Azzolini Riccardo 2019-11-14

Interfacce grafiche

1 JComboBox

Un `JComboBox` è una *lista drop-down*, dalla quale l'utente può selezionare un elemento.

1.1 Esempio

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

public class ComboBoxTest extends JFrame {
    public ComboBoxTest() {
        super("Testing JComboBox");

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

        String[] names = {
                "bug1.gif", "bug2.gif", "travelbug.gif", "buganim.gif"
        };
        Icon[] icons = new Icon[names.length];
        for (int i = 0; i < icons.length; i++) {
            icons[i] = new ImageIcon(names[i]);
        }

        JLabel label = new JLabel(icons[0]);

        JComboBox<String> imagesComboBox = new JComboBox<>(names);
        imagesComboBox.setMaximumRowCount(3);
        imagesComboBox.addItemListener(new ItemListener() {
            @Override
            public void itemStateChanged(ItemEvent event) {
                if (event.getStateChange() == ItemEvent.SELECTED) {
                    Icon newIcon =
                        icons[imagesComboBox.getSelectedIndex()];
```

```
                    label.setIcon(newIcon);
                }
            }
        });
        contentPane.add(imagesComboBox);

        contentPane.add(label);

        setSize(350, 100);
        setVisible(true);
    }

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

- Le stringhe contenute nell'array `names` sono usate sia come nomi dei file delle icone che come valori del `JComboBox`. L'array `icons` contiene invece le icone vere e proprie.
- Nella costruzione del `JComboBox` viene specificato l'array di valori selezionabili. Si usa poi il metodo `setMaximumRowCount` per limitare il numero di elementi visualizzati contemporaneamente nella tendina di selezione: siccome ci sono in totale 4 elementi, verrà visualizzata una barra di scorrimento.
- Il listener viene definito come istanza di una **classe interna anonima** che implementa `ItemListener`. Questa tecnica si usa spesso (per scrivere i listener in modo più compatto), purché non sia necessario creare istanze diverse dello stesso listener: siccome una classe anonima non ha nome, è impossibile invocarne il costruttore, quindi essa ha una sola istanza, creata automaticamente dalla dichiarazione della classe.
- Quando viene selezionato un valore, si usa il metodo `imagesComboBox.getSelectedIndex()` per ottenere l'indice di tale valore all'interno dell'array `names`, e si assegna alla `label` l'icona corrispondente.

2 JList

Una `JList` è una lista che visualizza una serie di elementi, e permette all'utente di selezionarne uno di essi, o, opzionalmente, anche più di uno.

2.1 Esempio

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

public class ListTest extends JFrame {
    private final String[] colorNames = {
            "Black", "Blue", "Cyan", "Dark Gray",
            "Gray", "Green", "Light Gray", "Magenta",
            "Orange", "Pink", "Red", "White",
            "Yellow"
    };
    private final Color[] colors = {
            Color.BLACK, Color.BLUE, Color.CYAN, Color.DARK_GRAY,
            Color.GRAY, Color.GREEN, Color.LIGHT_GRAY, Color.MAGENTA,
            Color.ORANGE, Color.PINK, Color.RED, Color.WHITE,
            Color.YELLOW
    };

    public ListTest() {
        super("List Test");

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

        JList<String> colorList = new JList<>(colorNames);
        colorList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        colorList.setVisibleRowCount(5);
        contentPane.add(new JScrollPane(colorList));

        colorList.addListSelectionListener(new ListSelectionListener() {
            @Override
            public void valueChanged(ListSelectionEvent event) {
                Color newColor =
                    colors[colorList.getSelectedIndex()];
                contentPane.setBackground(newColor);
            }
        });

        setSize(350, 150);
        setVisible(true);
    }
```

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

- Come il `JComboBox` nell'esempio precedente, la `JList` viene costruita specificando l'array di valori selezionabili.
- Si usa il metodo `colorList.setSelectionMode` per specificare che può essere selezionato solo un elemento alla volta (cioè che la selezione è singola, e non multipla).
- Con `colorList.setVisibleRowCount(5)`, il numero di elementi della lista contemporaneamente visibili viene impostato a 5. Siccome ci sono più di 5 elementi, è necessaria una barra di scorrimento, ma (a differenza di un `JComboBox`) questa non viene aggiunta automaticamente: bisogna invece mettere la `JList` in un `JScrollPane`, che viene poi aggiunto al `contentPane` (mentre la lista *non* viene aggiunta direttamente).
- Il listener, che implementa l'interfaccia `ListSelectionListener`, cambia il colore di sfondo della finestra, usando `contentPane.setBackground`, ogni volta che la selezione cambia.

3 JTextArea

Una `JTextArea` è un campo di testo multiriga (mentre un `JTextField` permette solo l'inserimento di una singola riga).

3.1 Esempio

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

public class TextAreaDemo extends JFrame {
    public TextAreaDemo() {
        super("TextArea Demo");

        Box box = Box.createHorizontalBox();

        String text = "This is a demo string to\n" +
```

```
                "illustrate copying text\n" +
                "from one textarea to\n" +
                "another textarea using an\n" +
                "external event\n";
        JTextArea textArea1 = new JTextArea(text, 10, 15);
        box.add(new JScrollPane(textArea1));

        JButton copyButton = new JButton("Copy >>>");
        box.add(copyButton);

        JTextArea textArea2 = new JTextArea(10, 15);
        textArea2.setEditable(false);
        box.add(new JScrollPane(textArea2));

        copyButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent event) {
                textArea2.setText(textArea1.getSelectedText());
            }
        });

        Container contentPane = getContentPane();
        contentPane.add(box);

        setSize(425, 200);
        setVisible(true);
    }

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

- `Box` è un contenitore di oggetti grafici.
- Gli argomenti passati al costruttore di `JTextArea` sono:
  1. il contenuto iniziale;
  2. il numero di righe;
  3. il numero di colonne.
- L'area di testo viene munita di barre di scorrimento usando `JScrollPane`. Esso aggiunge sia una barra orizzontale che una verticale, ma ciascuna di esse vie-

ne visualizzata solo se è necessaria, cioè se il contenuto fuoriesce in direzione orizzontale/verticale dall'area visibile.

- Per `textArea2`, viene impedito all'utente di modificare il contenuto (usando `setEditable(false)`, come per un `JTextBox`), ma esso può comunque essere alterato dal programma.
- Quando l'utente clicca il bottone, il listener copia il testo selezionato da `textArea1` a `textArea2`.

4 Layout manager

I **layout manager** determinano come i componenti vengono disposti all'interno della finestra. Essi sono implementazioni dell'interfaccia `java.awt.LayoutManager`.

4.1 `FlowLayout`

`FlowLayout` posiziona i componenti come le parole in un paragrafo di testo, da sinistra a destra e dall'alto verso il basso. Come, appunto, un paragrafo di testo, i componenti possono essere allineati a sinistra, centrati, o allineati a destra.

4.1.1 Esempio

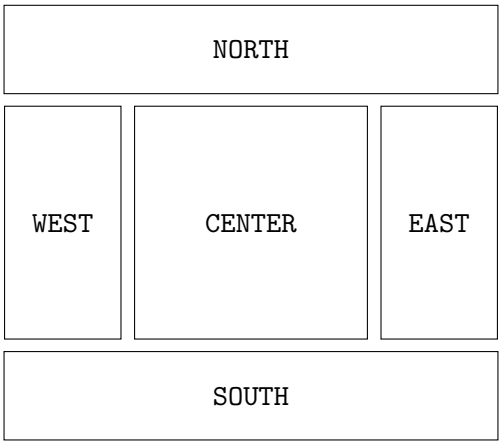
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class FlowLayoutDemo extends JFrame {
    private Container contentPane;
    private FlowLayout layout;

    public FlowLayoutDemo() {
        super("FlowLayout Demo");

        layout = new FlowLayout();

        contentPane = getContentPane();
        contentPane.setLayout(layout);

        JButton leftButton = new JButton("Left");
        contentPane.add(leftButton);
        leftButton.addActionListener(
```

```
            new AlignmentHandler(FlowLayout.LEFT)
        );

        JButton centerButton = new JButton("Center");
        contentPane.add(centerButton);
        centerButton.addActionListener(
            new AlignmentHandler(FlowLayout.CENTER)
        );

        JButton rightButton = new JButton("Right");
        contentPane.add(rightButton);
        rightButton.addActionListener(
            new AlignmentHandler(FlowLayout.RIGHT)
        );

        setSize(300, 75);
        setVisible(true);
    }

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

    private class AlignmentHandler implements ActionListener {
        private final int alignment;

        private AlignmentHandler(int alignment) {
            this.alignment = alignment;
        }

        @Override
        public void actionPerformed(ActionEvent event) {
            layout.setAlignment(alignment);
            layout.layoutContainer(contentPane);
        }
    }
}
```

Quando viene cambiato l'allineamento del `FlowLayout` dopo la creazione della finestra, è necessario chiamare il metodo `layoutContainer` per aggiornare il layout. Questo è un primo esempio di istruzione di "ridisegnamento" manuale: esse sono talvolta necessarie per applicare effettivamente alcuni cambiamenti.

4.2 BorderLayout

`BorderLayout` suddivide la finestra in 5 parti: `NORTH`, `SOUTH`, `EAST`, `WEST`, e `CENTER`.



4.2.1 Esempio

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

public class BorderLayoutDemo extends JFrame implements ActionListener {
    private BorderLayout layout;
    private JButton[] buttons;

    public BorderLayoutDemo() {
        super("BorderLayout Demo");

        layout = new BorderLayout(5, 5);

        Container contentPane = getContentPane();
        contentPane.setLayout(layout);

        String[] names = {
                "Hide North", "Hide South",
                "Hide East", "Hide West",
                "Hide Center"
        };
        buttons = new JButton[names.length];
        for (int i = 0; i < buttons.length; i++) {
```

```
            buttons[i] = new JButton(names[i]);
            buttons[i].addActionListener(this);
        }
        contentPane.add(buttons[0], BorderLayout.NORTH);
        contentPane.add(buttons[1], BorderLayout.SOUTH);
        contentPane.add(buttons[2], BorderLayout.EAST);
        contentPane.add(buttons[3], BorderLayout.WEST);
        contentPane.add(buttons[4], BorderLayout.CENTER);

        setSize(300, 200);
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent event) {
        for (JButton button : buttons) {
            if (event.getSource() == button) {
                button.setVisible(false);
            } else {
                button.setVisible(true);
            }
        }
        layout.layoutContainer(getContentPane());
    }

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

- In questo esempio, la classe `BorderLayoutDemo` funge sia da finestra che da listener per gli eventi (invece di usare, ad esempio, una classe interna).
- Come parametri del costruttore di `BorderLayout` vengono specificate le distanze ("gap") orizzontale e verticale tra componenti adiacenti (in questo caso, entrambe sono impostate a 5 pixel).
- Quando si aggiunge un elemento al `contentPane`, è necessario specificare in quale delle 5 parti del `BorderLayout` posizionarlo, passando un'apposita costante come secondo parametro al metodo `add`.

4.3 GridLayout

`GridLayout` posiziona i componenti nelle celle di una matrice, disponendoli da sinistra a destra e dall'alto verso il basso, in base all'ordine in cui vengono aggiunti alla finestra. Tutte le celle hanno le stesse dimensioni.

4.3.1 Esempio

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

public class GridLayoutDemo extends JFrame implements ActionListener {
    private Container contentPane;
    private GridLayout grid1, grid2;
    private boolean toggle = true;

    public GridLayoutDemo() {
        super("GridLayout Demo");

        grid1 = new GridLayout(2, 3, 5, 5);
        grid2 = new GridLayout(3, 2);

        contentPane = getContentPane();
        contentPane.setLayout(grid1);

        String[] names = {
                "one", "two", "three", "four", "five", "six"
        };
        JButton[] buttons = new JButton[names.length];
        for (int i = 0; i < buttons.length; i++) {
            buttons[i] = new JButton(names[i]);
            buttons[i].addActionListener(this);
            contentPane.add(buttons[i]);
        }

        setSize(300, 150);
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent event) {
        if (toggle) {
```

```
            contentPane.setLayout(grid2);
        } else {
            contentPane.setLayout(grid1);
        }
        toggle = !toggle;
        contentPane.validate();
    }

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

- Il costruttore di `GridLayout` permette di specificare il numero di righe e di colonne, e, opzionalmente, i gap orizzontali e verticali:
  - `grid1` ha 2 righe, 3 colonne, e gap di 5 pixel tra componenti adiacenti;
  - `grid2` ha 3 righe, 2 colonne, e nessun gap.
- `contentPane.validate()` è un altro modo di aggiornare il layout della finestra, come `layout.layoutContainer(contentPane)` negli esempi precedenti.

4.4 `GridBagLayout`

`GridBagLayout` realizza una struttura a matrice, simile a `GridLayout`, ma gli elementi:

- si possono estendere su più righe e/o colonne;
- non devono avere per forza tutti le stesse dimensioni;
- possono essere posizionati indipendentemente dall'ordine di aggiunta alla finestra.

Per specificare il posizionamento di ciascun componente si usa un oggetto `GridBagConstraints`.

4.4.1 Esempio

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

public class GridBagDemo extends JFrame {
    private Container contentPane;
    private GridBagLayout layout;
    private GridBagConstraints constraints;
```

```
public GridBagDemo() {
    super("GridBagLayout");

    contentPane = getContentPane();
    layout = new GridBagLayout();
    contentPane.setLayout(layout);

    constraints = new GridBagConstraints();

    JTextArea textArea1 = new JTextArea("TextArea1", 5, 10);
    JTextArea textArea2 = new JTextArea("TextArea2", 2, 2);

    String[] names = {"Iron", "Steel", "Brass"};
    JComboBox<String> comboBox = new JComboBox<>(names);

    JTextField textField = new JTextField("TextField");
    JButton button1 = new JButton("Button 1");
    JButton button2 = new JButton("Button 2");
    JButton button3 = new JButton("Button 3");

    constraints.fill = GridBagConstraints.BOTH;
    // fill = BOTH, weightx = 0, weighty = 0
    addComponent(textArea1, 0, 0, 1, 3);

    constraints.fill = GridBagConstraints.HORIZONTAL;
    // fill = HORIZONTAL, weightx = 0, weighty = 0
    addComponent(button1, 0, 1, 2, 1);

    // fill = HORIZONTAL, weightx = 0, weighty = 0
    addComponent(comboBox, 2, 1, 2, 1);

    constraints.weightx = 1000;
    constraints.weighty = 1;
    constraints.fill = GridBagConstraints.BOTH;
    // fill = BOTH, weightx = 1000, weighty = 1
    addComponent(button2, 1, 1, 1, 1);

    constraints.weightx = 0;
    constraints.weighty = 0;
    // fill = BOTH, weightx = 0, weighty = 0
    addComponent(button3, 1, 2, 1, 1);

    // fill = BOTH, weightx = 0, weighty = 0
```

```
        addComponent(textField, 3, 0, 2, 1);

        // fill = BOTH, weightx = 0, weighty = 0
        addComponent(textArea2, 3, 2, 1, 1);

        setSize(300, 150);
        setVisible(true);
    }

    private void addComponent(
            Component component,
            int row, int column,
            int width, int height
    ) {
        constraints.gridy = row;
        constraints.gridx = column;
        constraints.gridwidth = width;
        constraints.gridheight = height;

        layout.setConstraints(component, constraints);
        contentPane.add(component);
    }

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

- `constraints.fill` specifica se e in quali direzioni un componente può essere ridimensionato per fare in modo che riempia completamente le celle della matrice che occupa (nel caso in cui la dimensione naturale sarebbe inferiore a quella delle celle):
  - `GridBagConstraints.NONE` impedisce di ridimensionare il componente;
  - `GridBagConstraints.HORIZONTAL` permette di ridimensionare il componente in larghezza, ma non in altezza;
  - `GridBagConstraints.VERTICAL` permette di ridimensionare il componente in altezza, ma non in larghezza;
  - `GridBagConstraints.BOTH` permette il ridimensionamento del componente sia in larghezza che in altezza.

- `addComponent` è un metodo utility, definito all'interno di questa classe, che facilita l'inserimento di un componente nella matrice, permettendo di specificare la riga, la colonna, la larghezza (cioè il numero di celle occupate orizzontalmente), e l'altezza (cioè il numero di celle occupate verticalmente). Esso imposta i campi corrispondenti dell'oggetto `constraints`, associa tale oggetto al componente (`layout.setConstraints`), e infine aggiunge il componente alla finestra.
- Le celle sono numerate a partire da `0, 0`, che si trova in alto a sinistra (la stessa numerazione usata solitamente per le matrici).
- `constraints.weightx` e `constraints.weighty` specificano come varia la dimensione, rispettivamente delle colonne e delle righe, quando la finestra viene ridimensionata (e, in generale, quando la dimensione naturale della matrice è minore di quella della finestra in cui si trova). Di default, valgono entrambi 0, quindi le celle hanno dimensione fissa. Se, invece, una colonna contiene almeno un componente con `weightx > 0` (o, analogamente, una riga contiene almeno un componente con `weighty > 0`), essa può essere ridimensionata. Lo spazio aggiuntivo disponibile viene distribuito in proporzione ai valori di `weightx/weighty`: le colonne/righe con valori più alti occupano una frazione maggiore di tale spazio.

  In questo esempio, possono aumentare solo le dimensioni della riga e della colonna contenenti `button2`, dato che esso è l'unico componente inserito con `weightx` e `weighty` maggiori di 0.

5 Menu

A una finestra (`JFrame`) può essere aggiunta una barra dei menu, implementata dalla classe `JMenuBar`. Tale barra può contenere vari tipi di componenti, tra cui:

- bottoni (`JMenuItem`);
- checkbox (`JCheckBoxMenuItem`);
- radio button (`JRadioButtonMenuItem`);
- menu, anche annidati (`JMenu`).

I vari tipi di bottoni che possono essere inseriti nei menu funzionano in modo analogo alle versioni "normali", usate nella parte principale della finestra.

5.1 Esempio

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

public class MenuTest extends JFrame {
    private final Color[] colorValues = {
            Color.BLACK, Color.BLUE, Color.RED, Color.GREEN
    };
    private JRadioButtonMenuItem[] colorItems, fontItems;
    private JCheckBoxMenuItem[] styleItems;
    private JLabel displayLabel;
    private int style = Font.PLAIN;

    public MenuTest() {
        super("Using JMenus");

        JMenuBar bar = new JMenuBar();
        setJMenuBar(bar);

        JMenu fileMenu = new JMenu("File");
        fileMenu.setMnemonic('F');
        bar.add(fileMenu);

        JMenuItem aboutItem = new JMenuItem("About...");
        aboutItem.setMnemonic('A');
        fileMenu.add(aboutItem);
        aboutItem.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent event) {
                JOptionPane.showMessageDialog(
                        MenuTest.this,
                        "This is an example\nof using menus",
                        "About",
                        JOptionPane.PLAIN_MESSAGE
                );
            }
        });

        JMenuItem exitItem = new JMenuItem("Exit");
        exitItem.setMnemonic('x');
        fileMenu.add(exitItem);
```

```
exitItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent event) {
        System.exit(0);
    }
});

JMenu formatMenu = new JMenu("Format");
formatMenu.setMnemonic('r');
bar.add(formatMenu);

ItemHandler itemHandler = new ItemHandler();

JMenu colorMenu = new JMenu("Color");
colorMenu.setMnemonic('C');
formatMenu.add(colorMenu);

String[] colors = {"Black", "Blue", "Red", "Green"};
colorItems = new JRadioButtonMenuItem[colors.length];
ButtonGroup colorGroup = new ButtonGroup();
for (int i = 0; i < colorItems.length; i++) {
    colorItems[i] = new JRadioButtonMenuItem(colors[i]);
    colorMenu.add(colorItems[i]);
    colorGroup.add(colorItems[i]);
    colorItems[i].addActionListener(itemHandler);
}
colorItems[0].setSelected(true);

formatMenu.addSeparator();

JMenu fontMenu = new JMenu("Font");
fontMenu.setMnemonic('n');
formatMenu.add(fontMenu);

String[] fontNames = {"Serif", "Monospaced", "SansSerif"};
fontItems = new JRadioButtonMenuItem[fontNames.length];
ButtonGroup fontGroup = new ButtonGroup();
for (int i = 0; i < fontItems.length; i++) {
    fontItems[i] = new JRadioButtonMenuItem(fontNames[i]);
    fontMenu.add(fontItems[i]);
    fontGroup.add(fontItems[i]);
    fontItems[i].addActionListener(itemHandler);
}
fontItems[0].setSelected(true);
```

```
        fontMenu.addSeparator();

        String[] styleNames = {"Bold", "Italic"};
        styleItems = new JCheckBoxMenuItem[styleNames.length];
        StyleHandler styleHandler = new StyleHandler();
        for (int i = 0; i < styleNames.length; i++) {
            styleItems[i] = new JCheckBoxMenuItem(styleNames[i]);
            fontMenu.add(styleItems[i]);
            styleItems[i].addItemListener(styleHandler);
        }

        displayLabel = new JLabel("Sample Text", SwingConstants.CENTER);
        displayLabel.setForeground(colorValues[0]);
        displayLabel.setFont(new Font("Serif", Font.PLAIN, 72));

        Container contentPane = getContentPane();
        contentPane.setBackground(Color.CYAN);
        contentPane.add(displayLabel, BorderLayout.CENTER);

        setSize(500, 200);
        setVisible(true);
    }

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

    private class ItemHandler implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent event) {
            for (int i = 0; i < colorItems.length; i++) {
                if (colorItems[i].isSelected()) {
                    displayLabel.setForeground(colorValues[i]);
                    break;
                }
            }

            for (JRadioButtonMenuItem fontItem : fontItems) {
                if (event.getSource() == fontItem) {
                    String name = fontItem.getText();
                    displayLabel.setFont(new Font(name, style, 72));
                    break;
```

```
                }
            }
        }
    }

    private class StyleHandler implements ItemListener {
        @Override
        public void itemStateChanged(ItemEvent event) {
            style = Font.PLAIN;
            if (styleItems[0].isSelected()) {
                style |= Font.BOLD;
            }
            if (styleItems[1].isSelected()) {
                style |= Font.ITALIC;
            }

            String name = displayLabel.getFont().getName();
            displayLabel.setFont(new Font(name, style, 72));
        }
    }
}
```

- Per prima cosa, si crea una barra dei menu (`JMenuBar`), e la si aggiunge alla finestra con il metodo `setJMenuBar` di `JFrame`.
- Nella barra vengono inseriti due menu: "File" e "Format":
  - "File" contiene due `JMenuItem`, cioè bottoni: "About…", che mostra un messaggio contenente delle informazioni sull'applicazione, ed "Exit", che termina il programma mediante la chiamata `System.exit(0)`.[1]
  - "Format" contiene due sotto-menu (ottenuti aggiungendo degli altri `JMenu` all'interno del `JMenu`): "Color", che permette di scegliere il colore del testo presente nella finestra mediante una serie di radio button (`JRadioButtonMenu Item`), e "Font", nel quale si può scegliere il carattere tipografico, attraverso una combinazione di radio button e checkbox (`JCheckBoxMenuItem`).
- Il metodo `addSeparator` aggiunge un separatore (una riga orizzontale) tra due elementi di un menu.
- Il metodo `setMnemonic` imposta il tasto da premere, in combinazione con Alt, per attivare un menu (o, in generale, un componente di un menu). Ad esempio, `fileMenu.setMnemonic('F')` permette di aprire il menu "File" premendo Alt-f.

[1] Il parametro di `System.exit` è un codice usato per segnalare al sistema operativo la causa della terminazione del programma: un valore diverso da 0 indica che si è verificato un errore.