Un approccio pratico a Java Swing e MVC

Sommario

Java Swing

Questa è una guida molto breve e semplificata, non copre tutto su Java Swing, serve solo come introduzione per semplificare il lavoro per eventuali progetti.

Wireframe

Per sviluppare un'interfaccia grafica (per un sito web, un'applicazione, un gioco etc…) è utile disegnare un **wireframe** fatto di **rettangoli**, **testo** e **icone** come quello in Figura 1.

Il **wireframe** serve perché è difficile progettare un'interfaccia **intuitiva** e **funzionale**. Una volta progettata l'interfaccia **scrivere il codice è semplice**.

Proviamo a implementare un esempio

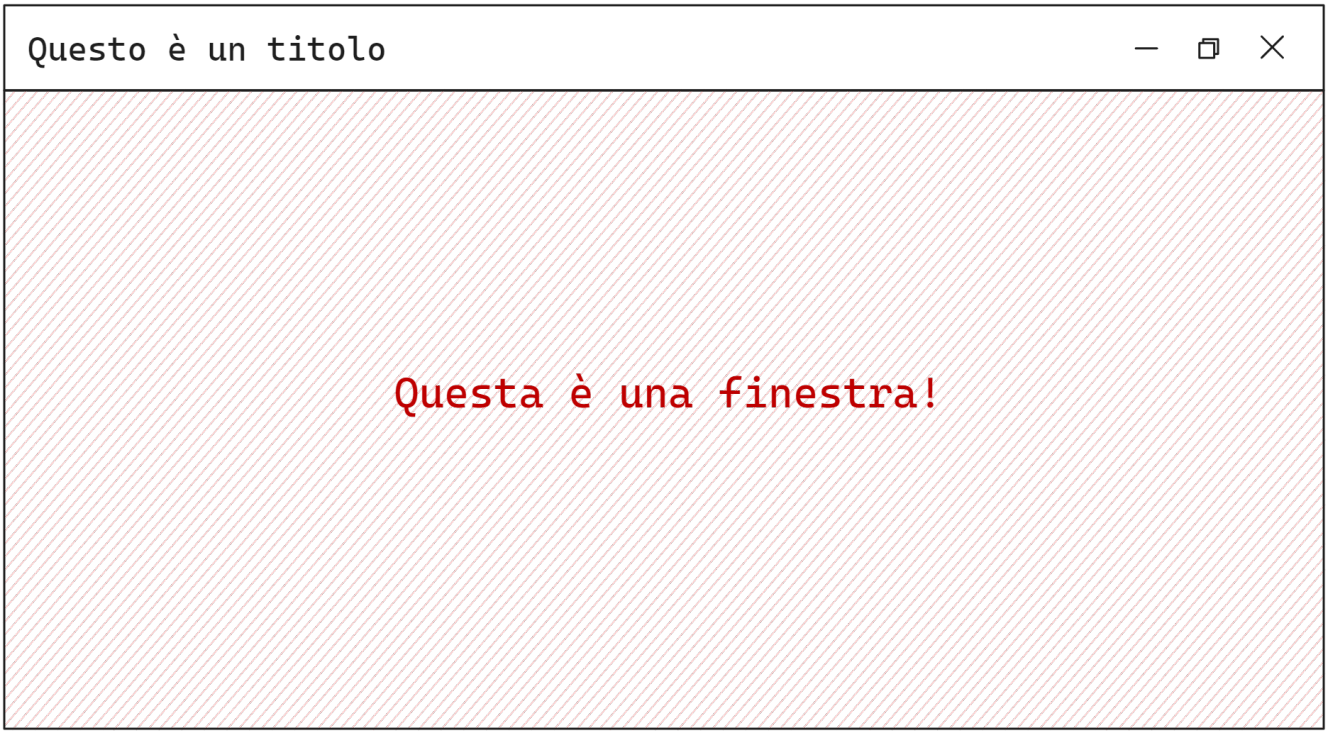


Figura 1: esempio di wireframe disegnato con excalidraw (doc)

Implementare un wireframe

Il codice completo che implementa il wireframe in Figura 1 è **in fondo alla spiegazione**

Creare la finestra con JFrame (doc)

```
JFrame frame = new JFrame("Questo \u00E8 un titolo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

frame.setSize(640, 400);
frame.setVisible(true);
```

Il costruttore `JFrame(String title)` imposta il titolo della finestra

- `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)` termina il programma quando la finestra viene chiusa
- `setVisible(true)` rende la finestra visibile

Aggiungere contenuti alla finestra con JPanel (doc)

```
JPanel panel = new JPanel();
JLabel label = new JLabel("Questa \u00E8 una finestra!");

panel.add(label);
frame.add(panel);
```

Sia `JFrame` sia `JPanel` sono `java.awt.Container`, quindi possiamo aggiungere contenuto (testo, immagini, pulsanti etc…) al loro interno tramite `add(Component comp)`.

- il `JPanel` aggiunto al frame occuperà l'intero spazio
- il contenuto viene aggiunto al `JPanel`
- `JLabel` serve a visualizzare testo (`"Questa è una finstra"` Figura 1)

Mancano ancora un po' di cose: il testo non è centrato, non ci sono i colori etc…

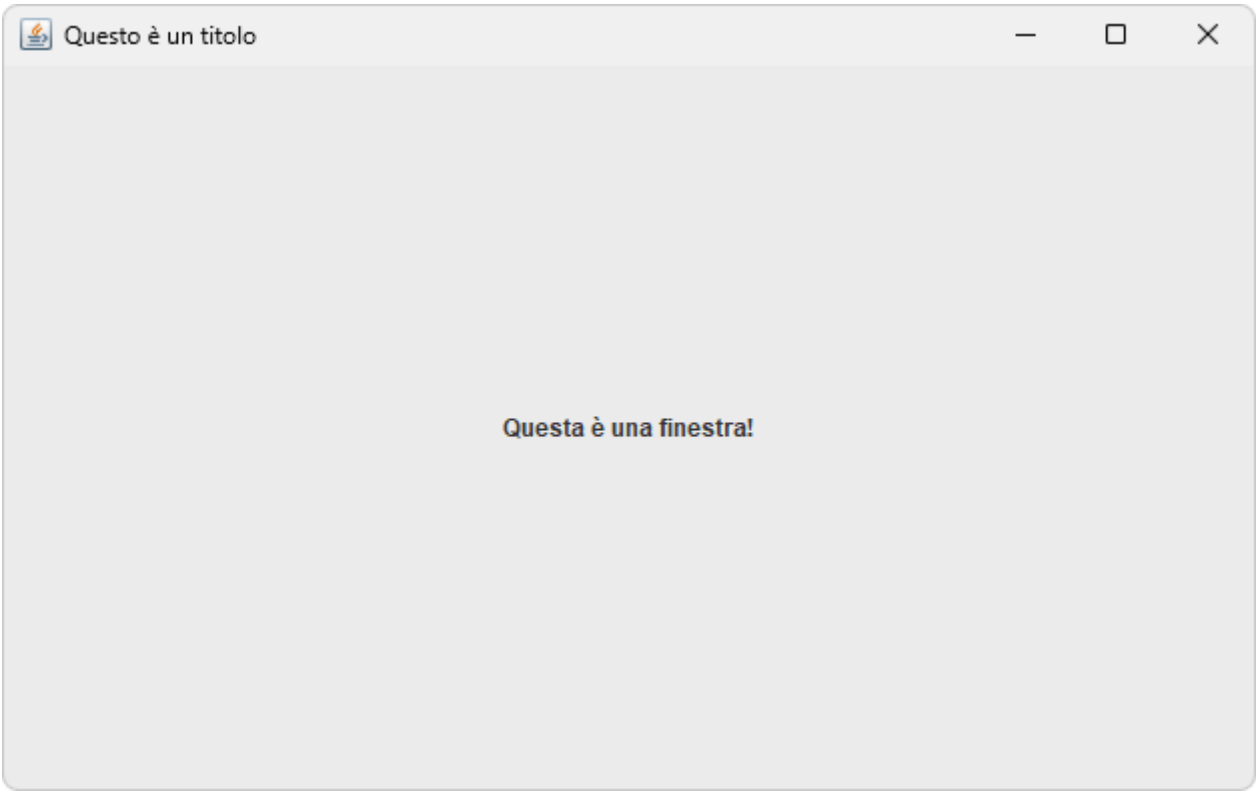
Centrare un elemento in un JPanel con GridBagLayout (doc)

```
JPanel panel = new JPanel(new GridBagLayout());
```

Il costruttore `JPanel(LayoutManager layout)` permette di specificare una **strategia** per **posizionare** e **dimensionare** il contenuto di un `JPanel` in **automatico**:
- non bisogna calcolare a mano `x`, `y`, `width` e `height` dei componenti, lo fa il `LayoutManager`
- funziona anche quando la **finestra viene ridimensionata**

> **i Nota**
>
> `LayoutManager` è un esempio di Strategy Pattern (doc)

Per **centrare un elemento** in un panel si usa un `GridBagLayout`



Colori, font e icone

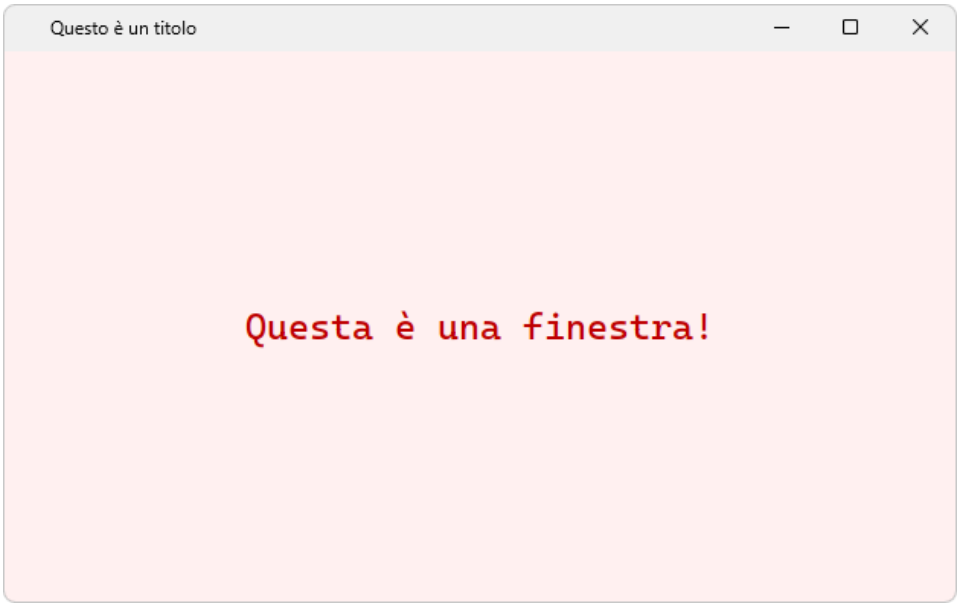
```
frame.setIconImage(new BufferedImage(1, 1, BufferedImage.TYPE_INT_ARGB));

JPanel panel = new JPanel(new GridBagLayout());
panel.setBackground(new Color(255, 240, 240));

JLabel label = new JLabel("Questa \u00E8 una finestra!");
label.setForeground(new Color(190, 0, 0));
label.setFont(new Font("Cascadia Code", Font.PLAIN, 24));
```

Nel **wireframe** in Figura 1 non c'era nessun'icona in alto a sinistra: per "levarla" ho creato un'immagine vuota di **1px** per **1px**

```
new BufferedImage(1, 1, BufferedImage.TYPE_INT_ARGB)
```

> **i Nota**
>
> Il font `"Cascadia Code"` non è installato di default, provate anche con altri font

Ora abbiamo una finestra che rispetta il **wireframe** in Figura 1



Codice finale

```java
import java.awt.Color;
import java.awt.Font;
import java.awt.GridBagLayout;
import java.awt.image.BufferedImage;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class App {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Questo \u00E8 un titolo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setIconImage(new BufferedImage(1, 1, BufferedImage.TYPE_INT_ARGB));

        JPanel panel = new JPanel(new GridBagLayout());
        panel.setBackground(new Color(255, 240, 240));

        JLabel label = new JLabel("Questa \u00E8 una finestra!");
        label.setForeground(new Color(190, 0, 0));
        label.setFont(new Font("Cascadia Code", Font.PLAIN, 24));

        panel.add(label);
        frame.add(panel);

        frame.setSize(640, 400);
        frame.setVisible(true);
    }
}
```

Interfacce con Swing senza variabili d'appoggio

Java mette a disposizione uno strumento che si chiama **instance initialization block**, un blocco di codice che viene eseguito dopo aver invocato il costruttore. È specialmente comodo quando si istanziano **classi anonime**.

```
class Persona {
  String nome;

  public Persona(String nome)  {
    this.nome = nome;
  }
}

class Main {
  public static void main(String[] args) {
    Persona dottorRossi = new Persona("Rossi") {
      {
        nome = "Dr. " + nome;
      }
    };

    System.out.println(dottorRossi.nome); // Dr. Rossi
  }
}
```

Possiamo sfruttare questa strategia per riscrivere il codice di prima senza variabili.

```
import java.awt.Color;
import java.awt.Font;
import java.awt.GridBagLayout;
import java.awt.image.BufferedImage;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class App {
    public static void main(String[] args) {
        new JFrame("Questo \u00E8 un titolo") {
            {
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setIconImage(new BufferedImage(1, 1, BufferedImage.TYPE_INT_ARGB));

                add(new JPanel(new GridBagLayout()) {
                    {
                        setBackground(new Color(255, 240, 240));

                        add(new JLabel("Questa \u00E8 una finestra!") {
                            {
                                setForeground(new Color(190, 0, 0));
                                setFont(new Font("Cascadia Code", Font.PLAIN, 24));
                            }
                        });
                    }
                });

                setSize(640, 400);
                setVisible(true);
            }
        };
    }
}
```

Layout Manager

Il costruttore `JPanel(LayoutManager layout)` permette di specificare una **strategia** per **posizionare** e **dimensionare** il contenuto di un `JPanel` in **automatico**:
- non bisogna calcolare a mano `x`, `y`, `width` e `height` dei componenti, lo fa il `LayoutManager`
- funziona anche quando la **finestra viene ridimensionata**

> **i Nota**
>
> `LayoutManager` è un esempio di Strategy Pattern (doc)

BorderLayout (doc)

Supponiamo di voler implementare questo wireframe

Figura 2: caso d'uso di un `BorderLayout`

Abbiamo un rettangolo con le statistiche in alto, e il restante spazio è occupato da un rettangolo centrale con un pulsante.

Barra delle statistiche e menu di gioco

```
frame.add(new JPanel(new BorderLayout(10, 10)) {
    {
        setBackground(new Color(240, 255, 240)); // verdignolo
        setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        add(
          new JPanel() {{ setBackground(new Color(220, 220, 255)); }},  // bluastro
          BorderLayout.NORTH
        );

        add(
          new JPanel() {{ setBackground(new Color(220, 220, 255)); }},
          BorderLayout.CENTER
        );
    }
});
```

Il costruttore `BorderLayout(int vgap, int hgap)` imposta uno “spazio” verticale e orizzontale fra due componenti.

Con `setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10))` impostiamo un bordo trasparente (per lasciare uno spazio dal bordo della finestra)

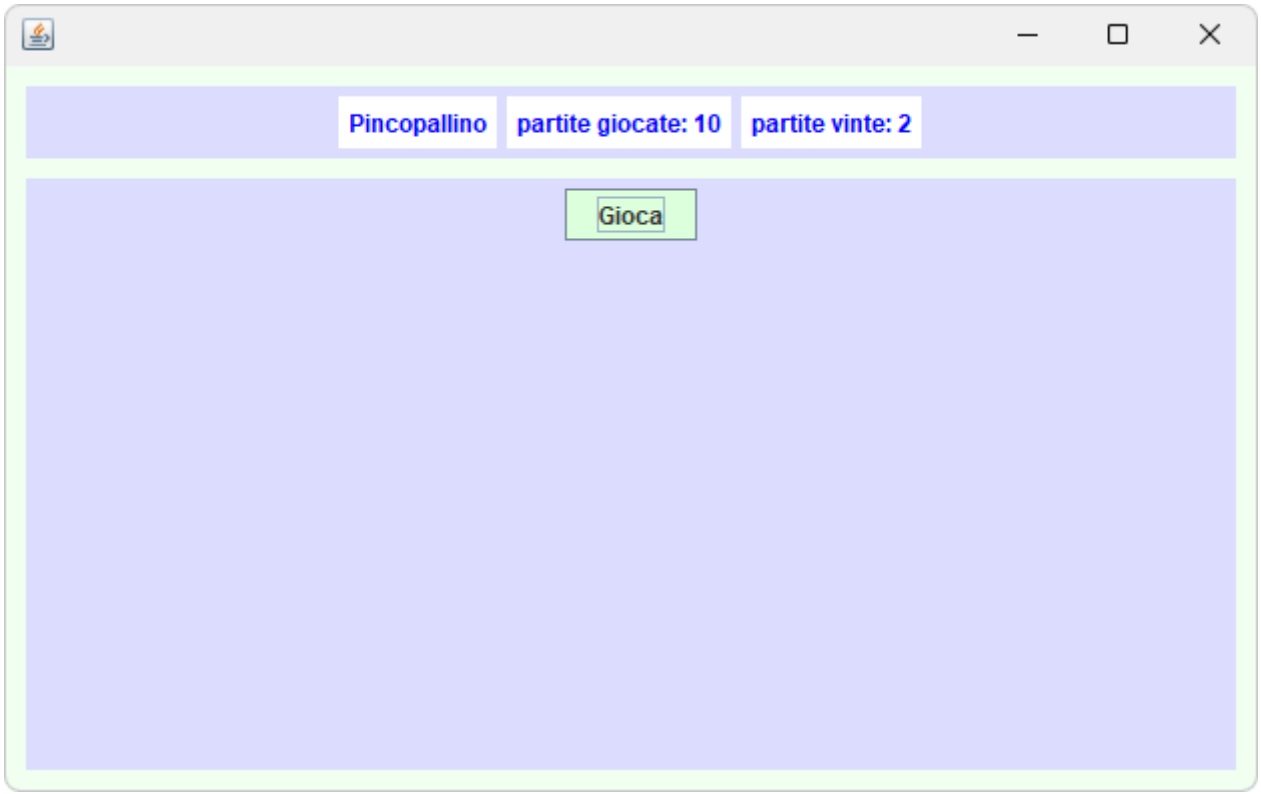
Quando aggiungo un elemento ad un container, posso specificare come deve essere trattato tramite il metodo `add(Component comp, Object constraints)`: in base al layout del container, `Object constraints` avrà un significato diverso.

> **i Nota**
>
> `BorderFactory` è un esempio di Factory Pattern (doc)

Aggiungere uno sfondo ai `JLabel`

```
frame.add(new JPanel(new BorderLayout(10, 10)) {
    {
        setBackground(new Color(240, 255, 240));
        setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        add(new JPanel() {
            {
                setBackground(new Color(220, 220, 255));

                String[] labels = {"Pincopallino", "partite giocate: 10", "partite vinte: 2"};

                for (String label : labels)
                    add(new JLabel(label) {
                        {
                            setForeground(Color.BLUE);
                            setBackground(Color.WHITE);
                            setOpaque(true);
                            setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
                        }
                    });
            }
        }, BorderLayout.NORTH);

        add(new JPanel() {
            {
                setBackground(new Color(220, 220, 255));
                add(new JButton("Gioca") {{ setBackground(new Color(220, 255, 220)); }});
            }
        }, BorderLayout.CENTER);
    }
});
```

Nota interessante: di default, lo sfondo di un `JLabel` è trasparente, per renderlo visibile bisogna usare `setOpaque(true)`



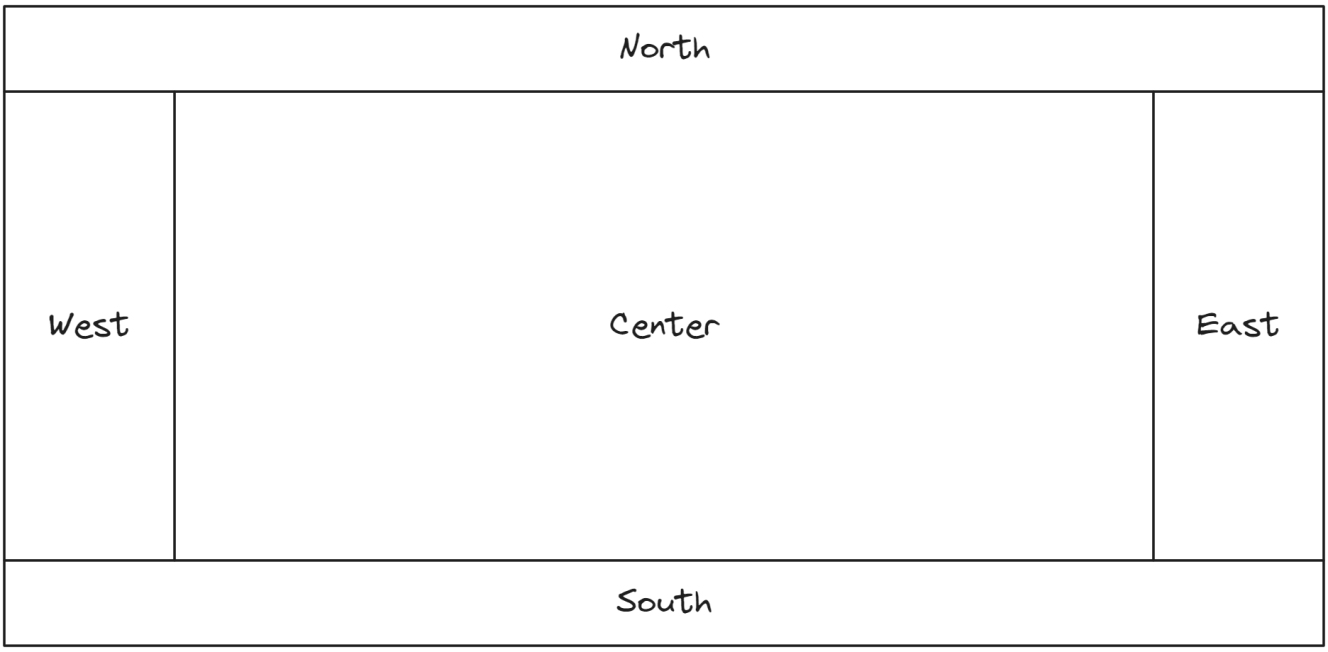
Come funziona il `BorderLayout` in generale?

```
new JPanel(new BorderLayout()) {
  {
    add(new JPanel(), BorderLayout.CENTER);
    add(new JPanel(), BorderLayout.NORTH);
    add(new JPanel(), BorderLayout.SOUTH);
    add(new JPanel(), BorderLayout.WEST);
    add(new JPanel(), BorderLayout.EAST);
  }
}
```

Il `BorderLayout` permette di specificare in quale posizione mettere un componente, secondo certe regole:

- il componente `CENTER` occuperà tutto lo spazio possibile
- i componenti `NORTH` e `SOUTH` avranno larghezza massima (indipendentemente dalla larghezza impostata) e avranno altezza minima, o, se impostata, l'altezza impostata
- i componenti `WEST` e `EAST` avranno altezza massima (indipendentemente dall'altezza impostata) e avranno larghezza minima, o, se impostata, la larghezza impostata

Il costruttore `BorderLayout(int vgap, int hgap)` imposta uno "spazio" verticale e orizzontale fra due componenti.



CardLayout (doc)

Menu, impostazioni e partita (come cambiare da una vista all'altra)

Il `CardLayout` è molto utile quando abbiamo più viste (menu principale, impostazioni, selezione partita etc…)

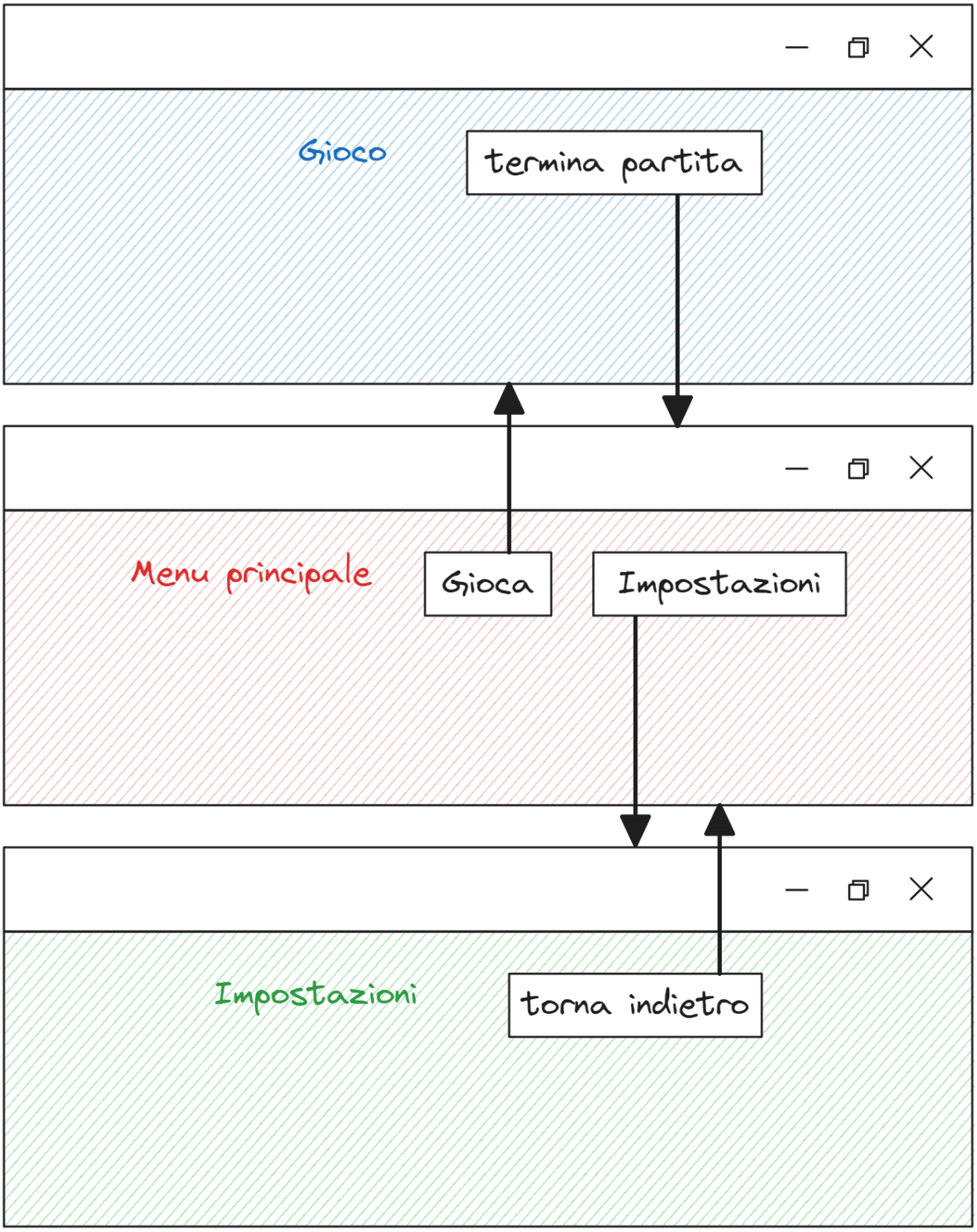


Figura 3: caso d'uso di un `CardLayout`

```
enum Screens {
  Menu, Settings, Game
}

// etc...

frame.add(new JPanel(new CardLayout()) {
  {
    add(new JPanel(), Screens.Menu.name());
    add(new JPanel(), Screens.Settings.name());
    add(new JPanel(), Screens.Game.name());
  }
});
```

L'idea sarebbe quella di associare ad ogni componente una `String` che lo identifica. In questo caso usiamo un `enum` per non sbagliare a scrivere il nome del componente.

In questo esempio, verrà visualizzato solo il `JPanel` associato a `"Menu"`, vediamo come poter cambiare da un `JPanel` all'altro.

```
enum Screens {
    Menu, Settings, Game;

    static void show(JPanel panel, Screens screen) {
        CardLayout layout = (CardLayout) panel.getLayout();
        layout.show(panel, screen.name());
    }

}
frame.add(new JPanel(new CardLayout()) {
  {
    JPanel panel = this;

    add(new JPanel() {
      {
        add(new JLabel("Menu principale"));
        add(new JButton("Gioca") {{
          addActionListener(e -> Screens.show(panel, Screens.Game));
        }});
        add(new JButton("Impostazioni") {{
          addActionListener(e -> Screens.show(panel, Screens.Settings));
        }});
      }
    }, Screens.Menu.name());

    add(new JPanel() {
      {
        add(new JLabel("Impostazioni"));
        add(new JButton("torna indietro") {{
          addActionListener(e -> Screens.show(panel, Screens.Menu));
        }});
      }
    }, Screens.Settings.name());

    add(new JPanel() {
        {
          add(new JLabel("Gioco"));
          add(new JButton("termina partita") {{
            addActionListener(e -> Screens.show(panel, Screens.Menu));
          }});
        }
    }, Screens.Game.name());
  }
});
```

GridLayout (doc)

Non è un layout particolarmente complesso: permette di specificare il numero di righe, il numero di colonne, e lo spazio fra due componenti.

```
frame.add(new JPanel(new GridLayout(4, 3, 10, 10)) {
  {
    setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

    for (int digit = 0; digit <= 9; digit++)
        add(new JButton(String.valueOf(digit)));
    add(new JButton("+"));
    add(new JButton("="));
  }
});
```

GridBagLayout (doc)

Il layout più flessibile

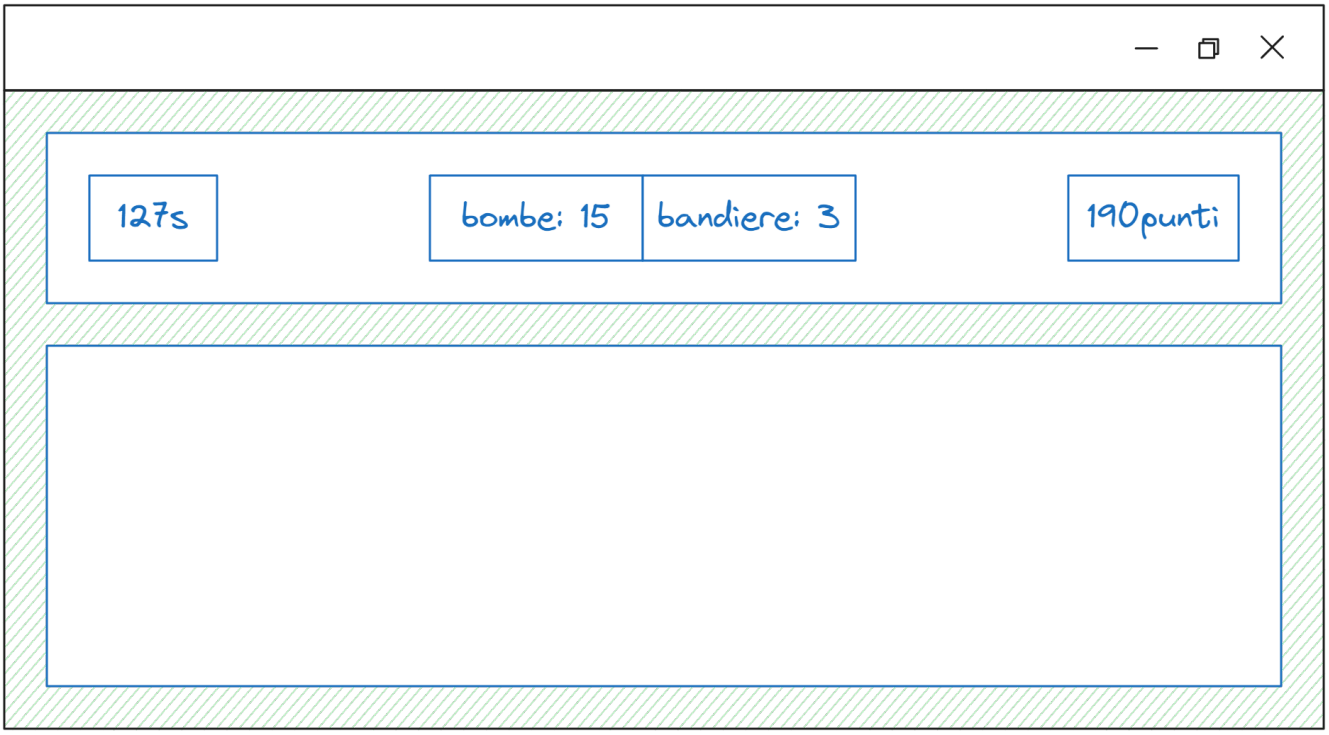


Figura 4: esempio di wireframe per il gioco “Minesweeper”

MVC