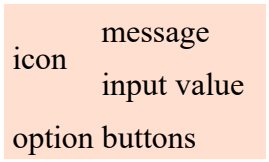
**Uso finestre di dialogo:** di input, di allarme o di scelta multipla

La classe `JOptionPane` permette di creare facilmente una *dialog box* standard di tipo "pop up" che consente all'utente di inserire dati o essere avvisato di qualcosa. Per informazioni sull'uso di tale classe si consulti How to Make Dialogs, nella sezione del *The Java Tutorial*.

Una *dialog box* appare, di solito, come mostrato a lato, ma si possono apportare modifiche al layout modificandone le proprietà con l'impostazione di opportuni **parametri** dei metodi di tipo **showXxxDialog**




**Parametri:** parentComponent, message, title, messageType, optionType, options, icon, initialValue oltre a selectionValues (opzioni di scelta in dialoghi di tipo input)

**parentComponent**: il Componente che contiene la dialog box; se tale primo parametro è **null** imposta il Frame di default usato come parent: la finestra di dialogo sarà centrata nello schermo e risulterà indipendente dal resto dell'applicazione.

**message**: un messaggio descrittivo, di solito una stringa; con altre possibilità:

- **Object[]** cioè un array di oggetti interpretato come una serie di messaggi memorizzati in uno stack verticale.L'interpretazione è ricorsiva: ogni oggetto è interpretato a seconda del tipo
- **Component** cioè un componente visualizzato nel dialog.
- **Icon** inglobata in un `JLabel` e visualizzata nel dialog
- **altro** cioè un'oggetto convertito in una stringa mediante il metodo `toString`. Il risultato è inglobato in un `JLabel` e visualizzato nel dialog

**title**: il titolo della finestra di dialogo che appare nella cornice superiore; il valore di default dipende dal tipo di dialog (la stringa "Input" per dialog in input, "Messaggio" per dialog in out, "Selezionare un'opzione" per dialog di conferma).

**messageType** : definisce lo stile del messaggio. Possibili valori, che prevedono relative icone di default, sono:

```
ERROR_MESSAGE
INFORMATION_MESSAGE
WARNING_MESSAGE
QUESTION_MESSAGE
PLAIN_MESSAGE
```

**optionType**: definisce il set dei bottoni di opzione (senza limiti) che appaiono in basso al dialog box:

```
DEFAULT_OPTION
YES_NO_OPTION
YES_NO_CANCEL_OPTION
OK_CANCEL_OPTION
```

**options**: una più dettagliata descrizione del set dei bottoni di opzione. Di solito un array di stringhe ma è previsto un array di oggetti che possono essere di tipo:

- **Component** cioè un componente aggiunto direttamente al bottone
- **Icon** usata come label in un `JButton`
- **altro** cioè un'oggetto convertito in una stringa mediante il metodo `toString`. Il risultato è usato come label in un `JButton`

**icon**: una icona decorativa. Il valore di default è stabilito dal parametro `messageType`.

**initialValue**: la scelta di default (valore in input)

La classe JOptionPane mette a disposizione **tre tipi di pannelli**: Confirm Dialog, Input Dialog e Message Dialog

- il primo tipo di pannello viene usato quando si deve chiedere all'utente di effettuare una scelta tra un gruppo di possibilità;
- il secondo torna utile quando si debba richiedere l'inserimento di una stringa di testo
- mentre il terzo viene usato per informare l'utente.

JOptionPane fornisce un gruppo di metodi statici che permettono di creare facilmente questi pannelli ricorrendo ad una sola riga di codice senza istanziare oggetti:

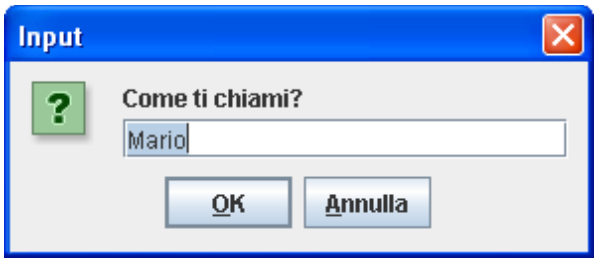
Alcuni *metodi* **`showInputDialog`** per visualizzare una finestra di dialogo in lettura:

| | |
|---|---|
| `static Object` | **`showInputDialog`**`(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue)`<br>Prompts the user for input in a **blocking dialog** where the initial selection, possible selections, and all other options can be specified. |
| `static String` | **`showInputDialog`**`(Object message)`<br>Shows a question-message dialog requesting input from the user. |
| `static String` | **`showInputDialog`**`(Object message, Object initialSelectionValue)`<br>Shows a question-message dialog requesting input from the user, with the input value initialized to `initialSelectionValue`. |

**Tali metodi ritornano null se si preme Annulla o si chiude la finestra**

JOptionPane.**showInputDialog** ("Come ti chiami?");


String nome = JOptionPane.**showInputDialog** ("Come ti chiami?", "Mario");



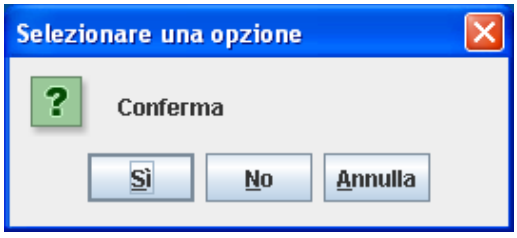

// **initialValue** è "Mario"

Tra i *metodi* **`showConfirmDialog`** per visualizzare una finestra di dialogo che chiede di **confermare** una domanda tipo yes/no/cancel, il più semplice :

| | |
|---|---|
| `static int` | **`showConfirmDialog`**`(Component parentComponent, Object message)`<br>Brings up a dialog with the options *Yes*, *No* and *Cancel*; with the title, **Select an Option**. |

**Ritorna -1 se si chiude la finestra**
**0 se si preme Sì**
**1 se si preme No**
**2 se si preme Annulla**

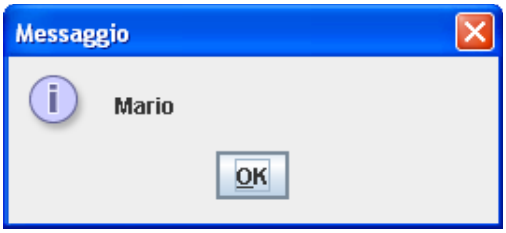
JOptionPane.**showConfirmDialog**(null, "Conferma");

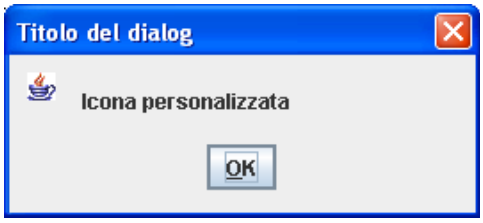


Alcuni *metodi* **`showMessageDialog`** per visualizzare una finestra di **informazione**:

| | |
|---|---|
| `static void` | **`showMessageDialog`**`(Component parentComponent, Object message)`<br>Brings up an information-message dialog titled "Message". |
| `static void` | **`showMessageDialog`**`(Component parentComponent, Object message, String title, int messageType, Icon icon)`<br>Brings up a dialog displaying a message, specifying all parameters. |

JOptionPane.**showMessageDialog**(null, nome);

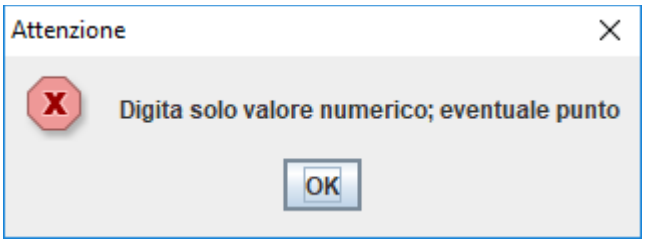



JOptionPane.**showMessageDialog** (**null**, "Icona personalizzata", "Titolo del dialog", JOptionPane.INFORMATION_MESSAGE, **new ImageIcon** ("JavaCup.gif"));

// icona (piccola immagine) personalizzata

Per **finestra di allerta** (look&feel di default per versioni più aggiornate):




JOptionPane.**showMessageDialog** (null,

"Digita solo valore numerico; eventuale punto",

"Attenzione",

JOptionPane.**ERROR_MESSAGE**);

Il *metodo* **`showOptionDialog`** costituisce Grand Unification dei metodi precedenti

| | |
|---|---|
| `static int` | **`showOptionDialog`**`(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)`<br>Brings up a dialog with a specified icon, where the initial choice is determined by the `initialValue` parameter and the number of choices |

Tutte le finestre di dialogo (*dialogs)* create sono **modali** cioè ogni metodo *showXxxDialog* blocca il thread corrente fino a quando l'interazione con l'utente non è stata completata

Esempio di applicazione con test di alcuni *metodi* di tipo ***showXxxDialog*** tra cui:

- finestra di dialogo in **lettura** con modifica del **titolo** di default della dialog box

  static String **showInputDialog** (Component parentComponent, Object message, **String title**, int messageType)

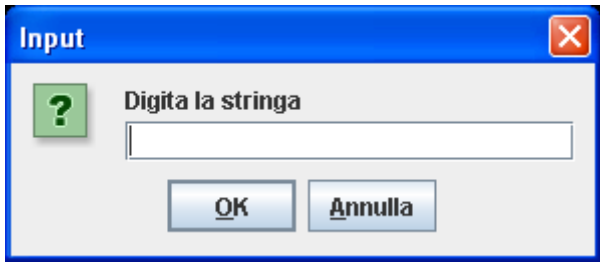
- finestra di dialogo in **scrittura** con modifica del **titolo** di default della dialog box

  static void **showMessageDialog** (Component parentComponent, Object message, **String title**, int messageType)

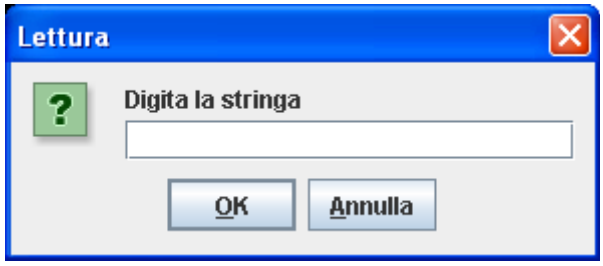
- uso del *metodo* che costituisce *Grand Unification*

  static int **showOptionDialog** (Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)

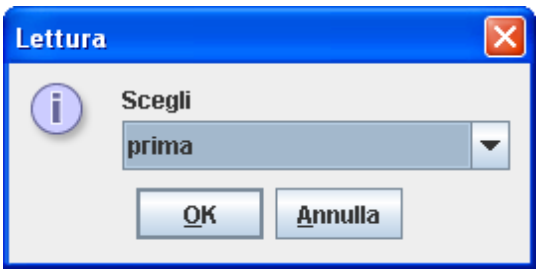
```
import javax.swing.*;     // per classi  JOptionPane e ImageIcon

class Dialog {

        private String input;

        public String getInput(){

                // uso finestra di dialogo in lettura
                 input=JOptionPane.showInputDialog("Digita la stringa");
```




```
                //  per stesso effetto .... ma con titolo "Lettura" ... invece di "Input"

                input=JOptionPane.showInputDialog(null,"Digita la stringa", "Lettura",
                                                        JOptionPane.QUESTION_MESSAGE);
```



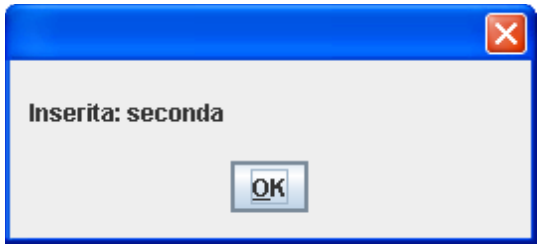

```
                // Mostra una finestra di dialogo che chiede all'utente di selezionare una stringa

                Object[] possibleValues = { "prima", "seconda" };
                Object sel_input = JOptionPane.showInputDialog(null, "Scegli", "Lettura",
                                        JOptionPane.INFORMATION_MESSAGE, null,
                                        possibleValues, possibleValues[0]);
```



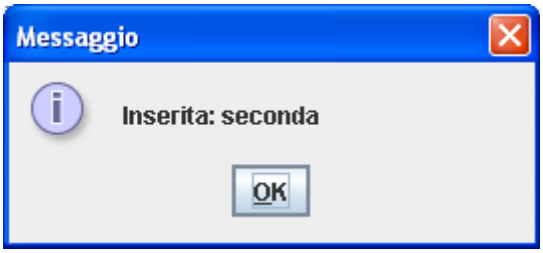


```
                if (sel_input != null)
                    input = sel_input.toString();  // oppure con casting   input = (String) sel_input;
                if (sel_input == null)  // alla pressione di Annulla oppure dell'icona di uscita
                        input ="\nHai annullato";

                return input;
        }
```

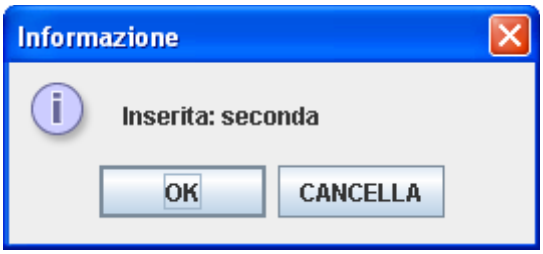
```
public void visualizza(String testo){    // uso finestra di dialogo in scrittura

        JOptionPane.showMessageDialog(null, testo, null,  JOptionPane.PLAIN_MESSAGE ); // senza icona
```



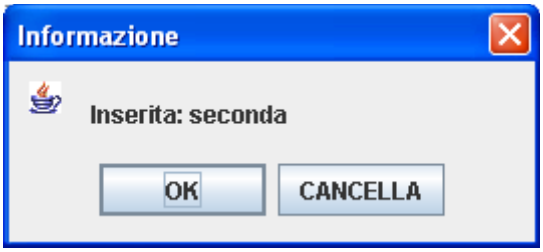

```
        JOptionPane.showMessageDialog(null, testo);    // icona di tipo "Informazione"
```



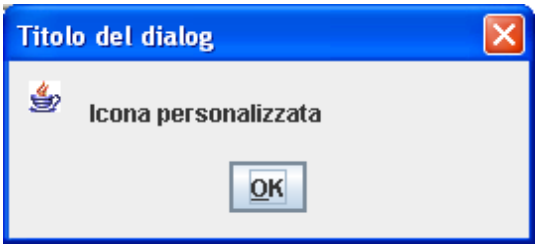

```
    // finestra di dialogo più flessibile
    Object[] options = { "OK", "CANCELLA" };
    JOptionPane.showOptionDialog (null, testo, "Informazione",  JOptionPane.DEFAULT_OPTION,
                                  JOptionPane.INFORMATION_MESSAGE, null, options,
                                  options[0]);                                  // icona di default
```




```
        JOptionPane.showOptionDialog (null, testo, "Informazione",  JOptionPane.DEFAULT_OPTION,
                                      JOptionPane.INFORMATION_MESSAGE,
                                      new ImageIcon ("JavaCup.gif"), options, options[0]);
```

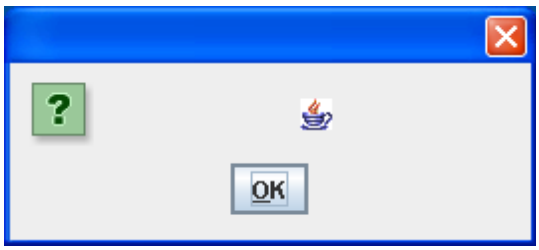



```
}
public static void main (String arg[]){
                Dialog o = new Dialog();
                String messaggio =  "Inserita: " + o.getInput();
                o.visualizza(messaggio);
}// fine main

}// fine applicazione
```



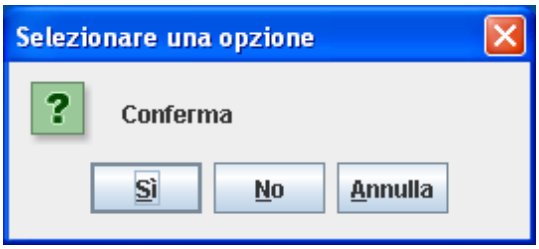

```
JOptionPane.showMessageDialog (null,
    "Icona personalizzata",
    "Titolo del dialog",
    JOptionPane.INFORMATION_MESSAGE,
    new ImageIcon("JavaCup.gif"));     // icona personalizzata
```

```
JButton b1 = new JButton("Bottone 1");
                    // Component come messaggio
JOptionPane.showMessageDialog (null, b1,
 null, JOptionPane.INFORMATION_MESSAGE );
```


```
ImageIcon img = new ImageIcon("JavaCup.gif");
                                // icona come messaggio
JOptionPane.showMessageDialog (null, img, null,
JOptionPane.QUESTION_MESSAGE );
```

```
JOptionPane.showConfirmDialog(null, "Conferma");
```



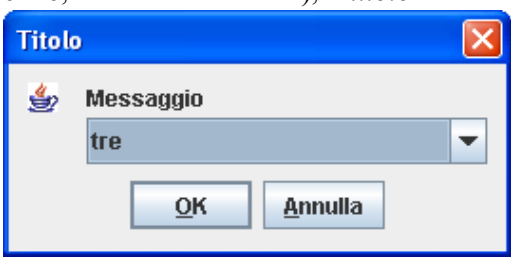

```
JLabel labelIcon = new JLabel(new ImageIcon("JavaCup.gif"));
                        // icona - logo come etichetta
Object[] options = {labelIcon, b1, b2};
JOptionPane.showOptionDialog (null, "Scegli", "Senza effetto",
JOptionPane.DEFAULT_OPTION,
JOptionPane.INFORMATION_MESSAGE, null, options,
options[0]);
```


```
Object[] options1 = {img, b1, b2};
 // icona - logo in bottone che premuto chiude dialog
JOptionPane.showOptionDialog (null, "Scegli bottone",
"Senza effetto", JOptionPane.DEFAULT_OPTION,
JOptionPane.INFORMATION_MESSAGE,
new ImageIcon ("JavaCup.gif"), options1, options1[0]);
  // icona personalizzata
```


NB:  personalizzazione del set di bottoni di opzione posti in basso

Per permettere *opzioni di scelta* in input, con metodo **showInputDialog**, volendo personalizzare l'*icona* ad esempio col logo di Java (immagine *JavaCup.gif* di  dimensioni 16x16, nella stessa cartella), il *titolo*:

```
Object[] possibleValues = { "uno", "due" ,"tre"};
JOptionPane.showInputDialog (null, "Messaggio",
"Titolo", JOptionPane.INFORMATION_MESSAGE,
new ImageIcon ("JavaCup.gif"), possibleValues, possibleValues[0]);
```




…. con modifica del Look & Feel


```
try {
     UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
} catch (Exception e) { }        // stile CDE/Motif. Scelta disponibile su qualunque piattaforma
```

**Look & Feel** : insieme delle due proprietà che caratterizzano un ambiente a finestre:

- l'aspetto dei componenti (ovvero la loro sintassi) – specifiche di visualizzazione
- la maniera in cui essi reagiscono alle azioni degli utenti (la loro semantica) – specifiche di interazione

La natura multipiattaforma di java ha spinto i progettisti di Swing a separare le problematiche di disegno grafico dei componenti da quelle inerenti al loro contenuto informativo, con la sorprendente conseguenza di permettere agli utenti di considerare il Look & Feel come una **proprietà del componente** da impostare a piacere. La distribuzione standard del JDK comprende di base due alternative: **Metal** e **Motif.**

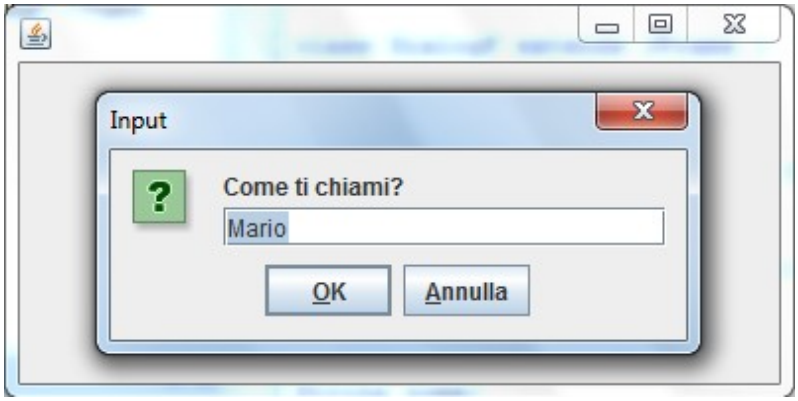
La prima definisce un Look & Feel multipiattaforma, progettato per risultare familiare agli utenti di ogni piattaforma; la seconda implementa una vista familiare agli utenti Unix. Le distribuzioni di java per Windows e Mac includono anche un L&F che richiama quello della piattaforma ospite.

Per impostare da programma un particolare look & feel è sufficiente chiamare il metodo

```
UIManager.setLookAndFeel(String className)
```

passando come parametro il nome di un l&f installato nel sistema.

Un'applicazione che crea una finestra in cui si apre una *dialog box* modale che risulta **centrata rispetto a quest'ultima**:




```
import javax.swing.*;
import java.awt.*;              // per Container

class DialogF extends JFrame {

public void visualizza(){
// per aprire una finestra
// nel punto x,y con larghezza 400, altezza 200
setBounds(100,100,400,200);
Container cont=getContentPane();
cont.setLayout(null);
setVisible(true);
String nome;
nome = JOptionPane.showInputDialog (cont,"Come ti chiami?", "Mario");
setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
}
public static void main (String arg[]){
                DialogF o = new DialogF();
                o.visualizza();
}// fine main
}// fine applicazione
```

La stessa applicazione (che crea una finestra in cui si apre una *dialog box* modale che risulta **centrata rispetto a quest'ultima**) **con modifica** del **Look & Feel**

```
import javax.swing.*;
import java.awt.*;              // per Container

class DialogF extends JFrame {

private Container cont;

public void visualizza(){
try {
        UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
 } catch (Exception e) { }      // stile CDE/Motif. Scelta disponibile su qualunque piattaforma

// per aprire una finestra
// nel punto x,y con larghezza 400, altezza 200
setBounds (100,100,400,200);

cont=getContentPane();

cont.setLayout(null);
setVisible(true);
String nome;
nome = JOptionPane.showInputDialog (cont,"Come ti chiami?", "Mario");

setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
}
public static void main (String arg[]){
                DialogF o = new DialogF();
                o.visualizza();
} // fine main
} // fine applicazione
```


**nb:**
Due parole sui metodi per visualizzare o nascondere i frames :

**setVisible(true)** per rendere visibile il frame che non lo è per default - deprecato **show()**

**setVisible(false)** ovviamente per nasconderlo - deprecato **hide()**

Per scegliere la posizione iniziale si può utilizzare il metodo **setBounds(int, int, int, int);** con argomenti nell'ordine di digitazione x ed y dell'angolo superiore sinistro e dimensioni (larghezza ed altezza) del frame.