

I Layout Manager (prima parte)

GUI Interface

OOP Comment:

Aggiungendo ad un frame molti componenti dell'interfaccia utente, il frame stesso può diventare abbastanza complesso: **per frame che contengono molti componenti è opportuno utilizzare l'ereditarietà**. Infatti, se il software che si sta sviluppando contiene molte finestre ricche di componenti ci si troverebbe a dover fronteggiare una soluzione il cui caso limite è di un metodo **main()** che contiene la definizione di tutte le finestre del programma.

Senza arrivare a questa situazione, si otterrebbero moduli che sono **molto accoppiati e poco coesi** (*al contrario dei principi della OOP*); le classi devono essere il più autonome possibile (low coupling), mentre il contrario i metodi al loro interno devono essere il più coesi possibile (* high cohesion).

Inoltre, gli stessi moduli sarebbero poco leggibili e non sarebbe facile manutenerli. In parole povere, non si sfruttarebbe le potenzialità dell'**Object Orientation**: non sarebbe possibile permettere l'**information hiding** dei contenuti dei frame e dei controlli in essi contenuti; non sarebbe nemmeno sfruttato l'**incapsulamento** delle implementazioni, mettendo insieme concetti eterogenei e finestre diverse tra loro.

Un Layout Manager è una politica di posizionamento dei componenti in un Container. Ogni Container ha un Layout Manager

***Alta coesione significa** che la classe è focalizzata su ciò che dovrebbe fare, cioè solo i metodi relativi all'intenzione della classe

I Layout Manager

Quando si dispongono i componenti all'interno di un Container sorge il problema di come gestire il posizionamento: infatti, sebbene sia possibile specificare le coordinate assolute di ogni elemento dell'interfaccia, queste possono cambiare nel corso della vita del programma allorquando la finestra principale venga ridimensionata.

Per semplificare il lavoro di impaginazione e risolvere questo tipo di problemi possibile ricorrere ai layout manager, oggetti che si occupano di gestire la strategia di posizionamento dei componenti all'interno di un contenitore.

Un gestore di layout è una qualsiasi classe che implementa l'interfaccia ***LayoutManager***; ogni container nasce con un certo Layout Manager ma `e possibile assegnare il pi`u opportuno per quel Container con il metodo:

public void setLayout(LayoutManager m);

Documantation: The Java LayoutManagers facilitates us to control the positioning and size of the components in GUI forms. **LayoutManager** is an **interface** that is implemented by all the classes of layout managers. There are the following classes that represent the layout managers:

- **[java.awt.FlowLayout](#)**
- **[java.awt.BorderLayout](#)**
- **[java.awt.GridLayout](#)**
- **[java.awt.GridBagLayout](#)**
- **[javax.swing.BoxLayout](#)**
- **[javax.swing.GroupLayout](#)**

FlowLayout Manager

Il primo layout da essere citato è il gestore a scorrimento (FlowLayout) che sono inseriti da sinistra verso destra con la loro *Preferred Size*, cioè la dimensione minima necessaria a disegnarlo interamente.

Usando il paragone con un editor di testi, ogni controllo rappresenta una “parola” che ha una sua propria dimensione. Come le parole in un editor vengono inseriti da sinistra verso destra finchè entrano in una riga, così viene fatto per i controlli inseriti da sinistra verso destra.

Quando un componente non entra in una “riga” viene posizionato in quella successiva. Il costruttore di default imposta l’allineamento centrale.

Constructors of FlowLayout class

- ***FlowLayout()*** : creates a flow layout with centered alignment and a default 5 unit horizontal and 5 vertical gap.
- ***FlowLayout(int align)*** : creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
- ***FlowLayout(int align, int hgap, int vgap)*** : creates a flow layout with the given alignment and the given horizontal and vertical gap.

```
public class EsempioFlowLayout{
    JFrame mioFrame;

    // constructor
    EsempioFlowLayout() {
        // creating a frame object
        mioFrame = new JFrame();

        // creating the buttons
        JButton b1 = new JButton("1"); JButton b2 = new JButton("2");
        JButton b3 = new JButton("3"); ..... JButton b10 = new JButton("10");

        // adding the buttons to frame
        mioFrame.add(b1); mioFrame.add(b2); ..... mioFrame.add(b10);
        mioFrame.setLayout(new FlowLayout()); //set to FlowLayout of MioFrame container
        mioFrame.setSize(300, 300);
        mioFrame.setVisible(true);
    } //costruc

    // main method
    public static void main(String args[]) {
        new EsempioFlowLayout();
    }
} //class
```



Ridimensionamento

E' opportuno osservare come l'allineamento dei componenti del frame si adatti durante l'esecuzione quando la stessa finestra viene ridimensionata (vedi immagine seguente).

Ovviamente se non esiste nessun allineamento per i componenti del frame tale che tutti siano contemporaneamente visibile, allora alcuni di questi risulteranno in parte o del tutto non visibili; per esempio, quando la finestra risulta troppo piccola



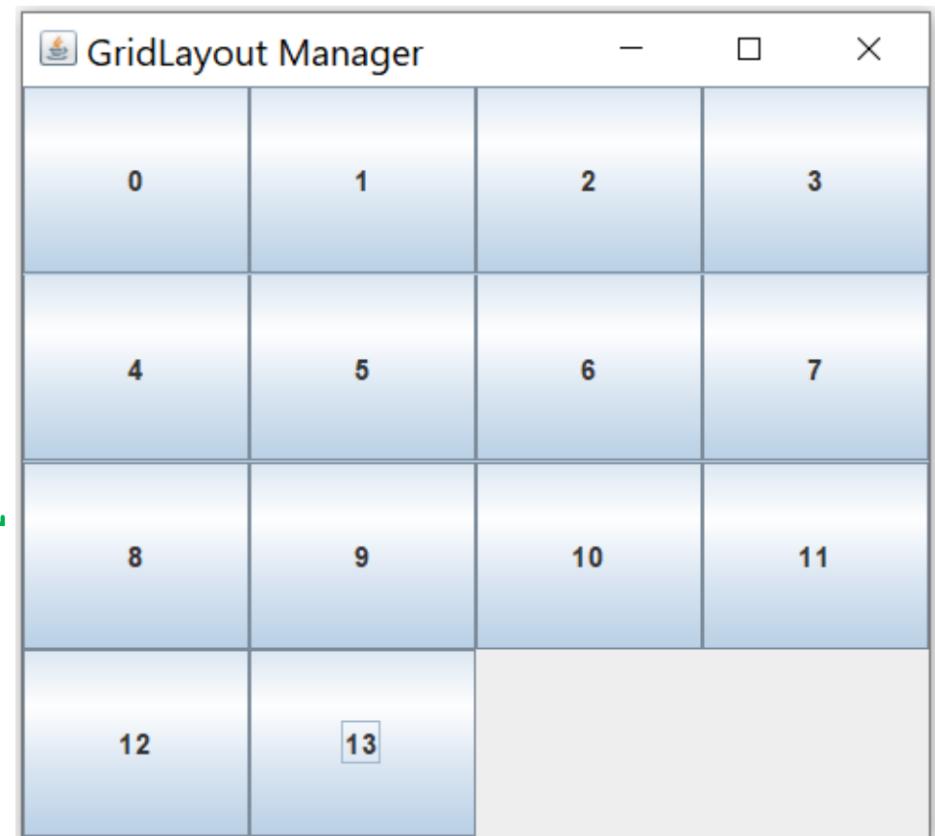
GridLayout Manager

Il gestore a griglia (**GridLayout**) suddivide il contenitore in una **griglia** di celle di uguali dimensioni. Le dimensioni della griglia vengono definite mediante il **costruttore**:

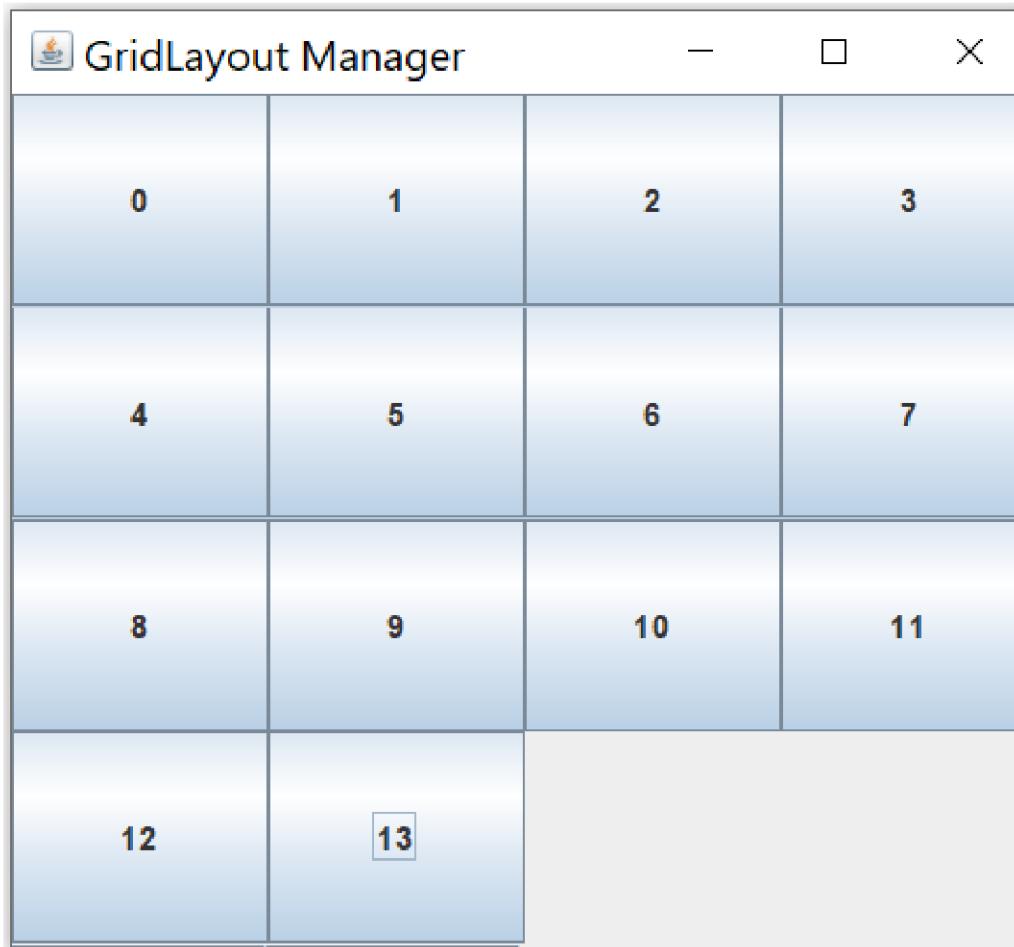
```
public GridLayout(int rows, int columns)
```

in cui i parametri **rows** e **columns** specificano rispettivamente *le righe e le colonne della griglia*. A differenza di quanto avviene con FlowLayout, **i componenti all'interno della griglia assumono automaticamente la stessa dimensione**, dividendo equamente lo spazio disponibile.

```
public class MioFrame extends JFrame {  
    public MioFrame() {  
        super("GridLayout");  
        Container c = this.getContentPane();  
        c.setLayout(new GridLayout(4, 4));  
  
        //aggiungo i bottoni al pannello secondo il GL  
        for(int i = 0; i<14; i++)  
            c.add(new JButton(String.valueOf(i)));  
        setSize(300, 300);  
        setVisible(true);  
    } //costruc  
} //class
```

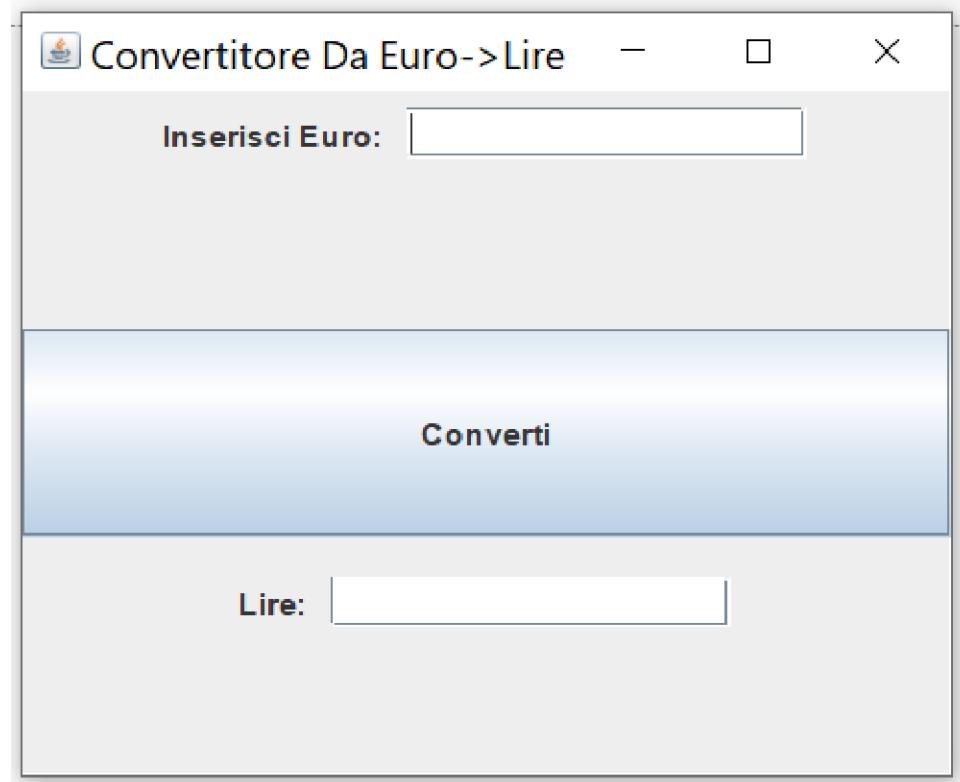


```
public class MioFrame extends JFrame {  
    public MioFrame(){  
        super("GridLayout");  
        Container c = this.getContentPane();  
        c.setLayout(new GridLayout(4, 4));  
  
        for(int i = 0; i<14; i++) c.add(new JButton(String.valueOf(i)));  
  
        setSize(300,300);  
        setVisible(true);  
    } //costruc  
} //class
```



Convertitore *euro --> lira* (interfaccia grafica)

```
public class MioFrame extends JFrame {  
    private JPanel p1 = new JPanel();  
    private JPanel p2 = new JPanel();  
    private JTextField txtEuro = new JTextField(15);  
    private JTextField txtLire = new JTextField(15);  
    private JButton btnConverti = new JButton("Converti");  
  
    public MioFrame () {  
        super("Convertitore Da Euro->Lire");  
        // inserisce le componenti nei pannelli  
        p1.add(new JLabel("Inserisci Euro: "));  
        p1.add(txtEuro);  
        p2.add(new JLabel("Lire: "));  
        p2.add(txtLire);  
  
        // inserisce le componenti nella finestra disponendole con una griglia 3x1  
        setLayout(new GridLayout(3,1,5,10));  
        add(p1);  
        add(btnConverti);  
        add(p2);  
        this.setBounds(400, 200, 370, 300);  
        this.setVisible(true);  
    } // costr  
} //class
```

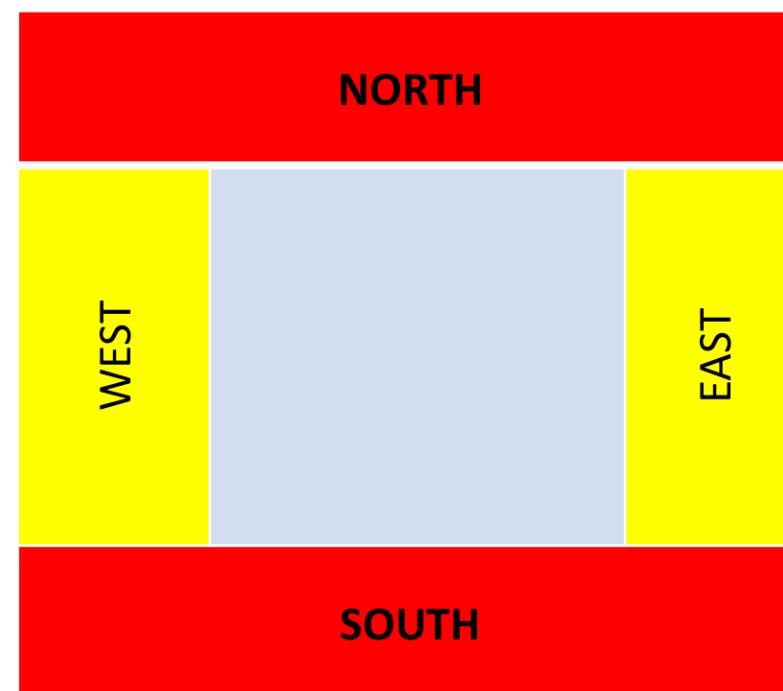


BorderLayout Manager

Il **BorderLayout** manager suddivide il contenitore esattamente in *cinque* aree, disposte a croce, (**NORTH – SOUTH - EAST - WEST**).

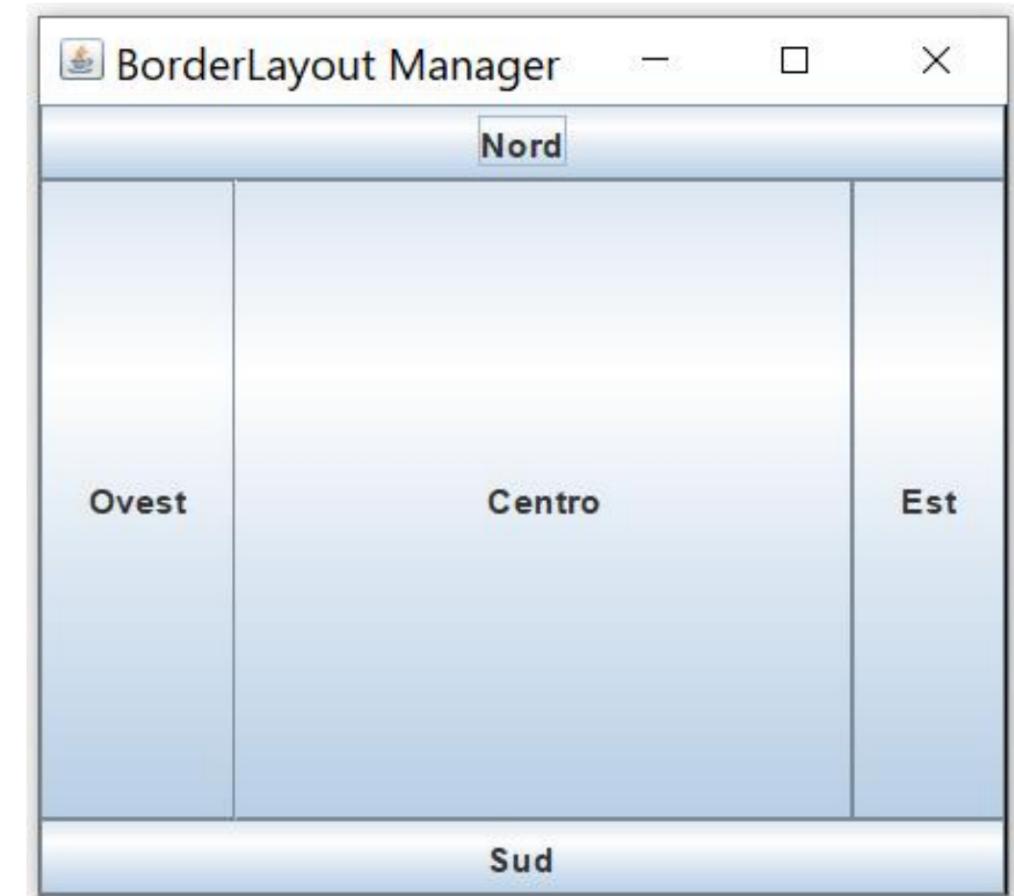
Ogni zona può contenere uno ed un solo widget (o Container): un secondo widget inserito in una zona sostituisce il precedente.
Se una o più zone non vengono riempite, allora i componenti nelle altre zone sono estesi a riempire le zone vuote.

Il programmatore può decidere in quale posizione aggiungere un controllo utilizzando la variante presente nei Container:



Esempio di 5 button che riempiono tutto il container del frame con una disposizione gestita dal BorderLayout

```
public class MioFrame extends JFrame {  
    JButton nord = new JButton("Nord");  
    JButton centro = new JButton("Centro");  
    JButton ovest=new JButton("Ovest");  
    JButton est = new JButton("Est");  
    JButton sud=new JButton("Sud");  
  
    public MioFrame() {  
        super("BorderLayout Manager");  
        Container c = this.getContentPane();  
        c.setLayout(new BorderLayout());  
        c.add(nord, BorderLayout.NORTH);  
        c.add(centro, BorderLayout.CENTER);  
        c.add(ovest, BorderLayout.WEST);  
        c.add(sud, BorderLayout.SOUTH);  
        c.add(est, BorderLayout.EAST);  
        setSize(300,300);  
        setVisible(true);  
    } //class  
} //class
```



GUI più complesse

I gestori di layout permettono maggiore versatilità nell'inserimento dei controlli nelle finestre. Tuttavia nella stragrande maggioranza delle situazioni un'intera finestra non può seguire un unico layout.

Si consideri la seguente GUI. Non è difficile convincersi che la parte in alto contenente una sorta di “*tastierino numerico*” segue un *GridLayout* mentre i bottoni in basso seguono un *FlowLayout* centrato



tastierino numerico
by GridLayout

Bottoni di conferma
by FlowLayout

Poichè ogni Container può seguire uno ed un unico layout, **occorre predisporre più Container, uno per ogni zona che ha un layout differente.**

Ovviamente ad ogni Container possono essere aggiunti direttamente i controlli oppure altri Container

La ***lastra dei contenuti*** è il Container "radice" ed altri si possono aggiungere all'interno.

I Container (insieme con i componenti contenuti) aggiunti in un altro Container si comporteranno come ogni altro ***widget*** la cui dimensione preferita è quella minima necessaria a visualizzare tutti i componenti all'interno.

Per creare nuovi Container conviene utilizzare la classe

`javax.swing.JPanel` che eredita da ***java.awt.Container***.

La forza di questa soluzione è data dall'alta modularità: ***è possibile usare un layout per il pannello interno e un altro layout per il ContentPane***.

Il pannello interno verrà inserito nella finestra coerentemente con il layout della lastra dei contenuti. Inoltre all'interno pannelli interni se ne possono inserire altri con loro layout e così via, come nel gioco delle scatole cinesi. Il numero di soluzioni diverse sono praticamente infinite.

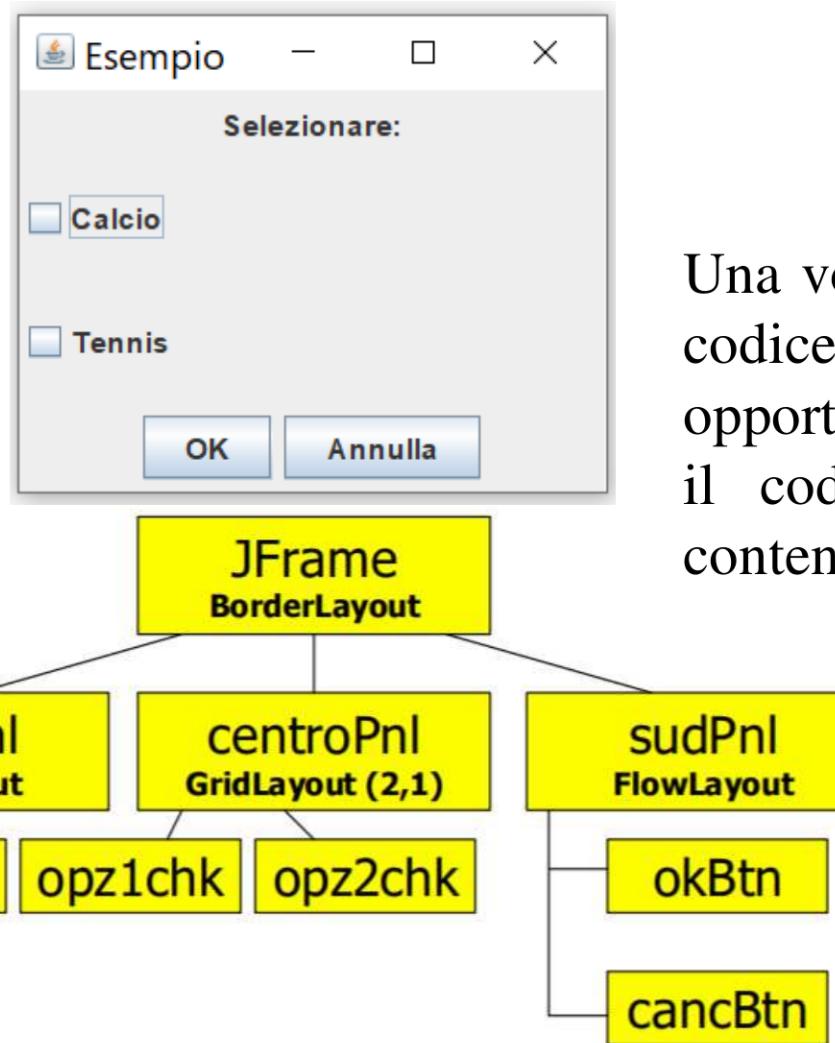
Progettazione top down di interfacce grafiche

Nella progettazione delle GUI si può ricorrere ad un approccio top-down, descrivendo l'insieme dei componenti a partire dal componente più esterno per poi procedere a mano a mano verso quelli più interni. Si può sviluppare una GUI come quella dell'esempio che segue seguendo questa procedura:

1. **Si definisce il tipo di top level container su cui si vuole lavorare (tipicamente un JFrame).**
2. **Si assegna un layout manager al content pane del JFrame, in modo da suddividerne la superficie in aree più piccole.**
3. **Per ogni area messa a disposizione dal layout manager è possibile definire un nuovo JPanel. Ogni sotto pannello può utilizzare un layout manager differente.**
4. **Ogni pannello identificato nel terzo passaggio può essere sviluppato ulteriormente, creando al suo interno ulteriori pannelli o disponendo dei controlli**

Il risultato della progettazione può essere rappresentato con un albero della GUI. L'obiettivo di tale albero è mostrare come i Container (*a partire dalla lastra dei contenuti*) sono stati suddivisi per inserire i componenti o altri Container.

Ogni componente (o Container) è rappresentato da un nodo i cui figli sono i componenti (o i Container) contenuti all'interno e il padre è il componente che lo contiene.



Una volta conclusa la fase progettuale, si può passare a scrivere il codice relativo all'interfaccia: in questo secondo momento, è opportuno adottare un approccio **bottom up**, realizzando dapprima il codice relativo ai componenti atomici, quindi quello dei contenitori e infine quello del JFrame

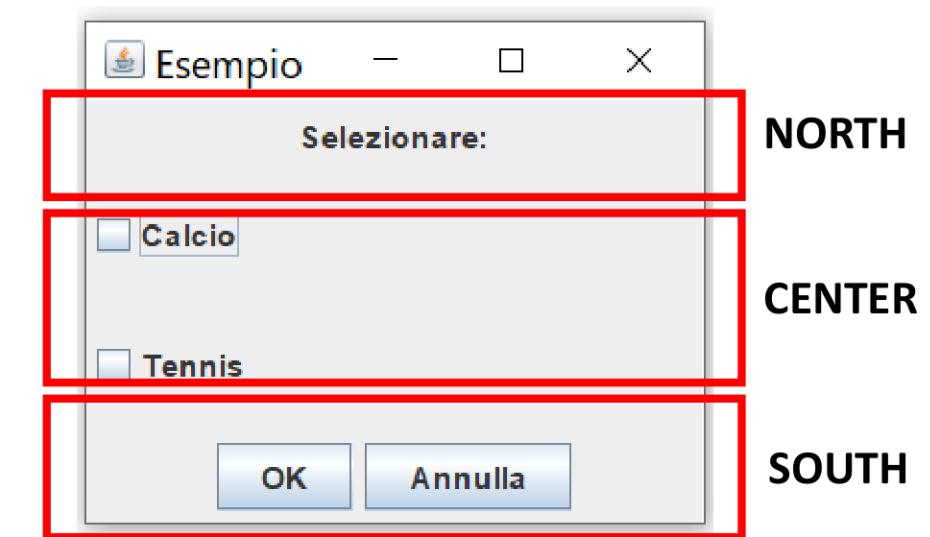
```

public class MioFrame extends JFrame {
    JPanel pannelloNord = new JPanel();
    JPanel pannelloCentro = new JPanel();
    JPanel pannelloSud = new JPanel();
    JLabel infoLabel = new JLabel("Selezionare:");
    JCheckBox opzioneCalcio = new JCheckBox("Calcio");
    JCheckBox opzioneTennis = new JCheckBox("Tennis");
    JButton okButton=new JButton("OK");
    JButton cancButton=new JButton("Annulla");

    public MioFrame() {
        super("Esempio");
        //lavoro sul pannello centrale
        pannelloCentro.setLayout(new GridLayout(2,1));
        pannelloCentro.add(opzioneCalcio);
        pannelloCentro.add(opzioneTennis);
        pannelloNord.add(infoLabel); //lavoro sul pannello NORD
        pannelloSud.add(okButton); //lavoro sul pannello SUD
        pannelloSud.add(cancButton);
        //lavoro sul pannello CONTAINER ROOT DEL FRAME
        getContentPane().add(pannelloNord,BorderLayout.NORTH);
        getContentPane().add(pannelloCentro,BorderLayout.CENTER);
        getContentPane().add(pannelloSud,BorderLayout.SOUTH);
        this.setSize(250, 200);
        //le due istruzioni successive hanno lo scopo di centrare la finestra sullo schermo.
        Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
        setLocation(((int) dim.getWidth()-this.getWidth())/2, ((int)dim.getHeight()-this.getHeight())/2);
        setVisible(true);
    } //costruct
}

```

Un Frame più complesso



NORTH

CENTER

SOUTH