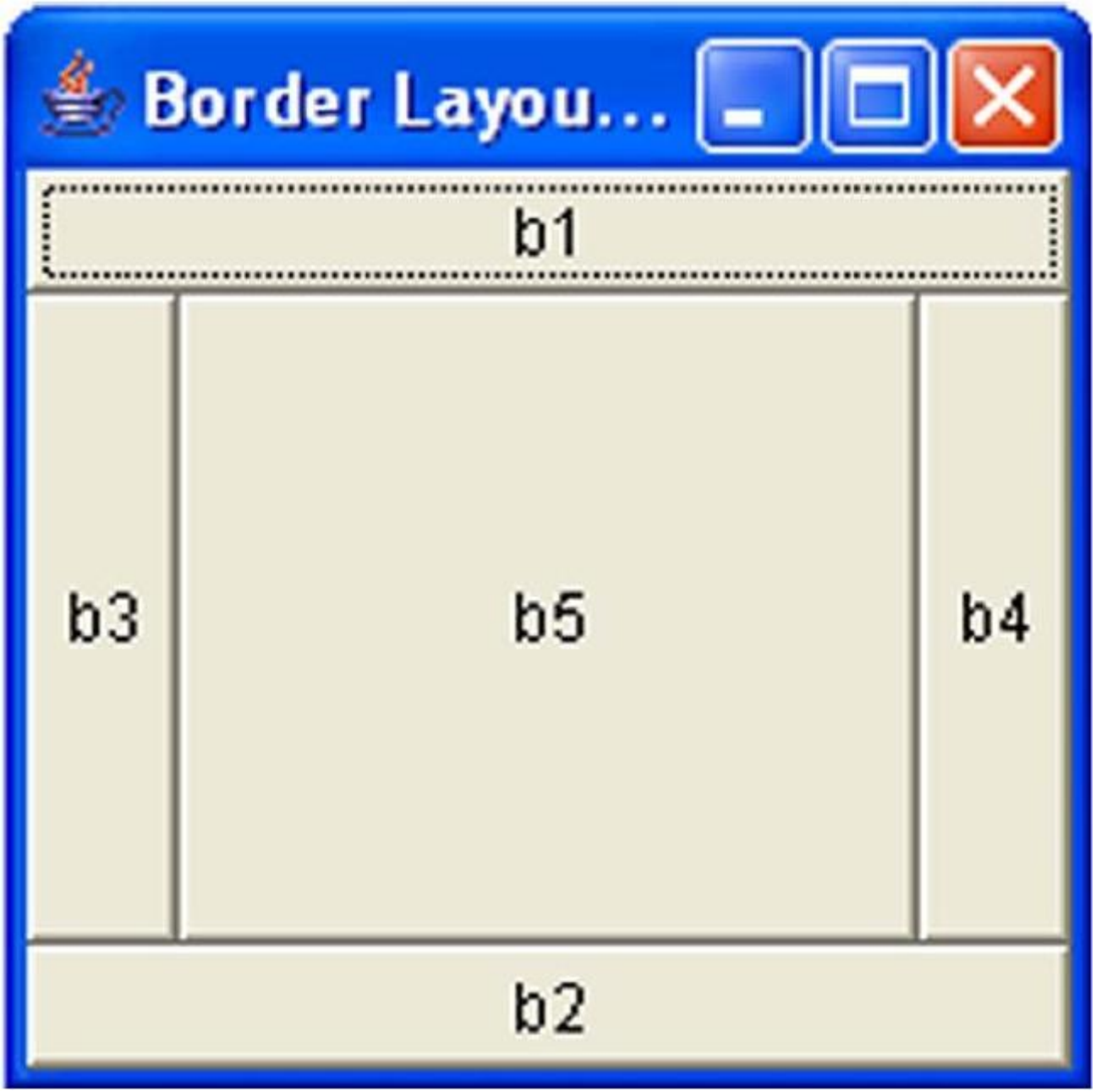
BorderLayout è il layout manager di default per i Frame,e i componenti si dispongono in cinque posizioni predefinite ridimensionandosi automaticamente:

- NORTH, SOUTH che si ridimensionano orizzontalmente
- EAST, WEST che si ridimensionano verticalmente
- CENTER che si ridimensiona orizzontalmente e verticalmente

I componenti con il BorderLayout si aggiungono utilizzano i metodi add(Component c,int position) o add(Component c,String position) e se si

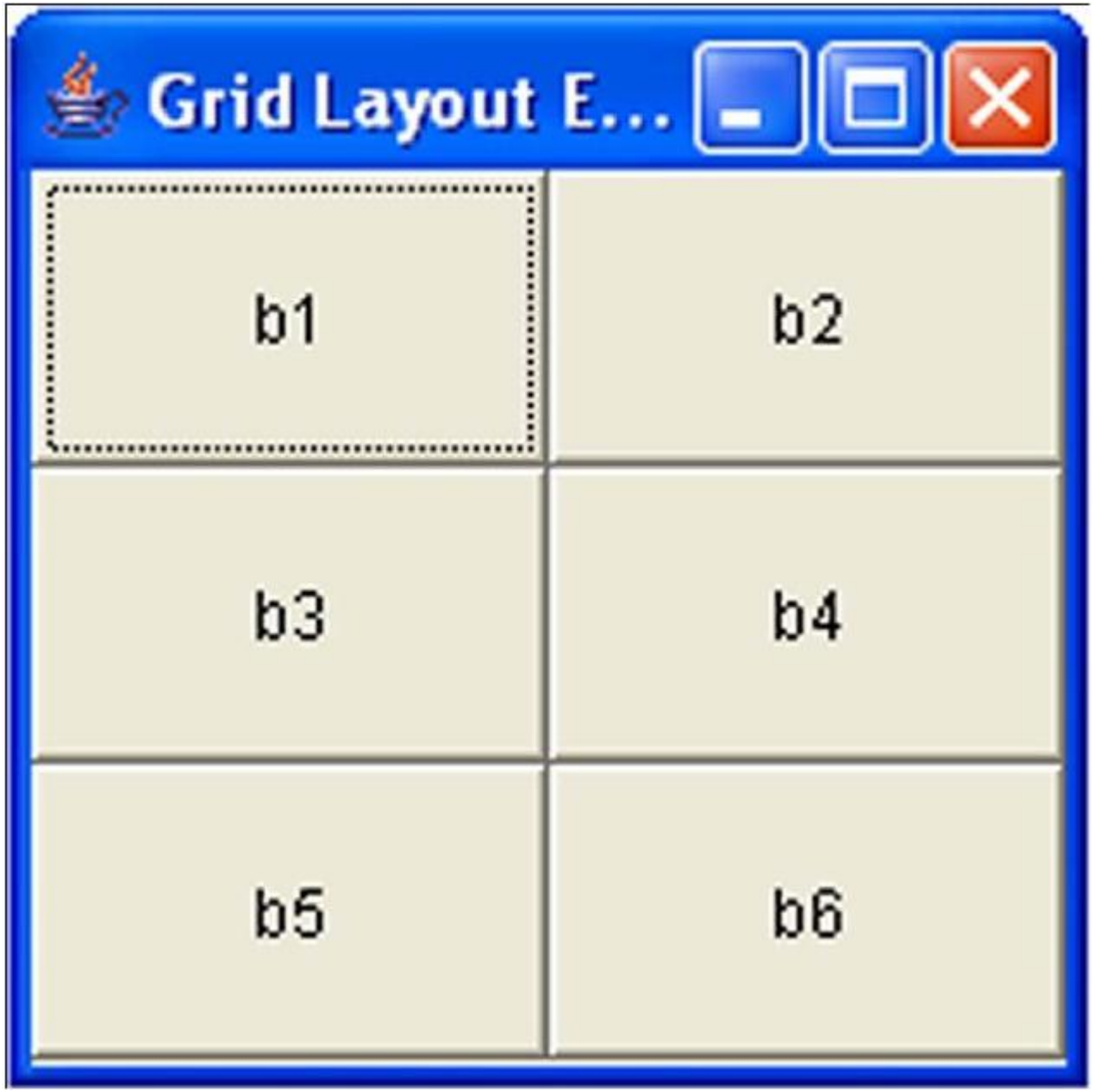
utilizza il metodo add(Component c), il componente verrà aggiunto al centro dal BorderLayout, e questo significa che un componente aggiunto in una certa area, si deformerà per occupare l'intera area.

```
public class BorderExample {
private Frame f;
private Button b[]={new Button("b1"),new Button("b2"),
  new Button("b3"),new Button("b4"), new Button("b5")};
public BorderExample() {f = new Frame("Border Layout Ex");}
public void setup() {
f.add(b[0], BorderLayout.NORTH);
f.add(b[1], BorderLayout.SOUTH);
f.add(b[2], BorderLayout.WEST);
f.add(b[3], BorderLayout.EAST);
f.add(b[4], BorderLayout.CENTER);
f.setSize(200,200);f.setVisible(true);}
public static void main(String args[]) {
new BorderExample().setup();}}
```


Border Layou...
b1
b3
b5
b4
b2

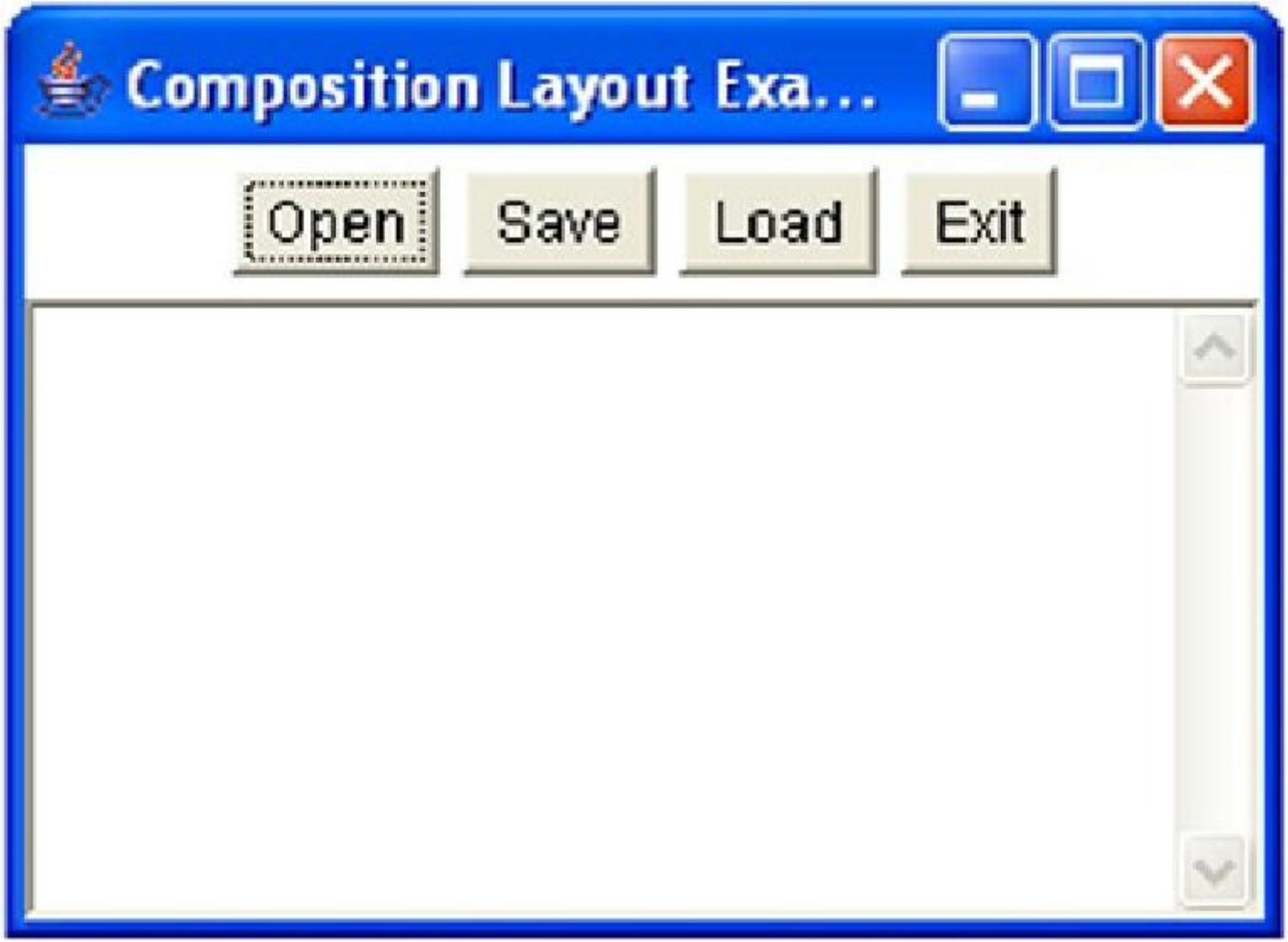
GridLayout dispone i componenti da sinistra verso destra e dall'alto verso il basso all'interno di una griglia, le regioni della griglia hanno la stessa dimensione e i componenti occupano tutto lo spazio possibile, nel costruttore del GridLayout si può specificare riga e colonna della griglia.

```
public class GridExample {
private Frame f;
private Button b[]=
{new Button("b1"),new Button("b2"), new Button("b3"),
 new Button("b4"), new Button("b5"), new Button("b6")};
public GridExample() {f = new Frame("Grid Layout Example");}
public void setup() {
f.setLayout(new GridLayout(3,2));
for (int i=0; i<6;++i) f.add(b[i]);
f.setSize(200,200);f.setVisible(true);}
public static void main(String args[]) {
new GridExample().setup();}}
```

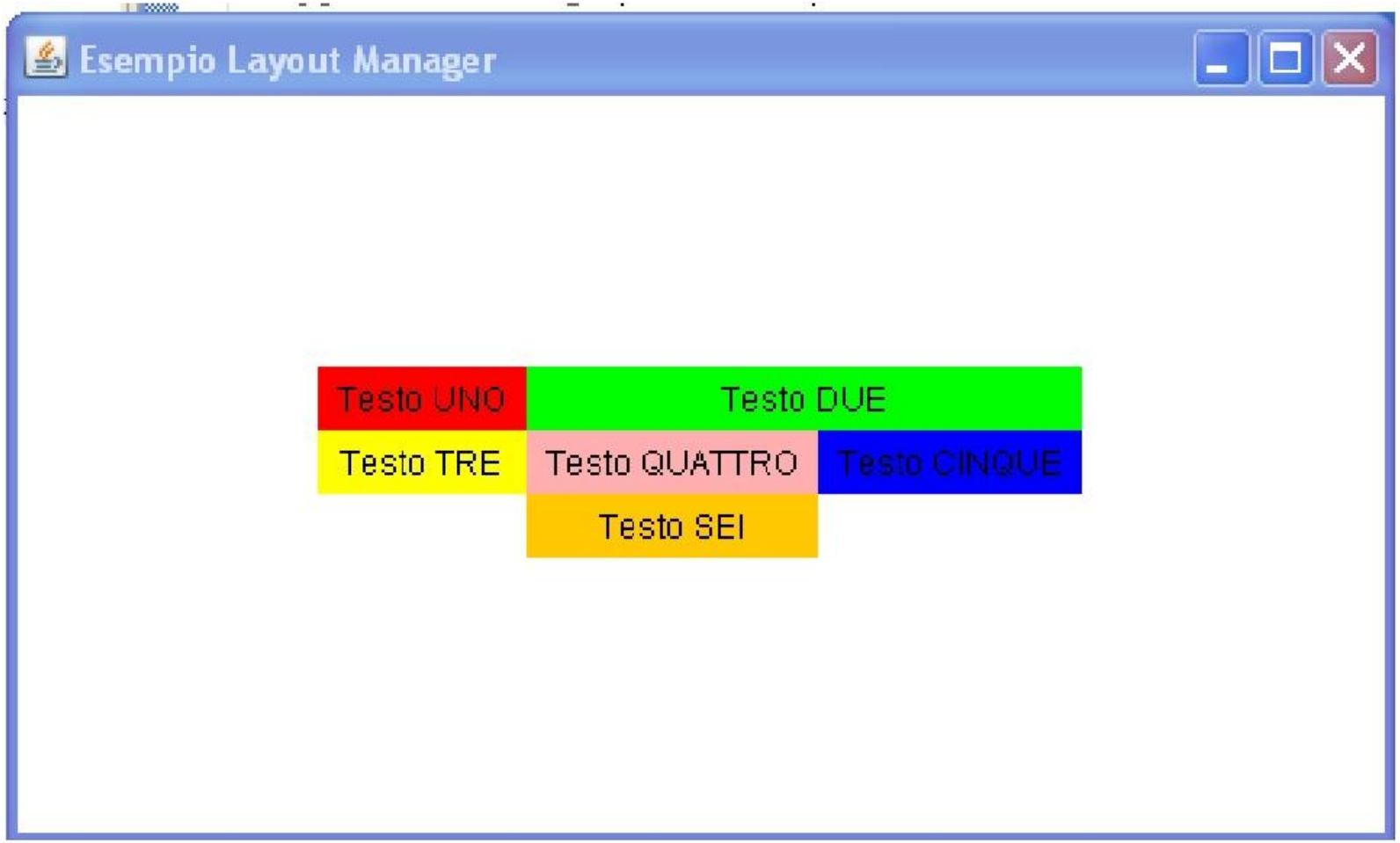

Grid Layout E...
b1
b2
b3
b4
b5
b6

Per creare GUI con layout complessi è possibile comporre vari layout.

```
public class CompositionExample {
 private Frame f;
 private TextArea ta;
 private Panel p;
 private Button b[]={new Button("Open"),new Button("Save"), new
Button("Load"), new Button("Exit")};
 public CompositionExample() {
  f = new Frame("Composition Layout ");
  p = new Panel(); ta = new TextArea();}
 public void setup() {
  for (int i=0; i<4;++i) p.add(b[i]);
  f.add(p,BorderLayout.NORTH);f.add(ta,BorderLayout.CENTER);
  f.setSize(350,200); f.setVisible(true);}
public static void main(String args[]) {
new CompositionExample().setup();}}
```


Composition Layout Exa...
Open
Save
Load
Exit

GridBagLayout può organizzare interfacce grafiche complesse da solo essendo anch'esso è capace di dividere il container in una griglia, ma, a differenza del GridLayout, può disporre i suoi componenti in modo tale che si estendano anche oltre un'unica cella, attraverso la classe GridBagConstraints.



CardLayout è un layout manager particolare che permetterà di posizionare i vari componenti uno sopra l'altro, come le carte in un mazzo.

```
public class CardTest {
private Panel p1, p2, p3;
private Label lb1, lb2, lb3;
private CardLayout cardLayout;
private Frame f;
public CardTest() {
f = new Frame ("CardLayout");
cardLayout = new CardLayout();
p1 = new Panel();p2 = new Panel();p3 = new Panel();
lb1=new Label("Primo pannello rosso");p1.setBackground(Color.red);
lb2=new Label("Secondo pannello verde");
p2.setBackground(Color.green);
lb3 = new Label("Terzo pannelloblue");p3.setBackground(Color.blue);}
public void setup()
{f.setLayout(cardLayout);p1.add(lb1);p2.add(lb2);p3.add(lb3);
```

```
f.add(p1, "uno");f.add(p2, "due");f.add(p3, "tre");
cardLayout.show(f, "uno");f.setSize(200,200);f.setVisible(true);}
}
private void slideShow() {
while (true) {
try {Thread.sleep(3000);
      cardLayout.next(f);}
catch (InterruptedException exc) {exc.printStackTrace();}}
}
public static void main (String args[]) {
CardTest cardTest = new CardTest();
cardTest.setup();cardTest.slideShow();}}
```

In pratica tre pannelli vengono aggiunti sfruttando un CardLayout, e ad ogni panel viene assegnato un alias (“uno”, “due”, e “tre”).

Viene settato il primo pannello da visualizzare con l'istruzione: cardLayout.show(f, "uno"); e dopo aver visualizzato la GUI, viene invocato il metodo slideShow() che tramite il metodo next(), mostra con intervalli di tre secondi i vari panel usando il metodo sleep() della class Thread.

Gestione degli eventi

Per gestione degli eventi, si intende la possibilità di associare l'esecuzione di una parte di codice, in corrispondenza ad un certo evento sulla GUI.

La gestione degli eventi in Java viene detta a "modello a delega", noto come "Observer", e con il modello a delega, esistono almeno tre oggetti per gestire gli eventi su una GUI:

1) il componente sorgente dell'evento ("event source")
2) l'evento stesso
3) il gestore dell'evento, detto "listener"

Ad esempio se si vuole che appaia una scritta al click di un bottone allora:

1)il bottone è la sorgente dell'evento
2)l'evento è il click del bottone che sarà un oggetto ActionEvent
3)il gestore dell'evento sarà un oggetto dell'interfaccia ActionListener e questa ridefinirà il metodo actionPerformed(ActionEvent ev) con il quale si gestisce l'evento.

JVM invocherà automaticamente questo metodo su quest'oggetto, quando l'utente premerà il bottone.

```
import java.awt.*;
public class DelegationModel {
private Frame f;
private Button b;
public DelegationModel() {
f = new Frame("Delegation Model");
b = new Button("Press Me");}
public void setup() {
b.addActionListener(new ButtonHandler());
f.add(b,BorderLayout.CENTER);
f.pack();f.setVisible(true);}
public static void main(String args[]) {
DelegationModel delegationModel = new DelegationModel();
delegationModel.setup();}}
```

L'istruzione b.addActionListener(new ButtonHandler()); fa in modo di comunicare alla JVM su quale oggetto di tipo Listener chiamare il metodo actionPerformed() per il bottone b.

Il metodo addActionListener(), si aspetta come parametro un oggetto di tipo ActionListener ed essendo ActionListener un'interfaccia, questo significa che si aspetta un oggetto istanziato da una classe che implementa tale interfaccia. La classe che gestisce l'evento (il listener) è la seguente:

```
import java.awt.*;
public class ButtonHandler implements ActionListener {
public void actionPerformed(ActionEvent e) {
System.out.println("È stato premuto il bottone");
System.out.println("E la sua etichetta è: "
+ e.getActionCommand());
}}
```

ogni volta che viene premuto il bottone viene stampato la sua etichetta ossia Press Me!!.

Altro esempio che Stampa la frase su un oggetto Label della stessa GUI:

```
import java.awt.*;
public class DelegModel1 {
private Frame f;
private Button b;
private Label l;
public DelegModel1() {
f = new Frame("Delegation Model 1");
b = new Button("Press Me");
l = new Label();}
public void setup() {
b.addActionListener(new TrueButtonHandler(l));
f.add(b,BorderLayout.CENTER); f.add(l,BorderLayout.SOUTH);
f.pack();f.setVisible(true);}
public static void main(String args[]) {
DelegModel1 delegModel1 = new DelegModel1();
delegationModel.setup();}}
```

Si Noti che quando viene istanziato l'oggetto listener TrueButtonHandler, viene passato al costruttore la label.
La classe che gestisce l'evento (il listener) è la seguente:

```
import java.awt.event.*;
import java.awt.*;
public class TrueButtonHandler implements ActionListener
{private Label l;
 private int counter;
 public TrueButtonHandler(Label l) {
 this.l = l;}
public void actionPerformed(ActionEvent e) {
l.setText(e.getActionCommand() + " - " + (++counter));
}
}
```

Lo stesso codice può essere scritto attraverso le classi innestate ossia con una classe definita all'interno di un'altra classe.

```
import java.awt.*;
import java.awt.event.*;
public class InnerDelegationModel {
private Frame f;
private Button b;
private Label l;
public InnerDelegationModel() {
f = new Frame("Delegation Model");
b = new Button("Press Me");
l = new Label();
}
public void setup() {
b.addActionListener(new InnerButtonHandler());
f.add(b,BorderLayout.CENTER);
f.add(l,BorderLayout.SOUTH);
```

```
f.pack();
f.setVisible(true);
}
public class InnerButtonHandler implements ActionListener {
private int counter;
public void actionPerformed(ActionEvent e) {
l.setText(e.getActionCommand() + " - " +
(++counter));
}
}
public static void main(String args[]) {
InnerDelegationModel delegationModel = new
InnerDelegationModel();
delegationModel.setup();
}
}
```

Una soluzione ancora più potente è rappresentata dall'utilizzo di una classe anonima per implementare il gestore dell'evento:

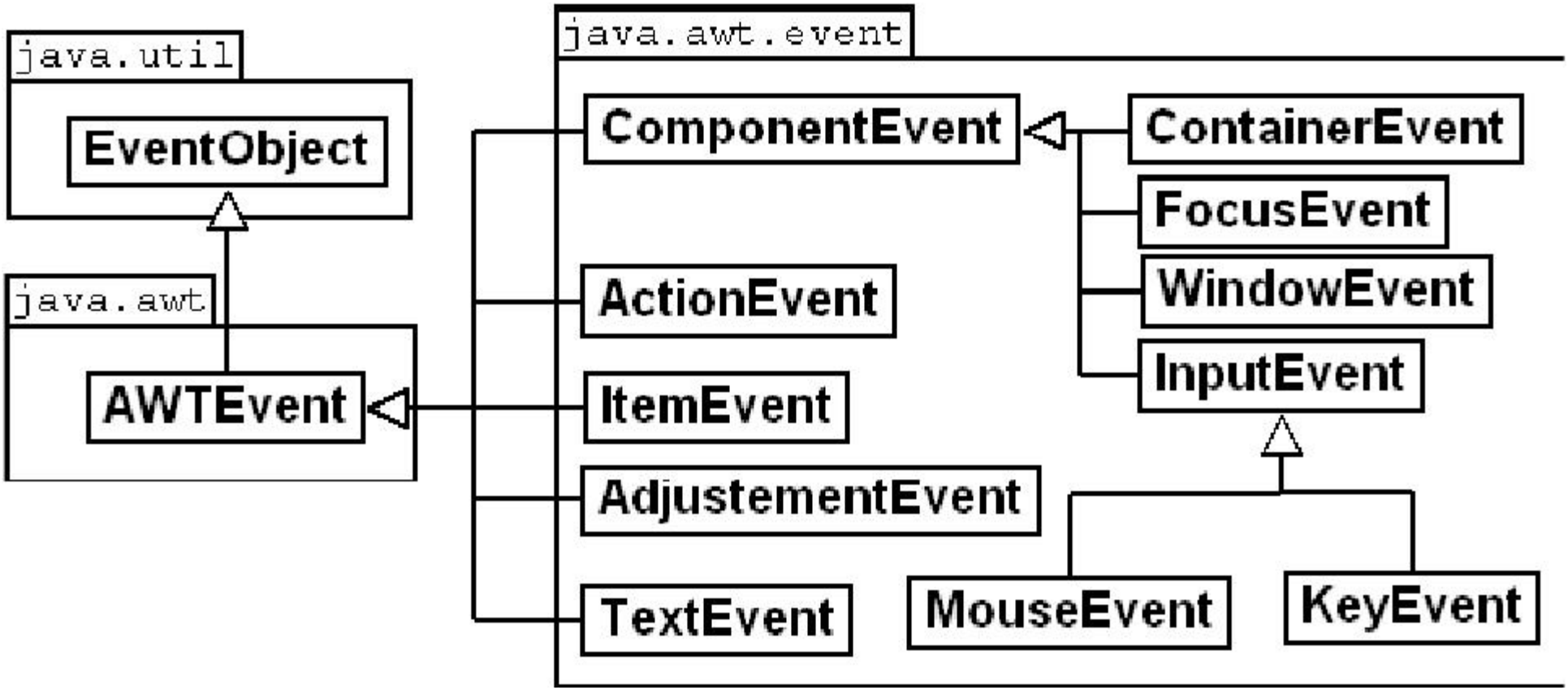
```
import java.awt.*;
import java.awt.event.*;
public class AnonymousDelegationModel {
private Frame f;
private Button b;
private Label l;
public AnonymousDelegationModel() {
f = new Frame("Delegation Model");
b = new Button("Press Me");
l = new Label();
}
public void setup() {
b.addActionListener( new ActionListener() {
    private int counter;
    public void actionPerformed(ActionEvent e) {
```

```
    l.setText(e.getActionCommand() + " - "+(++counter));}
  }
 );
f.add(b,BorderLayout.CENTER);
f.add(l,BorderLayout.SOUTH);
f.pack();
f.setVisible(true);}

public static void main(String args[]) {
AnomynousDelegationModel delegationModel = new
AnomynousDelegationModel();
delegationModel.setup();
}
}
```

La classe anonima ha una sintassi sicuramente più compatta.

Esistono naturalmente vari tipi di eventi che possono essere generati dai componenti e dai container di una GUI. Esiste nella libreria una gerarchia di classi di tipo evento che viene qui riportata sommariamente.



Evento	Descrizione	Interfaccia	Metodi
ActionEvent	Azione (generica)	ActionListener	actionPerformed
ItemEvent	Selezione	ItemListener	itemStateChanged
MouseEvent	Azioni effettuate con il mouse	MouseListener	mousePressed mouseReleased mouseEntered mouseExited mouseClicked
MouseEvent	Movimenti del mouse	MouseMotionListener	mouseDragged mouseMoved
KeyEvent	Pressione di tasti	KeyListener	keyPressed keyReleased keyTyped
WindowEvent	Azioni effettuate su finestre	WindowListener	windowClosing windowOpened windowIconified windowDeiconified windowClosed windowActivated windowDeactivated

Come primo esempio vediamo come chiudere un'applicazione grafica, chiudendo il frame principale. Se f è il reference del frame principale, il seguente codice, implementa una classe anonima che definendo il metodo windowClosing() , permette all'applicazione di terminare l'applicazione:

```
f.addWindowListener( new WindowListener() {
public void windowClosing (WindowEvent ev) {
System.exit(0);}
public void windowClosed (WindowEvent ev) {}
public void windowOpened (WindowEvent ev) {}
public void windowActivated (WindowEvent ev) {}
public void windowDeactivated (WindowEvent ev) {}
public void windowIconified (WindowEvent ev) {}
public void windowDeiconified (WindowEvent ev) {}} );
```

purtroppo, implementando l'interfaccia WindowListener, si ereditano ben 7 metodi e anche se ne utilizza uno, si devono comunque riscriverli tutti, questo problema, è mitigato dall'esistenza delle classi dette "Adapter", classi che implementano un listener, riscrivendo ogni metodo ereditato

senza codice applicativo. Per esempio la classe WindowAdapter è implementata come segue:

```
public abstract class WindowAdapter implements
WindowListener {
public void windowClosing (WindowEvent ev) {}
public void windowClosed (WindowEvent ev) {}
public void windowOpened (WindowEvent ev) {}
public void windowActivated (WindowEvent ev) {}
public void windowDeactivated (WindowEvent ev) {}
public void windowIconified (WindowEvent ev) {}
public void windowDeiconified (WindowEvent ev) {}}
```

Quindi se invece di implementare l'interfaccia listener si estende la classe adapter, non dobbiamo riscrivere tutti i metodi ereditati e la nostra classe anonima diventerà molto più compatta:

```
f.addWindowListener( new WindowAdapter() {
public void windowClosing (WindowEvent ev) {
System.exit(0);}} );
```

La classe Applet

Le applet danno la possibilità di lanciare le nostre applicazioni direttamente da pagine web.

In pratica un'applet è un applicazione Java, che può essere direttamente linkata ad una pagina HTML, mediante un tag speciale: il tag "<APPLET>".

Un'applet per definizione deve estendere la classe Applet del package java.applet e ne eredita i metodi e può ovverridarli, l'applet non ha un metodo main(), ma i metodi ereditati sono invocati direttamente dalla JVM del browser.

Swing

Swing fa parte di un gruppo di librerie note come **Java Foundation Classes (JFC)** che oltre a Swing, includono:

1) **Java 2D**: una libreria che permette agli sviluppatori di incorporare grafici di alta qualità, testo, effetti speciali ed immagini all'interno di applet ed applicazioni.
2) **Accessibility**: una libreria che permette a strumenti diversi dai soliti monitor (per esempio schermi Braille) di accedere alle informazioni sulle GUI
3) **Supporto al Drag and Drop** (DnD): una serie di classi che permettono di gestire il trascinamento dei componenti grafici.
4) **Supporto al Pluggable Look and Feel**: offre la possibilità di cambiare lo stile delle GUI che utilizzano Swing.

I componenti AWT sono stati forniti già dalla prima versione di Java (JDK 1.0), mentre Swing è stata inglobata come libreria ufficiale solo dalla versione 1.2 in poi.

La Sun raccomanda fortemente l'utilizzo di Swing piuttosto che di AWT nelle applicazioni Java. Swing ha infatti molti "pro" ed un solo "contro" rispetto ad AWT.

A differenza di AWT, Swing non fa chiamate native al sistema operativo dirette per sfruttarne i componenti grafici, bensì li ricostruisce da zero. Ovviamente, questa caratteristica di Swing, rende AWT nettamente più veloce.

Le classi di Swing quindi, sono molto più complicate di quelle di AWT, e permettono di creare interfacce grafiche senza nessun limite di fantasia.

Le classi di Swing si distinguono da quelle di AWT principalmente perché i loro nomi iniziano con una " J". Ad esempio, la classe di Swing equivalente alla classe Button di AWT si chiama JButton , il package di riferimento non è più java.awt ma javax.swing.

Ci sono alcune situazioni in cui per ottenere un componente Swing equivalente a quello AWT non basterà aggiungere una J davanti al nome AWT.

Ad esempio la classe equivalente a Choice di AWT si chiama JComboBox in Swing, e l'equivalente di Checkbox è JCheckBox con la "B" maiuscola.

Una classe importante è la Class JOptionPane. Tramite questa classe si può creare una finestra di dialogo che permette agli utenti di immettere dati o li informa di qualcosa.

Es: String s = JOptionPane.*showInputDialog("Immetti il tuo nome")* crea una finestra di dialogo su cui appare scritto "Immetti il tuo nome" e uno spazio per scrivere in cui quello che sarà scritto sarà assegnato alla variabile String s.

Es: JOptionPane.*showMessageDialog(null,"Il tuo nome è" + s)* apre una finestra in cui apparirà la stringa data come secondo parametro.

Esiste anche una terza libreria grafica non ufficiale che fu sviluppata originariamente da IBM e poi donata al mondo Open Source: la **SWT** (the Standard Widget Toolkit).

È una libreria sviluppata in C++ altamente performante e dallo stile piacevole, che bisogna inglobare nelle nostre applicazioni ed essendo scritta in C++, esistono versioni diverse per sistemi operativi diversi.

Un esempio di utilizzo di SWT, è l'interfaccia grafica di Eclipse