Azzolini Riccardo 2019-11-11

Interfacce grafiche

1 Interfacce grafiche

Le **interfacce grafiche** (**GUI**, **Graphical User Interface**) sono costruite tramite dei **componenti grafici**, che, in Java, sono istanze di apposite classi.

Alcuni esempi di componenti sono:

- barre di scorrimento;
- bottoni;
- menu;
- barre dei menu;
- *combo box*.

2 Swing

Uno dei modi per realizzare interfacce grafiche in Java è *Swing*, le cui classi sono raccolte nel package `javax.swing`. Siccome Swing è basato su AWT (Abstract Window Toolkit), per usarlo è solitamente necessario importare anche i contenuti dei package `java.awt` e `java.awt.event`.

3 `JLabel`

La classe `JLabel` realizza un componente grafico (*label*, etichetta) che permette di mostrare nell'interfaccia una singola riga di testo non modificabile dall'utente, e/o un'immagine.

Questo è il principale tipo di componente usato per l'output.

3.1 Esempio

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

public class LabelTest extends JFrame {
    public LabelTest() {
        super("Testing JLabel");

        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());

        JLabel label1 = new JLabel("Label with text");
        label1.setToolTipText("This is label1");
        contentPane.add(label1);

        Icon bug = new ImageIcon("bug1.gif");
        JLabel label2 = new JLabel(
                "Label with text and icon",
                bug,
                SwingConstants.LEFT
        );
        label2.setToolTipText("This is label2");
        contentPane.add(label2);

        JLabel label3 = new JLabel();
        label3.setText("Label with icon and text at the bottom");
        label3.setIcon(bug);
        label3.setHorizontalTextPosition(SwingConstants.CENTER);
        label3.setVerticalTextPosition(SwingConstants.BOTTOM);
        label3.setToolTipText("This is label3");
        contentPane.add(label3);

        setSize(275, 170);
        setVisible(true);
    }

    public static void main(String[] args) {
        LabelTest application = new LabelTest();
        application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

- `JFrame` è la classe che implementa le finestre. Per realizzare una finestra, si definisce quindi una sottoclasse di `JFrame`.
- Tipicamente, la costruzione della finestra avviene interamente nel costruttore, mentre il `main` si limita a creare un'istanza della classe per avviare l'applicazione. In questo caso, il codice di costruzione è, appunto, contenuto nel costruttore di default:

  1. Si chiama il costruttore della superclasse, passando come argomento il titolo della finestra.
  2. La finestra è composta da varie parti (titolo, barra dei menu, ecc.). Una di queste è la parte principale, il "contenuto". L'oggetto corrispondente a questa parte, che è un'istanza della classe `Container`, si ottiene mediante il metodo `getContentPane` di `JFrame`.
  3. La parte principale della finestra può essere graficamente organizzata in modi (*layout*) diversi. Qui viene usato il più semplice, `FlowLayout`, che dispone i componenti come le parole all'interno di un paragrafo, cioè da sinistra a destra e dall'alto verso il basso (e, quindi, tende a dare risultati non desiderati quando viene ridimensionata la finestra, perché componenti che prima si trovavano su "righe" diverse possono essere automaticamente messi sulla stessa riga).
  4. Viene costruita la prima `JLabel`, `label1`, che contiene del testo (passato come argomento al costruttore) e un tooltip (impostato successivamente, mediante `setToolTipText`), che viene visualizzato quando si posiziona il mouse sopra al componente.

     Una volta creato, un componente esiste solo come oggetto, indipendentemente da qualsiasi finestra. Per inserirlo nella parte principale della finestra, si usa il metodo `add` del `contentPane`.
  5. Si costruisce una seconda istanza di `JLabel`, assegnata a `label2`, che contiene una riga di testo e un'icona. Per specificare l'icona, viene passato al costruttore un oggetto della classe `ImageIcon`, che implementa l'interfaccia `Icon` e permette di caricare l'icona da un file immagine. L'ultimo parametro del costruttore determina l'allineamento orizzontale della label (in questo caso, la costante `SwingConstants.LEFT` specifica un allineamento a sinistra).[1] Come per `label1`, successivamente alla costruzione viene impostato il tooltip, e infine `label2` viene aggiunta alla finestra.
  6. La terza `JLabel`, `label3`, viene creata con il costruttore di default, mentre il suo contenuto viene impostato in seguito, mediante i metodi `setText` e `setIcon`.[2] Si usano poi i metodi `setHorizontalTextPosition` e `setVertical`

[1] Con un `FlowLayout`, l'impostazione dell'allineamento della label non ha effetto.

[2] Usare un costruttore di `JLabel` con parametri, oppure usare quello di default e poi i metodi per l'impostazione dei contenuti, è esattamente equivalente.

`TextPosition` per stabilire la posizione del testo rispetto all'immagine. Infine, viene impostato il tooltip, e la label viene aggiunta alla finestra.

7. Invocando il metodo `setSize`, si impostano le dimensioni della finestra: 275 pixel di larghezza per 170 pixel di altezza. Solitamente, per scegliere la dimensione migliore è necessario fare alcune prove.
8. La finestra viene resa visibile, mediante la chiamata `setVisible(true)`. Ciò viene fatto solo alla fine della costruzione per garantire che l'utente veda solo la finestra completa.

- Il codice contenuto nel metodo statico `main`:

  1. Crea un'istanza di questa classe (`LabelTest`), avviando così l'applicazione.
  2. Imposta cosa succede quando viene chiusa la finestra: specificando come parametro del metodo `setDefaultCloseOperation` (definito da `JFrame`) la costante `JFrame.EXIT_ON_CLOSE`, fa in modo che la chiusura della finestra causi la terminazione di tutta l'applicazione. Se ciò non venisse fatto, di default il processo continuerebbe a eseguire dopo la chiusura della finestra (e, ad esempio, eventuali altre finestre potrebbero rimanere aperte).

4 Event handling

Le GUI reagiscono alle azioni dell'utente mediante gli **eventi**. Esistono numerosi tipi di eventi: movimenti del mouse, click su bottoni, ecc.

È quindi l'utente che guida l'esecuzione, dato che gli eventi possono arrivare in qualsiasi momento e ordine. Si dice allora che le applicazione grafiche sono *event driven*.

La gestione di un evento si basa su un modello a tre parti:

- **Sorgente dell'evento**: il componente grafico con cui l'utente interagisce.
- **Oggetto evento**: incapsula informazioni relative all'evento che si è verificato. È un'istanza di una delle sottoclassi di `java.awt.AWTEvent`.
- **Event listener** ("ascoltatore"): quando un evento si verifica, viene notificato, riceve l'oggetto evento, e risponde eseguendo una qualche azione.

  Un listener è un'istanza di una classe che implementa una delle interfacce contenute nel package `java.awt.event`.[3] Ognuna di queste interfacce specifica un singolo metodo (*event handler*), che viene chiamato quando si verifica un evento, e riceve come parametro l'oggetto evento.

[3] Alcune interfacce listener sono invece contenute in `javax.swing.event`.

Esistono numerosi tipi di event listener, ciascuno dei quali è preposto a raccogliere eventi di un certo tipo, e implementa un'interfaccia diversa.

Il programmatore deve:

- creare uno o più listener, implementando i metodi di gestione specificati dalle interfacce corrispondenti ai tipi di eventi da gestire;
- registrare un listener presso ogni sorgente di eventi che vuole gestire.

Grazie al meccanismo della registrazione, un listener non ha bisogno di conoscere le sorgenti degli eventi di cui si dovrà occupare. Spetta invece a ciascuna delle sorgenti mantenere dei riferimenti ai listener registrati, e inviare ciascun tipo di evento ai listener corrispondenti.

5 Campi di testo

- Il componente `JTextField` è un'area in cui l'utente può inserire una singola riga di testo.
- Il componente `JPasswordField`, che estende `JTextField`, nasconde (di default) i caratteri inseriti dall'utente.

6 Esempio di campi di testo ed eventi

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

public class TextFieldTest extends JFrame {
    private JTextField textField1, textField2, textField3;
    private JPasswordField passwordField;

    public TextFieldTest() {
        super("Testing JTextField and JPasswordField");

        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());

        textField1 = new JTextField(10);
        contentPane.add(textField1);

        textField2 = new JTextField("Enter text here");
```

```
        contentPane.add(textField2);

        textField3 = new JTextField("Uneditable text field", 20);
        textField3.setEditable(false);
        contentPane.add(textField3);

        passwordField = new JPasswordField("Hidden text");
        contentPane.add(passwordField);

        TextFieldHandler handler = new TextFieldHandler();
        textField1.addActionListener(handler);
        textField2.addActionListener(handler);
        textField3.addActionListener(handler);
        passwordField.addActionListener(handler);

        setSize(325, 100);
        setVisible(true);
    }

    public static void main(String[] args) {
        TextFieldTest application = new TextFieldTest();
        application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    private class TextFieldHandler implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent event) {
            String message = "";
            if (event.getSource() == textField1) {
                message = "textField1: " + event.getActionCommand();
            } else if (event.getSource() == textField2) {
                message = "textField2: " + event.getActionCommand();
            } else if (event.getSource() == textField3) {
                message = "textField3: " + event.getActionCommand();
            } else if (event.getSource() == passwordField) {
                message = "passwordField: "
                        + new String(passwordField.getPassword());
            }
            JOptionPane.showMessageDialog(null, message);
        }
    }
}
```

Come nell'esempio precedente, la finestra è realizzata all'interno del costruttore per

default:

1. Il primo `JTextField`, `textField1`, viene creato specificando, come argomento del costruttore, il numero di *colonne* di larghezza.[4]
2. Per `textField2`, viene specificato il testo che contiene all'inizio (che, comunque, potrà poi essere modificato dall'utente).
3. Nella costruzione di `textField3`, si specificano sia il testo iniziale che la larghezza. Si usa poi il metodo `setEditable` per rendere il contenuto del campo non modificabile, come una label. C'è però una differenza tra un campo di testo non modificabile e una label: il campo di testo potrà essere reso nuovamente modificabile in un secondo momento (ad esempio, in seguito a un qualche evento).
4. Per il `JPasswordField` viene specificato un testo di default, che verrà visualizzato come "••••••••••••".
5. Si costruisce un'istanza di `TextFieldHandler`, che è una classe interna privata definita dentro a `TextFieldTest`. Tale oggetto viene poi registrato come listener per gli eventi generati dai vari campi di testo.[5] Questi eventi si verificano quando l'utente preme invio all'interno di uno dei campi.

La classe `TextFieldHandler` implementa l'interfaccia `ActionListener`, che specifica un unico metodo, `actionPerformed`: questo viene chiamato quando si verifica un evento, e riceve come parametro l'oggetto evento, che in questo caso è un'istanza della classe `ActionEvent`.

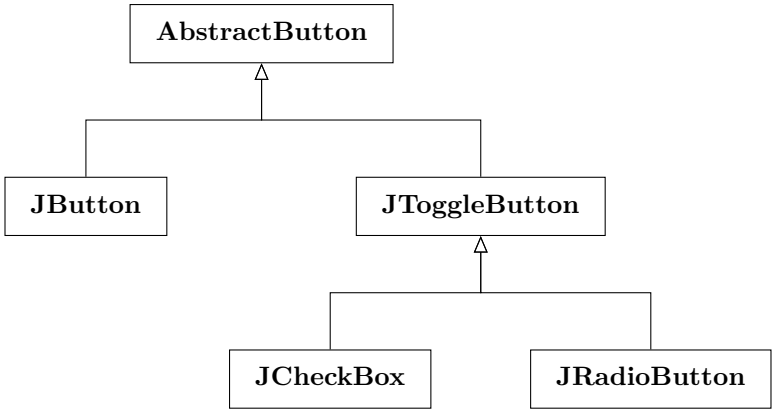
- Il metodo `getSource()` dell'oggetto evento restituisce il componente grafico che ha generato l'evento.
- `event.getActionCommand()` restituisce, in questo caso, il contenuto del campo di testo da cui ha avuto origine l'evento.
- Per ottenere il testo contenuto da `passwordField`, si usa il metodo `getPassword()`.[6] Esso restituisce un array di caratteri, che deve quindi essere convertito in una stringa per poterlo assegnare alla variabile `message`.
- `JOptionPane.showMessageDialog` mostra all'utente una finestra, intitolata "Message", che contiene il testo passato come secondo argomento (in questo caso, la stringa assegnata a `message`) e un pulsante "OK".

[4] Una colonna corrisponde approssimativamente alla larghezza di un carattere.
[5] In questo esempio, si usa lo stesso oggetto come listener per tutte le sorgenti di eventi, ma si potrebbero anche usare oggetti diversi.
[6] In alternativa, si potrebbe usare anche in questo caso il metodo `event.getActionCommand()`.

7 Bottoni

In generale, i bottoni sono componenti che l'utente clicca per effettuare determinate azioni.

Esistono vari tipi di bottoni, implementati dalle varie sottoclassi di `javax.swing.AbstractButton`:



In particolare, `JButton` genera semplicemente un evento quando viene cliccato, mentre i `JToggleButton` hanno uno stato (selezionato / non selezionato) che l'utente può cambiare con un click:

- lo stato di un `JCheckBox` è indipendente da quello di altri bottoni;
- lo stato di un `JRadioButton` dipende anche da quello di altri `JRadioButton`, appartenenti a uno stesso *gruppo*: in ogni momento, al massimo uno dei `JRadioButton` di un gruppo può essere selezionato.

7.1 Esempio di `JButton`

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

public class ButtonTest extends JFrame {
    public ButtonTest() {
        super("Testing Buttons");

        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());

        JButton plainButton = new JButton("Plain Button");
```

```
        contentPane.add(plainButton);

        Icon bug1 = new ImageIcon("bug1.gif");
        Icon bug2 = new ImageIcon("bug2.gif");
        JButton fancyButton = new JButton("Fancy Button", bug1);
        fancyButton.setRolloverIcon(bug2);
        contentPane.add(fancyButton);

        ButtonHandler handler = new ButtonHandler();
        plainButton.addActionListener(handler);
        fancyButton.addActionListener(handler);

        setSize(275, 100);
        setVisible(true);
    }

    public static void main(String[] args) {
        ButtonTest application = new ButtonTest();
        application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    private class ButtonHandler implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent event) {
            JOptionPane.showMessageDialog(
                    ButtonTest.this,
                    "You pressed: " + event.getActionCommand()
            );
        }
    }
}
```

- `plainButton` è un bottone che contiene solo del testo.
- `fancyButton`, invece, ha del testo, un'icona "normale", e un'icona diversa che viene mostrata quando il mouse passa sul bottone.
- Per entrambi i bottoni viene registrato un handler che mostra un messaggio contenente il nome del bottone cliccato dall'utente. Per ottenere il nome, si usa il metodo `event.getActionCommand()`.

7.2 Esempio di JCheckBox

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

public class CheckBoxTest extends JFrame {
    private JTextField field;
    private JCheckBox bold, italic;

    public CheckBoxTest() {
        super("JCheckBox Test");

        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());

        field = new JTextField("watch the font style change", 20);
        field.setFont(new Font("Serif", Font.PLAIN, 14));
        contentPane.add(field);

        bold = new JCheckBox("Bold");
        contentPane.add(bold);

        italic = new JCheckBox("Italic");
        contentPane.add(italic);

        CheckBoxHandler handler = new CheckBoxHandler();
        bold.addItemListener(handler);
        italic.addItemListener(handler);

        setSize(275, 100);
        setVisible(true);
    }

    public static void main(String[] args) {
        CheckBoxTest application = new CheckBoxTest();
        application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    private class CheckBoxHandler implements ItemListener {
        @Override
        public void itemStateChanged(ItemEvent event) {
            int fontStyle = Font.PLAIN;
```

```
            if (bold.isSelected()) {
                fontStyle |= Font.BOLD;
            }
            if (italic.isSelected()) {
                fontStyle |= Font.ITALIC;
            }
            field.setFont(new Font("Serif", fontStyle, 14));
        }
    }
}
```

- Il metodo `field.setFont` imposta il carattere tipografico usato per il contenuto del campo di testo. L'argomento è un'istanza della classe `Font`, il cui costruttore ha tre argomenti:
  1. il nome del carattere;[7]
  2. lo stile del carattere, che può essere normale (`Font.PLAIN`), oppure grassetto (`Font.BOLD`) e/o corsivo (`Font.ITALIC`);
  3. la dimensione del carattere.
- La classe `CheckBoxHandler` implementa l'interfaccia `ItemListener`, che si usa quando non interessa tanto il verificarsi di un evento puntuale, ma piuttosto un cambiamento di stato. Essa specifica il singolo metodo `itemStateChanged`.
- Il metodo `isSelected` di `JCheckBox` restituisce `true` se il checkbox è selezionato, e `false` altrimenti.
- Il metodo `itemStateChanged` di `CheckBoxHandler` modifica il carattere del campo di testo, calcolando lo stile in base allo stato dei due checkbox. Il calcolo funziona perché i valori delle costanti `Font.PLAIN`, `Font.BOLD` e `Font.ITALIC` possono essere combinati con l'operazione di OR bit a bit (oppure con la somma). In particolare, siccome `Font.PLAIN` vale 0, essa funge da valore "neutro", al quale possono essere aggiunte le altre costanti.

  Non è comunque necessario conoscere i valori esatti delle costanti per poter effettuare questo calcolo, e sarebbe sbagliato (dal punto di vista della progettazione in piccolo) usare direttamente i valori numerici invece delle costanti simboliche.

[7] `"Serif"` specifica un generico carattere *con grazie*. Altri caratteri generici disponibili sono `"SansSerif"` (*senza grazie*) e `"Monospaced"` (*non proporzionale*, cioè nel quale tutti i caratteri hanno la stessa larghezza).

7.3 Esempio di JRadioButton

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

public class RadioButtonTest extends JFrame {
    private JTextField field;

    public RadioButtonTest() {
        super("RadioButton Test");

        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());

        field = new JTextField("watch the font style change", 25);
        contentPane.add(field);

        JRadioButton plainButton = new JRadioButton("Plain", true);
        contentPane.add(plainButton);

        JRadioButton boldButton = new JRadioButton("Bold", false);
        contentPane.add(boldButton);

        JRadioButton italicButton = new JRadioButton("Italic", false);
        contentPane.add(italicButton);

        JRadioButton boldItalicButton =
            new JRadioButton("Bold/Italic", false);
        contentPane.add(boldItalicButton);

        ButtonGroup radioGroup = new ButtonGroup();
        radioGroup.add(plainButton);
        radioGroup.add(boldButton);
        radioGroup.add(italicButton);
        radioGroup.add(boldItalicButton);

        Font plainFont = new Font("Serif", Font.PLAIN, 14);
        Font boldFont = new Font("Serif", Font.BOLD, 14);
        Font italicFont = new Font("Serif", Font.ITALIC, 14);
        Font boldItalicFont =
            new Font("Serif", Font.BOLD | Font.ITALIC, 14);
```

```
        field.setFont(plainFont);
        plainButton.addItemListener(new RadioButtonHandler(plainFont));
        boldButton.addItemListener(new RadioButtonHandler(boldFont));
        italicButton.addItemListener(new RadioButtonHandler(italicFont));
        boldItalicButton.addItemListener(
            new RadioButtonHandler(boldItalicFont)
        );

        setSize(300, 100);
        setVisible(true);
    }

    public static void main(String[] args) {
        RadioButtonTest application = new RadioButtonTest();
        application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    private class RadioButtonHandler implements ItemListener {
        private final Font font;

        public RadioButtonHandler(Font f) {
            font = f;
        }

        @Override
        public void itemStateChanged(ItemEvent event) {
            field.setFont(font);
        }
    }
}
```

- Il secondo parametro del costruttore di `JRadioButton` specifica lo stato di selezione iniziale: in questo caso, si specifica che deve essere inizialmente selezionato `plainButton`, poiché il carattere iniziale del campo di testo è "normale" ("plain").
- Per specificare quali `JRadioButton` sono alternativi l'uno all'altro, li si aggiunge a uno stesso gruppo, `radioGroup`, che è un'istanza della classe `ButtonGroup`.
- Siccome, tra i `JRadioButton` del gruppo, ne può essere selezionato uno solo alla volta, per ottenere tutte le possibili combinazioni di grassetto e corsivo sono necessari 4 `JRadioButton` (mentre, nell'esempio precedente, erano sufficienti 2 `JCheckBox`).
- La classe `RadioButtonHandler` ha un costruttore con un parametro che specifica il carattere da impostare quando si verifica l'evento. Per ogni `JRadioButton`, viene

quindi registrata un'istanza diversa di `RadioButtonHandler`, in modo da impostare il carattere corrispondente al bottone selezionato.