

11. Interfaccia grafica

Ecco una parte della gerarchia delle classi predefinite in Java che costituiscono i componenti principali dell'interfaccia grafica.

Queste classi sono definite in libreria, altrimenti dette **package**. Per poterle usare in un file è necessario includere il package. Questo deve essere fatto all'inizio del file. L'istruzione per includere un package è:

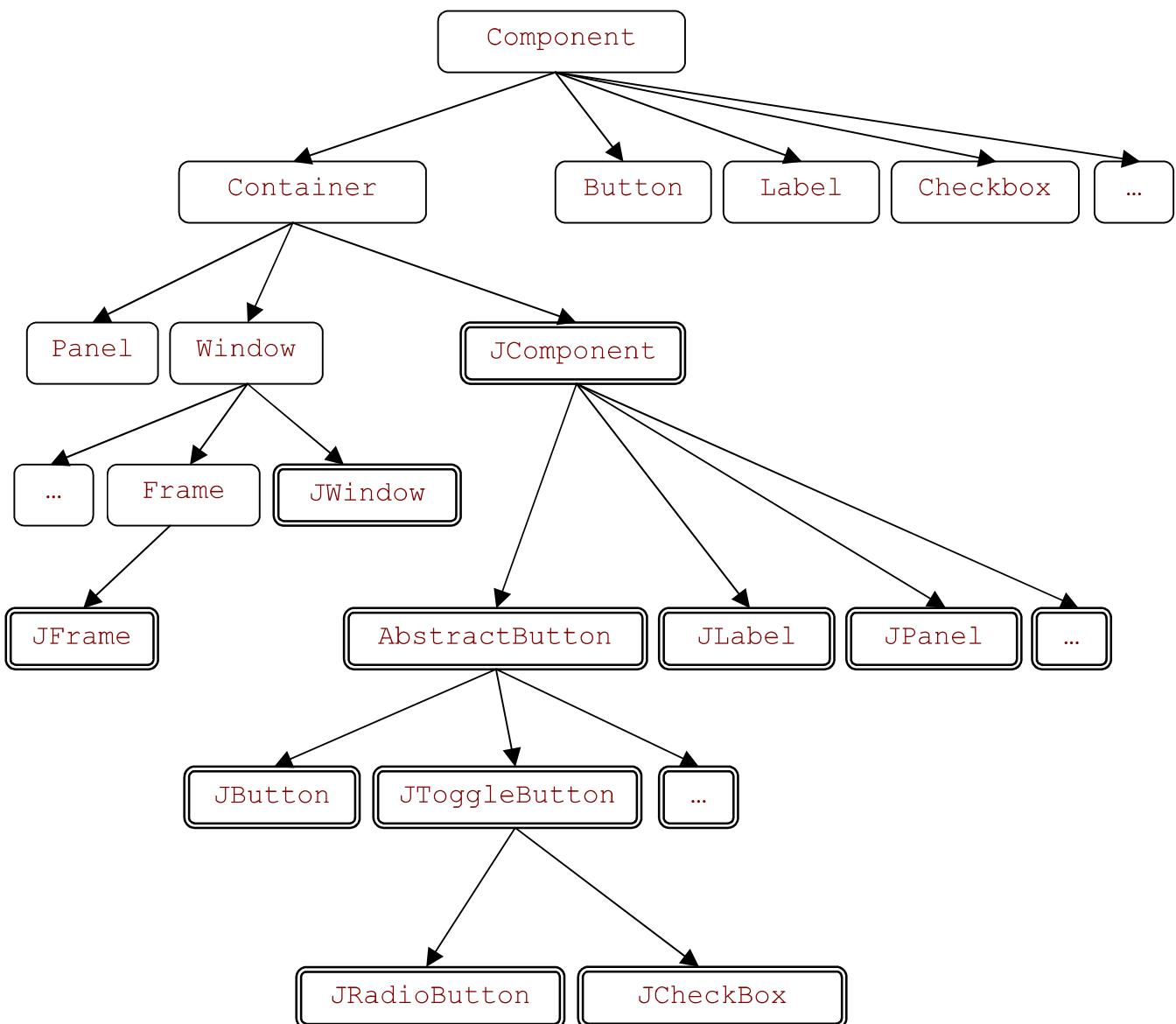
```
include nomepackage.nomeclassadaincludere;
```

oppure

```
include nomepackage.*;
```

se si vogliono includere tutte le classi del package.

Per quanto riguarda il disegno, le classi con il bordo semplice sono contenute nel package `java.awt` mentre le classi con il bordo doppio sono contenute nel package `javax.swing`



12. Visualizzare una finestra **JFrame**.

La classe **JFrame** descrive una finestra di quelle che vengono normalmente usate come **finestre principali** delle applicazioni. Si tratta cioè di una finestra che ha un **bordo** che permette di ridimensionarla, e una **barra del titolo** con i pulsanti per chiuderla, minimizzarla e ripristinarla.

Tramite la classe **JFrame** possiamo **creare** una finestra e **visualizzarla** sullo schermo. Vediamo come.

```
import javax.swing.*;

class ProgrammaX1
{
    public static void main(String[] args)
    {
        JFrame mainFrame = new JFrame("La mia finestra");
        mainFrame.setSize(300, 300);
        mainFrame.show();
    }
}
```

Note:

- i) Innanzitutto ricordiamo che, come abbiamo detto nella sezione precedente, la classe **JFrame** è contenuta nel package **javax.swing** e dunque è necessario importare o la classe oppure tutto il package. In questo esempio abbiamo deciso per comodità di importare tutto il package con l'istruzione

```
import javax.swing.*;
```

- ii) Innanzitutto dobbiamo **creare** un oggetto **JFrame** attraverso il suo costruttore. La classe **JFrame** ha molti costruttori; in particolare uno dei suoi costruttori prende come argomento una **stringa di testo** che comparirà come **titolo** della finestra. Ad ogni modo c'è anche un **costruttore di default** (senza argomenti) che costruisce una finestra senza titolo. In questo esempio abbiamo scelto di usare il costruttore che permette di specificare il titolo:

```
JFrame mainFrame = new JFrame("La mia finestra");
```

- iii) Creare una finestra non è sufficiente a vederla sullo schermo; infatti quando una finestra viene creata inizialmente è **invisibile**. Per visualizzarla è necessario chiamare un metodo specifico che tutti gli oggetti di tipo **Window** (e quindi anche gli oggetti di tipo **JFrame**, che è sottoclasse di **Window**) possiedono: il metodo **show**. Prima di fare ciò però vogliamo assegnare una dimensione alla finestra. Possiamo farlo con il metodo **setSize** che prende come argomenti la **larghezza** e l'**altezza** che la finestra dovrà assumere:

```
mainFrame.setSize(300, 300);
```

Infine, chiamiamo il metodo **show** per visualizzare la finestra sullo schermo:

```
mainFrame.show();
```

- iv) Osserviamo che quando “chiudiamo” la finestra con il pulsante apposito, essa scompare dallo schermo; ciononostante il programma sembra stare ancora girando, perché nella finestra da cui abbiamo lanciato il comando **java** non ci viene restituito il prompt dei comandi. Infatti il comportamento standard di un oggetto **JFrame** è il seguente: alla pressione del pulsante di chiusura la finestra viene nascosta, ma non eliminata. Pertanto il programma non termina. Dovremo essere noi – vedremo nell'esercizio successivo come – ad accorgerci che la finestra viene chiusa e a dare le istruzioni per fare terminare definitivamente il programma. Per ora possiamo “uccidere” brutalmente il programma premendo i tasti **CTRL+C**.

13. Uscire dal programma alla chiusura della finestra (solo JDK 1.3)

Gli oggetti `JFrame` hanno un metodo che permette di scegliere (all'interno di un insieme predefinito) quale operazione eseguire quando la finestra viene chiusa con l'apposito pulsante. Si tratta del metodo `setDefaultCloseOperation`. Le possibilità tra cui è possibile scegliere sono:

- (1) non fare nulla
- (2) nascondere la finestra (questa è l'operazione predefinita, che viene fatta se noi non scegliamo nulla)
- (3) nascondere e distruggere la finestra
- (4) nascondere e distruggere la finestra, ed uscire dal programma

A ciascuna di queste possibilità è associato un codice (un numero intero); il metodo `setDefaultCloseOperation` prende come argomento tale codice. Fortunatamente non abbiamo bisogno di ricordarci e nemmeno conoscere i codici associati alle diverse operazioni: essi sono infatti memorizzati in costanti predefinite.

Tali costanti sono:

1. `WindowConstants.DO NOTHING ON CLOSE` per non fare nulla.
2. `WindowConstants.HIDE ON CLOSE` per nascondere la finestra.
3. `WindowConstants.DISPOSE ON CLOSE` per nascondere e distruggere la finestra.
4. `JFrame.EXIT ON CLOSE` per nascondere e distruggere la finestra, ed uscire dal programma.

Si noti che i nomi delle costanti sono preceduti dal nome della classe in cui sono definite.

L'ultima possibilità è quella che interessa a noi. Per ottenere l'effetto desiderato sarà dunque sufficiente aggiungere al programma precedente la seguente istruzione:

```
mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

14. Aggiungere un pulsante `JButton` ad una finestra

Vogliamo adesso aggiungere un bottone alla nostra finestra. Un bottone è un **componente grafico**; sono detti componenti grafici tutti quegli oggetti che appartengono alla classe `Component` (e sue sottoclassi).

In generale gli unici componenti grafici che sono in grado di contenerne altri sono i **contenitori**, ossia quelli che appartengono alla classe `Container` (e sue sottoclassi). La classe `Container` ha infatti il metodo `add` che permette di aggiungere all'interno del contenitore un altro componente grafico. Quando il contenitore viene visualizzato, vengono visualizzati anche tutti gli oggetti al suo interno.

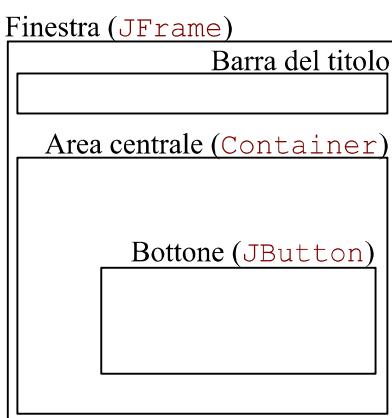
Poiché un contenitore è anche un componente grafico (leggi: la classe `Container` è sottoclasse della classe `Component`), è possibile aggiungere in un contenitore un altro contenitore.

La nostra finestra `JFrame` è un contenitore; tuttavia non possiamo aggiungere componenti grafici direttamente ad essa. Infatti una finestra `JFrame` contiene già altri componenti; fra questi:

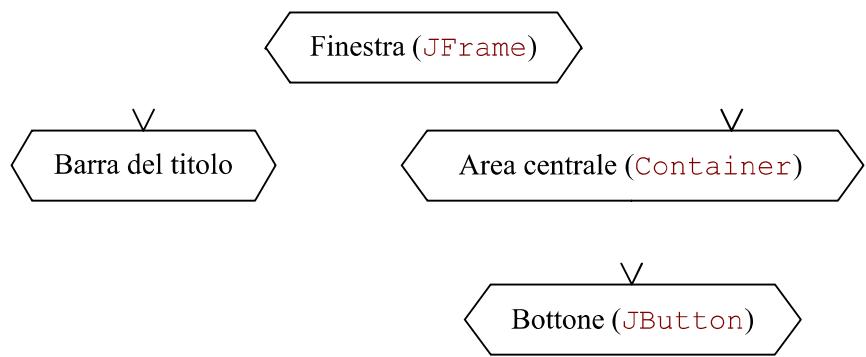
- (i) la **barra del titolo**
- (ii) l'**area centrale** che per il momento è vuota.

Se vogliamo aggiungere alla finestra un bottone, non dobbiamo aggiungerlo alla finestra stessa, bensì alla sua **area centrale** che è a sua volta un contenitore. Possiamo rappresentare questa situazione graficamente:

Rappresentazione a scatole



Rappresentazione ad albero



Quindi per aggiungere un bottone dobbiamo innanzitutto **ottenere dall'oggetto JFrame un riferimento alla sua area centrale** (un oggetto **Container**); dopodiché potremo **creare il bottone e aggiungere all'area centrale il bottone creato**.

Per fare tutte queste cose useremo un approccio leggermente diverso da quello usato nell'esercizio precedente. Poiché stiamo cercando di **personalizzare** la finestra, creeremo una classe apposita che rappresenti proprio la **nostra** finestra.

Naturalmente tale classe sarà una sottoclassificazione di **JFrame** in modo da ereditare il comportamento e le capacità di quest'ultima, e da permetterci di aggiungere ciò che desideriamo (in questo caso, un bottone).

Ecco il codice della nostra finestra, che chiameremo **MyFrame**:

```
import java.awt.*;
import javax.swing.*;

public class MyFrame extends JFrame
{
    public MyFrame()
    {
        super("La mia finestra");
        setSize(300,300);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container areaCentrale = getContentPane();
        JButton pulsante = new JButton("Pulsante");
        areaCentrale.add(bottone);
    }
}
```

Note:

- Per cominciare, possiamo inserire le operazioni per l'aggiunta del bottone nel costruttore. Inoltre possiamo spostare nel costruttore alcune delle operazioni che prima avevamo inserito nel metodo **main**. Come prima cosa invochiamo il costruttore della superclasse (ossia **JFrame**) tramite la parola chiave **super**, e gli passiamo il titolo da dare alla finestra. In secondo luogo impostiamo le dimensioni con il metodo **setSize**. Infine impostiamo l'operazione di uscita.

```
super("La mia finestra");
setSize(300,300);
setDefaultCloseOperation(EXIT_ON_CLOSE);
```

Si osservi qui la differenza tra la chiamata di un metodo dall'**esterno** di un oggetto (bisogna far precedere il nome del metodo dal nome dell'oggetto seguito dal punto) e la chiamata all'**interno** dell'oggetto stesso, in cui non è necessario specificare il nome dell'oggetto.

- Ora è necessario ottenere un riferimento all'area centrale della finestra **JFrame**. In Java l'area centrale dell'oggetto si chiama **content pane**; per ottenere il riferimento si può chiamare il metodo **getContentPane** definito in **JFrame**. Il metodo restituisce un riferimento ad un oggetto **Container** che memorizziamo in una variabile appropriata.

```
Container areaCentrale = getContentPane();
```

Attenzione: la classe **Container** è definita nel package **java.awt** quindi è necessario importarlo!

- A questo punto possiamo creare un pulsante. La classe che descrive i pulsanti è la classe **JButton**. Anche essa ha molti costruttori; quello che interessa a noi in questo momento è il costruttore che prende come argomento una stringa di testo da visualizzare sul pulsante.

```
JButton pulsante = new JButton("Pulsante");
```

- Infine, aggiungiamo il pulsante all'area centrale attraverso il metodo **add**. Il metodo **add** prende come argomento qualunque oggetto di tipo **Component**, quindi anche un oggetto **JButton**.

```
areaCentrale.add(bottone);
```

Adesso che abbiamo definito la classe che rappresenta la nostra finestra, dobbiamo creare un oggetto di tipo `MyFrame` nel metodo `main`, e visualizzarlo sullo schermo. Il `main` sarà più corto di quello degli esercizi precedenti, perché abbiamo spostato alcune istruzioni nel costruttore di `MyFrame`.

```
class ProgrammaX2
{
    public static void main(String[] args)
    {
        MyFrame mainFrame = new MyFrame();
        mainFrame.show();
    }
}
```

Ed ecco il risultato:



15. Rilevare la pressione del pulsante

Java gestisce l'interazione tra un utente e l'interfaccia grafica del programma con un meccanismo **ad eventi**. Quando un utente utilizza un programma da noi creato, solitamente interagisce con l'interfaccia grafica con il mouse e la tastiera; tramite di essi può cliccare su pulsanti, selezionare elementi in un elenco, scrivere in una casella di testo, etc. **Ciascuna di queste interazioni viene chiamata “evento” (c’è un tipo di evento diverso per ogni tipo di interazione)**. Quello che noi dobbiamo fare è semplicemente **catturare l’evento**, ossia fare in modo di accorgerci che esso è accaduto.

In Java un evento è rappresentato da un oggetto della classe `EventObject`. Tale classe ha svariate sottoclassi; ciascuna descrive un tipo specifico di evento. Il tipo di evento associato alla pressione di un pulsante è chiamato **action event**; la classe che lo descrive è `ActionEvent` (che naturalmente è una sottoclasse di `EventObject`).

Quando l’evento si verifica il pulsante che è stato premuto crea automaticamente un oggetto di tipo `ActionEvent` che contiene le informazioni sull’evento in questione (ad esempio, **quale** pulsante è stato premuto). Da un punto di vista pratico, pertanto, dobbiamo riuscire a farci “consegnare” l’oggetto evento appena costruito.

Perché la nostra classe possa ricevere l’oggetto evento generato dal pulsante essa deve:

1. essere in grado di ricevere un oggetto `ActionEvent`;
2. dichiarare esplicitamente al pulsante che vuole ricevere gli oggetti `ActionEvent` da esso generati.

Perché una classe sia in grado di ricevere degli oggetti evento essa deve definire al suo interno determinati metodi. Questi metodi sono quelli il creatore dell’evento (ad esempio il pulsante) chiamerà per “passare” l’oggetto evento a chi desidera riceverlo (ad esempio la nostra finestra).

I metodi da definire cambiano a seconda del tipo di evento; l’elenco dei metodi per un certo tipo di evento si trova nella corrispondente **interfaccia**. Ad esempio i metodi necessari per l’evento di tipo `ActionEvent` si trovano nell’interfaccia `ActionListener`. Si tratta in realtà di un metodo solo:

```
public void actionPerformed(ActionEvent e)
```

Dunque la nostra classe deve definire questo metodo. Quando il pulsante viene premuto esso innanzitutto crea un oggetto di tipo `ActionEvent`; dopodiché **chiama** il metodo `actionPerformed` che noi abbiamo definito e gli passa come argomento l’oggetto che ha appena costruito. Pertanto le istruzioni che mettiamo nel metodo costituiscono la nostra **risposta** alla pressione del bottone.

Definire il metodo non è sufficiente: dobbiamo dichiarare **nell’intestazione della nostra classe** che vogliamo implementare l’interfaccia corrispondente (ossia in questo caso `ActionListener`). Questo si fa con la seguente dichiarazione:

```
public class MyFrame extends JFrame implements ActionListener
```

Questo serve per **garantire** che nella nostra classe ci sia il metodo `actionPerformed` (altrimenti noi potremmo decidere che vogliamo ricevere l’evento dal pulsante e poi quando il pulsante cerca di chiamare il metodo non lo trova!). Perché il fatto di scrivere `implements ActionListener` è una garanzia? Perché se noi dichiariamo nell’intestazione di una classe che vogliamo **implementare un’interfaccia** il compilatore ci compilerà il file **soltanto se nella classe sono definiti TUTTI i metodi elencati nell’interfaccia**.

A questo punto abbiamo soddisfatto il punto 1, ossia abbiamo messo la nostra classe in grado di ricevere un oggetto `ActionEvent` (implementa il metodo che serve e dichiara ufficialmente che lo fa). Resta da comunicare al pulsante che la classe vuole ricevere i suoi eventi. Questo si fa chiamando un metodo del pulsante (classe `JButton`) che è in grado di “iscrivere” un oggetto nella lista di coloro che ricevono l’evento. Il metodo si chiama `addActionListener` ed accetta soltanto oggetti che implementano l’interfaccia `ActionListener`:

```
public void addActionListener(ActionListener l)
```

Attenzione: questo metodo non dobbiamo definirlo noi, è già definito nella classe `JButton`. Noi dobbiamo solo chiamarlo e passargli come argomento il nostro `ActionListener`, ossia la finestra stessa. Vediamo ora come esempio un programma che “riceve” l’evento corrispondente alla pressione del bottone, e quando lo riceve stampa a video la scritta “Click!”.

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MyFrame extends JFrame implements ActionListener
{
    public MyFrame()
    {
        super("La mia finestra");
        JButton pulsante = new JButton("Pulsante");
        Container areaCentrale = getContentPane();
        areaCentrale.add(pulsante);
        pulsante.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e)
    {
        System.out.println("Click!");
    }
}

```

Note:

- Le parti sottolineate sono quelle aggiunte rispetto alla versione precedente. Si osservi innanzitutto che abbiamo importato un nuovo **package**: si tratta di **java.awt.event** che contiene tutte le classi e interfacce che hanno a che vedere con gli eventi (in particolare, a noi servono **ActionEvent** e **ActionListener**).
- Il metodo **actionPerformed** contiene una sola istruzione per stampare sul video una scritta. Durante l'esecuzione, cliccando sul pulsante, si può verificare che il metodo viene chiamato **ogni volta** che il pulsante viene premuto.
- Si osservi l'istruzione:

```
pulsante.addActionListener(this)
```

Essa chiama il metodo **addActionListener** dell'oggetto **pulsante**, passandogli come parametro **this** ossia l'oggetto corrente. Questo perché è l'oggetto **MyFrame** stesso che stiamo definendo che riceverà l'evento creato dal pulsante.

Quesiti

- Q1.** Nell'esercizio precedente abbiamo fatto sì che fosse la finestra stessa a ricevere l'evento della pressione del pulsante. Si provi ora a costruire una **classe separata AscoltaPulsante** che implementa **ActionListener**; quindi si crei un oggetto di tale classe che riceve l'evento in questione. Si ricordi di importare nella nuova classe i package necessari!

Perché la classe **AscoltaPulsante** implementi **ActionListener** essa deve avere la dichiarazione **implements** e inoltre definire il metodo **actionPerformed**. D'altro canto non sarà più necessario che **MyFrame** implementi **ActionListener**, pertanto da essa si possono rimuovere la dichiarazione **implements** e il metodo **actionPerformed**.

Nel costruttore di **MyFrame** si dovrà innanzitutto costruire un oggetto di tipo **AscoltaPulsante** (sarà lui a dover ricevere l'evento):

```
AscoltaPulsante ap = new AscoltaPulsante();
```

Quindi si dovrà registrare presso il pulsante non più la finestra stessa (**this**), ma l'oggetto appena creato:

```
pulsante.addActionListener(ap);
```

- Q2.** Si modifichi la classe **AscoltaPulsante** in modo che invece di stampare la scritta "Click!" stampi **il numero di volte che il pulsante è stato premuto sino a quel momento**.

16. Uscire dal programma alla chiusura della finestra (JDK 1.1 e 1.2)

Nell'esercizio precedente abbiamo visto come ricevere eventi di tipo `ActionEvent`, che vengono creati da un pulsante quando viene premuto. In questo esercizio vedremo invece come ricevere gli eventi di tipo `WindowEvent`, che vengono creati da una finestra quando le accade qualcosa. In particolare vogliamo accorgerci di quando la finestra viene chiusa utilizzando il pulsante con la X in alto a destra, in modo da poter uscire dal programma.

Mentre c'è un solo evento di tipo `ActionEvent` (la pressione del pulsante: è l'unico caso in cui viene creato un oggetto `ActionEvent`), ci sono molti eventi di tipo `WindowEvent`, e sono tutti quegli eventi che hanno a che vedere con una finestra. Ogni volta che uno di questi eventi accade la finestra crea un oggetto di tipo `WindowEvent`; quindi chiama **un metodo diverso** a seconda dell'evento che si è verificato. Dunque l'interfaccia corrispondente all'evento della finestra, ossia `WindowListener`, contiene una lista di più metodi, uno per ogni diversa circostanza in cui l'evento `WindowEvent` viene creato.

Qui di seguito è riportato l'elenco dei metodi, e per ciascuno la circostanza in cui viene chiamato:

- `public void windowOpened(WindowEvent we)` : finestra aperta per la prima volta.
- `public void windowDeactivated(WindowEvent we)` : finestra disattivata (l'utente passa ad un'altra finestra).
- `public void windowActivated(WindowEvent we)` : finestra riattivata.
- `public void windowIconified(WindowEvent we)` : finestra minimizzata.
- `public void windowDeiconified(WindowEvent we)` : finestra ripristinata (dopo essere stata minimizzata).
- `public void windowClosing(WindowEvent we)` : l'utente manifesta l'intenzione di chiudere la finestra. **Questa è la circostanza particolare che ci interessa.**
- `public void windowClosed(WindowEvent we)` : finestra effettivamente chiusa.

Creiamo ora una classe `AscoltaFinestra` analoga alla precedente `AscoltaPulsante`, ma che implementa `WindowListener`:

```
import java.awt.event.*;

class AscoltaFinestra implements WindowListener
{
    public void windowOpened(WindowEvent we)
    {
    }

    public void windowDeactivated(WindowEvent we)
    {
    }

    public void windowActivated(WindowEvent we)
    {
    }

    public void windowIconified(WindowEvent we)
    {
    }

    public void windowDeiconified(WindowEvent we)
    {
    }

    public void windowClosing(WindowEvent we)
    {
        System.exit(1);
    }

    public void windowClosed(WindowEvent we)
    {
    }
}
```

Note

- a. Come si può osservare, la classe `AscoltaFinestra` definisce tutti i metodi elencati in `WindowListener`: questo è **obbligatorio** se si dichiara di implementare l'interfaccia `WindowListener`. A noi interessa solo compiere un'azione (in particolare, uscire dal programma) quando l'utente dichiara di voler chiudere la finestra. Poiché in tale circostanza viene chiamato il metodo `windowClosing`, mettiamo l'istruzione per uscire dal programma dentro di esso. Per quanto riguarda gli altri metodi, essi vengono chiamati in circostanze in cui non ci interessa fare niente. Pertanto li lasciamo senza istruzioni. (Si osservi che c'è una differenza fra un metodo **definito esplicitamente senza istruzioni**, come in questo caso, e un metodo **dichiarato ma non definito** come succede nelle interfacce. Nel primo caso esiste un'area in cui vengono conservate le istruzioni del metodo, solo che è vuota. Nel secondo caso tale area non esiste.)

Per fare sì che alla chiusura della finestra il programma esca, ci resta solo da **registrare presso la finestra** il fatto che c'è un oggetto (di tipo `AscoltaFinestra`) interessato a ricevere i suoi eventi. Riprendiamo pertanto la nostra finestra `MyFrame` e **aggiungiamo** le nuove istruzioni (lasciando anche quelle per ascoltare il pulsante!).

```
import java.awt.*;
import javax.swing.*;

public class MyFrame extends JFrame
{
    public MyFrame()
    {
        super("La mia finestra");
        JButton pulsante = new JButton("Pulsante");
        Container areaCentrale = getContentPane();
        areaCentrale.add(pulsante);
        AscoltaPulsante ap = new AscoltaPulsante();
        pulsante.addActionListener(ap);
        AscoltaFinestra af = new AscoltaFinestra();
addWindowListener(af);
    }
}
```

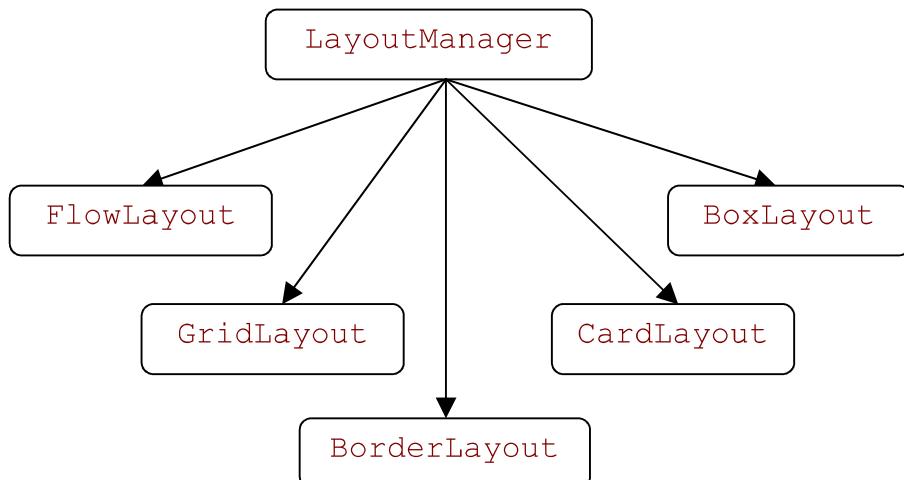
Note

- b. Le istruzioni aggiunte sono quelle riportate sottolineate. Osserviamo che la chiamata al metodo `addWindowListener` non è preceduta dal nome di alcun oggetto: questo perché il metodo viene chiamato sulla finestra stessa (`this`).

17. Posizionamento dei componenti: i Layout Manager

Supponiamo di aggiungere più bottoni ad una stessa finestra: in tal caso vorremmo anche decidere come posizionarli uno rispetto all'altro.

La decisione su come disporre i componenti grafici in una finestra viene presa da un oggetto chiamato **Layout Manager**. Ci sono diversi tipi di Layout Manager: ciascuno dispone i componenti in un modo diverso. Ciascun tipo di Layout Manager è descritto da una classe diversa; tutte però derivano da **LayoutManager**. Nel seguente schema sono riportati i principali Layout Manager:



Vediamo adesso un esempio di come si può scegliere un particolare Layout Manager. Modifichiamo il programma dell'esercizio 16 in modo da inserire non uno, ma sei bottoni. Quindi sceglieremo un Layout Manager e vediamo come dispone i componenti nella finestra.

```

import java.awt.*;
import javax.swing.*;

public class MyFrame extends JFrame
{
    public MyFrame()
    {
        super("La mia finestra");
        setSize(200,200);
        JButton b1 = new JButton("1");
        JButton b2 = new JButton("2");
        ...
        JButton b6 = new JButton("6");
        Container areaCentrale = getContentPane();
areaCentrale.setLayout(new FlowLayout());
        areaCentrale.add(b1);
        areaCentrale.add(b2);
        ...
        areaCentrale.add(b6);
        AscoltaFinestra af = new AscoltaFinestra();
        addWindowListener(af);
    }
}
  
```

Note

- L'istruzione chiave è quella sottolineata. Innanzitutto essa corrisponde in realtà a due istruzioni, in quanto è una forma abbreviata per

```
LayoutManager lm = new FlowLayout();
areaCentrale.setLayout(lm);
```

Con l'istruzione `new FlowLayout()` si crea un nuovo oggetto Layout Manager del tipo desiderato, in questo caso appunto `FlowLayout`. Bisogna poi assegnare il Layout Manager alla finestra che dovrà utilizzarlo, ossia nell'esempio `areaCentrale`. Questo viene fatto chiamando il metodo `setLayout` a cui si passa il Layout Manager appena creato.

Il costruttore di `FlowLayout` non prende nessun argomento. Vedremo che i costruttori di alcuni Layout Manager prendono degli argomenti.

Caratteristiche del Flow Layout



- Il costruttore è senza argomenti;
- i componenti vengono disposti in fila, da sinistra a destra;
- se non ci stanno in una riga, vengono disposti su due, e così via;
- i componenti su ciascuna riga sono centrati rispetto alla finestra;
- ciascun componente è della dimensione minima necessaria per visualizzarlo (nel caso di un bottone, tale dimensione minima dipende dalla scritta che esso contiene).

Vediamo ora le caratteristiche degli altri Layout Manager. Per ciascuno diamo anche un esempio di utilizzo.

Caratteristiche del Grid Layout



- Il `GridLayout` crea una griglia le cui dimensioni (righe ? colonne) devono essere specificate nel costruttore;
- viene inserito un componente in ciascun riquadro della griglia; se ci sono più componenti che riquadri (ad esempio 7 componenti in una griglia 2?3) vengono aggiunte delle colonne. Se ci sono meno componenti che riquadri, il numero di colonne viene aggiustato in modo che non ci siano colonne completamente vuote;
- i riquadri hanno tutti la stessa dimensione;
- i componenti vengono allargati ad occupare l'intero riquadro.

Esempio:

```
areaCentrale.setLayout(new GridLayout(3,2));
```

Caratteristiche del Border Layout



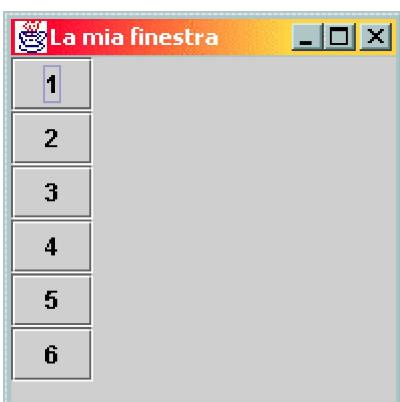
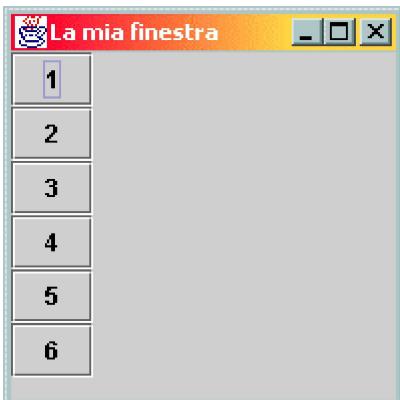
- Il `BorderLayout` divide la finestra in 5 parti, chiamate NORTH, SOUTH, WEST, EAST e CENTER;
- in ciascuna parte viene inserito un componente; poiché ci sono 5 sezioni possono essere inseriti solo cinque componenti;
- le aree NORTH e SOUTH hanno larghezza massima e altezza minima, mentre le aree WEST ed EAST hanno altezza massima e larghezza minima. L'area CENTER occupa lo spazio restante;
- se una delle aree ai bordi è vuota il suo spazio viene occupato dall'area CENTER;
- quando si aggiunge un componente bisogna anche specificare in quale area lo si vuole inserire.

Esempio:

```
areaCentrale.setLayout(new BorderLayout());
areaCentrale.add(b1, BorderLayout.NORTH);
areaCentrale.add(b2, BorderLayout.SOUTH);
...
```

Si noti che le costanti numeriche NORTH, SOUTH, etc. sono definite nella classe `BorderLayout` e pertanto è necessario far precedere i loro nomi dal nome della classe stessa.

Caratteristiche del Box Layout



- Il Box Layout dispone i componenti in verticale **oppure** in orizzontale, a seconda di quanto specificato nel costruttore attraverso le costanti Y_AXIS (verticale) e X_AXIS (orizzontale);
- il costruttore vuole anche come parametro un riferimento alla finestra di cui sarà il Layout Manager;
- i componenti vengono disposti in colonna (o in riga) anche qualora non stessero nella finestra;
- la dimensione dei componenti è la minima possibile.

Esempio:

```
areaCentrale.setLayout(new BoxLayout(areaCentrale, BoxLayout.Y_AXIS));
```

Le costanti X_AXIS e Y_AXIS sono definite nella classe `BoxLayout`.

Quesito

Si provino ad usare i Layout Manager illustrati in questo esercizio e si verifichi il risultato.

18. Posizionamento dei componenti: i Componenti Invisibili

I componenti invisibili sono dei componenti che occupano spazio ma non si vedono, pertanto possono essere utilizzati per introdurre degli spazi tra altri componenti. Ci sono due tipi di componenti invisibili: Area Rigida e Colla. Il Layout Manager che permette di sfruttare meglio i componenti invisibili è il Box Layout; ad ogni modo essi possono essere inseriti anche con gli altri Layout Manager, sebbene i risultati non siano sempre quelli più ovvii.

Area Rigida (`RigidArea`)

Un'Area Rigida è un rettangolo di dimensione fissata che occupa lo spazio corrispondente. Per creare un'Area Rigida è necessario chiamare il metodo statico `createRigidArea` della classe `Box`. I metodi che servono per creare degli oggetti che non possono essere creati direttamente tramite il costruttore sono detti

metodi factory ossia **metodi-fabbrica**. Il metodo `createRigidArea` è per l'appunto un metodo-fabbrica. esso prende come argomento un oggetto di tipo `Dimension`. La classe `Dimension` è una classe molto semplice che si limita a raggruppare in un unico oggetto due interi rappresentati rispettivamente una larghezza ed una altezza.

Ecco l'istruzione necessaria per creare, ad esempio, un'Area Rigida di dimensioni 20?20:

```
Component areaRigida = Box.createRigidArea(new Dimension(20,20));
```

Ovviamente perché l'Area Rigida prenda il suo posto nella finestra è necessario inserirvela come qualunque altro componente. È necessario ricordare che i componenti vengono visualizzati nella finestra nell'ordine in cui vengono inseriti, pertanto ad esempio per inserire un'Area Rigida fra due bottoni è necessario inserire nell'ordine il primo bottone, l'Area Rigida e quindi il secondo bottone. Un'Area Rigida può anche avere una delle due dimensioni pari a 0; in tal caso più che di un'area si tratterà di un segmento, e occuperà spazio solo in una dimensione.

Esempio con Box Layout orizzontale, due bottoni separati da un'Area Rigida 20?0.

```
... vengono creati due pulsanti, p1 e p2 ...
areaCentrale.setLayout(new BoxLayout(areaCentrale,BoxLayout.X_AXIS));
areaCentrale.add(p1);
areaCentrale.add(Box.createRigidArea(new Dimension(20,0)));
areaCentrale.add(p2);
... etc etc ...
```



È importante tenere presente che la dimensione impostata per l'Area Rigida al momento della costruzione è una **dimensione minima**; pertanto quei Layout Manager che massimizzano la dimensione di un componente non ne tengono conto (si provi ad esempio ad inserire un'Area Rigida in un Grid Layout: essa occuperà un posto, ma la dimensione sarà comunque quella del riquadro della griglia).

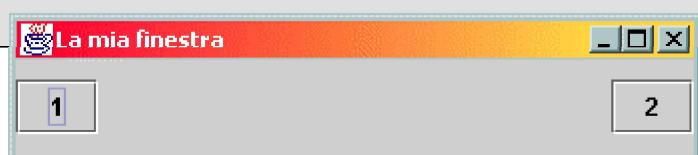
Colla (Glue)

Il secondo tipo di componente invisibile è la Colla. La Colla non ha una dimensione fissa, ma tende ad espandersi il più possibile (se il Layout Manager glielo consente). Potremmo dire che la Colla ha dimensione minima nulla, e dimensione massima infinita. Essa viene usata per distanziare due componenti il più possibile. La Colla può essere orizzontale o verticale. Una Colla orizzontale ha altezza 0, mentre la sua larghezza è espandibile. Una Colla verticale ha larghezza 0, ed è la sua altezza ad essere espandibile. I due tipi di Colla si possono creare con due metodi-fabbrica della classe `Box`, rispettivamente `createHorizontalGlue` e `createVerticalGlue`:

```
Component collaOrizzontale = Box.createHorizontalGlue();
Component collaVerticale = Box.createVerticalGlue();
```

Esempio con Box Layout orizzontale, due bottoni separati da una Colla orizzontale.

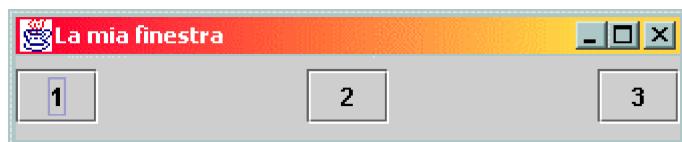
```
... vengono creati due pulsanti, p1 e p2 ...
areaCentrale.setLayout(new
BoxLayout(areaCentrale,BoxLayout.X_AXIS));
areaCentrale.add(p1);
areaCentrale.add(Box.createHorizontalGlue());
areaCentrale.add(p2);
... etc etc ...
```



Poiché la dimensione **minima** della Colla è nulla, con alcuni Layout Manager essa non funziona. Ad esempio il Flow Layout visualizza tutti i componenti alla loro dimensione minima, pertanto in esso la Colla non si vede. Se ci sono più oggetti Colla le loro dimensioni vengono il più possibile uguagliate.

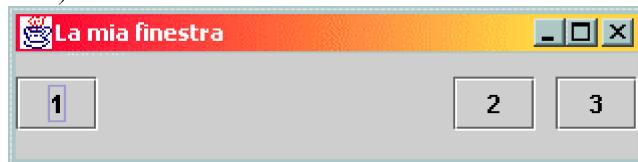
Esempio con Box Layout orizzontale, tre pulsanti separati da Colle orizzontali.

```
... vengono creati tre pulsanti, p1, p2 e p3 ...
areaCentrale.setLayout(new BoxLayout(areaCentrale,BoxLayout.X_AXIS));
areaCentrale.add(p1);
areaCentrale.add(Box.createHorizontalGlue());
areaCentrale.add(p2);
areaCentrale.add(Box.createHorizontalGlue());
areaCentrale.add(p3);
... etc etc ...
```



Quesiti

Q1. Si provi a costruire una finestra con i bottoni disposti come nella seguente figura (tra i bottoni 2 e 3 ci sono 10 pixel di distanza):

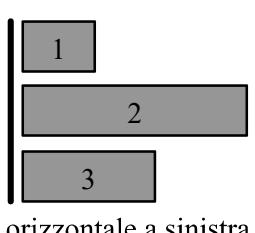


Q2. Si provi ora a costruire una finestra con i bottoni disposti come nella seguente figura:

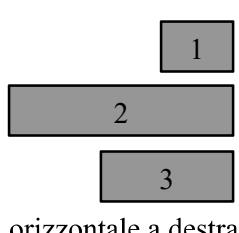


19. Posizionamento dei componenti: allineamento in un Box Layout

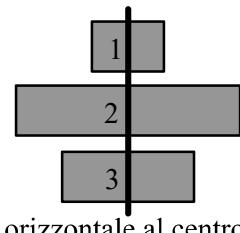
Quando si utilizza un Box Layout è possibile specificare anche l'**allineamento** dei componenti uno rispetto all'altro. L'allineamento **orizzontale** decide se un componente si trova sulla destra o sulla sinistra rispetto ad un altro, mentre l'allineamento **verticale** decide se un componente si trova in alto o in basso rispetto ad un altro. Qui sotto sono riportati alcuni esempi; come si può vedere dalle figure l'allineamento orizzontale influenza solo su componenti disposti verticalmente (Box Layout con Y_AXIS), mentre l'allineamento



orizzontale a sinistra

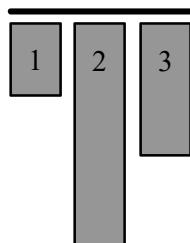


orizzontale a destra

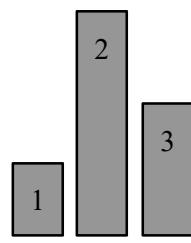


orizzontale al centro

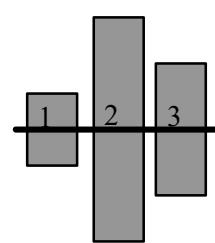
verticale influisce solo sui componenti disposti orizzontalmente (Box Layout con X_AXIS).



verticale in alto



verticale in basso



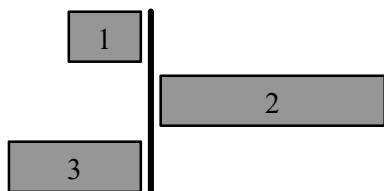
verticale al centro

L'allineamento di un componente è stabilito da due parametri decimali di tipo `float` chiamati `AlignmentX` (allineamento orizzontale) e `AlignmentY` (allineamento verticale). Per modificarne il valore si possono usare i metodi del componente `setAlignmentX` e `setAlignmentY` rispettivamente.

I parametri passati a questi metodi devono essere valori `float` compresi fra 0 e 1. In particolare:

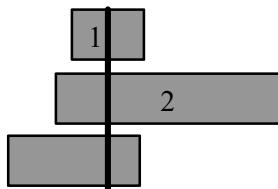
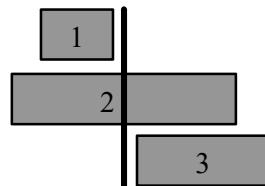
- il valore 0 significa allineamento a sinistra e in alto, rispettivamente;
- il valore 1 significa allineamento a destra e in basso, rispettivamente;
- il valore 0.5 significa allineamento centrale;
- i valori intermedi significano posizioni intermedie fra quelle elencate, ma sono molto poco usati.

Per ottenere gli allineamenti raffigurati è necessario assegnare a tutti i componenti uno stesso valore; è tuttavia anche possibile ottenere effetti diversi impostando valori diversi per ciascun componente. Ecco alcuni esempi:



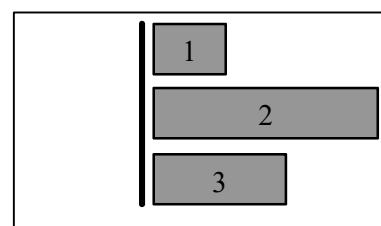
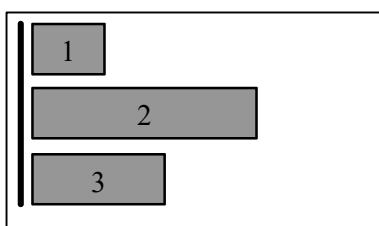
In questo caso il primo componente ha allineamento pari a 1 (destra), il secondo pari a 0.5 (centro) e il terzo pari a 0 (sinistra).

In questo caso il primo e terzo componente hanno allineamento orizzontale pari a 1 (destra), mentre il secondo componente ha allineamento pari a 0 (sinistra). Si immagini una linea verticale, come quella in figura, che deve coincidere con i lati destri o sinistri a seconda dell'allineamento impostato.



In questo caso il primo componente ha allineamento pari a 0.5 (centro), il secondo pari a 0.25 (un quarto sulla sinistra) e il terzo pari a 0.75 (un quarto sulla destra).

Quanto detto spiega come i componenti vengono disposti **uno rispetto all'altro**, ma non come vengono disposti all'interno della finestra. Ad esempio nella figura sottostante si vede come dei componenti allineati a sinistra possano essere disposti diversamente all'interno della finestra:



Ebbene: la disposizione dei componenti rispetto alla finestra viene **calcolata automaticamente** a partire dall'allineamento reciproco, e pertanto non è possibile stabilirla indipendentemente dal resto. In generale possiamo dire che:

- se i componenti sono tutti allineati a sinistra, saranno a sinistra anche nella finestra;
- se sono tutti allineati a destra, saranno a destra anche nella finestra;
- se sono tutti al centro, saranno al centro anche nella finestra;

- altrimenti saranno in qualche posizione intermedia.

Andiamo ora a sperimentare quanto detto fino a qui; modifichiamo ossia il programma degli ultimi esercizi in modo da vedere i diversi allineamenti.

```
... siamo nel costruttore della finestra ...
 JButton p1 = new JButton("1");
 JButton p2 = new JButton("222222"); //più lungo
 JButton p3 = new JButton("333"); //lunghezza intermedia
 areaCentrale.setLayout(new BoxLayout(areaCentrale,BoxLayout.Y_AXIS));
 p1.setAlignmentX(1);
 p2.setAlignmentX(1);
 p3.setAlignmentX(1);
 areaCentrale.add(p1);
 areaCentrale.add(p2);
 areaCentrale.add(p3);
 ... etc etc ...
```

Quesito

Si provi ad eseguire il programma con diversi valori per l'allineamento; in particolare si provi a disporre i componenti come nelle figure precedenti.

20. Posizionamento dei componenti: Pannelli annidati

I Layout Manager che abbiamo visto permettono un numero limitato di disposizioni dei componenti. Alcuni poi (ad es. il Border Layout) permettono addirittura un numero limitato di componenti!

Ad esempio, con nessuno dei Layout Manager visti sinora è possibile ottenere la seguente finestra:



Il segreto nell'ottenere le più svariate combinazioni di pulsanti consiste nel **usare delle finestre intermedie**, dette **Pannelli**, che vengono inserite una dentro l'altra. Ciascun Pannello è un'area rettangolare all'interno della quale è possibile disporre altri componenti, utilizzando tutto ciò che abbiamo visto sinora (Layout Manager, Componenti Invisibili, Allineamento, etc). A sua volta, tuttavia, un pannello è un componente, e pertanto può essere inserito all'interno di una finestra. Un pannello è descritto dalla classe **JPanel** e può essere costruito con **new JPanel()**.

Consideriamo ad esempio la finestra qui sopra. Essa può essere realizzata nel modo seguente:

- Si crea un Pannello (chiamiamolo **pulsanti**) in cui vengono inseriti i due pulsanti. Il Pannello usa un Flow Layout (in modo che i pulsanti stiano al centro), e tra i due pulsanti viene inserita un'Area Rigida per speararli un po'.
- Si assegna all'area centrale della finestra un Border Layout.
- Si inserisce il Pannello **pulsanti** nella zona SOUTH dell'Area Centrale.

La figura a fianco schematizza graficamente quanto detto.

