Capitolo 16

Il disegno in Java

Andrea Gini

Il disegno in Java

Dopo aver concluso la panoramica sui principali componenti grafici Java, è giunto il momento di vedere come si utilizzano le primitive di disegno, grazie alle quali è possibile creare componenti ex novo.

Dal punto di vista del programmatore, un componente grafico non è altro che un oggetto al quale viene assegnata un'area di schermo su cui disegnare, e che è in grado di ascoltare gli eventi di mouse o tastiera. In questo capitolo verranno illustrate le primitive di disegno grafico e la gestione di eventi di mouse e tastiera, quindi si vedrà come combinare i concetti appresi per creare componenti interattivi.

JComponent e il meccanismo di disegno

Il primo passo per creare un componente nuovo è creare una sottoclasse di un componente esistente. I componenti Swing visti nei capitoli precedenti sono tutti sottoclassi di JComponent, un componente privo di forma che offre il supporto ai soli eventi di mouse e tastiera: esso si presta perciò a fare da base per la creazione di controlli grafici di qualsiasi tipo.

Il sistema grafico di Java funziona grazie a un meccanismo a call back: ogni sottoclasse di JComponent dispone di un metodo paintComponent(Graphics g), che viene chiamato direttamente dal sistema in tutte le circostanze in cui è necessario dipingere il componente sullo schermo. Durante il ciclo di vita di un'applicazione, la necessità di ridisegnare un determinato componente si verifica soprattutto in tre circostanze: in occasione della prima visualizzazione, nel corso di

un ridimensionamento o nel caso in cui l'area del componente sia stata "danneggiata", ossia sia stata coperta momentaneamente da un'altra finestra.

Il metodo paintComponent() non deve mai essere invocato direttamente: per motivi di prestazioni e di architettura, infatti, il refresh dello schermo viene avviato solo in alcune circostanze (non ha senso ridisegnare lo schermo più di 30 volte al secondo, dal momento che l'occhio non è in grado di discernere le differenze). Nei casi in cui l'utente desideri richiamare in modo esplicito il refresh, deve farlo tramite il metodo repaint(). Esso invia al sistema una richiesta di refresh che sarà gestita dal thread che si occupa del disegno, appena possibile. Esistono due versioni significative del metodo repaint:

```
public void repaint()
public void repaint(int x, int y, int width, int height)
```

Il primo richiede che l'intero componente venga ridisegnato; il secondo effettua il repaint unicamente nell'area specificata dai parametri (x e y specificano la posizione dell'angolo in alto a sinistra, mentre width e height sono rispettivamente la larghezza e l'altezza). Questo metodo permette di limitare l'area di disegno al frammento che ha subito modifiche dall'ultimo repaint, una strategia che può avere un impatto significativo sulle performance nel caso di programmi grafici che devono effettuare calcoli molto complessi.

L'oggetto Graphics

Come illustrato nel paragrafo precedente, per creare un componente nuovo è sufficiente definire una sottoclasse di JComponent e dichiarare al suo interno un metodo caratterizzato dalla firma:

```
public void paintComponent(Graphics g)
```

L'oggetto Graphics, che il metodo paintComponent() riceve come parametro, incapsula l'area in cui il è possibile disegnare. Graphics dispone di metodi di disegno. I più importanti sono quelli che permettono di disegnare linee, cerchi, stringhe e rettangoli:

```
boolean drawImage(Image img, int x, int y, ImageObserver observer)
void drawLine(int x1, int y1, int x2, int y2)
void drawOval(int x, int y, int width, int height)
 void drawRect(int x, int y, int width, int height)
void drawString(String str, int x, int y)
```

È possibile disegnare anche cerchi e rettangoli pieni:

```
void fillOval(int x, int y, int width, int height)
void fillRect(int x, int y, int width, int height)
```

Le coordinate usate come argomento su metodi di un oggetto Graphics sono considera-

te in relazione all'angolo in alto a sinistra del componente da disegnare. La coordinata x, dunque,cresce da sinistra a destra, e la y dall'alto in basso. Ogni JComponent possiede una quadrupla di metodi che permettono di leggere e impostare il colore di primo piano e il font del componente:

```
Color getForeground()
void setForeground(Color c)
Font getFont()
void setFont(Font f)
```

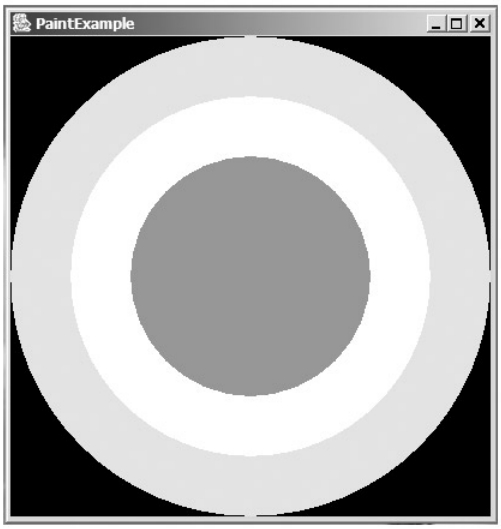
Al momento di invocare il metodo paintComponent su un determinato componente, il sistema di disegno imposta sull'oggetto Graphics il colore e il font del componente stesso. Durante il disegno, questi parametri possono naturalmente essere modificati, invocando sull'oggetto Graphics i seguenti metodi:

```
void Color getColor()
void setColor(Color c)
Font getFont()
void setFont(Font font)
```

Il prossimo esempio mostra come creare un semplice componente a partire da JComponent. all'interno del metodo paintComponent() sono presenti le direttive per disegna un quadrato nero di 400 x 400 pixel, con all'interno tre cerchi concentrici di colore verde, bianco e rosso:

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

public class PaintExample extends JComponent {
  public void paintComponent(Graphics g) {
    g.setColor(Color.BLACK);
    g.fillRect(0,0,400,400);
    g.setColor(Color.GREEN);
    g.fillOval(0,0,400,400);
    g.setColor(Color.WHITE);
    g.fillOval(50,50,300,300);
    g.setColor(Color.RED);
    g.fillOval(100,100,200,200);
  }

  public static void main(String argv[]) {
    JFrame f = new JFrame("PaintExample");
    f.setSize(410,430);
    f.getContentPane().add(new PaintExample());
    f.setVisible(true);
  }
}
```

Figura 16.1 – *Un primo esempio di componente grafico personalizzato.*



I metodi di disegno vengono eseguiti in sequenza. Quando una direttiva grafica viene eseguita, il suo disegno si sovrappone a quanto eventualmente già presente sullo schermo. Il metodo setColor() modifica il colore del pennello, pertanto esso ha effetto su tutte le istruzioni successive. Il componente PaintExample può essere inserito all'interno delle interfacce grafiche come qualsiasi altro componente. Per creare componenti di reale utilità, è necessario imparare a gestire le circostanze in cui il componente deve essere ridimensionato.

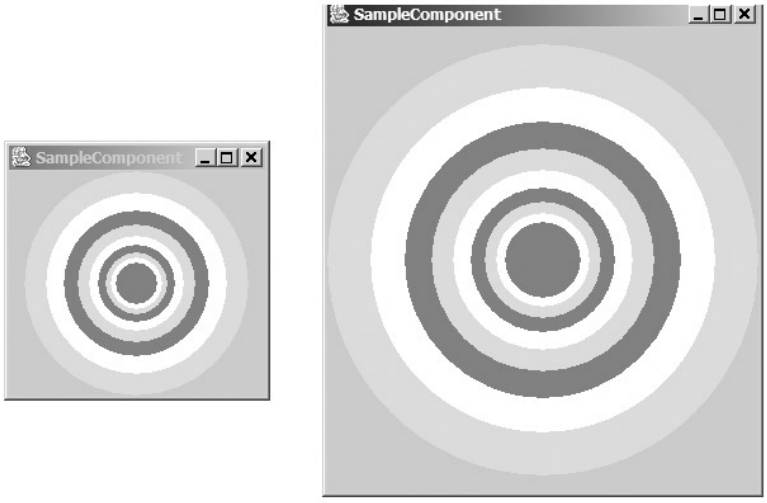
Adattare il disegno alle dimensioni del clip

Quando si disegna un componente è bene tener conto delle dimensioni del clip, in modo da adattare il disegno di conseguenza. Il metodo getSize(), presente in tutti i componenti grafici Java, restituisce un oggetto di tipo Dimension, che a sua volta possiede come attributi pubblici width e height, pari rispettivamente alla larghezza e all'altezza dell'area di disegno. Conoscendo queste misure, è possibile creare un disegno che sia proporzionale alla superficie da riempire. Nel prossimo esempio, viene creato un componente al cui interno vengono dipinti una serie di cerchi concentrici di colore rosso, bianco e verde. Ridimensionando il componente, il disegno viene ricalcolato in modo da adattarsi alle nuove dimensioni:

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

public class SampleComponent extends JComponent {
```

```
  private Color[] colors = {Color.RED,Color.GREEN,Color.WHITE};

  public void paintComponent(Graphics g) {
    // Calcola il diametro a partire dalle dimensioni del componente
    Dimension size = getSize();
    int d = Math.min(size.width, size.height);
    // Disegna una serie di cerchi concentrici
    for ( int i = 1; i <10 ; i++) {
      // sceglie a rotazione il colore
      g.setColor(colors[i%3]);
      // Calcola le coordinate del cerchio
      int x = (size.width - d) / 2;
      int y = (size.height - d) / 2;
      g.fillOval(x, y, d,d);
      // Riduce le dimensioni del diametro
      d = d - (d / 10 * 2);
    }
  }
  public static void main(String argv[]) {
    JFrame f = new JFrame();
    f.setSize(500, 300);
    f.getContentPane().add(new SampleComponent());
    f.setVisible(true);
  }
}
```

Figura 16.2 – *Un componente grafico capace di adattarsi al cambiamento delle dimensioni.*



Disegnare immagini

L'oggetto Graphics dispone anche di un metodo drawImage(), che permette di disegnare immagini GIF o JPEG presenti su disco. Il metodo drawImage() ha diversi formati. I due più importanti sono i seguenti:

```
boolean drawImage(Image img, int x, int y, ImageObserver observer)
boolean drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)
```

Il primo disegna l'immagine a partire dalle coordinate specificate con i parametri x e y, rispettandone le dimensioni originali. Il secondo permette anche di indicare un'area, alla quale l'immagine verrà adattata aumentandone o riducendone le dimensioni. Entrambi i metodi richiedono come parametro un ImageObserver, ossia un oggetto cui viene notificato in modo asincrono il progredire del caricamento dell'immagine. Tutti i componenti grafici Java implementano l'interfaccia ImageObserver, pertanto all'interno del metodo paint questo parametro può essere tranquillamente impostato a this. Per caricare un'immagine da disco, è possibile utilizzare una chiamata di questo tipo, fornendo come parametro l'indirizzo dell'immagine da caricare:

```
Image image = Toolkit.getDefaultToolkit().getImage(url);
```

Il prossimo esempio crea un componente che carica un'immagine e la ridimensiona in modo da coprire tutta l'area disponibile: se si cambiano le dimensioni del frame, l'immagine verrà a sua volta ridimensionata. Per lanciare il programma è necessario specificare da riga di comando l'indirizzo di un'immagine. Per esempio, la chiamata seguente:

```
java DrawImageExample c:\paperino.gif
```

avvia il programma caricando l'immagine paperino.gif dalla radice del disco C:

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

public class DrawImageExample extends JComponent  {
  private Image image;

  public DrawImageExample(String location) {
    image = Toolkit.getDefaultToolkit().getImage(location);
  }

  public void paintComponent(Graphics g) {
    Dimension size = getSize();
    g.drawImage(image, 0, 0, size.width,size.height,this);
  }

  public static void main(String argv[]) {
```

```
    if(argv.length != 1)
      throw new IllegalArgumentException("Use: java PaintImageExample <image>");
    JFrame f = new JFrame("DrawImageExample");
    f.setSize(600, 500);
    f.getContentPane().add(new DrawImageExample(argv[0]));
    f.setVisible(true);
  }
}
```

Figura 16.3 – *Una dimostrazione del metodo drawImage().*

Disegnare il testo

L'oggetto Graphics contiene anche metodi per disegnare stringhe di testo:

```
void drawString(String str, int x, int y)
```

Questo metodo richiede semplicemente una stringa e le coordinate del punto in cui disegnarla. Posizionare il testo all'interno di un componente non certo un compito facile, visto anche che con la maggior parte dei font le dimensioni dei caratteri cambiano a seconda del carattere da visualizzare

(una "i" occupa meno spazio di una "O"). Per permettere un posizionamento agevole, possibile utilizzare l'oggetto FontMetrics, che può essere ricavato direttamente da Graphics grazie al metodo:

```
FontMetrics getFontMetrics()
FontMetrics getFontMetrics(Font f)
```

Il primo di questi metodi restituisce il FontMetrics relativo al font attualmente in uso all'interno dell'oggetto Graphics, mentre il secondo permette di ottenere quello di un qualsiasi font di sistema. Grazie a FontMetrics possibile conoscere i principali parametri tipografici del font:

```
int charWidth(char ch)
int getAscent()
int getDescent()
int getHeight()
int getLeading()
int getMaxAdvance()
int getMaxAscent()
int[] getWidths()
boolean hasUniformLineMetrics()
int stringWidth(String str)
```

La conoscenza di questi parametri consente di ottenere un controllo pressoché totale sul modo di visualizzare i caratteri a schermo. Ai fini di un uso normale, i due metodi più importanti sono int stringWidth(String s) e int getHeight(), che restituiscono rispettivamente la larghezza in pixel di una determinata stringa in quel font e l'altezza del font stesso (intesa come distanza tra l'estremità inferiore di un carattere discendente, come la p minuscola, e l'estremità superiore dei caratteri più alti, come la I maiuscola). Il seguente esempio mostra come si possano utilizzare queste misure per disporre una stringa al centro di un pannello. All'interno di questo programma viene utilizzato il componente JFontChooser, il cui sorgente è stato descritto nel capitolo 15.

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

public class DrawStringExample extends JComponent  {
  private String text;

  public DrawStringExample(String text) {
    this.text = text;
  }

  public void paintComponent(Graphics g) {
    FontMetrics metrics = g.getFontMetrics();
    Dimension size = getSize();
    int fontHeight = metrics.getHeight();
    int textWidth = metrics.stringWidth(text);
```

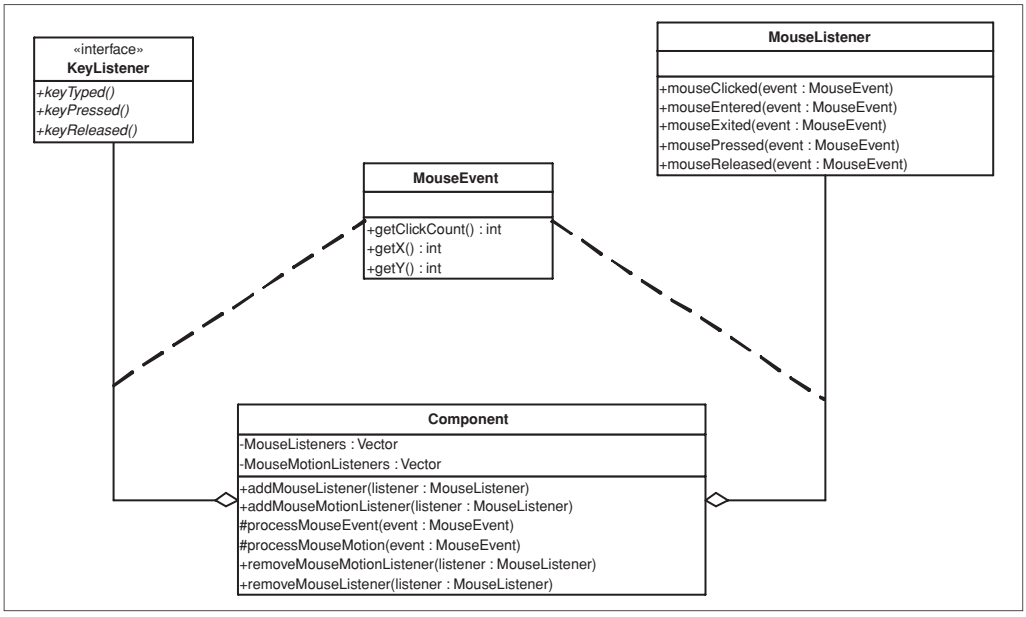
```
    int x = (size.width - textWidth) / 2;
    int y = (size.height + (fontHeight/2)) / 2;
    g.drawString(text,x,y);
  }

  public static void main(String argv[]) {
    if(argv.length != 1)
      throw new IllegalArgumentException("Use: java PaintImageExample <string>");
    JFrame f = new JFrame("ImageExample");
    f.getContentPane().setLayout(new BorderLayout());

    FontChooser fc = new FontChooser();
    final DrawStringExample dse = new DrawStringExample(argv[0]);

    fc.addPropertyChangeListener(new PropertyChangeListener() {
      public void propertyChange(PropertyChangeEvent e) {
        if(e.getPropertyName().equals("font"))
          dse.setFont((Font)e.getNewValue());
          dse.repaint();
      }
    });
    f.setSize(600, 500);
    f.getContentPane().add(BorderLayout.NORTH,fc);
    f.getContentPane().add(BorderLayout.CENTER,dse);

    fc.setFont(dse.getFont());
    f.setVisible(true);
  }
}
```

Figura 16.4 – *FontMetrics consente di posizionare in modo preciso le scritte sullo schermo.*

Eventi di mouse

Ogni componente grafico è in grado di notificare gli eventi generati dal mouse. In particolare, è predisposto per lavorare con due tipi di ascoltatori: MouseListener e MouseMotionListener. Il primo è specializzato nella gestione degli eventi relativi alla pressione dei pulsanti del mouse, mentre il secondo si occupa dello spostamento del puntatore. I metodi dell'interfaccia MouseListener intercettano la pressione di un pulsante del mouse, il suo rilascio, il clic (generato dopo una sequenza di pressione-rilascio), l'entrata e l'uscita del puntatore dall'area del componente:

Figura 16.5 – *Diagramma di classe degli eventi del mouse.*




```
public void mousePressed(MouseEvent e);
public void mouseReleased(MouseEvent e);
public void mouseClicked(MouseEvent e);
public void mouseEntered(MouseEvent e);
public void mouseExited(MouseEvent e);
```

MouseMotionListener, invece, ascolta lo spostamento del mouse, sia libero (mouseMoved()) sia associato alla pressione di un pulsante (mouseDragged()):

```
public void mouseMoved(MouseEvent e)
public void mouseDragged(MouseEvent e)
```

Entrambi gli ascoltatori utilizzano come evento MouseEvent, il quale dispone di un insieme di metodi che consentono di conoscere il componente che ha generato l'evento, la posizione del mouse e il numero di clic consecutivi registrati:

```
Component getComponent()
int getClickCount()
Point getPoint()
int getX()
int getY()
```

MouseEvent dispone anche di un metodo boolean isPopupTrigger(), che restituisce true quando l'evento generato può essere interpretato come una richiesta di menu contestuale secondo le convenzioni della piattaforma sottostante. L'uso di questo metodo è stato illustrato nel capitolo 13, nel paragrafo su JPopupMenu. Il package java.awt.event contiene le classi MouseAdapter e MouseMotionAdapter, che forniscono un'implementazione vuota dei due ascoltatori. Il package javax.swing.event, invece, contiene la classe MouseInputAdapter, che fornisce un'implementazione vuota di entrambe le interfacce.

MouseEvent è una sottoclasse di InputEvent, una classe che dispone di un gruppo di metodi che permettono di sapere quali pulsanti erano premuti al momento del clic:

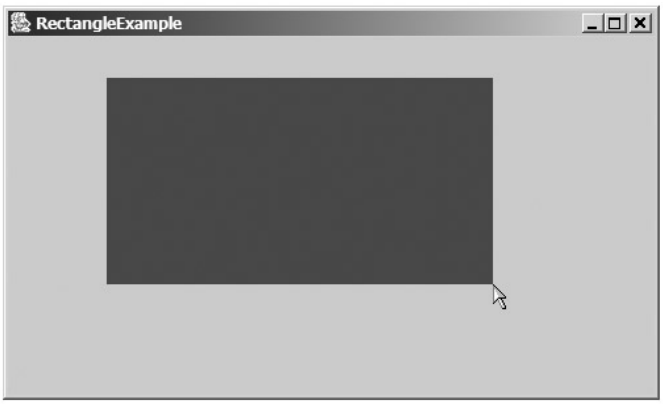
```
int getModifiers()
boolean isAltDown()
boolean isAltGraphDown()
boolean isControlDown()
boolean isMetaDown()
boolean isShiftDown()
```

Il prossimo esempio mostra come utilizzare gli eventi del mouse per creare semplici disegni sullo schermo. La classe principale è un JComponent, il cui metodo paintComponent() dipinge un rettangolo. Le coordinate e le dimensioni del rettangolo vengono aggiornate secondo i clic e gli spostamenti del mouse da un apposito MouseInputAdapter:

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

public class RectangleExample extends JComponent  {
  private Point corner1 = new Point(0,0);
  private Point corner2 = new Point(0,0);

  public RectangleExample() {
    MouseInputAdapter m = new RectangleExampleMouseListener();
    addMouseListener(m);
    addMouseMotionListener(m);
```

```
  }
  public void paintComponent(Graphics g) {
    int x = Math.min(corner1.x,corner2.x);
    int y = Math.min(corner1.y,corner2.y);
    int width  = Math.abs(corner1.x - corner2.x);
    int height = Math.abs(corner1.y - corner2.y);

    g.fillRect(x, y, width, height);
  }
  class RectangleExampleMouseListener extends MouseInputAdapter {
    public void mousePressed(MouseEvent e) {
      corner1 = e.getPoint();
      corner2 = corner1;
      repaint();
    }
    public void mouseDragged(MouseEvent e) {
      corner2 = e.getPoint();
      repaint();
    }
  }
  public static void main(String argv[]) {
    JFrame f = new JFrame();
    f.setSize(500, 300);
    RectangleExample r = new RectangleExample();
    r.setForeground(Color.BLUE);
    f.getContentPane().add(r);
    f.setVisible(true);
  }
}
```

Figura 16.6 – *Un componente attivo, capace di reagire agli eventi del mouse.*



Eventi di tastiera

I componenti prevedono anche il supporto agli eventi di tastiera, mediante l'ascoltatore KeyListener, caratterizzato dai seguenti metodi:

```
void keyPressed(KeyEvent e)
void keyReleased(KeyEvent e)
void keyTyped(KeyEvent e)
```

L'oggetto KeyEvent possiede i seguenti metodi:

```
char getKeyChar()
int getKeyCode()
```

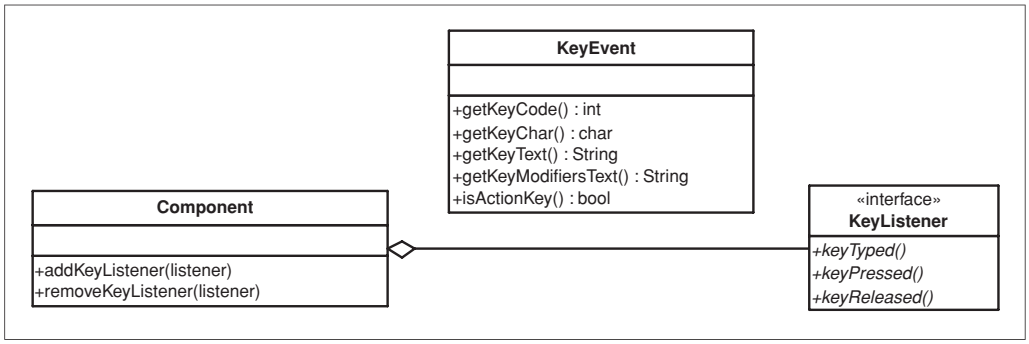
Il primo restituisce il carattere corrispondente al tasto premuto. Alcuni tasti non corrispondono a caratteri (Invio, Esc e così via), quindi per identificare i tasti viene usato un codice numerico detto key code. La classe KeyEvent possiede una serie di costanti corrispondenti ai codici carattere di qualsiasi tasto della tastiera. Ecco qualche esempio:

```
VK_ENTER
VK_ESCAPE
VK_EURO_SIGN
VK_F1
VK_F2
```

I tasti alfanumerici restituiscono anche il carattere a cui corrispondono, tramite il metodo getKeyChar(). Anche KeyEvent è sottoclasse di InputEvent e dispone dei metodi necessari a verificare la precisa combinazione dei tasti premuta dall'utente:

```
int getModifiers()
boolean isAltDown()
boolean isAltGraphDown()
boolean isControlDown()
boolean isMetaDown()
boolean isShiftDown()
```

Figura 16.7 – *Le classi relative alla gestione degli eventi di tastiera.*

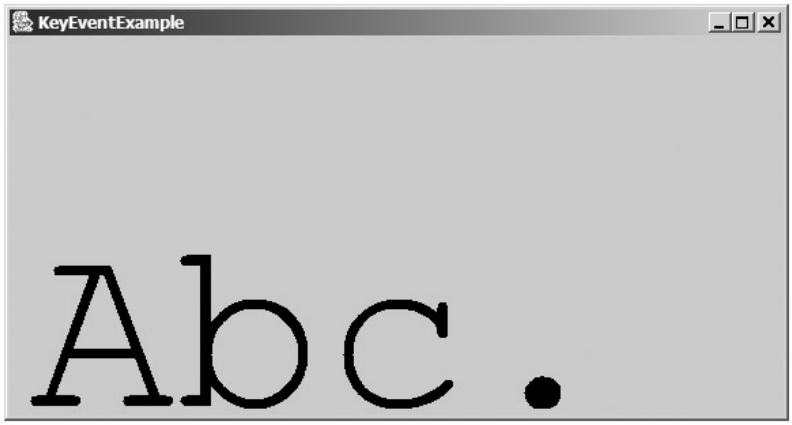


Il seguente esempio crea un componente al cui interno viene visualizzata la lettera "A". Premendo i tasti alfanumerici o il tasto backspace è possibile creare una frase. Mediante i tasti cursore è possibile spostare la scritta sullo schermo:

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

public class KeyEventExample extends JFrame {

  private int x = 30;
  private int y = 220;
  private String s = "A";

  public KeyEventExample() {
    getContentPane().add(new InnerComponent());
    setSize(500,400);
    addKeyListener(new KeyHandler());
  }
  class InnerComponent extends JComponent {
    public void paintComponent(Graphics g) {
      g.setFont(new Font("monospaced",5,180));
      if(!s.equals(""))
        g.drawString(s,x,y);
    }
  }
  class KeyHandler extends KeyAdapter {
    public void keyPressed(KeyEvent e) {
      int c = e.getKeyCode();
        switch(c) {
          case KeyEvent.VK_UP :
```

```
            y--;
            break;
          case KeyEvent.VK_DOWN :
            y++;
            break;
          case KeyEvent.VK_LEFT :
            x--;
            break;
          case KeyEvent.VK_RIGHT :
            x++;
            break;
          case KeyEvent.VK_BACK_SPACE :
            int length = s.length() > 1 ? s.length()-1 : 0;
            s = s.substring(0,length);
            break;
          default :
            s = s+Character.toString(e.getKeyChar());
            break;
        }
        repaint();
    }
  }

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

Figura 16.8 – *Un componente capace di reagire agli eventi della tastiera.*



Disegno a mano libera

Per concludere questo capitolo è utile un esempio riepilogativo. Anche se i concetti appena illustrati costituiscono solo una frazione delle reali possibilità offerte da Java, essi permettono tuttavia di creare programmi grafici di una certa complessità.

Un uso adeguato degli eventi del mouse e delle primitive di disegno permette di realizzare un semplice programma di disegno a mano libera. Il disegno viene memorizzato sotto forma di poly linee: oggetti grafici composti da una sequenza di linee collegate tra loro in modo tale da dare l'illusione di un tratto continuo. Ogni volta che l'utente preme il pulsante viene creato un Vector, all'interno del quale vengono immagazzinate le coordinate di ogni punto in cui il mouse si viene a trovare durante il drag. Quando il pulsante viene rilasciato, la poly linea viene interrotta. Il Vector polyLineList contiene tutte le poly linee che compongono il disegno. Ogni volta che l'area del componente necessita di un repaint, le poly linee vengono ridipinte una a una sullo schermo:

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

public class Painter extends JComponent {

 // contiene un Vector per ogni poly line
 private Vector polyLineList = new Vector();

 // Costruttore della classe principale
 public Painter() {
  super();
  MouseInputListener m = new MyMouseInputListener();
  addMouseListener(m);
  addMouseMotionListener(m);
 }
 public void paintComponent(Graphics g) {
  // disegna ogni poly line
  Iterator polyLineIterator = polyLineList.iterator();
  while(polyLineIterator.hasNext()) {
   Vector polyLine = (Vector)polyLineIterator.next();
   Iterator pointIterator = polyLine.iterator();
   // disegna ogni linea della poly line
   Point oldPoint = (Point)pointIterator.next();
   while(pointIterator.hasNext()) {
    Point newPoint = (Point)pointIterator.next();
    g.drawLine(oldPoint.x,oldPoint.y,newPoint.x,newPoint.y);
    oldPoint = newPoint;
   }
```

```
    }
  }

  class MyMouseInputListener extends MouseInputAdapter {
    // pulsante premuto
    public void mousePressed(MouseEvent e) {
      // crea una nuova poly line e la inserisce nella lista
      Vector polyLine = new Vector();
      polyLine.add(new Point(e.getX(),e.getY()));
      polyLineList.add(polyLine);
    }
    public void mouseDragged(MouseEvent e) {
      // aggiunge un punto alla poly line
      Vector polyLine = (Vector)polyLineList.lastElement();
      polyLine.add(e.getPoint());
      repaint();
    }
  }

  public static void main(String argv[]) {
    Painter p = new Painter();
    JFrame f = new JFrame("Painter");
    f.getContentPane().add(p);
    f.setSize(400,300);
    f.setVisible(true);
  }
}
```

Figura 16.9 – *Le direttive di disegno e la manipolazione degli eventi del mouse permettono di esprimere il proprio estro creativo.*