Capitolo 14

Controlli per inserimento dati

ANDREA GINI

Tipologie di controlli

JTextField

I JTextField (campi di testo) sono oggetti grafici che permettono di editare una singola riga di testo. Premendo il tasto Invio viene generato un ActionEvent, per segnalare agli ascoltatori che il testo è stato immesso. È possibile creare un JTextField mediante i seguenti costruttori:

```
JTextField()
JTextField(String text)
```

I seguenti metodi, invece, permettono di impostare o di leggere le principali proprietà dell'oggetto:

```
setText(String text)
setColumns(int columns)
setFont(Font f)
```

L'ascoltatore di default per JTextField è ActionListener, che viene invocato alla pressione del tasto Invio. Gli oggetti ActionEvent generati da un JTextField permettono di accedere direttamente al testo contenuto nel componente tramite il metodo getActionCommand(). L'esempio seguente mostra come creare un JTextField e un ascoltatore che reagisca all'inserimento di testo:

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

public class JTextFieldExample extends JFrame {
  private JTextField textField;
  private JLabel label;

  public JTextFieldExample() {
    super("JTextField");
    setSize(200, 80);
    getContentPane().setLayout(new BorderLayout());

    textField = new JTextField();
    label = new JLabel();
    getContentPane().add(BorderLayout.NORTH, textField);
    textField.addActionListener(new EnterTextListener());
    getContentPane().add(BorderLayout.SOUTH, label);

    setVisible(true);
  }
  class EnterTextListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      label.setText("Testo inserito: " + textField.getText());
      textField.setText("");
    }
  }
  public static void main(String argv[]) {
    JTextFieldExample jtf = new JTextFieldExample();
  }
}
```

Figura 14.1 – *Il programma JTextFieldExample.*

JPasswordField

JPasswordField è una sottoclasse di JTextField specializzata nell'inserimento di password. Le principali differenze rispetto alla superclasse sono due: la prima è che in JPasswordField i caratteri digitati vengono visualizzati di default tramite asterischi (***); la seconda è che il testo in chiaro viene

restituito sotto forma di array di char e non come stringa. Il metodo setEchoChar(char c) permette di impostare qualsiasi carattere al posto dell'asterisco di default. Il metodo char[] getPassword(), invece, restituisce il contenuto del campo di testo in chiaro, sotto forma di array di char.

Figura 14.2 – *JPasswordField impedisce di vedere quello che l'utente sta digitando.*

JComboBox

I JComboBox offrono all'utente la possibilità di effettuare una scelta a partire da un elenco di elementi, anche molto lungo. A riposo il componente si presenta come un pulsante, con l'etichetta corrispondente al valore attualmente selezionato. Un clic del mouse provoca la comparsa di un menu provvisto di barra laterale di scorrimento, che mostra le opzioni disponibili. Se si imposta un JComboBox come editabile, esso si comporterà a riposo come un JTextField, permettendo all'utente di inserire valori non presenti nella lista.

È possibile creare un JComboBox usando i seguenti costruttori. Il secondo costruttore permette di inizializzare il componente con una lista di elementi di tipo String, Icon o JLabel.

```
JComboBox()
JComboBox(Object[] items)
```

Un gruppo di metodi permette di aggiungere, togliere o manipolare gli elementi dell'elenco, così come si fa con un Vector:

```
void addItem(Object anObject
void removeItem(Object anObject)
void removeItemAt(int anIndex)
void removeAllItems()
Object getItemAt(int index)
int getItemCount()
void insertItemAt(Object anObject, int index)
```

Per operare sull'elemento correntemente selezionato, sono disponibili tre metodi che permettono di ottenere l'elemento (che viene restituito sotto forma di Object) e di impostarlo mediante un indice numerico o l'oggetto stesso:

```
Object getSelectedItem()
void setSelectedIndex(int anIndex)
void setSelectedItem(Object anObject)
```

L'ultimo metodo interessante è quello che permette di rendere un JComboBox editabile:

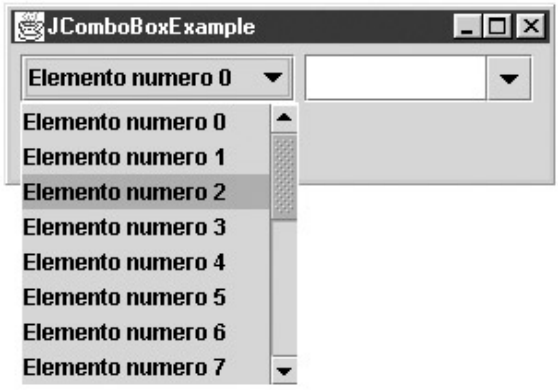
```
void setEditable(boolean b)
```

Nel seguente esempio vengono creati due JComboBox: uno editabile e l'altro no. All'interno del primo è possibile inserire gli elementi digitandoli direttamente nel componente e premendo Invio. Come ascoltatori vengono usati due ActionListener: EditListener si occupa di aggiungere alla lista i nuovi elementi, mentre SelectionListener viene invocato da entrambi i componenti al fine di aggiornare una JLabel con il valore dell'elemento selezionato.

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

public class JComboBoxExample extends JFrame {
   private JComboBox uneditableComboBox;
   private JLabel label;
   private JComboBox editableComboBox;
   private String[] items;

   public JComboBoxExample() {
     // Imposta le proprietà del top level container
     super("JComboBoxExample");
     setBounds(10, 35, 300, 100);
     getContentPane().setLayout(new FlowLayout(FlowLayout.LEFT));

     // Crea 20 elementi
     items = new String[20];
     for(int i = 0; i < 20; i++)
        items[i] = "Elemento numero " + String.valueOf(i);

     // Inizializza un ComboBox non editabile
     uneditableComboBox = new JComboBox(items);
     ActionListener selectionListener = new SelectionListener();
     uneditableComboBox.addActionListener(selectionListener);

     label = new JLabel();

     // Inizializza un JComboBox editabile
     editableComboBox = new JComboBox();
     editableComboBox.setEditable(true);
     editableComboBox.addActionListener(new EditListener());
     editableComboBox.addActionListener(selectionListener);

     getContentPane().add(uneditableComboBox);
     getContentPane().add(editableComboBox);
     getContentPane().add(label);
```

```
        setVisible(true);
    }

    class SelectionListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            JComboBox cb = (JComboBox)e.getSource();
            String selectedItem = (String)cb.getSelectedItem();
            label.setText("Selezionato: " + selectedItem);
        }
    }
    class EditListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            JComboBox cb = (JComboBox)e.getSource();
            String selectedItem = (String)cb.getSelectedItem();
            editableComboBox.addItem(selectedItem);
            editableComboBox.setSelectedItem("");
        }
    }
    public static void main(String argv[]) {
        JComboBoxExample b = new JComboBoxExample();
    }
}
```

Figura 14.3 – *JComboBox permette di scegliere un elemento da un elenco.*



JList

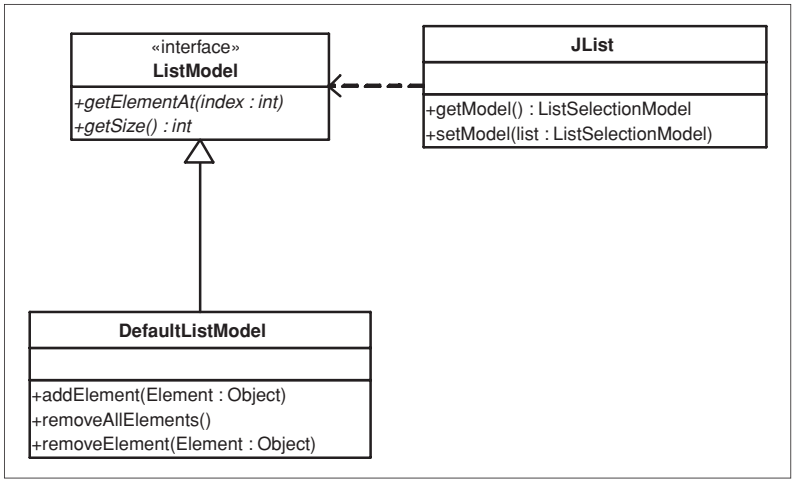
JList è un altro componente che permette di scegliere tra elementi che compongono un elenco. Diversamente da JComboBox consente di selezionare più di un elemento per volta, utilizzando il tasto Shift per selezionare elementi contigui o il tasto Ctrl per elementi separati.

Per utilizzare in modo completo JList è necessario comprendere la sua struttura interna: come

si può vedere dalla figura 14.4, JList mantiene gli elementi dell'elenco in un oggetto conforme all'interfaccia ListModel. Il package javax.swing contiene DefaultListModel, un'implementazione di JList di uso generico che permette di aggiungere o togliere a piacere elementi dall'elenco. Contrariamente a quanto sembra suggerire il nome, DefaultListModel *non* è il modello di default: se si crea una JList a partire da un vettore, esso utilizzerà un proprio ListModel non modificabile, al quale non potranno essere aggiunti o tolti elementi. Se si vuole creare una JList più flessibile, occorre procedere nel modo seguente:

```
listModel = new DefaultListModel();
listModel.addElement("Elemento 1");
listModel.addElement("Elemento 2");
....
list = new JList(listModel);
```

Figura 14.4 – *Diagramma delle classi di JList.*



Per visualizzare correttamente JList è indispensabile montarlo all'interno di un JScrollPane (un pannello dotato di barra di scorrimento) e aggiungere quest'ultimo al pannello principale. In caso contrario, non sarà possibile visualizzare tutti gli elementi presenti nell'elenco.

```
list = new JList(listModel);
JScrollPane scroll = new JScrollPane(list);
panel.add(scroll);
```

I costruttori permettono di creare un JList a partire da un oggetto di tipo ListModel o da un generico vettore di oggetti:

```
JList(ListModel dataModel)
JList(Object[] listData)
```

Come già accennato, il secondo costruttore produrrà una JList non modificabile. Gli elementi del vettore listData possono essere di tipo String, JLabel o Icon. Qualsiasi altro oggetto verrà visualizzato tramite la stringa restituita dal metodo toString().

Il seguente metodo permette di impostare la modalità di selezione:

```
void setSelectionMode(int selectionMode)
```

Il parametro selectionMode può assumere i seguenti valori: JList.SINGLE_SELECTION se si desidera che sia possibile selezionare un solo elemento per volta; JList.SINGLE_INTERVAL_SELECTION se si vuole permettere la selezione di un singolo intervallo per volta; JList.MULTIPLE_INTERVAL_SELECTION se non si vuole porre restrizioni al numero di elementi o intervalli selezionabili.

Un gruppo di metodi permette di operare sull'elemento selezionato, quando questo è unico:

```
Object getSelectedValue()
int getSelectedIndex()
void setSelectedIndex(int index)
boolean isSelectedIndex(int index)
void clearSelection()
```

Il metodo getSelectedIndex() restituisce l'indice del primo elemento selezionato, o il valore -1 se al momento non è selezionato alcun elemento. isSelectedIndex(int index) consente invece di sapere se un determinato elemento è selezionato o meno. Una quadrupla di metodi permette di operare su selezioni multiple:

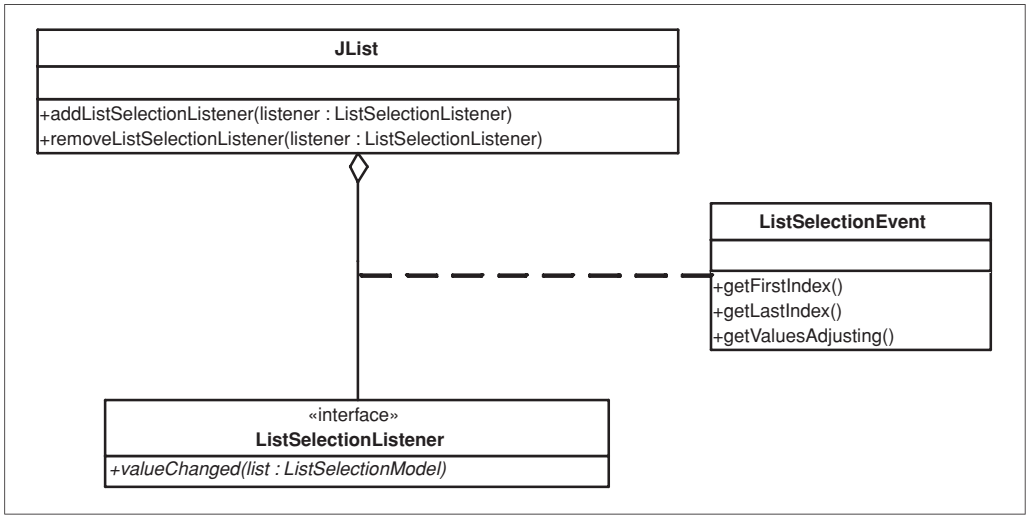
```
int[] getSelectedIndices()
Object[] getSelectedValues()
void setSelectedIndices(int[] indices)
void setSelectionInterval(int anchor, int lead)
```

Ogni volta che l'utente seleziona un elemento, viene notificato un ListSelectionEvent ai ListSelectionListener registrati. Per gestire l'elenco degli ascoltatori sono disponibili i caratteristici metodi:

```
void addListSelectionListener(ListSelectionListener listener)
void removeListSelectionListener(ListSelectionListener listener)
```

I metodi di ListSelectionEvent permettono di conoscere gli indici di inizio e di fine della selezione. Tramite il metodo boolean getValueIsAdjusting() è possibile sapere se l'utente sta ancora operando sulla selezione, o se ha terminato rilasciando il pulsante del mouse.

Figura 14.5 – *Modello di eventi di JList.*

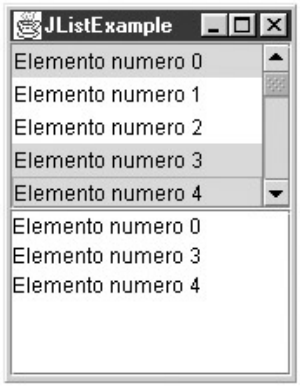


L'esempio seguente crea un JList con 20 elementi selezionabili a intervalli non contigui, e un'area di testo non modificabile che elenca gli elementi attualmente selezionati:

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

public class JListExample extends JFrame {

   private JList list;
   private JTextArea output;

   public JListExample() {
      super("JListExample");
      setSize(170, 220);
      getContentPane().setLayout(new GridLayout(0, 1));

      // Crea 20 elementi
      String[] items = new String[20];
      for(int i = 0; i < 20;i++)
         items[i] = "Elemento numero " + String.valueOf(i);

      // Inizializza una JList
      list = new JList(items);
      list. setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
```

```
        ListSelectionListener selectionListener = new SelectionListener();
        list.addListSelectionListener(selectionListener);

        // Crea la TextArea di output
        output = new JTextArea();
        output.setEditable(false);

        // assembla la GUI
        getContentPane().add(new JScrollPane(list));
        getContentPane().add(new JScrollPane(output));
        setVisible(true);
    }
    class SelectionListener implements ListSelectionListener {
        public void valueChanged(ListSelectionEvent e) {
            if(!e.getValueIsAdjusting()) {
                JList list = (JList)e.getSource();
                output.setText("");
                Object[] selectedItems = list.getSelectedValues();
                for(int i = 0; i < selectedItems.length; i++)
                        output.append(selectedItems[i].toString() + "\n");
                }
            }
        }
    public static void main(String argv[]) {

        JListExample b = new JListExample();
    }
}
```

Figura 14.6 – *Premendo il tasto Ctrl è possibile selezionare più di un elemento dall'elenco.*



JSlider

JSlider è un cursore a slitta, che permette di inserire in maniera continua valori numerici compresi tra un massimo e un minimo, eliminando di fatto la possibilità di inserire valori scorretti. Le proprietà più importanti di un JSlider sono il valore minimo, il valore massimo e l'orientamento, che può essere orizzontale o verticale. Il costruttore principale permette di specificare questi attributi al momento della creazione, e di impostare la posizione iniziale del cursore:

```
JSlider(int orientation, int min, int max, int value)
```

Le quattro proprietà fondamentali del componente possono essere impostate anche mediante i seguenti metodi:

```
void setOrientation(int orientation)
void setMinimum(int min)
void setMaximum(int max)
void setValue(int value)
```

Alcuni metodi permettono di impostare altre proprietà visuali del componente:

```
void setInverted(boolean b)
void setPaintTicks(boolean b)
void setPaintLabels(boolean b)
```

Il primo di questi metodi permette di scegliere se la scala debba essere disegnata da destra a sinistra (true) o da sinistra verso destra (false); il secondo se disegnare o meno il righello; il terzo, infine, se disegnare le etichette. Una coppia di metodi consente di impostare la spaziatura tra le tacche del righello:

```
void setMajorTickSpacing(int)
void setMinorTickSpacing(int)
```

Esiste infine la possibilità di personalizzare ulteriormente l'aspetto del componente, specificando etichette non regolari. Per farlo, bisogna chiamare il metodo setLabelTable(HashTable h), passando come parametro una HashTable che contenga coppie chiave-valore composte da un oggetto di tipo Int e un Component: ogni Component verrà disegnato in corrispondenza della tacca specificata dell'intero passato come chiave. Nelle righe seguenti viene mostrata la creazione di un JSlider con etichette testuali. Naturalmente, è possibile utilizzare al posto delle label qualsiasi tipo di componente, per esempio dei JLabel contenenti icone, o addirittura pulsanti programmati per riposizionare il cursore su valori preimpostati.

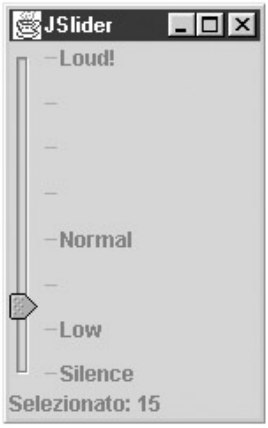
```
slider = new JSlider(JSlider.VERTICAL, 0, 70, 15);
slider.setMajorTickSpacing(10);
slider.setPaintTicks(true);
```

```
Hashtable labelTable = new Hashtable();
labelTable.put(new Integer(0), new JLabel("Silence"));
labelTable.put(new Integer(10), new JLabel("Low"));
labelTable.put(new Integer(30), new JLabel("Normal"));
labelTable.put(new Integer(70), new JLabel("Loud!"));
slider.setLabelTable(labelTable);
```

Il metodo Hashtable createStandardLabels(int incremen,int start) permette di creare tabelle con configurazioni standard, a partire dal valore specificato dal parametro start con la progressione data dal parametro increment:

```
s.setLabelTable(s.createStandardLabels(10,5));
```

Figura 14.7 – *Con il metodo setLabelTable() è possibile personalizzare il righello.*



Eventi JSlider

JSlider utilizza ChangeListener come ascoltatore e ChangeEvent come evento. I metodi per la gestione dell'elenco degli ascoltatori sono conformi alle consuete convenzioni di naming:

```
void addChangeListener(ChangeListener l)
void removeChangeListener(ChangeListener l)
```

Un ascoltatore di tipo ChangeListener deve implementare il metodo stateChanged(ChangeEvent e). Per conoscere lo stato di un JSlider bisogna interrogare il componente attraverso due metodi: getValue(), che restituisce il valore intero su cui il cursore è attualmente posizionato; getValueIsAdjusting(), che restituisce true se l'azione di modifica è tuttora in corso. Un ascoltatore come il seguente effettua un'azione solamente quando il cursore viene rilasciato, e scarta tutti gli eventi di aggiustamento:

```
class SliderChangeListener implements ChangeListener {
  public void stateChanged(ChangeEvent e) {
     JSlider slider = (JSlider)e.getSource();
     if(!slider.getValueIsAdjusting())
        label.setText("Selezionato: " + String.valueOf(slider.getValue()));
  }
}
```

Esempio d'uso

Le seguenti righe di codice chiariranno meglio l'uso di questo componente:

```
slider = new JSlider(JSlider.HORIZONTAL, 0, 60, 15);
slider.setMajorTickSpacing(10);
slider.setMinorTickSpacing(5);
slider.setPaintTicks(true);
slider.setPaintLabels(true);
```

Il costruttore crea un JSlider orizzontale, la cui scala varia tra 0 e 60 con il cursore posizionato sul 15. Seguono due metodi che impostano la spaziatura tra le tacche del righello: il primo imposta la spaziatura tra le tacche più grandi, il secondo tra quelle più piccole. Gli ultimi due metodi attivano il disegno del righello e della guida numerata, normalmente disattivati. Questa sequenza di metodi permette di creare un gran numero di cursori a slitta, come si può vedere nel prossimo esempio:

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

public class JSliderExample extends JFrame {

  private JSlider slider1;
  private JSlider slider2;
  private JSlider slider3;
  private JLabel label;

  public JSliderExample() {
    super("JSlider");
    setSize(220, 240);
    getContentPane().setLayout(new FlowLayout(FlowLayout.LEFT));
    ChangeListener listener = new SliderChangeListener();
    slider1 = new JSlider(JSlider.HORIZONTAL, 0, 60, 15);
    slider1.setMajorTickSpacing(10);
    slider1.setMinorTickSpacing(5);
    slider1.setPaintTicks(true);
    slider1.setPaintLabels(true);
    slider1.addChangeListener(listener);
    slider2 = new JSlider(JSlider.HORIZONTAL, 0, 60, 10);
    slider2.setMajorTickSpacing(15);
```

```
    slider2.setMinorTickSpacing(5);
    slider2.setPaintTicks(true);
    slider2.setPaintLabels(true);
    slider2.addChangeListener(listener);
    slider3 = new JSlider(JSlider.HORIZONTAL, 0, 60, 30);
    slider3.setMajorTickSpacing(5);
    slider3.setMinorTickSpacing(1);
    slider3.setPaintTicks(true);
    slider3.setPaintLabels(true);
    slider3.addChangeListener(listener);
    label = new JLabel("Selezionato: " + String.valueOf(slider1.getValue()));
    getContentPane().add(slider1);
    getContentPane().add(slider2);
    getContentPane().add(slider3);
    getContentPane().add(label);
    setVisible(true);
  }

  class SliderChangeListener implements ChangeListener {
    public void stateChanged(ChangeEvent e) {
      JSlider source = (JSlider)e.getSource();
      label.setText("Selezionato: " + String.valueOf(source.getValue()));
    }
  }

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

Figura 14.8 – *Il programma JSliderExample.*

JTextArea

Il package Swing dispone di un insieme completo di componenti di testo. Dopo JTextField, già esaminato all'inizio di questo capitolo, si prenderà ora in esame JTextArea, che crea un'area di testo: un oggetto grafico da utilizzare quando si intende lavorare su testi di lunghezza arbitraria privi di attributi di stile.

La costruzione di una JTextArea non presenta particolarità degne di nota: è importante invece fare attenzione, quando si assembla l'interfaccia grafica, a collocare il componente all'interno di un JScrollPane. In caso contrario, non sarà possibile navigare un testo più lungo di una schermata:

```
JTextArea ta = new JTextArea();
getContentPane().add(BorderLayout.CENTER, new JScrollPane(ta));
```

Una JTextArea può essere editabile o meno: tale comportamento può essere modificato tramite il metodo setEditable(boolean b). Un gruppo di metodi permette di impostare il font e i colori di primo piano e di sfondo:

```
void setFont(Font font)
void setForeground(Color fg)
void setBackground(Color bg)
```

Manipolazione del testo

È possibile editare il testo nel componente direttamente con la tastiera e il mouse, oppure da programma, ricorrendo ad alcuni metodi:

```
void setText(String t)
void append(String str)
void insert(String str, int pos)
void replaceRange(String str, int start, int end)
```

Il primo di questi permette di impostare il testo del componente, cancellando il contenuto precedente. Il secondo consente invece di aggiungere una stringa in coda al testo preesistente. I metodi insert() e replaceRange(), infine, servono a inserire del testo in una determinata posizione, o a rimpiazzare un frammento di testo, specificato dai parametri start e end, con una stringa. Questi metodi richiedono come parametri valori interi, che fanno riferimento alla posizione del cursore rispetto all'inizio del documento: se la text area in questione contenesse nella prima riga la frase "La vispa Teresa" e nella seconda "avea tra l'erbetta", sarebbe possibile dire che la parola "Teresa" è compresa tra gli offset 9 e 15, mentre "erbetta" si trova tra gli indici 27 e 34 (va contato anche il ritorno carrello).

JTextArea dispone anche di una coppia di metodi che permettono di comunicare con i dispositivi mediante gli stream, al fine di caricare o salvare il testo:

```
void read(Reader in, Object descriptor)
void write(Writer out)
```

Il metodo read() richiede come parametro un descriptor, ossia un oggetto che fornisce informazioni aggiuntive sul formato del testo: dal momento che JTextArea opera unicamente su testo privo di attributi, è possibile impostare tale parametro a null.

Per leggere il contenuto dell'area di testo sono disponibili due versioni del metodo getText(). Una è priva di argomenti e restituisce tutto il testo presente all'interno del componente; l'altra, invece, richiede di specificare la posizione di inizio e la lunghezza del frammento:

```
String getText()
String getText(int offs, int len)
```

Cursore e selezione

Un gruppo di metodi permette di conoscere la posizione attuale del cursore ed eventualmente di modificarla:

```
int getCaretPosition()
void setCaretPosition(int position)
void moveCaretPosition(int pos)
```

È anche possibile operare selezioni (mettere in evidenza una porzione di testo) in maniera programmatica, indicandone l'inizio e la fine:

```
int getSelectionStart()
int getSelectionEnd()

void select(int selectionStart, int selectionEnd)
void selectAll()
String getSelectedText()
void replaceSelection(String content)
```

Mediante le selezioni è anche possibile operare sulla clipboard di sistema, ricorrendo ai metodi cut(), copy() e paste().