I Canvas

Tra i vari **contenitori Java** il **Canvas** (*area di disegno* o *tela*) è una semplice superficie di disegno particolarmente utile per visualizzare immagini o per effettuare altre operazioni grafiche.

Un oggetto di tipo `Canvas` è semplicemente una **porzione rettangolare di schermo**, una specie di pannello che possiede un proprio metodo **paint()** che, pur diverso dall'omonimo metodo di un Applet, usa anch'esso oggetti della classe *Graphics* ed è richiamato dal sistema in fase di creazione e aggiornamento in modo che l'oggetto Graphics associato possa aggiornarsi .

Una istanza della classe `Canvas` non è molto utile: riceve eventi da tastiera e da mouse, ma non li tratta, ed il metodo `paint()` non disegna alcunché.

Generalmente la classe **`Canvas` è usata** solo **come base per subclassi**, che possono generare oggetti più utili: una subclasse di `Canvas` generalmente farà almeno un'*override* del metodo `paint()` e possibilmente anche dei metodi di gestione degli eventi come quelli del mouse e della tastiera.

Quando il contenuto del canvas dovrà essere modificato al di fuori del metodo `paint()`, sarà possibile richiamare il *contesto grafico* con i metodi `getGraphics()`. Vedremo come questi metodi possono essere usati per i canvas creando ad esempio un bottone di forma personalizzata e colore modificato da eventi

Pur se è possibile disegnare direttamente in ogni Component, ed è comune disegnare direttamente su applet e finestre, è di solito meglio collocare ogni operazione di disegno nelle sottoclassi del `Canvas`: nei casi in cui sia necessario disegnare su un'applet che contiene anche dei bottoni, l'uso di un canvas è pressochè inevitabile.

// esempio di **Applicazione** con uso GUI (necessita una finestra o **JFrame**)
// Per disegnare uso di classe che **eredita da Canvas**

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

public class Graf{

    public Graf () {

            JFrame f = new JFrame("Finestra"); // crea frame invisibile

            Container c = f.getContentPane();  // recupera il "muro grezzo"

            Disegno d = new Disegno();         // classe esterna che eredita da Canvas

            c.add (d);
            f.setSize(300,180);                  // per impostare le dimensioni e la posizione:  misure in pixel
            f.setLocation(200,100);              // (0,0) angolo sup. sin.
            f.setResizable(true);    // per ridimensionare (per default non ridimensionabili) con mouse
            f.setVisible(true);                                          // mostra il frame (dimensioni 300x150)
            f.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String [] args) {
          Graf ogg = new Graf();
    }
}// fine classe principale
```

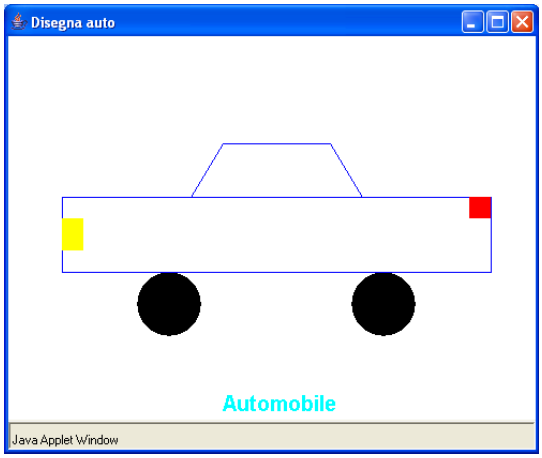
```
// classe esterna eventualmente nello stesso file Graf.java

class Disegno extends Canvas  {

      public void paint(Graphics g)  {
       g.setColor(Color.red);
       g.fillRect(20,20,100,80);     // rettangolo pieno
       g.setColor(Color.blue);
       g.drawRect(30,30,80,60);      // linea rettangolare
       g.setColor(Color.black);
       g.drawString("ciao", 50,60);

       Font f1 = new Font("Times", Font.PLAIN, 20);      // cambiare font
       g.setFont(f1);
       g.drawString("ciao", 200,60);

       // per recuperare le proprietà del font
       f1 = g.getFont();
       int size = f1.getSize();
       int style = f1.getStyle();
       String name = f1.getName();

       g.drawString("Font di dimensioni " + size, 50,140);

       System.out.println("Font di dimensioni "+ size +" stile " + style + " e nome " + name);
        }
    }// fine classe che eredita da Canvas
```

Esercizio: disegnare automobile

Creare un Applet che disegni un'automobile in modo stilizzato:

1) direttamente nella finestra del browser.

2) con un **canvas in una finestra separata**. Si costruisca una classe AutoC con gli opportuni metodi grafici già usati per disegnarla direttamente nella finestra del browser. (con soluzione)



nb: **applicazione** con disegno su JPanel della stessa automobile stilizzata

Soluzione: esempio di Applet che usa **canvas in una finestra separata**

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

public class AutoC extends JApplet  {
        Auto A;
         public void init()   {
              JFrame f1 = new JFrame ("Disegna auto");
              A = new Auto();                                    // istanza che eredita da Canvas
              f1.setSize (500, 420);
              f1.setLocation (50,50);
              f1.setResizable(true);    // con mouse ridimensionabile
              Container c =f1.getContentPane();
              c.add(A);
              f1.setVisible(true);
            }
       }

class Auto extends Canvas  {
            public void paint(Graphics g)  {
              Font f = new Font("Times Roman", Font.BOLD, 20);
              g.setFont(f);
              g.fillOval(120, 220, 60,60);    // una ruota piena
              g.fillOval(320, 220, 60,60);    // altra ruota piena
              g.setColor(Color.blue);         // carrozzeria blu
              g.drawRect(50, 150, 400, 70);
              g.drawLine(170,150,200,100);
              g.drawLine(330,150,300,100);
              g.drawLine(300,100,200,100);
              g.setColor(Color.yellow);       // luci gialle
              g.fillRect(50, 170, 20, 30);
              g.setColor(Color.red);          // luci rosse
              g.fillRect(430, 150, 20, 20);
              g.setColor(Color.cyan);         // testo cyan
              g.drawString("Automobile", 200, 350);
                }
          }
```

Codice HTML:

```
<html>
<head><title>Disegno di un Auto </title></head>
<body>
<object  code ="AutoC.class"
            width= "500"
             height= "150">></object>
</body>
</html>
```

nb: per eseguire la **pagina html** con **opzione Run** in JCreator, salvare tale pagina web nella stessa cartella del bytecode (**classes**).

Esercizio: Disegno di un auto in **altra finestra** alla pressione di un pulsante (**evento della GUI**)

Con codice HTML:

```
<html>
        <head>
                <title>Disegno di un Auto come evento </title>
        </head>
        <body>
                <object code ="Autoe.class" width = "350" height ="70" >
                </object>
        </body>
</html>
```

```
// Applet Autoe.java
// esempio di Applet con gestione eventi della GUI: creazione di eventi personalizzati
// Uso canvas

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

public class Autoe extends JApplet {

        Auto A;

        public void init() {

             JButton B = new JButton ("Disegna un auto");

            GestiscePulsante A1 = new GestiscePulsante();   // crea ascoltatore

            B.addActionListener (A1);     // lega la classe di ascolto all'origine dell'evento: cioè
                                          // il pulsante B
            add(B);
           }

        public void Disegna()  {

             JFrame f1 = new JFrame("Disegna auto");
             A = new Auto();

             f1.setSize (300, 200);
             f1.setLocation (50,50);
             f1.setResizable(true);    // con mouse ridimensionabile
             Container c = f1.getContentPane();  // recupera il "muro grezzo"

             c.add(A);
             f1.setVisible(true);
           }
```

```
    // definizione classe d'ascolto interna cioè innestata non statica
     private class GestiscePulsante implements ActionListener{ // interfaccia per eventi di azione

                public void actionPerformed (ActionEvent e) {   // unico metodo da ridefinire

                   Disegna();
                 }
          }     // fine classe d'ascolto
         }

class Auto extends Canvas  {                  // classe esterna per disegnare auto

       public void paint(Graphics g)  {

          Font f = new Font("Times Roman", Font.BOLD, 20);
                                  // tipo, aspetto, dimensione carattere
          g.setFont(f);

          g.fillOval(120, 220, 60,60);    // una ruota piena
          g.fillOval(320, 220, 60,60);    // altra ruota piena
          g.setColor(Color.blue);          // carrozzeria blu
          g.drawRect(50, 150, 400, 70);
          g.drawLine(170,150,200,100);
          g.drawLine(330,150,300,100);
          g.drawLine(300,100,200,100);
          g.setColor(Color.yellow);       // luci gialle
          g.fillRect(50, 170, 20, 30);
          g.setColor(Color.red);          // luci rosse
          g.fillRect(430, 150, 20, 20);

          g.setColor(Color.cyan);         // testo cyan
          g.drawString("Automobile", 200, 350);
           }
       }
```

Classi innestate e interne: definizione

Una classe innestata non è altro che una classe che viene definita all'interno di un'altra classe.

Il vantaggio di implementare una classe all'interno di un'altra, riguarda principalmente il **risparmio di codice**. Infatti, la classe innestata ha accesso alle variabili di istanza della classe in cui è *innestata*.

N.B. : se compiliamo una classe che contiene una classe innestata, saranno creati due file:
"nomeClasseEsterna.class" e "nomeClasseEsterna**$**nomeClasseinnestata.class".

N.B. : fino alla versione 1.3 di Java, il termine "classe innestata" non era stato ancora adottato. Si parlava invece di "classe interna" ("inner class").
*Dalla versione 1.4 in poi, la Sun ha deciso di sostituire il termine "classe interna", con il termine "classe innestata" o classe annidata ("**nested class**"). Il termine "classe interna" deve ora essere utilizzato solo per le classi **innestate** che **non** sono **dichiarate statiche**. In realtà la stragrande maggior parte delle volte si utilizzano classi interne.*