EBook Gratuito

APPRENDIMENTO swing

Free unaffiliated eBook created from
Stack Overflow contributors.

#swing

Sommario

Di

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: swing

It is an unofficial and free swing ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official swing.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capitolo 1: Iniziare con lo swing

Osservazioni

Swing è stato sostituito da JavaFX . In genere, Oracle consiglia di sviluppare nuove applicazioni con JavaFX. Ancora: Swing sarà supportato in Java per il prossimo futuro. JavaFX si integra bene anche con Swing, per consentire applicazioni di transizione senza problemi.

Si consiglia vivamente di avere la maggior parte dei componenti Swing sul thread di invio eventi. È facile dimenticare di raggruppare la configurazione della GUI in una chiamata `invokeLater` . Dalla documentazione Java:

> Il codice di gestione degli eventi Swing viene eseguito su un thread speciale noto come thread di invio degli eventi. La maggior parte del codice che richiama i metodi Swing viene eseguito anche su questo thread. Ciò è necessario perché la maggior parte dei metodi degli oggetti Swing non sono "thread-safe": il loro richiamo da più thread rischia di interferire con i thread o errori di coerenza della memoria. Alcuni metodi del componente Swing sono etichettati come "thread safe" nella specifica API; questi possono essere tranquillamente richiamati da qualsiasi thread. Tutti gli altri metodi del componente Swing devono essere richiamati dal thread di invio dell'evento. I programmi che ignorano questa regola potrebbero funzionare correttamente la maggior parte del tempo, ma sono soggetti a errori imprevedibili e difficili da riprodurre.

Inoltre, a meno che non sia una buona ragione, assicurati *sempre di* aver chiamato `setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE)` altrimenti potresti dover gestire una perdita di memoria se ti dimentichi di distruggere la JVM.

Examples

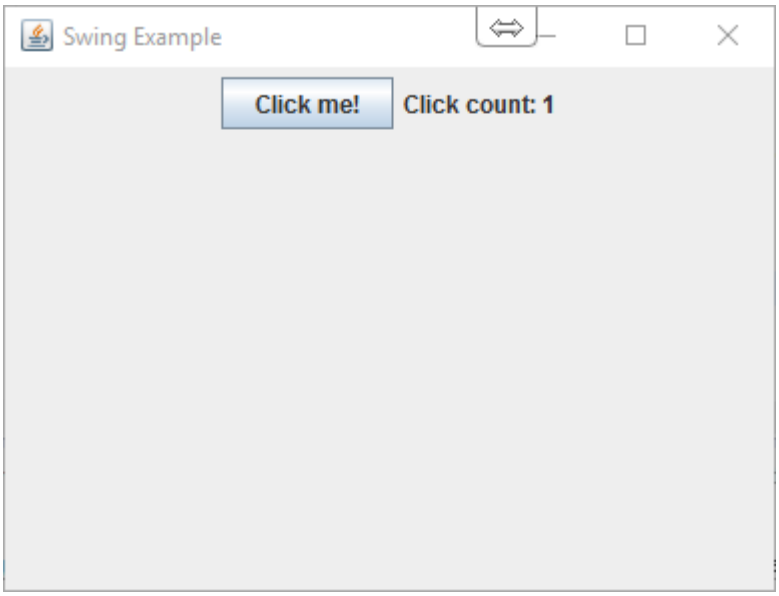
Incremento con un pulsante

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;

/**
 * A very simple Swing example.
 */
public class SwingExample {
    /**
     * The number of times the user has clicked the button.
     */
    private long clickCount;

    /**
     * The main method: starting point of this application.
```

```
     *
     * @param arguments the unused command-line arguments.
     */
    public static void main(final String[] arguments) {
        new SwingExample().run();
    }

    /**
     * Schedule a job for the event-dispatching thread: create and show this
     * application's GUI.
     */
    private void run() {
        SwingUtilities.invokeLater(this::createAndShowGui);
    }

    /**
     * Create the simple GUI for this application and make it visible.
     */
    private void createAndShowGui() {
        // Create the frame and make sure the application exits when the user closes
        // the frame.
        JFrame mainFrame = new JFrame("Counter");
        mainFrame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        // Add a simple button and label.
        JPanel panel = new JPanel();
        JButton button = new JButton("Click me!");
        JLabel label = new JLabel("Click count: " + clickCount);
        panel.add(button);
        panel.add(label);
        mainFrame.getContentPane().add(panel);

        // Add an action listener to the button to increment the count displayed by
        // the label.
        button.addActionListener(actionEvent -> {
            clickCount++;
            label.setText("Click count: " + clickCount);
        });

        // Size the frame.
        mainFrame.setBounds(80, 60, 400, 300);
        //Center on screen
        mainFrame.setLocationRelativeTo(null);
        //Display frame
        mainFrame.setVisible(true);
    }
}
```

Risultato

Come il pulsante con l'etichetta "Click me!" viene premuto il conteggio dei clic aumenterà di uno:



"Ciao mondo!" sul titolo della finestra con lambda

```
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;

public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Hello World!");
            frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
            frame.setSize(200, 100);
            frame.setVisible(true);
        });
    }
}
```

All'interno del metodo `main` :
Sulla prima riga viene chiamato `SwingUtilities.invokeLater` viene `SwingUtilities.invokeLater` un'espressione lambda con un blocco di codice `() -> {...}` . Esegue l'espressione lambda passata sull'EDT, che è l'abbreviazione di Event Dispatch Thread, invece del thread principale. Questo è necessario, perché all'interno del blocco di codice dell'espressione lambda, ci saranno componenti Swing che verranno creati e aggiornati.

All'interno del blocco di codice dell'espressione lambda:
Sulla prima riga, una nuova istanza `JFrame` chiamata `frame` viene creata usando il `new JFrame("Hello World!")` . Questo crea un'istanza di finestra con "Hello World!" sul suo titolo. Successivamente sulla seconda riga il `frame` è configurato su `EXIT_ON_CLOSE` . Altrimenti la finestra verrà chiusa, ma l'esecuzione del programma rimarrà attiva. La terza riga configura l'istanza del `frame` con una larghezza di 200 pixel e un'altezza di 100 pixel utilizzando il metodo `setSize` . Fino ad ora l'esecuzione non mostrerà nulla. Solo dopo aver chiamato `setVisible(true)` sulla quarta riga, l'istanza del `frame` è configurata per apparire sullo schermo.

"Ciao mondo!" sul titolo della finestra con compatibilità

Usando `java.lang.Runnable` facciamo il nostro "Hello World!" esempio disponibile per gli utenti Java con versioni risalenti alla versione 1.2:

```
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;

public class Main {
    public static void main(String[] args){
        SwingUtilities.invokeLater(new Runnable(){

            @Override
            public void run(){
                JFrame frame = new JFrame("Hello World!");
                frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
                frame.setSize(200, 100);
                frame.setVisible(true);
            }
        });
    }
}
```

Leggi Iniziare con lo swing online: https://riptutorial.com/it/swing/topic/2191/iniziare-con-lo-swing

Capitolo 2: Gestione del layout

Examples

Layout del bordo

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

JPanel root = new JPanel(new BorderLayout());

root.add(new JButton("East"), EAST);
root.add(new JButton("West"), WEST);
root.add(new JButton("North"), NORTH);
root.add(new JButton("South"), SOUTH);
root.add(new JButton("Center"), CENTER);

JFrame frame = new JFrame();
frame.setContentPane(root);
frame.pack();
frame.setVisible(true);
```

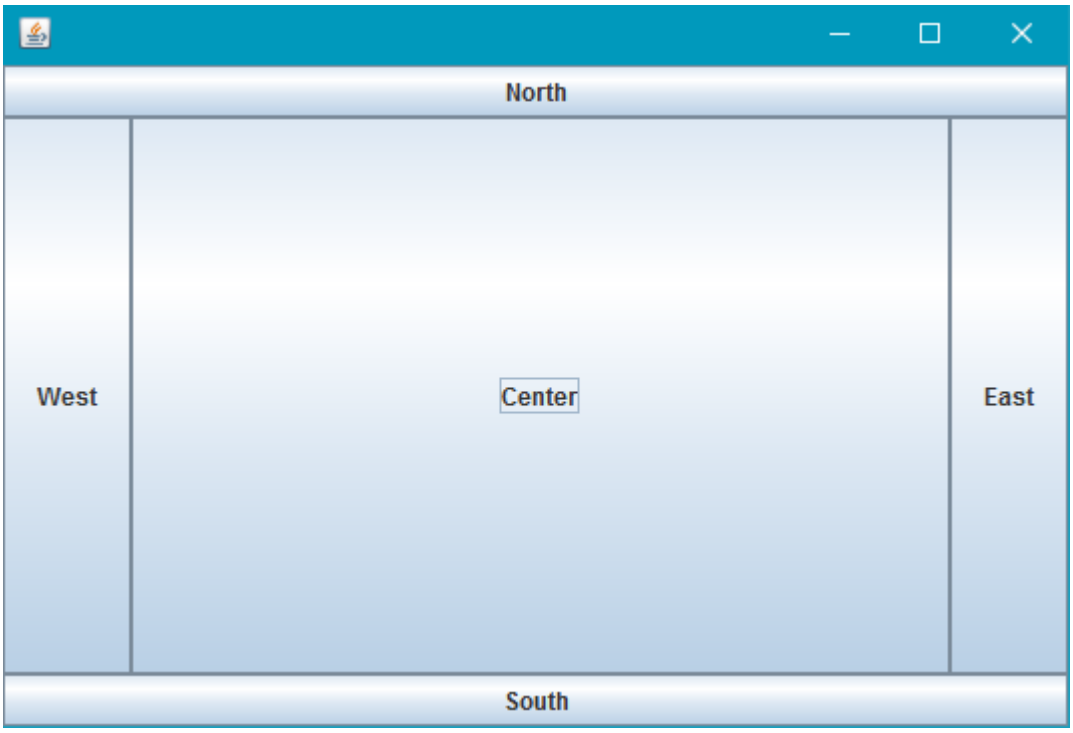
Il layout dei bordi è uno dei più semplici gestori di layout. Il modo di usare un gestore di layout è impostare il gestore di un `JPanel` .

Gli slot di Border Layout seguono le seguenti regole:

- Nord e Sud: altezza preferita
- Est e Ovest: larghezza preferita
- Centro: spazio rimanente massimo

`BorderLayout` slot di `BorderLayout` possono essere vuoti. Il gestore del layout compenserà automaticamente eventuali spazi vuoti, ridimensionandoli quando necessario.

Ecco come appare questo esempio:



Layout del flusso

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

public class FlowExample {
    public static void main(String[] args){
        SwingUtilities.invokeLater(new Runnable(){

            @Override
            public void run(){
                JPanel panel = new JPanel();
                panel.setLayout(new FlowLayout());

                panel.add(new JButton("One"));
                panel.add(new JButton("Two"));
                panel.add(new JButton("Three"));
                panel.add(new JButton("Four"));
                panel.add(new JButton("Five"));

                JFrame frame = new JFrame();
                frame.setContentPane(Panel);
                frame.pack();
                frame.setVisible(true);
            }
        });
    }
}
```

Il layout del flusso è il gestore di layout più semplice che Swing ha da offrire. Il layout del flusso cerca di mettere tutto su una riga e, se il layout supera la larghezza, avvolge la linea. L'ordine è specificato dall'ordine in cui aggiungi componenti al tuo pannello.

Screenshots:

Layout della griglia

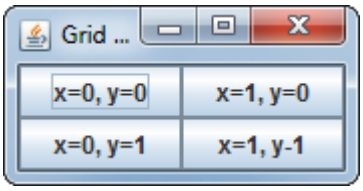
`GridLayout` consente di disporre i componenti sotto forma di griglia.

Si passa il numero di righe e colonne che si desidera che la griglia abbia al `GridLayout` di `GridLayout` , ad esempio il `new GridLayout(3, 2)` creerà un `GridLayout` con 3 righe e 2 colonne.

Quando aggiungi componenti a un contenitore con `GridLayout` , i componenti verranno aggiunti riga per riga, da sinistra a destra:

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

public class Example {
    public static void main(String[] args){
        SwingUtilities.invokeLater(Example::createAndShowJFrame);
    }

    private static void createAndShowJFrame(){
        JFrame jFrame = new JFrame("Grid Layout Example");

        // Create layout and add buttons to show restraints
        JPanel jPanel = new JPanel(new GridLayout(2, 2));
        jPanel.add(new JButton("x=0, y=0"));
        jPanel.add(new JButton("x=1, y=0"));
        jPanel.add(new JButton("x=0, y=1"));
        jPanel.add(new JButton("x=1, y-1"));

        jFrame.setContentPane(jPanel);
        jFrame.pack();
        jFrame.setLocationRelativeTo(null);
        jFrame.setVisible(true);
    }
}
```

Questo crea e mostra una `JFrame` che assomiglia a:



Una descrizione più dettagliata è disponibile: GridLayout

Leggi Gestione del layout online: https://riptutorial.com/it/swing/topic/5417/gestione-del-layout

Capitolo 3: Grafica

Examples

Usando la classe Graphics

Intro

La classe `Graphics` ti permette di disegnare su componenti java come un `Jpanel` , può essere usato per disegnare stringhe, linee, forme e immagini. Ciò avviene *sovrascrivendo* il `paintComponent(Graphics g)` del `JComponent` si sta disegnando utilizzando l'oggetto `Graphics` ricevuto come argomento per eseguire il disegno:

`Board` classe

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

public class Board extends JPanel{

    public Board() {
        setBackground(Color.WHITE);
    }

    @override
    public Dimension getPreferredSize() {
        return new Dimension(400, 400);
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        // draws a line diagonally across the screen
        g.drawLine(0, 0, 400, 400);
        // draws a rectangle around "hello there!"
        g.drawRect(140, 180, 115, 25);
    }
}
```

classe wrapper `DrawingCanvas`

```
import javax.swing.*;

public class DrawingCanvas extends JFrame {

    public DrawingCanvas() {

        Board board = new Board();

        add(board); // adds the Board to our JFrame
```

```
        pack(); // sets JFrame dimension to contain subcomponents

        setResizable(false);
        setTitle("Graphics Test");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        setLocationRelativeTo(null); // centers window on screen
    }

    public static void main(String[] args) {
        DrawingCanvas canvas = new DrawingCanvas();
        canvas.setVisible(true);
    }
}
```

Colori

Per disegnare forme con colori diversi è necessario impostare il colore dell'oggetto `Graphics` prima di ogni chiamata di disegno utilizzando `setColor` :

```
g.setColor(Color.BLUE);  // draws a blue square
g.fillRect(10, 110, 100, 100);

g.setColor(Color.RED);  // draws a red circle
g.fillOval(10, 10, 100, 100);

g.setColor(Color.GREEN);  // draws a green triangle
int[] xPoints = {0, 200, 100};
int[] yPoints = {100, 100, 280};
g.fillPolygon(xPoints, yPoints, 3);
```

Disegno di immagini

Le immagini possono essere disegnate su un `JComponent` usando il metodo `drawImage` della classe `Graphics` :

caricamento di un'immagine

```
BufferedImage img;
try {
    img = ImageIO.read(new File("stackoverflow.jpg"));
} catch (IOException e) {
    throw new RuntimeException("Could not load image", e);
}
```

disegnare l'immagine

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
```

```
    int x = 0;
    int y = 0;

    g.drawImage(img, x, y, this);
}
```

La `x` e `y` specificano la posizione dei **top-sinistra** dell'immagine.

Utilizzo del metodo Repaint per creare un'animazione di base

La classe MyFrame estende JFrame e contiene anche il metodo principale

```
import javax.swing.JFrame;


public class MyFrame extends JFrame{

    //main method called on startup
    public static void main(String[] args) throws InterruptedException {

        //creates a frame window
        MyFrame frame = new MyFrame();

        //very basic game loop where the graphics are re-rendered
        while(true){
            frame.getPanel().repaint();

            //The program waits a while before rerendering
            Thread.sleep(12);
        }
    }

    //the MyPanel is the other class and it extends JPanel
    private MyPanel panel;

    //constructor that sets some basic staring values
    public MyFrame(){
        this.setSize(500, 500);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //creates the MyPanel with paramaters of x=0 and y=0
        panel = new MyPanel(0,0);
        //adds the panel (which is a JComponent because it extends JPanel)
        //into the frame
        this.add(panel);
        //shows the frame window
        this.setVisible(true);
    }

    //gets the panel
    public MyPanel getPanel(){
        return panel;
    }

}
```

La classe MyPanel che estende JPanel e ha il metodo paintComponent

```
import java.awt.Graphics;
import javax.swing.JPanel;


public class MyPanel extends JPanel{

    //two int variables to store the x and y coordinate
    private int x;
    private int y;

    //construcor of the MyPanel class
    public MyPanel(int x, int y){
        this.x = x;
        this.y = y;
    }

    /*the method that deals with the graphics
        this method is called when the component is first loaded,
         when the component is resized and when the repaint() method is
        called for this component
    */
    @Override
    public void paintComponent(Graphics g){
        super.paintComponent(g);

        //changes the x and y varible values
        x++;
        y++;

        //draws a rectangle at the x and y values
        g.fillRect(x, y, 50, 50);
    }

}
```

Leggi Grafica online: https://riptutorial.com/it/swing/topic/5153/grafica

Capitolo 4: JList

Examples

Modifica gli elementi selezionati in una JList

Dato un `JList` come

```
JList myList = new JList(items);
```

gli elementi selezionati nell'elenco possono essere modificati tramite `ListSelectionModel` di `JList` :

```
ListSelectionModel sm = myList.getSelectionModel();
sm.clearSelection();                      // clears the selection
sm.setSelectionInterval(index, index);    // Sets a selection interval
                                          // (single element, in this case)
```

In alternativa, `JList` fornisce anche alcuni metodi utili per manipolare direttamente gli indici selezionati:

```
myList.setSelectionIndex(index);            // sets one selected index
                                            // could be used to define the Default Selection

myList.setSelectedIndices(arrayOfIndexes);  // sets all indexes contained in
                                            // the array as selected
```

Leggi JList online: https://riptutorial.com/it/swing/topic/5413/jlist

Capitolo 5: Layout della griglia

Examples

Come funziona GridLayout

Un `GridLayout` è un gestore di layout che posiziona componenti all'interno di una griglia con dimensioni di cella uguali. È possibile impostare il numero di righe, colonne, il divario orizzontale e il divario verticale utilizzando i seguenti metodi:

- `setRows(int rows)`
- `setColumns(int columns)`
- `setHgap(int hgap)`
- `setVgap(int vgap)`

oppure puoi impostarli con i seguenti costruttori:

- `GridLayout(int rows, int columns)`
- `GridLayout(int rows, int columns, int hgap, int vgap)`

Se il numero di righe o colonne è sconosciuto, è possibile impostare la rispettiva variabile su `0` . Per esempio:

```
new GridLayout(0, 3)
```

Ciò farà sì che `GridLayout` abbia 3 colonne e tutte le righe necessarie.

L'esempio seguente mostra come un `GridLayout` stende componenti con valori diversi per righe, colonne, spazio orizzontale, distanza verticale e dimensioni dello schermo.

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.GridLayout;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSpinner;
import javax.swing.SpinnerNumberModel;
import javax.swing.WindowConstants;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class GridLayoutExample {

    private GridLayout gridLayout;
    private JPanel gridPanel, contentPane;
    private JSpinner rowsSpinner, columnsSpinner, hgapSpinner, vgapSpinner;
```

```
    public void createAndShowGUI() {
        gridLayout = new GridLayout(5, 5, 3, 3);

        gridPanel = new JPanel(gridLayout);

        final ChangeListener rowsColumnsListener = new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                gridLayout.setRows((int) rowsSpinner.getValue());
                gridLayout.setColumns((int) columnsSpinner.getValue());
                fillGrid();
            }
        };

        final ChangeListener gapListener = new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                gridLayout.setHgap((int) hgapSpinner.getValue());
                gridLayout.setVgap((int) vgapSpinner.getValue());
                gridLayout.layoutContainer(gridPanel);
                contentPane.revalidate();
                contentPane.repaint();
            }
        };

        rowsSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getRows(), 1, 10, 1));
        rowsSpinner.addChangeListener(rowsColumnsListener);

        columnsSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getColumns(), 1, 10,
1));
        columnsSpinner.addChangeListener(rowsColumnsListener);

        hgapSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getHgap(), 0, 50, 1));
        hgapSpinner.addChangeListener(gapListener);

        vgapSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getVgap(), 0, 50, 1));
        vgapSpinner.addChangeListener(gapListener);

        JPanel actionPanel = new JPanel();
        actionPanel.add(new JLabel("Rows:"));
        actionPanel.add(rowsSpinner);
        actionPanel.add(Box.createHorizontalStrut(10));
        actionPanel.add(new JLabel("Columns:"));
        actionPanel.add(columnsSpinner);
        actionPanel.add(Box.createHorizontalStrut(10));
        actionPanel.add(new JLabel("Horizontal gap:"));
        actionPanel.add(hgapSpinner);
        actionPanel.add(Box.createHorizontalStrut(10));
        actionPanel.add(new JLabel("Vertical gap:"));
        actionPanel.add(vgapSpinner);

        contentPane = new JPanel(new BorderLayout(0, 10));
        contentPane.add(gridPanel);
        contentPane.add(actionPanel, BorderLayout.SOUTH);

        fillGrid();

        JFrame frame = new JFrame("GridLayout Example");
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.setContentPane(contentPane);
        frame.setSize(640, 480);
```

```
        frame.setLocationByPlatform(true);
        frame.setVisible(true);
    }

    private void fillGrid() {
        gridPanel.removeAll();
        for (int row = 0; row < gridLayout.getRows(); row++) {
            for (int col = 0; col < gridLayout.getColumns(); col++) {
                JLabel label = new JLabel("Row: " + row + " Column: " + col);
                label.setHorizontalAlignment(JLabel.CENTER);
                label.setBorder(BorderFactory.createLineBorder(Color.GRAY));
                gridPanel.add(label);
            }
        }
        contentPane.revalidate();
        contentPane.repaint();
    }

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            @Override
            public void run() {
                new GridLayoutExample().createAndShowGUI();
            }
        });
    }

}
```

Leggi Layout della griglia online: https://riptutorial.com/it/swing/topic/2780/layout-della-griglia

Capitolo 6: Layout GridBag

Sintassi

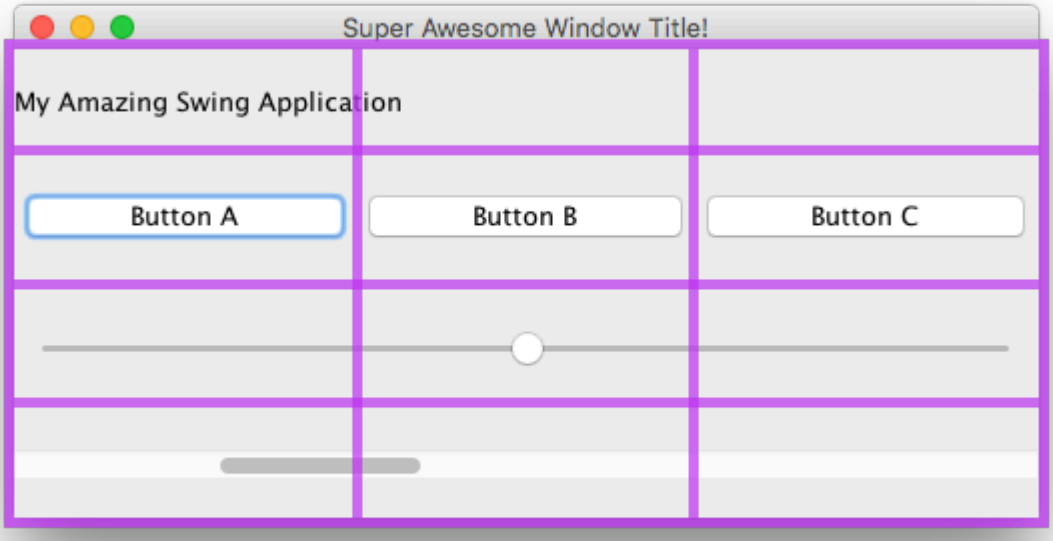
- frame.setLayout (new GridBagLayout ()); // Imposta GridBagLayout per il frame
- pane.setLayout (new GridBagLayout ()); // Imposta GridBagLayout per Panel
- JPanel pane = new JPanel (new GridBagLayout ()); // Imposta GridBagLayout per Panel
- GridBagConstraints c = new GridBagConstraints () // Inizializza GridBagConstraint

Examples

Come funziona GridBagLayout?

I layout vengono utilizzati ogni volta che si desidera che i componenti non vengano semplicemente visualizzati l'uno accanto all'altro. Il `GridBagLayout` è utile, poiché divide la tua finestra in righe e colonne e tu decidi in quale riga e colonna inserire i componenti, nonché quante righe e colonne sono grandi il componente.

Prendiamo questa finestra come esempio. Le linee della griglia sono state contrassegnate per mostrare il layout.



Qui, ho creato 6 componenti, disposti utilizzando un GridBagLayout.

Componente	Posizione	Taglia
`JLabel` : "My Amazing Swing Application"	`0, 0`	`3, 1`

Componente	Posizione	Taglia
`JButton` : "Pulsante A"	`0, 1`	`1, 1`
`JButton` : "Pulsante B"	`1, 1`	`1, 1`
`JButton` : "Pulsante C"	`2, 1`	`1, 1`
`JSlider`	`0, 2`	`3, 1`
`JScrollBar`	`0, 3`	`3, 1`

Nota che la posizione `0, 0` è in alto a sinistra: i valori di `x` (colonna) aumentano da sinistra a destra, i valori di `y` (riga) aumentano dall'alto verso il basso.

Per iniziare a disporre i componenti in `GridBagLayout` , prima imposta il layout del tuo `JFrame` o del riquadro del contenuto.

```
frame.setLayout(new GridBagLayout());
//OR
pane.setLayout(new GridBagLayout());
//OR
JPanel pane = new JPanel(new GridBagLayout()); //Add the layout when creating your content
pane
```

Si noti che non si definisce mai la dimensione della griglia. Questo viene fatto automaticamente quando aggiungi i tuoi componenti.

Successivamente, sarà necessario creare un oggetto `GridBagConstraints` .

```
GridBagConstraints c = new GridBagConstraints();
```

Per assicurarti che i tuoi componenti riempiano le dimensioni della finestra, potresti voler impostare il peso di tutti i componenti su 1. Il peso viene usato per determinare come distribuire lo spazio tra colonne e righe.

```
c.weightx = 1;
c.weighty = 1;
```

Un'altra cosa che potresti voler fare è assicurarti che i componenti occupino tutto lo spazio orizzontale possibile.

```
c.fill = GridBagConstraints.HORIZONTAL;
```

Puoi anche impostare altre opzioni di riempimento se lo desideri.

```
GridBagConstraints.NONE //Don't fill components at all
GridBagConstraints.HORIZONTAL //Fill components horizontally
GridBagConstraints.VERTICAL //Fill components vertically
GridBagConstraints.BOTH //Fill components horizontally and vertically
```

Quando crei i componenti, vorrai impostare dove dovrebbe essere posizionato sulla griglia e quante piastrelle della griglia dovrebbe usare. Ad esempio, per posizionare un pulsante nella terza riga nella seconda colonna e occupare uno spazio di griglia 5 x 5, effettuare le seguenti operazioni. Tieni presente che la griglia inizia da `0, 0` , non `1, 1` .

```
JButton button = new JButton("Fancy Button!");
c.gridx = 2;
c.gridy = 1;
c.gridwidth = 5;
c.gridheight = 5;
pane.add(buttonA, c);
```

Quando aggiungi componenti alla tua finestra, ricorda di passare i vincoli come parametro. Questo può essere visto nell'ultima riga nell'esempio di codice sopra.

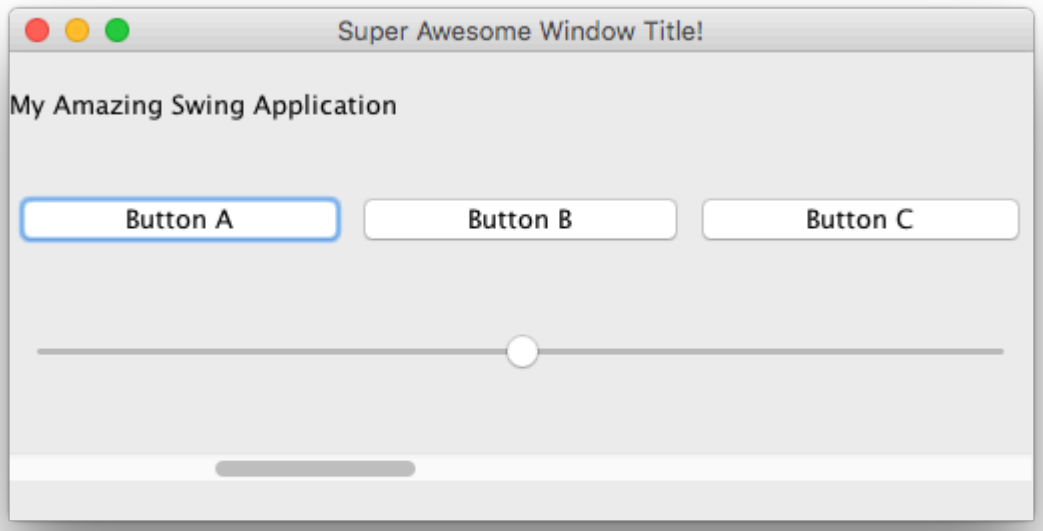
È possibile riutilizzare lo stesso `GridBagConstraints` per ogni componente, cambiarlo dopo aver aggiunto un componente non cambia il componente aggiunto in precedenza.

Esempio

Ecco il codice per l'esempio all'inizio di questa sezione.

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(new GridBagLayout()); //Create a pane to house all content, and give
it a GridBagLayout
frame.setContentPane(pane);

GridBagConstraints c = new GridBagConstraints();
c.weightx = 1;
c.weighty = 1;
c.fill = GridBagConstraints.HORIZONTAL;

JLabel headerLabel = new JLabel("My Amazing Swing Application");
c.gridx = 0;
c.gridwidth = 3;
c.gridy = 0;
pane.add(headerLabel, c);

JButton buttonA = new JButton("Button A");
c.gridx = 0;
c.gridwidth = 1;
c.gridy = 1;
pane.add(buttonA, c);

JButton buttonB = new JButton("Button B");
c.gridx = 1;
c.gridwidth = 1;
c.gridy = 1;
pane.add(buttonB, c);
```

```
JButton buttonC = new JButton("Button C");
c.gridx = 2;
c.gridwidth = 1;
c.gridy = 1;
pane.add(buttonC, c);

JSlider slider = new JSlider(0, 100);
c.gridx = 0;
c.gridwidth = 3;
c.gridy = 2;
pane.add(slider, c);

JScrollBar scrollBar = new JScrollBar(JScrollBar.HORIZONTAL, 20, 20, 0, 100);
c.gridx = 0;
c.gridwidth = 3;
c.gridy = 3;
pane.add(scrollBar, c);

frame.setVisible(true); //Show the window
```



Leggi Layout GridBag online: https://riptutorial.com/it/swing/topic/3698/layout-gridbag

Capitolo 7: MigLayout

Examples

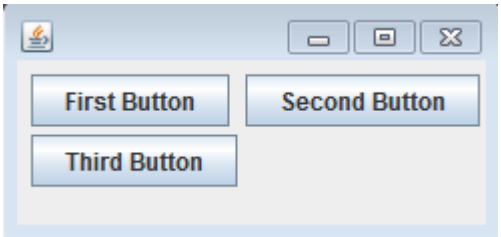
Elementi di avvolgimento

Questo esempio dimostra come posizionare 3 pulsanti in totale con 2 pulsanti nella prima riga. Quindi si verifica un avvolgimento, quindi l'ultimo pulsante si trova in una nuova riga.

I vincoli sono stringhe semplici, in questo caso "avvolgere" mentre si posiziona il componente.

```
public class ShowMigLayout {

    // Create the elements
    private final JFrame demo = new JFrame();
    private final JPanel panel = new JPanel();
    private final JButton button1 = new JButton("First Button");
    private final JButton button2 = new JButton("Second Button");
    private final JButton button3 = new JButton("Third Button");

    public static void main(String[] args) {
        ShowMigLayout showMigLayout = new ShowMigLayout();
        SwingUtilities.invokeLater(showMigLayout::createAndShowGui);
    }

    public void createAndShowGui() {
        // Set the position and the size of the frame
        demo.setBounds(400, 400, 250, 120);

        // Tell the panel to use the MigLayout as layout manager
        panel.setLayout(new MigLayout());

        panel.add(button1);
        // Notice the wrapping
        panel.add(button2, "wrap");
        panel.add(button3);

        demo.add(panel);
        demo.setVisible(true);
    }
}
```

Produzione:



Leggi MigLayout online: https://riptutorial.com/it/swing/topic/2966/miglayout

Capitolo 8: Nozioni di base

Examples

Ritarda un'attività UI per un periodo specifico

Tutte le operazioni relative allo Swing avvengono su una filettatura dedicata (la EDT - **E** vent **D** ispatch **T** hread). Se questo thread viene bloccato, l'interfaccia utente diventa non reattiva.

Pertanto, se si desidera ritardare un'operazione, non è possibile utilizzare `Thread.sleep` . Utilizzare invece `javax.swing.Timer` . Ad esempio il seguente `Timer` invertirà il testo di su una `JLabel`

```
  int delay = 2000;//specify the delay for the timer
  Timer timer = new Timer( delay, e -> {
    //The following code will be executed once the delay is reached
    String revertedText = new StringBuilder( label.getText() ).reverse().toString();
    label.setText( revertedText );
  } );
  timer.setRepeats( false );//make sure the timer only runs once
```

Di seguito è riportato un esempio eseguibile completo che utilizza questo `Timer` : l'interfaccia utente contiene un pulsante e un'etichetta. Premendo il pulsante si invertirà il testo dell'etichetta dopo un ritardo di 2 secondi

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

public final class DelayedExecutionExample {

  public static void main( String[] args ) {
    EventQueue.invokeLater( () -> showUI() );
  }

  private static void showUI(){
    JFrame frame = new JFrame( "Delayed execution example" );

    JLabel label = new JLabel( "Hello world" );
    JButton button = new JButton( "Reverse text with delay" );
    button.addActionListener( event -> {
      button.setEnabled( false );
      //Instead of directly updating the label, we use a timer
      //This allows to introduce a delay, while keeping the EDT free
      int delay = 2000;
      Timer timer = new Timer( delay, e -> {
        String revertedText = new StringBuilder( label.getText() ).reverse().toString();
        label.setText( revertedText );
        button.setEnabled( true );
      } );
      timer.setRepeats( false );//make sure the timer only runs once
      timer.start();
    } );

    frame.add( label, BorderLayout.CENTER );
```

```
    frame.add( button, BorderLayout.SOUTH );
    frame.pack();
    frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
    frame.setVisible( true );
  }
}
```

Ripeti un'attività UI a intervalli fissi

L'aggiornamento dello stato di un componente Swing deve avvenire sul thread di invio eventi (EDT). `javax.swing.Timer` attiva `ActionListener` `javax.swing.Timer` , rendendolo una buona scelta per eseguire le operazioni di Swing.

L'esempio seguente aggiorna il testo di una `JLabel` ogni due secondi:

```
//Use a timer to update the label at a fixed interval
int delay = 2000;
Timer timer = new Timer( delay, e -> {
  String revertedText = new StringBuilder( label.getText() ).reverse().toString();
  label.setText( revertedText );
} );
timer.start();
```

Di seguito viene fornito un esempio completo che utilizza questo `Timer` : l'interfaccia utente contiene un'etichetta e il testo dell'etichetta verrà ripristinato ogni due secondi.

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

public final class RepeatTaskFixedIntervalExample {
  public static void main( String[] args ) {
    EventQueue.invokeLater( () -> showUI() );
  }
  private static void showUI(){
    JFrame frame = new JFrame( "Repeated task example" );
    JLabel label = new JLabel( "Hello world" );

    //Use a timer to update the label at a fixed interval
    int delay = 2000;
    Timer timer = new Timer( delay, e -> {
      String revertedText = new StringBuilder( label.getText() ).reverse().toString();
      label.setText( revertedText );
    } );
    timer.start();

    frame.add( label, BorderLayout.CENTER );
    frame.pack();
    frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
    frame.setVisible( true );
  }
}
```

Esecuzione di un'attività UI un numero fisso di volte

In `ActionListener` collegato a `javax.swing.Timer` , è possibile tenere traccia del numero di volte in

cui il `Timer` eseguito `ActionListener` . Una volta raggiunto il numero di volte richiesto, è possibile utilizzare il metodo `Timer#stop()` per arrestare il `Timer` .

```
Timer timer = new Timer( delay, new ActionListener() {
  private int counter = 0;
  @Override
  public void actionPerformed( ActionEvent e ) {
    counter++;//keep track of the number of times the Timer executed
    label.setText( counter + "" );
    if ( counter == 5 ){
      ( ( Timer ) e.getSource() ).stop();
    }
  }
});
```

Di seguito è riportato un esempio eseguibile completo che utilizza questo `Timer` : mostra un'interfaccia in cui il testo dell'etichetta conta da zero a cinque. Una volta che viene raggiunto il cinque, il `Timer` si ferma.

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

public final class RepeatFixedNumberOfTimes {
  public static void main( String[] args ) {
    EventQueue.invokeLater( () -> showUI() );
  }
  private static void showUI(){
    JFrame frame = new JFrame( "Repeated fixed number of times example" );
    JLabel label = new JLabel( "0" );

    int delay = 2000;
    Timer timer = new Timer( delay, new ActionListener() {
      private int counter = 0;
      @Override
      public void actionPerformed( ActionEvent e ) {
        counter++;//keep track of the number of times the Timer executed
        label.setText( counter + "" );
        if ( counter == 5 ){
          //stop the Timer when we reach 5
          ( ( Timer ) e.getSource() ).stop();
        }
      }
    });
    timer.setInitialDelay( delay );
    timer.start();

    frame.add( label, BorderLayout.CENTER );
    frame.pack();
    frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
    frame.setVisible( true );
  }
}
```

Creare il tuo primo JFrame

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingUtilities;

public class FrameCreator {

    public static void main(String args[]) {
        //All Swing actions should be run on the Event Dispatch Thread (EDT)
        //Calling SwingUtilities.invokeLater makes sure that happens.
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame();
            //JFrames will not display without size being set
            frame.setSize(500, 500);

            JLabel label = new JLabel("Hello World");
            frame.add(label);

            frame.setVisible(true);
        });
    }

}
```

Come puoi notare se esegui questo codice, l'etichetta si trova in una posizione molto brutta. Questo è difficile da cambiare in un buon modo usando il metodo `add` . Per consentire un posizionamento più dinamico e flessibile, controlla i gestori del layout di Swing .

Creazione della sottoclasse JFrame

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingUtilities;

public class CustomFrame extends JFrame {

    private static CustomFrame statFrame;

    public CustomFrame(String labelText) {
        setSize(500, 500);

        //See link below for more info on FlowLayout
        this.setLayout(new FlowLayout());

        JLabel label = new JLabel(labelText);
        add(label);

        //Tells the JFrame what to do when it's closed
        //In this case, we're saying to "Dispose" on remove all resources
        //associated with the frame on close
        this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }

    public void addLabel(String labelText) {
        JLabel label = new JLabel(labelText);
        add(label);
        this.validate();
    }
```

```
    public static void main(String args[]) {
        //All Swing actions should be run on the Event Dispatch Thread (EDT)
        //Calling SwingUtilities.invokeLater makes sure that happens.
        SwingUtilities.invokeLater(() -> {
            CustomFrame frame = new CustomFrame("Hello Jungle");
            //This is simply being done so it can be accessed later
            statFrame = frame;
            frame.setVisible(true);
        });

        try {
            Thread.sleep(5000);
        } catch (InterruptedException ex) {
            //Handle error
        }

        SwingUtilities.invokeLater(() -> statFrame.addLabel("Oh, hello world too."));
    }

}
```

Per ulteriori informazioni su FlowLayout qui .

Ascoltando un evento

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class CustomFrame extends JFrame {

    public CustomFrame(String labelText) {
        setSize(500, 500);

        //See link below for more info on FlowLayout
        this.setLayout(new FlowLayout());

        //Tells the JFrame what to do when it's closed
        //In this case, we're saying to "Dispose" on remove all resources
        //associated with the frame on close
        this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        //Add a button
        JButton btn = new JButton("Hello button");
        //And a textbox
        JTextField field = new JTextField("Name");
        field.setSize(150, 50);
        //This next block of code executes whenever the button is clicked.
        btn.addActionListener((evt) -> {
            JLabel helloLbl = new JLabel("Hello " + field.getText());
            add(helloLbl);
            validate();
        });
        add(btn);
        add(field);
```

```
    }

    public static void main(String args[]) {
        //All Swing actions should be run on the Event Dispatch Thread (EDT)
        //Calling SwingUtilities.invokeLater makes sure that happens.
        SwingUtilities.invokeLater(() -> {
            CustomFrame frame = new CustomFrame("Hello Jungle");
            //This is simply being done so it can be accessed later
            frame.setVisible(true);
        });
    }

}
```

Crea un popup "Attendi ..."

Questo codice può essere aggiunto a qualsiasi evento come un listener, un pulsante, ecc. Apparirà un `JDialog` blocco che rimarrà fino al completamento del processo.

```
final JDialog loading = new JDialog(parentComponent);
JPanel p1 = new JPanel(new BorderLayout());
p1.add(new JLabel("Please wait..."), BorderLayout.CENTER);
loading.setUndecorated(true);
loading.getContentPane().add(p1);
loading.pack();
loading.setLocationRelativeTo(parentComponent);
loading.setDefaultCloseOperation(JDialog.DO_NOTHING_ON_CLOSE);
loading.setModal(true);

SwingWorker<String, Void> worker = new SwingWorker<String, Void>() {
    @Override
    protected String doInBackground() throws InterruptedException
        /** Execute some operation */
    }
    @Override
    protected void done() {
        loading.dispose();
    }
};
worker.execute(); //here the process thread initiates
loading.setVisible(true);
try {
    worker.get(); //here the parent thread waits for completion
} catch (Exception e1) {
    e1.printStackTrace();
}
```

Aggiunta di JButtons (Hello World Pt.2)

Supponendo che tu abbia creato correttamente un JFrame e che Swing sia stato importato ...

Puoi importare interamente Swing

```
import javax.Swing.*;
```

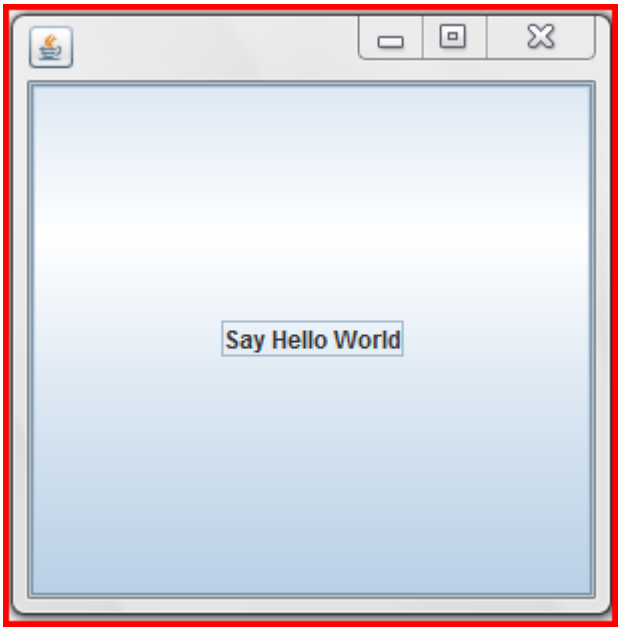
oppure È possibile importare i componenti / il telaio dell'oscillazione che si intende utilizzare

```
import javax.Swing.Jframe;
import javax.Swing.JButton;
```

Adesso giù per aggiungere il Jbutton ...

```
public static void main(String[] args) {

    JFrame frame = new JFrame(); //creates the frame
    frame.setSize(300, 300);
    frame.setVisible(true);

    ////////////////////////ADDING BUTTON BELOW//////////////////////////////
    JButton B = new JButton("Say Hello World");
    B.addMouseListener(new MouseAdapter() {

        public void mouseReleased(MouseEvent arg0) {
            System.out.println("Hello World");
        }

    });
    B.setBounds(0, 0,frame.getHeight(), frame.getWidth());
    B.setVisible(true);
    frame.add(B);
    ////////////////////////////////////////////////////////////////////////
}
```

Eseguendo / compilando questo codice dovresti ottenere qualcosa di simile a questo ...



Quando si fa clic sul pulsante ... "Hello World" dovrebbe apparire anche nella tua console.

Leggi Nozioni di base online: https://riptutorial.com/it/swing/topic/5415/nozioni-di-base

Capitolo 9: Pattern MVP

Examples

Esempio di MVP semplice

Per illustrare un semplice esempio di utilizzo del pattern MVP, prendere in considerazione il codice seguente che crea un'interfaccia utente semplice con solo un pulsante e un'etichetta. Quando si fa clic sul pulsante, l'etichetta si aggiorna con il numero di volte in cui si fa clic sul pulsante.

Abbiamo 5 classi:

- Modello: il POJO per mantenere lo stato (M in MVP)
- Visualizza - La classe con codice UI (V in MVP)
- ViewListener - Interfaccia che fornisce metodi per rispondere alle azioni nella vista
- Presentatore: risponde all'input e aggiorna la vista (P in MVP)
- Applicazione: la classe "principale" per riunire tutto e avviare l'app

Una classe "modello" minima che mantiene solo una variabile `count` singolo.

```
/**
 * A minimal class to maintain some state
 */
public class Model {
    private int count = 0;

    public void addOneToCount() {
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

Un'interfaccia minima per notificare gli ascoltatori:

```
/**
 * Provides methods to notify on user interaction
 */
public interface ViewListener {
    public void onButtonClicked();
}
```

La classe view costruisce tutti gli elementi dell'interfaccia utente. La vista e *solo* la vista dovrebbero avere riferimento agli elementi dell'interfaccia utente (ad esempio, nessun pulsante, campi di testo, ecc. Nel relatore o in altre classi).

```
/**
```

```
 * Provides the UI elements
 */

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.WindowConstants;

public class View {
    // A list of listeners subscribed to this view
    private final ArrayList<ViewListener> listeners;
    private final JLabel label;

    public View() {
        final JFrame frame = new JFrame();
        frame.setSize(200, 100);
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.setLayout(new GridLayout());

        final JButton button = new JButton("Hello, world!");

        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(final ActionEvent e) {
                notifyListenersOnButtonClicked();
            }
        });
        frame.add(button);

        label = new JLabel();
        frame.add(label);

        this.listeners = new ArrayList<ViewListener>();

        frame.setVisible(true);
    }

    // Iterate through the list, notifying each listner individualy
    private void notifyListenersOnButtonClicked() {
        for (final ViewListener listener : listeners) {
            listener.onButtonClicked();
        }
    }

    // Subscribe a listener
    public void addListener(final ViewListener listener) {
        listeners.add(listener);
    }

    public void setLabelText(final String text) {
        label.setText(text);
    }
}
```

La logica di notifica può anche essere codificata in questo modo in Java8:

```
        ...
        final Button button = new Button("Hello, world!");
        // In order to do so, our interface must be changed to accept the event parametre
        button.addActionListener((event) -> {
            notifyListeners(ViewListener::onButtonClicked, event);
            // Example of calling methodThatTakesALong, would be the same as callying:
            // notifyListeners((listener, long)->listener.methodThatTakesALong(long), 10L)
            notifyListeners(ViewListener::methodThatTakesALong, 10L);
        });
        frame.add(button);
        ...

/**
 * Iterates through the subscribed listeneres notifying each listener individually.
 * Note: the {@literal '<T>' in private <T> void} is a Bounded Type Parametre.
 *
 * @param <T>      Any Reference Type (basically a class).
 *
 * @param consumer A method with two parameters and no return,
 *                 the 1st parametre is a ViewListner,
 *                 the 2nd parametre is value of type T.
 *
 * @param data     The value used as parametre for the second argument of the
 *                 method described by the parametre consumer.
 */
private <T> void notifyListeners(final BiConsumer<ViewListener, T> consumer, final T data) {
    // Iterate through the list, notifying each listener, java8 style
    listeners.forEach((listener) -> {

        // Calls the funcion described by the object consumer.
        consumer.accept(listener, data);

        // When this method is called using ViewListener::onButtonClicked
        // the line: consumer.accept(listener,data); can be read as:
        // void accept(ViewListener listener, ActionEvent data) {
        //     listener.onButtonClicked(data);
        // }

    });
}
```

L'interfaccia deve essere sottoposta a refactoring per poter utilizzare ActionEvent come parametro:

```
public interface ViewListener {
    public void onButtonClicked(ActionEvent evt);
    // Example of methodThatTakesALong signature
    public void methodThatTakesALong(long );
}
```

Qui è necessario un solo metodo-notifica, il metodo listener effettivo e il suo parametro vengono passati come parametri. Nel caso in cui ciò sia necessario, questo può essere usato anche per qualcosa di un po 'meno ingegnoso dell'effettiva gestione degli eventi, funziona tutto il tempo che c'è un metodo nell'interfaccia, ad esempio:

```
        notifyListeners(ViewListener::methodThatTakesALong, -1L);
```

Il relatore può prendere visione e aggiungersi come ascoltatore. Quando si fa clic sul pulsante nella vista, la vista notifica tutti i listener (incluso il relatore). Ora che il relatore viene avvisato, può intraprendere l'azione appropriata per aggiornare il modello (ovvero lo stato dell'applicazione), quindi aggiornare la vista di conseguenza.

```
/**
 * Responsible to responding to user interaction and updating the view
 */
public class Presenter implements ViewListener {
    private final View view;
    private final Model model;

    public Presenter(final View view, final Model model) {
        this.view = view;
        view.addListener(this);
        this.model = model;
    }

    @Override
    public void onButtonClicked() {
        // Update the model (ie. the state of the application)
        model.addOneToCount();
        // Update the view
        view.setLabelText(String.valueOf(model.getCount()));
    }
}
```

Per mettere tutto insieme, la vista può essere creata e iniettata nel presentatore. Allo stesso modo, un modello iniziale può essere creato e iniettato. Mentre entrambi *possono* essere creati nel presenter, iniettarli nel costruttore consente di eseguire test molto più semplici.

```
public class Application {
    public Application() {
        final View view = new View();
        final Model model = new Model();
        new Presenter(view, model);
    }

    public static void main(String... args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new Application();
            }
        });
    }
}
```

Leggi Pattern MVP online: https://riptutorial.com/it/swing/topic/5154/pattern-mvp

Capitolo 10: StyledDocument

Sintassi

- doc.insertString (index, text, attributes); // gli attributi dovrebbero essere un AttributeSet

Examples

Creazione di un documento DefaultStyled

```
try {
    StyledDocument doc = new DefaultStyledDocument();
    doc.insertString(0, "This is the beginning text", null);
    doc.insertString(doc.getLength(), "\nInserting new line at end of doc", null);
    MutableAttributeSet attrs = new SimpleAttributeSet();
    StyleConstants.setBold(attrs, true);
    doc.insertString(5, "This is bold text after 'this'", attrs);
} catch (BadLocationException ex) {
    //handle error
}
```

DefaultStyledDocuments sarà probabilmente la tua risorsa più utilizzata. Possono essere creati direttamente e sottoclasse la classe astratta `StyledDocument` .

Aggiunta di StyledDocument a JTextPane

```
try {
        JTextPane pane = new JTextPane();
        StyledDocument doc = new DefaultStyledDocument();
        doc.insertString(0, "Some text", null);
        pane.setDocument(doc); //Technically takes any subclass of Document
    } catch (BadLocationException ex) {
        //handle error
    }
```

Il JTextPane può quindi essere aggiunto a qualsiasi modulo della GUI Swing.

Copia di DefaultStyledDocument

`StyledDocuments` generalmente non implementa il clone e quindi deve copiarli in un modo diverso se necessario.

```
try {
        //Initialization
        DefaultStyledDocument sourceDoc = new DefaultStyledDocument();
        DefaultStyledDocument destDoc = new DefaultStyledDocument();
        MutableAttributeSet bold = new SimpleAttributeSet();
        StyleConstants.setBold(bold, true);
        MutableAttributeSet italic = new SimpleAttributeSet();
```

```
        StyleConstants.setItalic(italic, true);
        sourceDoc.insertString(0, "Some bold text. ", bold);
        sourceDoc.insertString(sourceDoc.getLength(), "Some italic text", italic);

        //This does the actual copying
        String text = sourceDoc.getText(0, sourceDoc.getLength()); //This copies text, but
loses formatting.
        for (int i = 0; i < text.length(); i++) {
            Element e = destDoc.getCharacterElement(i); //A Elment describes a particular part
of a document, in this case a character
            AttributeSet attr = e.getAttributes(); //Gets the attributes for the character
            destDoc.insertString(destDoc.getLength(), text.substring(i, i+1), attr); //Gets
the single character and sets its attributes from the element
        }
    } catch (BadLocationException ex) {
        //handle error
    }
```

Serializzazione di DefaultStyledDocument in RTF

Utilizzando la libreria AdvancedRTFEditorKit è possibile serializzare un documento `DefaultStyledDocument` su una stringa RTF.

```
try {
   DefaultStyledDocument writeDoc = new DefaultStyledDocument();
   writeDoc.insertString(0, "Test string", null);

   AdvancedRTFEditorKit kit = new AdvancedRTFEditorKit();
   //Other writers, such as a FileWriter, may be used
   //OutputStreams are also an option
   Writer writer = new StringWriter();
   //You can write just a portion of the document by modifying the start
   //and end indexes
   kit.write(writer, writeDoc, 0, writeDoc.getLength());
   //This is the RTF String
   String rtfDoc = writer.toString();

   //As above this may be a different kind of reader or an InputStream
   StringReader reader = new StringReader(rtfDoc);
   //AdvancedRTFDocument extends DefaultStyledDocument and can generally
   //be used wherever DefaultStyledDocument can be.
   //However for reading, AdvancedRTFDocument must be used
   DefaultStyledDocument readDoc = new AdvancedRTFDocument();
   //You can insert at different values by changing the "0"
   kit.read(reader, readDoc, 0);
   //readDoc is now the same as writeDoc
} catch (BadLocationException | IOException ex) {
    //Handle exception
    ex.printStackTrace();
}
```

Leggi StyledDocument online: https://riptutorial.com/it/swing/topic/5416/styleddocument

Capitolo 11: Swing Workers e l'EDT

Sintassi

- classe astratta pubblica SwingWorker <T, V>
- T - il tipo di risultato restituito da doInBackground di questo SwingWorker e metodi.
- V - il tipo utilizzato per eseguire risultati intermedi con i metodi di pubblicazione e di processo di SwingWorker.
- T doInBackground () - La funzione astratta che deve essere ignorata. Il tipo di ritorno è T.

Examples

Thread di invio principale ed evento

Come ogni altro programma java, ogni programma swing inizia con un metodo principale. Il metodo principale è avviato dal thread principale. Tuttavia, i componenti di Swing devono essere creati e aggiornati sul thread di invio eventi (o breve: EDT). Per illustrare la dinamica tra il thread principale e l'EDT dai un'occhiata a questo Hello World! esempio.

Il thread principale è appena usato per delegare la creazione della finestra all'EDT. Se l'EDT non è ancora stato avviato, la prima chiamata a `SwingUtilities.invokeLater` l'infrastruttura necessaria per l'elaborazione dei componenti di Swing. Inoltre, l'EDT rimane attivo in background. Il thread principale sta per morire direttamente dopo l'avvio della configurazione di EDT, ma l'EDT rimarrà attivo fino a quando l'utente non uscirà dal programma. Questo può essere ottenuto premendo il riquadro di chiusura `JFrame` visibile. Ciò interromperà l'EDT e il processo del programma sarà interamente.

Trova i primi N numeri pari e visualizza i risultati in un JTextArea dove i calcoli vengono eseguiti in background.

```
import java.awt.EventQueue;
import java.awt.GridLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JFrame;
import javax.swing.JTextArea;
import javax.swing.SwingWorker;

class PrimeNumbersTask extends SwingWorker<List<Integer>, Integer> {
    private final int numbersToFind;
```

```
    private final JTextArea textArea;

    PrimeNumbersTask(JTextArea textArea, int numbersToFind) {
        this.numbersToFind = numbersToFind;
        this.textArea = textArea;
    }

    @Override
    public List<Integer> doInBackground() {
        final List<Integer> result = new ArrayList<>();
        boolean interrupted = false;
        for (int i = 0; !interrupted && (i < numbersToFind); i += 2) {
            interrupted = doIntenseComputing();
            result.add(i);
            publish(i); // sends data to process function
        }
        return result;
    }

    private boolean doIntenseComputing() {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            return true;
        }
        return false;
    }

    @Override
    protected void process(List<Integer> chunks) {
        for (int number : chunks) {
            // the process method will be called on the EDT
            // thus UI elementes may be updated in here
            textArea.append(number + "\n");
        }
    }
}

public class SwingWorkerExample extends JFrame {
    private JTextArea textArea;

    public SwingWorkerExample() {
        super("Java SwingWorker Example");
        init();
    }

    private void init() {
        setSize(400, 400);
        setLayout(new GridLayout(1, 1));
        textArea = new JTextArea();
        add(textArea);

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose();
                System.exit(0);
            }
        });
    }

    public static void main(String args[]) throws Exception {
```

```
        SwingWorkerExample ui = new SwingWorkerExample();
        EventQueue.invokeLater(() -> {
            ui.setVisible(true);
        });

        int n = 100;
        PrimeNumbersTask task = new PrimeNumbersTask(ui.textArea, n);
        task.execute(); // run async worker which will do long running task on a
        // different thread
        System.out.println(task.get());
    }
}
```

Leggi Swing Workers e l'EDT online: https://riptutorial.com/it/swing/topic/3431/swing-workers-e-l-edt

Capitolo 12: timer in JFrame

Examples

Timer in JFrame

Supponiamo che tu abbia un pulsante nel tuo programma Java che conta un tempo. Ecco il codice per il timer di 10 minuti.

```
private final static long REFRESH_LIST_PERIOD=10 * 60 * 1000; //10 minutes

Timer timer = new Timer(1000, new ActionListener() {

   @Override
   public void actionPerformed(ActionEvent e) {
    if (cnt > 0) {
     cnt = cnt - 1000;
     btnCounter.setText("Remained (" + format.format(new Date(cnt)) + ")");
    } else {
     cnt = REFRESH_LIST_PERIOD;
     //TODO
    }

   }
  });

  timer.start();
```

Leggi timer in JFrame online: https://riptutorial.com/it/swing/topic/6745/timer-in-jframe

Capitolo 13: Usando Look and Feel

Examples

Utilizzando il sistema L & F

Swing supporta parecchi L & F nativi.
Puoi sempre installarne facilmente uno senza richiedere una specifica classe L & F:

```
public class SystemLookAndFeel
{
    public static void main ( final String[] args )
    {
        // L&F installation should be performed within EDT (Event Dispatch Thread)
        // This is important to avoid any UI issues, exceptions or even deadlocks
        SwingUtilities.invokeLater ( new Runnable ()
        {
            @Override
            public void run ()
            {
                // Process of L&F installation might throw multiple exceptions
                // It is always up to you whether to handle or ignore them
                // In most common cases you would never encounter any of those
                try
                {
                    // Installing native L&F as a current application L&F
                    // We do not know what exactly L&F class is, it is provided by the
UIManager
                    UIManager.setLookAndFeel ( UIManager.getSystemLookAndFeelClassName () );
                }
                catch ( final ClassNotFoundException e )
                {
                    // L&F class was not found
                    e.printStackTrace ();
                }
                catch ( final InstantiationException e )
                {
                    // Exception while instantiating L&F class
                    e.printStackTrace ();
                }
                catch ( final IllegalAccessException e )
                {
                    // Class or initializer isn't accessible
                    e.printStackTrace ();
                }
                catch ( final UnsupportedLookAndFeelException e )
                {
                    // L&F is not supported on the current system
                    e.printStackTrace ();
                }

                // Now we can create some natively-looking UI
                // This is just a small sample frame with a single button on it
                final JFrame frame = new JFrame ();
                final JPanel content = new JPanel ( new FlowLayout () );
                content.setBorder ( BorderFactory.createEmptyBorder ( 50, 50, 50, 50 ) );
```

```
                content.add ( new JButton ( "Native-looking button" ) );
                frame.setContentPane ( content );
                frame.setDefaultCloseOperation ( WindowConstants.EXIT_ON_CLOSE );
                frame.pack ();
                frame.setLocationRelativeTo ( null );
                frame.setVisible ( true );
            }
        } );
    }
}
```

Questi sono i supporti JDK nativi di L & Fs (OS -> L & F):

OS	Nome L & F	Classe L & F
Solaris, Linux con GTK +	GTK +	com.sun.java.swing.plaf.gtk.GTKLookAndFeel
Altro Solaris, Linux	Motivo	com.sun.java.swing.plaf.motif.MotifLookAndFeel
Windows classico	finestre	com.sun.java.swing.plaf.windows.WindowsLookAndFeel
Windows XP	Windows XP	com.sun.java.swing.plaf.windows.WindowsLookAndFeel
Windows Vista	Windows Vista	com.sun.java.swing.plaf.windows.WindowsLookAndFeel
Macintosh	Macintosh	com.apple.laf.AquaLookAndFeel *
IBM UNIX	IBM	javax.swing.plaf.synth.SynthLookAndFeel *
HP UX	HP	javax.swing.plaf.synth.SynthLookAndFeel *

* questi L & F sono forniti dal fornitore del sistema e il nome effettivo della classe L & F può variare

Usando la L & F personalizzata

```
public class CustomLookAndFeel
{
    public static void main ( final String[] args )
    {
        // L&F installation should be performed within EDT (Event Dispatch Thread)
        // This is important to avoid any UI issues, exceptions or even deadlocks
        SwingUtilities.invokeLater ( new Runnable ()
        {
            @Override
            public void run ()
            {
                // Process of L&F installation might throw multiple exceptions
                // It is always up to you whether to handle or ignore them
                // In most common cases you would never encounter any of those
                try
```

```
                {
                    // Installing custom L&F as a current application L&F
                    UIManager.setLookAndFeel ( "javax.swing.plaf.metal.MetalLookAndFeel" );
                }
                catch ( final ClassNotFoundException e )
                {
                    // L&F class was not found
                    e.printStackTrace ();
                }
                catch ( final InstantiationException e )
                {
                    // Exception while instantiating L&F class
                    e.printStackTrace ();
                }
                catch ( final IllegalAccessException e )
                {
                    // Class or initializer isn't accessible
                    e.printStackTrace ();
                }
                catch ( final UnsupportedLookAndFeelException e )
                {
                    // L&F is not supported on the current system
                    e.printStackTrace ();
                }

                // Now we can create some pretty-looking UI
                // This is just a small sample frame with a single button on it
                final JFrame frame = new JFrame ();
                final JPanel content = new JPanel ( new FlowLayout () );
                content.setBorder ( BorderFactory.createEmptyBorder ( 50, 50, 50, 50 ) );
                content.add ( new JButton ( "Metal button" ) );
                frame.setContentPane ( content );
                frame.setDefaultCloseOperation ( WindowConstants.EXIT_ON_CLOSE );
                frame.pack ();
                frame.setLocationRelativeTo ( null );
                frame.setVisible ( true );
            }
        } );
    }
}
```

Puoi trovare una lista enorme di Swing L & Fs disponibili nell'argomento qui: Java Look and Feel (L & F)
Tieni presente che alcuni di questi L & F potrebbero essere obsoleti a questo punto.

Leggi Usando Look and Feel online: https://riptutorial.com/it/swing/topic/3627/usando-look-and-feel

Capitolo 14: Utilizzo di Swing per interfacce utente grafiche

Osservazioni

Chiudere l'applicazione alla chiusura della finestra

È facile dimenticare di chiudere l'applicazione quando la finestra è chiusa. Ricorda di aggiungere la seguente riga.

```
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the JFrame is closed
```

Examples

Creazione di una finestra vuota (JFrame)

Creare il JFrame

Creare una finestra è facile. Devi solo creare un `JFrame` .

```
JFrame frame = new JFrame();
```

Intitolando la finestra

Potresti voler dare un titolo alla tua finestra. Puoi farlo passando una stringa quando crei `JFrame` o chiamando `frame.setTitle(String title)` .

```
JFrame frame = new JFrame("Super Awesome Window Title!");
//OR
frame.setTitle("Super Awesome Window Title!");
```

Impostazione della dimensione della finestra

La finestra sarà il più piccola possibile quando è stata creata. Per ingrandirlo, puoi impostarne le dimensioni in modo esplicito:

```
frame.setSize(512, 256);
```

Oppure puoi avere la dimensione del frame stessa in base alla dimensione del suo contenuto con il metodo `pack()` .

```
frame.pack();
```

I metodi `setSize()` e `pack()` si escludono a vicenda, quindi utilizzare uno o l'altro.

Cosa fare alla fine della finestra

Si noti che l'applicazione **non si** chiuderà quando la finestra è stata chiusa. Puoi chiudere l'applicazione dopo che la finestra è stata chiusa dicendo a `JFrame` di farlo.

```
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
```

In alternativa, puoi dire alla finestra di fare qualcos'altro quando è chiuso.

```
WindowConstants.DISPOSE_ON_CLOSE //Get rid of the window
WindowConstants.EXIT_ON_CLOSE //Quit the application
WindowConstants.DO_NOTHING_ON_CLOSE //Don't even close the window
WindowConstants.HIDE_ON_CLOSE //Hides the window - This is the default action
```

Creazione di un riquadro del contenuto

Un passaggio facoltativo consiste nel creare un riquadro del contenuto per la tua finestra. Questo non è necessario, ma se vuoi farlo, crea un `JPanel` e chiama `frame.setContentPane(Component component)` .

```
JPanel pane = new JPanel();
frame.setContentPane(pane);
```

Mostrando la finestra

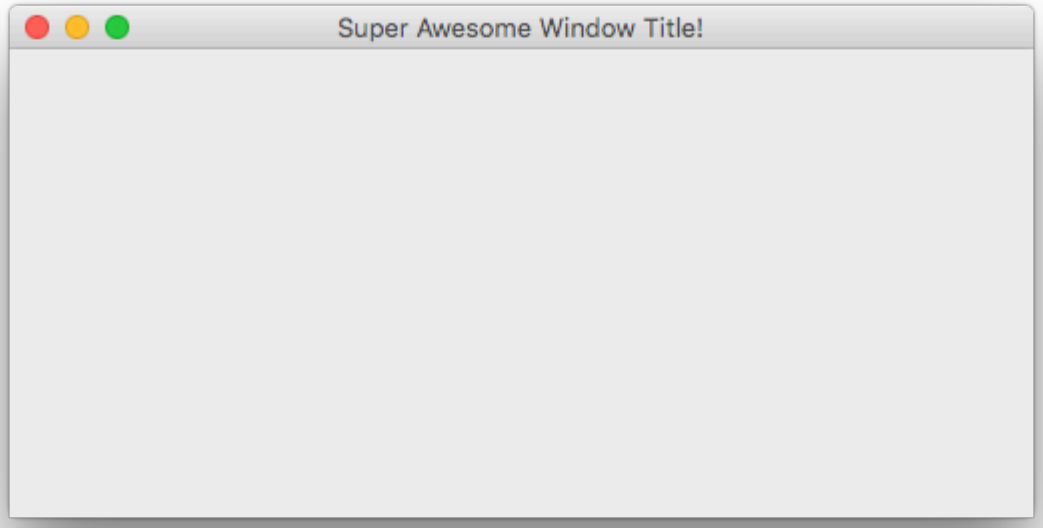
Dopo averlo creato, dovrai creare i tuoi componenti, quindi mostrare la finestra. Mostrare la finestra è fatto come tale.

```
frame.setVisible(true);
```

Esempio

Per quelli di voi che amano copiare e incollare, ecco un esempio di codice.

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(); //Create the content pane
frame.setContentPane(pane); //Set the content pane

frame.setVisible(true); //Show the window
```



Aggiunta di componenti

Un componente è una sorta di elemento dell'interfaccia utente, ad esempio un pulsante o un campo di testo.

Creazione di un componente

La creazione di componenti è quasi identica alla creazione di una finestra. Invece di creare un `JFrame` , tuttavia, lo crei. Ad esempio, per creare un `JButton` , fai quanto segue.

```
JButton button = new JButton();
```

Molti componenti possono avere parametri passati a loro una volta creati. Ad esempio, a un pulsante può essere dato del testo da visualizzare.

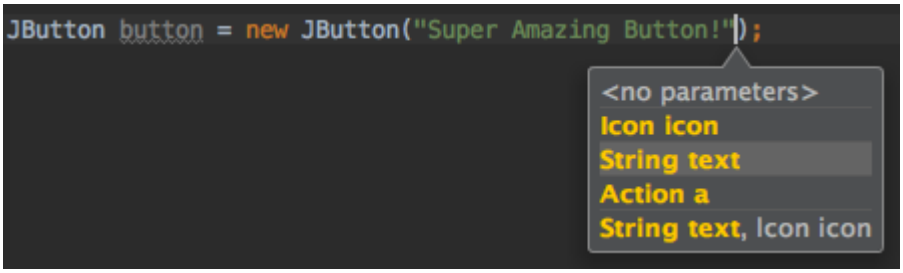
```
JButton button = new JButton("Super Amazing Button!");
```

Se non si desidera creare un pulsante, è possibile trovare un elenco di componenti comuni in un

altro esempio in questa pagina.

I parametri che possono essere passati a loro variano da un componente all'altro. Un buon modo per verificare ciò che possono accettare è guardando i paramter all'interno del tuo IDE (se ne usi uno). Le scorciatoie predefinite sono elencate di seguito.

- IntelliJ IDEA - Windows / Linux: `CTRL + P`
- IntelliJ IDEA - OS X / macOS: `CMD + P`
- Eclipse: `CTRL + SHIFT + Space`
- NetBeans: `CTRL + P`



Mostrando il componente

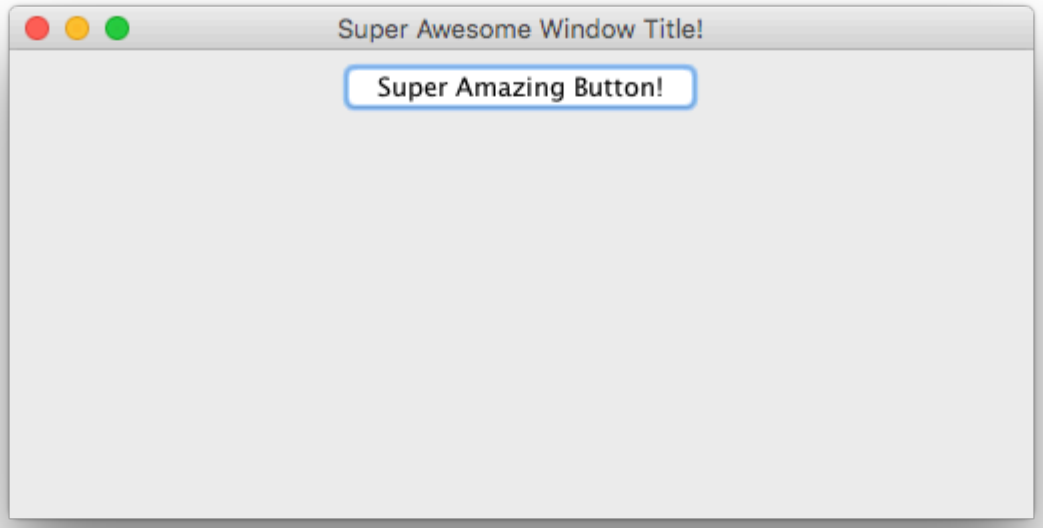
Dopo che un componente è stato creato, in genere devi impostare i suoi parametri. Dopo, devi metterlo da qualche parte, ad esempio sul tuo `JFrame` , o sul pannello dei contenuti, se ne hai creato uno.

```
frame.add(button); //Add to your JFrame
//OR
pane.add(button); //Add to your content pane
//OR
myComponent.add(button); //Add to whatever
```

Esempio

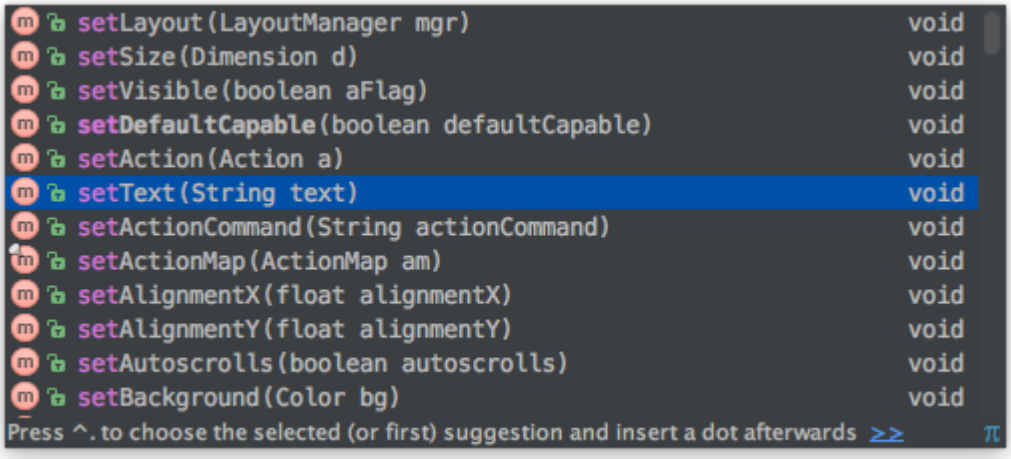
Ecco un esempio di creazione di una finestra, impostazione di un riquadro del contenuto e aggiunta di un pulsante.

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(); //Create the content pane
frame.setContentPane(pane); //Set the content pane

JButton button = new JButton("Super Amazing Button!"); //Create the button
pane.add(button); //Add the button to the content pane

frame.setVisible(true); //Show the window
```



Impostazione dei parametri per i componenti

I componenti hanno vari parametri che possono essere impostati per loro. Variano da componente a componente, quindi un buon modo per vedere quali parametri possono essere impostati per i componenti è iniziare a digitare `componentName.set` e lasciare che il completamento automatico del tuo IDE (Se si utilizza un IDE) suggerisca metodi. Il collegamento predefinito in molti IDE, se non viene visualizzato automaticamente, è `CTRL + Space` .



Parametri comuni condivisi tra tutti i componenti

Descrizione	Metodo
Imposta la dimensione più piccola che può essere il componente (solo se il gestore di layout onora la proprietà minimumSize)	`setMinimumSize(Dimension minimumSize)`

Descrizione	Metodo
Imposta la dimensione massima che può essere il componente (solo se il gestore di layout onora la proprietà maximumSize)	`setMaximumSize(Dimension maximumSize)`
Imposta la dimensione perferita del componente (solo se il gestore del layout onora la proprietà preferredSize)	`setPreferredSize(Dimension preferredSize)`
Mostra o nasconde il componente	`setVisible(boolean aFlag)`
Imposta se il componente deve rispondere all'input dell'utente	`setEnabled(boolean enabled)`
Imposta il carattere del testo	`setFont(Font font)`
Imposta il testo del suggerimento	`setToolTipText(String text)`
Imposta il Colore sfondo del componente	`setBackground(Color bg)`
Imposta il colore di primo piano (colore del carattere) del componente	`setForeground(Color bg)`

Parametri comuni in altri componenti

Componenti comuni	Descrizione	Metodo
`JLabel` , `JButton` , `JCheckBox` , `JRadioButton` , `JToggleButton` , `JMenu` , `JMenuItem` , `JTextArea` , `JTextField`	Imposta il testo visualizzato	`setText(String text)`
`JProgressBar` , `JScrollBar` , `JSlider` , `JSpinner`	Imposta un valore numerico tra i valori minimo e massimo del componente	`setValue(int n)`
`JProgressBar` , `JScrollBar` , `JSlider` , `JSpinner`	Imposta è il più piccolo valore possibile che può essere la proprietà `value`	`setMinimum(int n)`
`JProgressBar` , `JScrollBar` , `JSlider` , `JSpinner`	Set è il più grande valore possibile che può essere la proprietà `value`	`setMaxmimum(int n)`
`JCheckBox` , `JToggleBox`	Imposta se il valore è vero o falso (Es .: Se una casella di controllo è selezionata?)	`setSelected(boolean b)`

Componenti comuni

Descrizione	Classe
Pulsante	`JButton`
casella di controllo	`JCheckBox`
Menu a discesa / casella combinata	`JComboBox`
Etichetta	`JLabel`
Elenco	`JList`
Barra dei menu	`JMenuBar`
Menu in una barra dei menu	`JMenu`
Articolo in un menu	`JMenuItem`
Pannello	`JPanel`
Barra di avanzamento	`JProgressBar`
Pulsante di scelta	`JRadioButton`
Barra di scorrimento	`JScrollBar`
Slider	`JSlider`
Spinner / Selezione numero	`JSpinner`
tavolo	`JTable`
Albero	`JTree`
Area di testo / campo di testo multilinea	`JTextArea`
Campo di testo	`JTextField`
Barra degli strumenti	`JToolBar`

Realizzare interfacce utente interattive

Avere un pulsante è tutto buono e buono, ma che senso ha se cliccarlo non fa nulla? `ActionListener` vengono usati per dire al tuo pulsante, o ad altri componenti di fare qualcosa quando è attivato.

L'aggiunta di `ActionListener` viene eseguita come tale.

```
buttonA.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        //Code goes here...
        System.out.println("You clicked the button!");
    }
});
```

Oppure, se stai usando Java 8 o successivo ...

```
buttonA.addActionListener(e -> {
    //Code
    System.out.println("You clicked the button!");
});
```

Esempio (Java 8 e versioni successive)

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(); //Create a pane to house all content
frame.setContentPane(pane);

JButton button = new JButton("Click me - I know you want to.");
button.addActionListener(e -> {
    //Code goes here
    System.out.println("You clicked me! Ouch.");
});
pane.add(buttonA);

frame.setVisible(true); //Show the window
```

Organizzazione del layout dei componenti

L'aggiunta di componenti uno dopo l'altro determina un'interfaccia utente difficile da utilizzare, poiché i componenti sono tutti **da qualche parte** . I componenti sono ordinati dall'alto verso il basso, ogni componente in una "fila" separata.

Per ovviare a questo e fornirti come sviluppatore la possibilità di `LayoutManager` facilmente di componenti di layout, Swing dispone di `LayoutManager` .

Questi LayoutManager sono trattati in modo più approfondito in Introduzione ai gestori di layout e negli argomenti di Layout Manager separati:

- Layout della griglia
- Layout GridBag

Leggi Utilizzo di Swing per interfacce utente grafiche online:
https://riptutorial.com/it/swing/topic/2982/utilizzo-di-swing-per-interfacce-utente-grafiche

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con lo swing	Community, Freek de Bruijn, Petter Friberg, Vogel612, XavCo7
2	Gestione del layout	explv, J Atkin, mayha, pietrocalzini, recke96, Squidward, XavCo7
3	Grafica	Adel Khial, Ashlyn Campbell, Squidward
4	JList	Andreas Fester, Squidward, user6653173
5	Layout della griglia	Lukas Rotter, user6653173
6	Layout GridBag	CraftedCart, Enwired, mayha, Vogel612
7	MigLayout	hamena314, keuleJ
8	Nozioni di base	DarkV1, DonyorM, elias, Robin, Squidward
9	Pattern MVP	avojak, ehzawad, Leonardo Pina, sjngm, Squidward
10	StyledDocument	DonyorM, Squidward
11	Swing Workers e l'EDT	dpr, isaias-b, rahul tyagi
12	timer in JFrame	SSD
13	Usando Look and Feel	Mikle Garin
14	Utilizzo di Swing per interfacce utente grafiche	CraftedCart, mayha, Michael, Vogel612, Winter