Capitolo 13

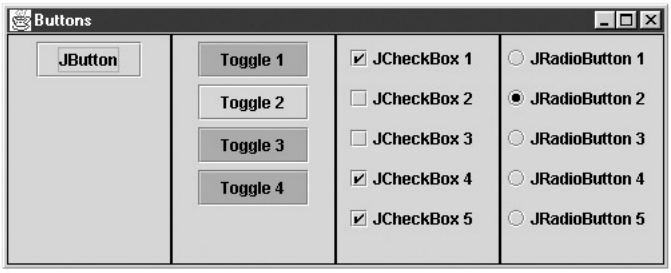
Bottoni e menu

ANDREA GINI

Pulsanti

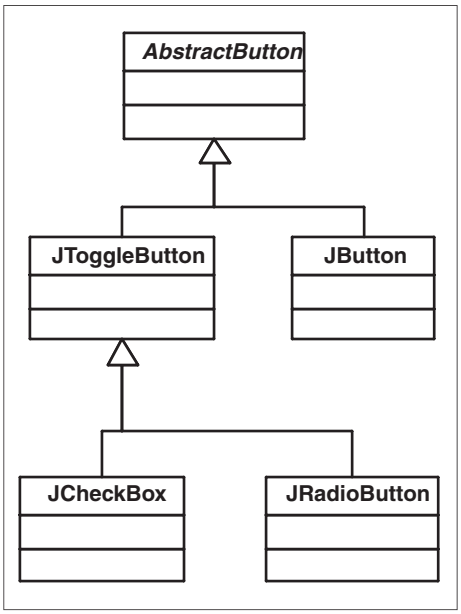
I pulsanti, grazie alla loro modalità di utilizzo estremamente intuitiva, sono sicuramente i controlli grafici più usati. Il package Swing offre quattro tipi di pulsanti, legati tra loro dalla gerarchia illustrata in figura 13.2.

Figura 13.1 – *Pulsanti disponibili in Java.*



JButton è l'implementazione del comune bottone push; JToggleButton è un pulsante on/off; JCheckBox è una casella di controllo, ossia un controllo grafico creato sul modello delle caselle di spunta dei questionari; JRadioButton serve a creare pulsanti di opzione, che permettono di scegliere una possibilità tra molte in modo mutuamente esclusivo.

Figura 13.2 – *Gerarchia dei principali pulsanti Java.*



AbstractButton: gestione dell'aspetto

La classe AbstractButton definisce l'interfaccia di programmazione comune a tutti i pulsanti. L'API di AbstractButton definisce un insieme di metodi per gestire l'aspetto del componente. In particolare, viene fornita la possibilità di associare a ogni controllo grafico un'etichetta di testo, un'icona o entrambi. È possibile impostare l'etichetta in formato HTML: basta aggiungere il prefisso <html> nel parametro di setText(String). Le seguenti righe di codice mostrano come creare un JButton con un'etichetta HTML:

```
JButton b = new JButton();
b.setText("<html><font size=-1><b><u>Esempio</u></b> di pulsante <b>HTML</b></font></html>"));
```

Figura 13.3 – *Su tutti i pulsanti Swing è possibile impostare un'etichetta HTML.*

I pulsanti Swing permettono di impostare un'icona diversa per ognuno degli stati in cui si possono trovare: normale, premuto, selezionato (valido per i controlli che mantengono lo stato come i CheckBox), disabilitato e rollover (lo stato in cui si trova il pulsante quando viene sorvolato dal puntatore del mouse).

```
void setIcon(Icon defaultIcon)
void setPressedIcon(Icon pressedIcon)
void setSelectedIcon(Icon selectedIcon)
void setDisabledIcon(Icon disabledIcon)
void setRolloverIcon(Icon rolloverIcon)
void setDisabledSelectedIcon(Icon disabledSelectedIcon)
void setRolloverSelectedIcon(Icon rolloverSelectedIcon)
```

Come icona si può utilizzare un oggetto di tipo ImageIcon. Per farlo, si ricorre al costruttore ImageIcon(String filename) che crea un'icona a partire da un'immagine di tipo GIF o JPEG, il cui percorso viene specificato nella stringa del parametro. Se non viene specificato diversamente, le immagini per gli stati premuto, selezionato e disabilitato vengono create in modo automatico a partire da quella di default. Le seguenti righe creano un pulsante decorato con l'immagine img.gif (può andare bene una qualsiasi immagine GIF o JPEG). Naturalmente, il file deve essere presente nella directory di lavoro, altrimenti verrà creato un pulsante vuoto:

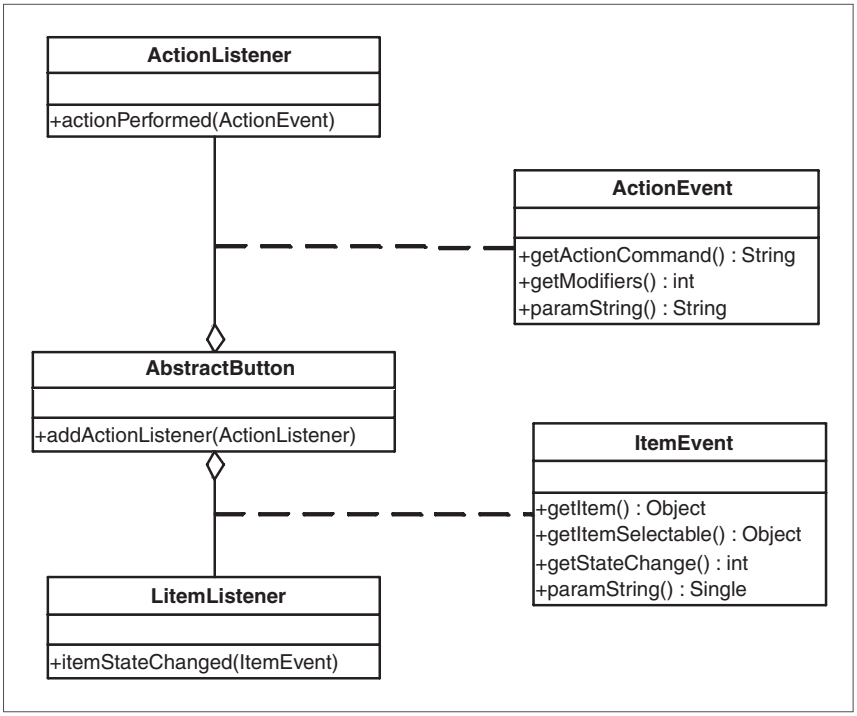
```
JButton b = new JButton();
b.setIcon(new ImageIcon("img.gif"));
```

Altri metodi importanti sono void setText(String text), che permette di impostare la scritta sopra il pulsante, setEnabled(boolean b) che permette di abilitare o disabilitare il componente, e setRolloverEnabled(boolean b) che attiva o disattiva l'effetto rollover.

Eventi dei pulsanti

I controlli derivati da AbstractButton prevedono due tipi di ascoltatore: ActionListener e ItemListener. Il primo ascolta l'evento realtivo alla pressione del pulsante. Il secondo ascolta invece i cambiamenti tra gli stati selezionato e non selezionato, ed è utile nei pulsanti di tipo JToggleButton. Nei paragrafi seguenti verranno illustrati esempi di uso di entrambi gli ascoltatori.

Figura 13.4 – *Pulsanti Java, ascoltatori ed eventi.*



JButton

JButton, il comune pulsante push, è la più importante sottoclasse di AbstractButton. I seguenti costruttori permettono di creare pulsanti e di definirne le principali proprietà visuali, ossia l'etichetta di testo e l'icona:

```
JButton(String text)
JButton(Icon icon)
JButton(String text, Icon icon)
```

Ogni top level container può segnalare un pulsante come DefaultButton. Esso verrà evidenziato in modo particolare e richiamato con la semplice pressione del tasto Invio. Le righe seguenti creano un pulsante, un JFrame e impostano il pulsante come DefaultButton.

```
JFrame f = new JFrame();
JButton b = new JButton("DefaultButton");
f.getContentPane().add(b);ty
f.getRootPane().setDefaultButton(b);
```

L'ascoltatore più utile per un JButton è ActionListener, che riceve eventi di tipo ActionEvent quando il pulsante viene premuto. Si vedrà ora un programma di esempio che illustra l'uso di due JButton, uno dei quali viene registrato come DefaultButton, e dei relativi ascoltatori.

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

public class JButtonExample extends JFrame {

  private JButton dialogButton;
  private JButton closeButton;

  public JButtonExample() {
    super("JButtonExample");
    closeButton = new JButton("Close");
    dialogButton = new JButton("Open Frame");
    dialogButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(closeButton, "Chiudi questa finestra per proseguire");
      }
    });
    closeButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          System.exit(0);
        }
        catch (Exception ex) {
        }
      }
    });
    getContentPane().setLayout(new FlowLayout(FlowLayout.CENTER));
    getRootPane().setDefaultButton(closeButton);
    getContentPane().add(closeButton);
    getContentPane().add(dialogButton);
    pack();
  }

  public static void main(String argv[]) {
    JButtonExample x = new JButtonExample();
    x.setVisible(true);
  }
}
```

Figura 13.5 – *Il programma JButtonExample.*

JToggleButton

JToggleButton è un pulsante che può trovarsi in due stati: premuto e rilasciato. Cliccando con il mouse si provoca il passaggio tra uno stato e l'altro. I costruttori permettono di creare JToggleButton e di impostarne le proprietà:

```
JToggleButton(String text, Icon icon, boolean selected)
JToggleButton(String text, boolean selected)
JToggleButton(Icon icon, boolean selected)
```

Il parametro selected permette di impostare lo stato iniziale del pulsante. Un JToggleButton, quando viene premuto, genera un ActionEvent e un ItemEvent. L'evento più significativo per questo tipo di pulsanti è il secondo, che segnala il cambiamento di stato, ossia il passaggio da premuto a rilasciato e viceversa. La classe ItemEvent dispone di due metodi importanti: Object getItem() e int getStateChange(). Il primo restituisce un reference al pulsante che ha generato l'evento (è necessario ricorrere al casting); il secondo sostituisce un intero che può assumere i valori ItemEvent.SELECTED oppure ItemEvent.DESELECTED, a seconda del valore assunto dal componente. È comunque possibile utilizzare ascoltatori di tipo ActionListener in modo simile all'esempio precedente. L'esempio che segue crea una finestra con un JToggleButton, che permette di aprire e di chiudere una finestra di dialogo non modale:

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

public class JToggleButtonExample extends JFrame {

  private JDialog dialog;
  private JToggleButton jDialogButton;

  public JToggleButtonExample() {
```

```
    super("JToggleButtonExample");
    setBounds(10, 35, 250, 70);
    FlowLayout fl = new FlowLayout(FlowLayout.CENTER);
    getContentPane().setLayout(fl);
    dialog = createDialog();

    jDialogButton = new JToggleButton("Open / Close Frame", false);
    jDialogButton.addItemListener(new JDialogButtonItemListener());
    getContentPane().add(jDialogButton);
    setVisible(true);
  }
  public JDialog createDialog() {
    JDialog d = new JDialog(this, "JDialog", false);
    d.setBounds(250, 20, 300, 100);
    d.getContentPane().setLayout(new BorderLayout());
    d.getContentPane().add(BorderLayout.CENTER, new JLabel("Finestra Aperta", JLabel.CENTER));
    d.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    return d;
  }
  // Ascoltatore di JDialogButton
  class JDialogButtonItemListener implements ItemListener {
    public void itemStateChanged(ItemEvent e) {
      int status = e.getStateChange();
      if ( status == ItemEvent.SELECTED )
        dialog.setVisible(true);
      else
        dialog.setVisible(false);
    }
  }

  public static void main(String argv[]) {
    JToggleButtonExample b = new JToggleButtonExample();
  }
}
```

Figura 13.6 – *Il programma JToggleButtonExample.*

Il codice dell'ascoltatore JDialogButtonItemListener è un po' più complesso di quello degli ActionListener dell'esempio precedente. Questo tipo di ascoltatore deve normalmente prevedere una verifica dello stato in cui si trova il pulsante, al fine di produrre la reazione appropriata. La verifica dello stato viene effettuata interrogando l'oggetto ItemEvent con il metodo getStateChange().

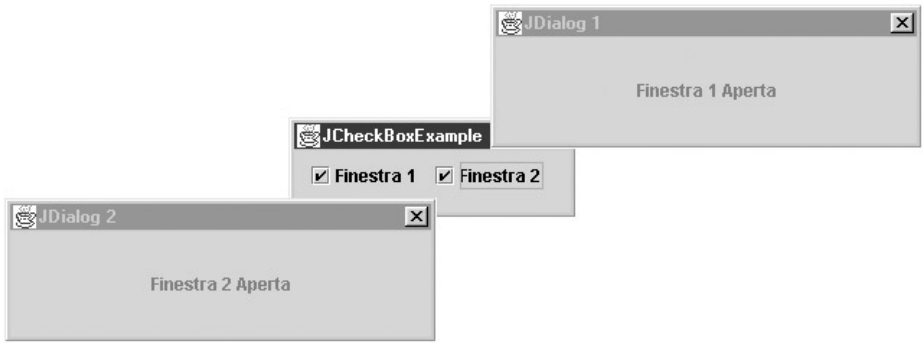
JCheckBox

JCheckBox è una sottoclasse di JToggleButton che crea caselle di controllo, con un aspetto simile a quello delle caselle di spunta dei questionari. Il suo funzionamento è analogo a quello della superclasse, ma di fatto tende a essere utilizzato in contesti in cui si offre all'utente la possibilità di scegliere una o più opzioni tra un insieme, come avviene per esempio nei pannelli di controllo. I costruttori disponibili sono gli stessi di JToggleButton e così pure la gestione degli eventi, quindi non sarà necessario ripetere quanto è stato già detto. Un esempio mostrerà un uso tipico di questo componente:

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

public class JCheckBoxExample extends JFrame {

  private JDialog dialog1;
  private JDialog dialog2;
  private JCheckBox checkBox1;
  private JCheckBox checkBox2;

  public JCheckBoxExample() {
    // Imposta le proprietà del top level container
    super("JCheckBoxExample");
    setBounds(10, 35, 200, 70);
    getContentPane().setLayout(new FlowLayout(FlowLayout.CENTER));

    // Crea due finestre di dialogo non modali,
    // inizialmente invisibili
    dialog1 = new JDialog(this, "JDialog 1", false);
    dialog1.setBounds(250, 20, 300, 100);
    dialog1.getContentPane().setLayout(new BorderLayout());
    dialog1.getContentPane().add(BorderLayout.CENTER new JLabel("Finestra 1 Aperta", JLabel.CENTER));
    dialog1.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    dialog2 = new JDialog(this, "JDialog 2", false);
    dialog2.setBounds(250, 150, 300, 100);
    dialog2.getContentPane().setLayout(new BorderLayout());
    dialog2.getContentPane().add(BorderLayout.CENTER, new JLabel("Finestra 2 Aperta", JLabel.CENTER));
    dialog2.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);

    // Crea i checkBox e li registra presso il loro ascoltatore
```

```
        ItemListener listener = new JCheckBoxItemListener();
        checkBox1 = new JCheckBox("Finestra 1");
        checkBox1.addItemListener(listener);
        checkBox2 = new JCheckBox("Finestra 2");
        checkBox2.addItemListener(listener);

        // Aggiunge i checkBox al top level container
        getContentPane().add(checkBox1);
        getContentPane().add(checkBox2);
        setVisible(true);
    }
    // ascoltatore JCheckBox
    class JCheckBoxItemListener implements ItemListener {
        public void itemStateChanged(ItemEvent e) {
            Object target = e.getItem();
            int status = e.getStateChange();

            if(target.equals(checkBox1) && status == ItemEvent.SELECTED)
                        dialog1.setVisible(true);
            else if(target.equals(checkBox1) && status == ItemEvent.DESELECTED)
                        dialog1.setVisible(false);
            else if(target.equals(checkBox2) && status == ItemEvent.SELECTED)
                        dialog2.setVisible(true);
            else if(target.equals(checkBox2) && status == ItemEvent.DESELECTED)
                        dialog2.setVisible(false);
        }
    }
    public static void main(String argv[]) {
        JCheckBoxExample b = new JCheckBoxExample();
    }
}
```

Figura 13.7 – *Il programma JCheckBoxExample.*



L'ascoltatore JCheckBoxItemListener presenta un grado di complessità maggiore del precedente. Esso infatti ascolta entrambi i controlli, e a ogni chiamata verifica quale dei due abbia generato l'evento, chiamando il metodo getItem() di ItemEvent, e quale stato esso abbia assunto, predisponendo la reazione opportuna.

JRadioButton

JRadioButton è una sottoclasse di JToggleButton, dotata dei medesimi costruttori. Questo tipo di controllo, chiamato pusante di opzione, viene usato tipicamente per fornire all'utente la possibilità di operare una scelta tra un insieme di possibilità, in contesti nei quali un'opzione esclude l'altra.

Per implementare questo comportamento di mutua esclusione, è necessario registrare i JRadioButton che costituiscono l'insieme presso un'istanza della classe ButtonGroup, come viene mostrato nelle righe seguenti:

```
ButtonGroup group = new ButtonGroup();
group.add(radioButton1);
group.add(radioButton2);
group.add(radioButton3);
```

Ogni volta che l'utente attiva uno dei pulsanti registrati presso il ButtonGroup, gli altri vengono automaticamente messi a riposo. Questo comportamento ha una conseguenza importante nella gestione degli eventi. Infatti, un gruppo di JRadioButton registrati presso un ButtonGroup genera *due* ItemEvent consecutivi per ogni clic del mouse: uno per la casella che viene selezionata e uno per quella deselezionata. Di norma, si è interessati unicamente al fatto che un particolare JRadioButton sia stato premuto, poiché la politica di mutua esclusione rende superflua la verifica dello stato. In questi casi, è consigliabile utilizzare un ActionListener come nell'esempio seguente, nel quale un gruppo di JRadioButton permette di modificare la scritta su un'etichetta di testo:

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

public class JRadiobuttonExample extends JFrame {

   private JRadioButton radioButton1;
   private JRadioButton radioButton2;
   private JRadioButton radioButton3;
   private JLabel label;

   public JRadiobuttonExample() {
      // Imposta le proprietà del top level container
      super("JRadiobuttonExample");
```

```
        setBounds(10, 35, 150, 150);
        getContentPane().setLayout(new FlowLayout(FlowLayout.CENTER));

        // Crea i radiobutton e la label
        radioButton1 = new JRadioButton("RadioButton 1");
        radioButton2 = new JRadioButton("RadioButton 2");
        radioButton3 = new JRadioButton("RadioButton 3");
        label = new JLabel();

        // Crea l'ascoltatore e registra i JRadioButton
        ActionListener listener = new JRadioButtonListener();
        radioButton1.addActionListener(listener);
        radioButton2.addActionListener(listener);
        radioButton3.addActionListener(listener);

        // Crea il ButtonGroup e registra i RadioButton
        ButtonGroup group = new ButtonGroup();
        group.add(radioButton1);
        group.add(radioButton2);
        group.add(radioButton3);

        // Aggiunge i componenti al top level container
        getContentPane().add(radioButton1);
        getContentPane().add(radioButton2);
        getContentPane().add(radioButton3);
        getContentPane().add(label);

        radioButton1.doClick();
        setVisible(true);
    }
    // Ascoltatore JRadioButton
    class JRadioButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            String target = e.getActionCommand();
            label.setText(target);
        }
    }
    public static void main(String argv[]) {
        JRadiobuttonExample b = new JRadiobuttonExample();
    }
}
```

Figura 13.8 – *Il programma JRadioButtonExample.*

JToolBar

Nelle moderne interfacce grafiche, l'insieme dei controlli viene suddiviso tra due luoghi: la Menu Bar, di cui si parlerà più avanti, e la Tool Bar, di cui ci si occupa ora. JToolBar è un contenitore che permette di raggruppare un insieme di controlli grafici in una riga, che solitamente viene posizionata al di sotto della barra dei menu. Sebbene sia utilizzata soprattutto come contenitore di pulsanti provvisti di icona, è possibile inserire al suo interno qualsiasi tipo di componente, come campi di testo o elenchi di selezione a discesa.

Ricorrendo al drag & drop è possibile staccare una Tool Bar dalla sua posizione originale e renderla fluttuante: in questo caso, essa verrà visualizzata in una piccola finestra separata dal frame principale. Allo stesso modo, è possibile afferrare una barra degli strumenti con il mouse e trascinarla in una nuova posizione.

L'uso di JToolBar all'interno dei propri programmi non presenta particolari difficoltà. È sufficiente crearne un'istanza, aggiungervi i componenti nell'ordine da sinistra a destra e posizionarla all'interno del contenitore principale:

```
JToolBar toolBar = new JToolBar();
JButton b = new JButton(new ImageIcon("img.gif"));
toolBar.add(b);
....
JFrame f = new JFrame();
f.getContentPane().setLayout(new BorderLayout());
f.getContentPane().add(BorderLayout.NORTH, toolBar);
```

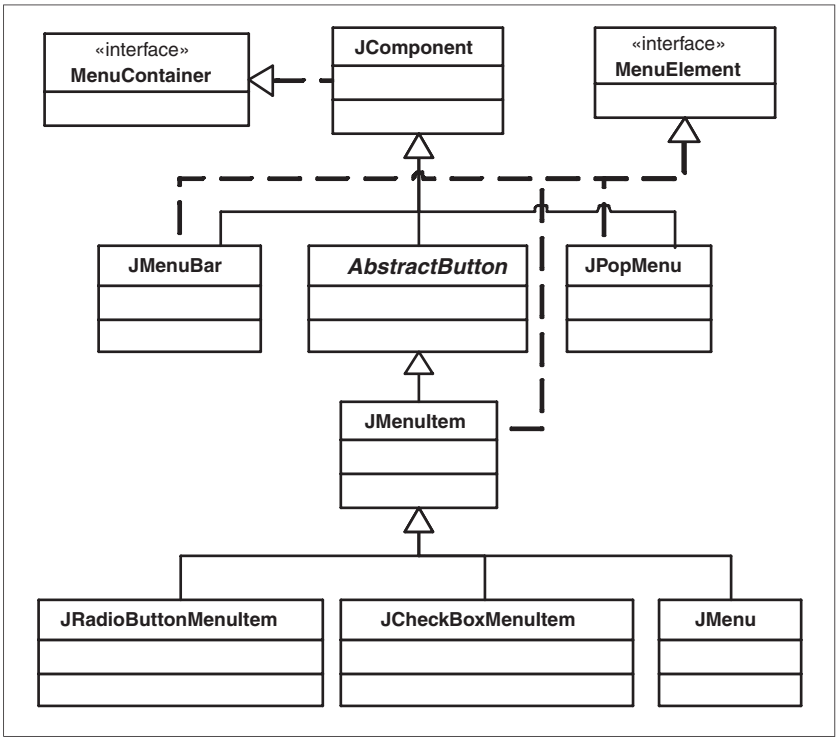
Tra i metodi di JToolBar, vale la pena segnalare add(Component c), che permette di aggiungere un componente alla barra, e addSeparator(), che aggiunge un segnale di separazione.

I menu

I menu sono controlli che permettono di accedere a un grande numero di opzioni in uno spazio ridotto, organizzato gerarchicamente. Ogni programma grafico dispone di una Menu Bar

organizzata per gruppi di funzioni: accesso al disco, operazioni sulla clipboard, opzioni e così via. Ogni menu può contenere sia elementi terminali (MenuItem) sia ulteriori menu nidificati.

Figura 13.9 – *Gerarchia dei componenti di tipo menu.*



In Swing anche i menu si assemblano in modo gerarchico, costruendo un oggetto per ogni elemento e aggiungendolo al proprio contenitore. La gerarchia delle classi in figura 13.8 mostra che ogni sottoclasse di JComponent è predisposta a contenere menu, capacità che viene garantita dall'interfaccia MenuContainer. Le classi JMenu e JPopupMenu sono contenitori appositamente realizzati per questo scopo. La classe JMenuItem implementa l'elemento di tipo più semplice: essendo sottoclasse di AbstractButton, ne eredita l'interfaccia di programmazione e il comportamento (vale a dire che tutti i metodi visti nel paragrafo AbstractButton possono essere utilizzati anche su questi componenti). JRadioButtonMenuItem e JCheckBoxMenuItem sono analoghi ai pulsanti JRadioButton e JCheckBox; oltre alla parentela diretta con JMenuItem, essi hanno in comune con JMenu e JPopupMenu l'interfaccia MenuElement, che accomuna tutti i componenti che possono comparire all'interno di un menu. Il metodo add(JMenu m), comune a tutte le classi, permette di innestare qualsiasi menu all'interno di qualunque altro. I seguenti costruttori permettono di

creare JMenuItem, JRadioButtonMenuItem e JCheckBoxMenuItem in maniera simile a come si può fare con i JButton. I parametri permettono di specificare l'etichetta, l'icona e lo stato:

```
JMenuItem(String text)
JMenuItem(String text, Icon icon)
JCheckBoxMenuItem(String text, Icon icon, boolean b)
JCheckBoxMenuItem(String text, boolean b)
JRadioButtonMenuItem(String text, boolean selected)
JRadioButtonMenuItem(String text, Icon icon, boolean selected)
```

Sebbene sia possibile posizionare una JMenuBar ovunque all'interno di un'interfaccia grafica, i top level container JFrame, JApplet e JDialog riservano a questo scopo una posizione esclusiva, situata appena sotto la barra del titolo. È possibile aggiungere un JMenu a un JFrame, a un JApplet o a un JDialog mediante il metodo setJMenuBar(JMenuBar), come si vede nelle righe di esempio:

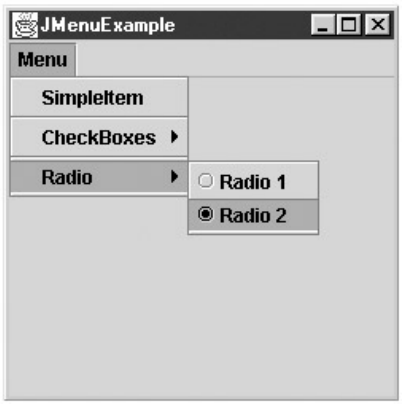
```
JMenuBar menubar = new JMenuBar();
....
JFrame f = new JFrame("A Frame");
f.setJMenuBar(menuBar)
```

La gestione degli eventi nei menu è del tutto simile a quella dei pulsanti: ogni volta che si seleziona un JMenuItem, esso lancia un ActionEvent ai suoi ascoltatori. Normalmente si usa ActionListener per i JMenuItem e ItemListener per i JCheckboxMenuItem, mentre per JRadioButtonMenuItem è possibile usare sia l'uno sia l'altro. Il seguente esempio illustra la costruzione di un menu ricorrendo a elementi di ogni tipo:

```
import javax.swing.*;

public class JMenuExample extends JFrame {
   public JMenuExample() {
      // Imposta le proprietà del top level container
      super("JMenuExample");
      setBounds(10, 35, 250, 250);
      // Crea menu, sottomenu e menuitems
      JMenuBar menubar = new JMenuBar();
      JMenu menu = new JMenu("Menu");
      JMenuItem simpleItem = new JMenuItem("SimpleItem");
      JMenu checkSubMenu = new JMenu("CheckBoxes");
      JCheckBoxMenuItem check1 = new JCheckBoxMenuItem("Check 1");
      JCheckBoxMenuItem check2 = new JCheckBoxMenuItem("Check 1");
      JMenu radioSubMenu = new JMenu("Radio");
      JRadioButtonMenuItem radio1 = new JRadioButtonMenuItem("Radio 1");
      JRadioButtonMenuItem radio2 = new JRadioButtonMenuItem("Radio 2");
      ButtonGroup group = new ButtonGroup();
```

```
        group.add(radio1);
        group.add(radio2);
        // Compone i menu
        checkSubMenu.add(check1);
        checkSubMenu.add(check2);
        radioSubMenu.add(radio1);
        radioSubMenu.add(radio2);
        menu.add(simpleItem);
        menu.addSeparator();// (new JSeparator());
        menu.add(checkSubMenu);
        menu.addSeparator();//.add(new JSeparator());
        menu.add(radioSubMenu);
        menubar.add(menu);
        // Aggiunge la barra dei menu al JFrame
        setJMenuBar(menubar);
        setVisible(true);
    }
    public static void main(String argv[]) {
        JMenuExample m = new JMenuExample();
    }
}
```

Figura 13.10 – *Il programma JMenuExample.*



JPopupMenu

I JPopupMenu implementano i menu contestuali presenti in quasi tutti i moderni sistemi a finestre. La costruzione di JPopupMenu è del tutto simile a quella di Jmenu, mentre diversa è la modalità di visualizzazione. Il metodo:

```
public void show(Component invoker, int x, int y)
```

visualizza il menu al di sopra del componente specificato dal parametro invoker, alle coordinate x e y (relative a invoker). Per associare un JPopupMenu alla pressione del pulsante destro del mouse su un oggetto grafico, è necessario registrare il componente interessato presso un MouseListener incaricato di chiamare il metodo show() al momento opportuno. Dal momento che alcuni sistemi a finestre mostrano il menu contestuale alla pressione del pulsante destro (evento mousePressed), mentre altri lo mostrano al momento del rilascio (evento mouseReleased), è bene ascoltare entrambi gli eventi, controllando la condizione isPopupTrigger() sull'evento MouseEvent. Esso infatti restituisce true solamente se l'evento corrente è quello che provoca il richiamo del menu contestuale nella piattaforma ospite:

```
class PopupListener extends MouseAdapter {
   // pulsante destro premuto (stile Motif)
   public void mousePressed(MouseEvent e) {
      if (e.isPopupTrigger()) {
            popup.show(e.getComponent(), e.getX(), e.getY());
      }
   }
   // pulsante destro premuto e rilasciato (stile Windows)
   public void mouseReleased(MouseEvent e) {
      if (e.isPopupTrigger()) {
         popup.show(e.getComponent(), e.getX(), e.getY());
      }
   }
}
```

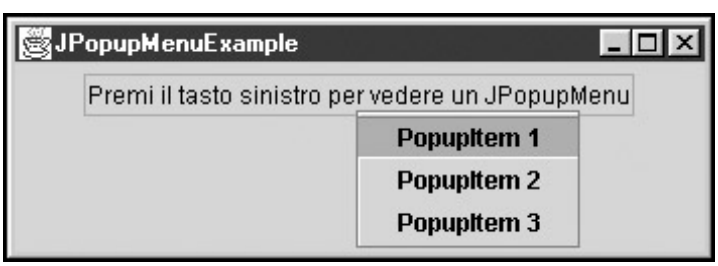
Questo accorgimento permette di creare programmi che rispecchiano il comportamento della piattaforma ospite, senza ambiguità che potrebbero disorientare l'utente.

Il seguente esempio crea un JTextField, cui aggiunge un MouseListener che si occupa di visualizzare un JPopupMenu alla pressione del pulsante destro:

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

public class JPopupMenuExample extends JFrame {
   private JPopupMenu popup;

   public JPopupMenuExample() {
      super("JPopupMenuExample");
      setBounds(10, 35, 350, 120);

      JTextField textField = new JTextField("Premi il pulsante sinistro per vedere un JPopupMenu");
      textField.setEditable(false);
```

```
        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(textField);

        popup = new JPopupMenu();
        JMenuItem popupItem1 = new JMenuItem("PopupItem 1");
        JMenuItem popupItem2 = new JMenuItem("PopupItem 2");
        JMenuItem popupItem3 = new JMenuItem("PopupItem 3");
        popup.add(popupItem1);
        popup.add(popupItem2);
        popup.add(popupItem3);

        // Aggiunge un MouseListener al componente
        // che deve mostrare il menu
        MouseListener popupListener = new PopupListener();
        textField.addMouseListener(popupListener);
        setVisible(true);
    }
    class PopupListener extends MouseAdapter {
        public void mousePressed(MouseEvent e) {
            if (e.isPopupTrigger()) {
                popup.show(e.getComponent(), e.getX(), e.getY());
            }
        }
        public void mouseReleased(MouseEvent e) {
            if (e.isPopupTrigger()) {
                popup.show(e.getComponent(), e.getX(), e.getY());
            }
        }
    }
    public static void main(String[] args) {
        JPopupMenuExample window = new JPopupMenuExample();
    }
}
```

Figura 13.11 – *Il programma JPopupMenuExample.*



Gestire gli eventi con le action

La maggior parte dei programmi grafici permette di accedere a una funzionalità in diverse maniere. I word processor, per esempio, consentono di effettuare un *cut* su clipboard in almeno tre modi distinti: dal menu Modifica, tramite il pulsante identificato dall'icona della forbice o tramite una voce del menu contestuale. Questa ridondanza è gradita all'utente, che ha la possibilità di utilizzare il programma secondo le proprie abitudini e il proprio grado di esperienza, ma può rivelarsi complicata da implementare per il programmatore. In Swing è possibile risolvere questo genere di problemi ricorrendo alle Action: oggetti che permettono di associare un particolare evento a un gruppo di controlli grafici, fornendo nel contempo la possibilità di gestire in modo centralizzato gli attributi e lo stato.

Descrizione dell'API

L'interfaccia Action, sottoclasse di ActionListener, eredita il metodo actionPerformed(ActionEvent e), con il quale si implementa la normale gestione degli eventi. Il metodo setEnabled(boolean b) permette di abilitare una Action; la chiamata a questo metodo provoca automaticamente l'aggiornamento dello stato di tutti i controlli grafici a esso associati. La coppia di metodi:

```
Object getValue(String key)
void putValue(String key, Object value)
```

permette di leggere o impostare coppie chiave-valore (in modo simile a quanto avviene nelle Hashtable) in cui la chiave è una stringa che descrive un attributo e il valore è l'attributo stesso. Tra le possibili chiavi si possono segnalare le seguenti:

```
Action.NAME
Action.SHORT_DESCRIPTION
Action.SMALL_ICON
```

Esse permettono di specificare, rispettivamente, il nome dell'azione (che verrà riportato sul pulsante), il testo da usare nei ToolTip e l'icona da esporre sui controlli abbinati alla Action. Per esempio, se si desidera impostare l'icona relativa a una Action, occorre utilizzare l'istruzione putValue in questo modo:

```
action.putValue(Action.SMALL_ICON, new ImageIcon("img.gif"))
```

La classe AbstractAction, implementazione dell'interfaccia Action, fornisce costruttori che permettono di impostare le proprietà in modo più intuitivo:

```
AbstractAction(String name)
AbstractAction(String name, Icon icon)
```

Uso delle Action

È possibile creare un oggetto Action estendendo la classe AbstractAction e fornendo il codice del metodo actionPerformed(ActionEvent e), in modo simile a quanto si farebbe per un oggetto di tipo ActionListener:

```
class MyAction extends AbstractAction {
  private Icon myIcon = new ImageIcon("img.gif");
  public MyAction() {
    super("My Action", myIcon);
  }
  public void actionPerformed(ActionEvent e) {
    // qui va il codice dell'ascoltatore
  }
}
```

È possibile definire una Action come classe anonima:

```
Action myAction = new AbstractAction("My Action", new ImageIcon("img.gif")) {
  public void actionPerformed(ActionEvent e) {
    // qui va il codice dell'ascoltatore
  }
});
```

Per abbinare una Action ai corrispondenti controlli grafici è sufficiente utilizzare il metodo add(Action a), presente in JMenu, JToolBar e JPopupMenu, come si vede nelle seguenti righe:

```
Action myAction = new MyAction();
JToolBar toolBar = new JToolBar();
JMenuBar menuBar = new JMenuBar();
JPopupMenu popup = new JPopupMenu();
// aggiunge un pulsante alla Tool Bar
toolBar.add(myAction);
// aggiunge un MenuItem alla Menu Bar
menuBar.add(myAction);
// aggiunge un MenuItem al Popup Menu
popup.add(myAction);
```

Dal momento che il metodo add(Action) restituisce il componente che viene creato, è possibile cambiarne l'aspetto anche dopo la creazione. Se si vuole aggiungere un MenuItem al menu, ma si desidera che esso sia rappresentato soltanto da una stringa di testo, senza icona, è possibile agire nel modo seguende:

```
JMenuItem mi = menuBar.add(myAction);
mi.setIcon(null);
```

Se durante l'esecuzione del programma si vogliono disabilitare i controlli abbinati a MyAction, è possibile farlo ricorrendo all'unica istruzione:

```
myAction.setEnabled(false);
```

che provvederà a disabilitare tutti i controlli legati a MyAction.