
Semplici applicazioni grafiche in Java

Applicazione grafica

- Visualizza informazioni all'interno di una finestra dotata di barra di titolo e cornice (frame)
 - La Java Virtual Machine esegue ogni frame su una thread separata
 - la gestione del frame e delle operazioni che genera è affidata ad una thread
 - Thread = flusso di esecuzione, processo, visione dinamica di un programma sequenziale
-

Finestre: classe javax.swing.JFrame

```
import javax.swing.*;
```

```
.....
```

```
.....
```

```
.....
```

```
JFrame frame =  
    new JFrame();  
frame.setSize(300, 400);  
frame.setTitle(  
    "An Empty Frame");  
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
frame.setVisible(true);
```



File EmptyFrameViewer.java

```
01: import javax.swing.*;
02:
03: public class EmptyFrameViewer
04: {
05:     public static void main(String[] args)
06:     {
07:         JFrame frame = new JFrame();
08:
09:         final int FRAME_WIDTH = 300;
10:         final int FRAME_HEIGHT = 400;
11:
12:         frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
13:         frame.setTitle("An Empty Frame");
14:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15:
16:         frame.setVisible(true);
17:     }
18: }
```

Disegnare figure

- per visualizzare qualcosa all'interno di un frame occorre definire un oggetto di tipo `JComponent` e aggiungerlo al frame
- si deve estendere la classe `JComponent` (pacchetto `javax.swing`)

```
public class RectangleComponent extends JComponent
{
    public void paintComponent(Graphics g)
    {
        // Recover Graphics2D
        Graphics2D g2 = (Graphics2D) g;
        . . .
    }
}
```

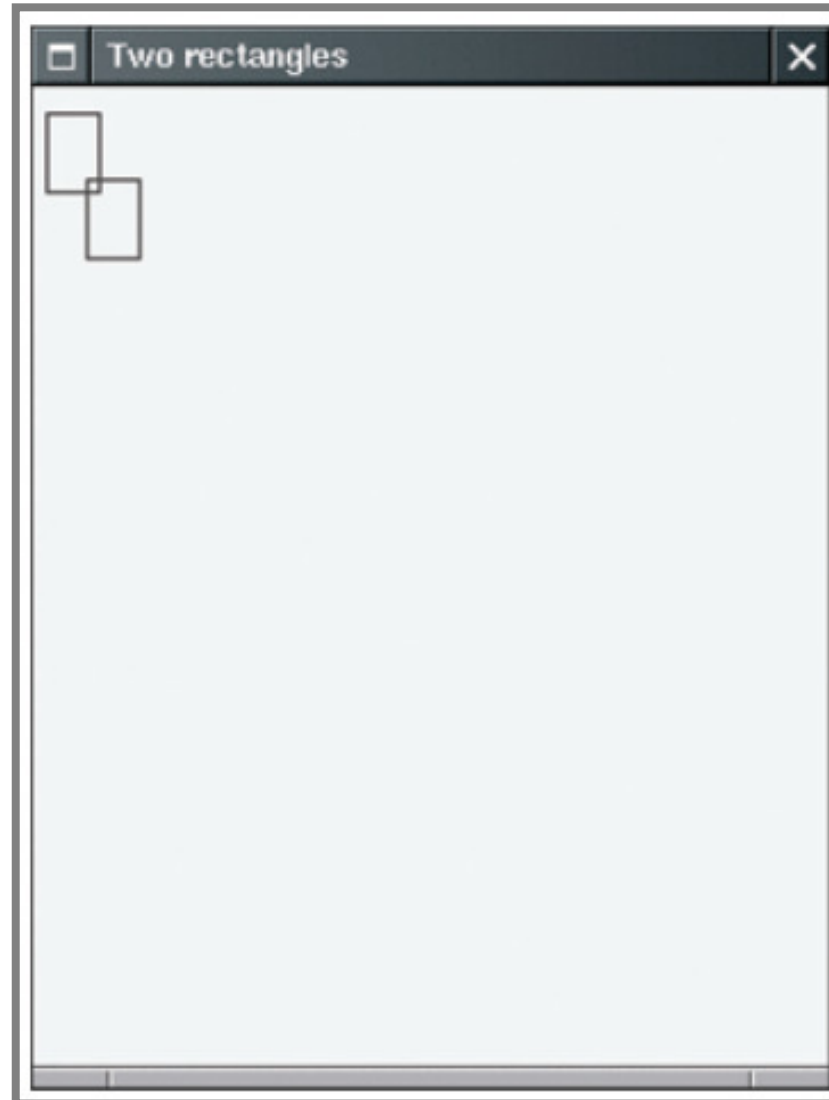
Metodo paintComponent

- Invocato automaticamente ogni volta che una componente necessita di essere ridisegnata
 - quando una finestra viene resa visibile la prima volta
 - quando una finestra viene ridimensionata
 - quando una finestra diviene nuovamente visibile dopo essere stata nascosta (`setVisible(false)`)
 - Le istruzioni di disegno sono inserite in questo metodo
-

Disegnare forme

- `Graphics` ci consente di manipolare lo stato grafico (ad es. colore). Classe astratta.
 - `Graphics2D`: astratta, estende `Graphics`, ha metodi per tracciare forme grafiche
 - **Cast a `Graphics2D` del parametro `Graphics` serve per usare metodo `draw`**
 - `Graphics` e `Graphics2D` sono in **`java.awt`**
-

Esempio: disegnare rettangoli in un frame



Classi del programma

- `RectangleComponent`: riscrive metodo `paintComponent` per tracciare rettangoli nella maniera voluta
 - `RectangleViewer`:
 - contiene il metodo `main` che costruisce un frame
 - aggiunge una componente al frame e rende il frame visibile
-

File RectangleComponent.java

```
01: import java.awt.Graphics; import java.awt.Graphics2D;
02: import java.awt.Rectangle; import javax.swing.JComponent;
03:
04: // A component that draws two rectangles.
05: public class RectangleComponent extends JComponent{
06:     public void paintComponent(Graphics g){
07:
08:         // Recover Graphics2D
09:         Graphics2D g2 = (Graphics2D) g;
10:
11:         // Construct a rectangle and draw it
12:         Rectangle box = new Rectangle(5, 10, 20, 30);
13:         g2.draw(box);
14:
15:         // Move rectangle 15 units to the right and 25 units down
16:         box.translate(15, 25);
17:
18:         // Draw moved rectangle
19:         g2.draw(box);
20:     }
21: }
```

File RectangleViewer.java

```
01: import javax.swing.JFrame;
02:
03: public class RectangleViewer
04: {
05:     public static void main(String[] args)
06:     {
07:         JFrame frame = new JFrame();
08:
09:         final int FRAME_WIDTH = 300;
10:         final int FRAME_HEIGHT = 400;
12:         frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
13:         frame.setTitle("Two rectangles");
14:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15:
16:         RectangleComponent component = new RectangleComponent();
17:         frame.add(component);
18:
19:         frame.setVisible(true);
20:     }
21: }
```

Ricapitoliamo i passi

1. Istanza un frame e settare parametri (ampiezza, altezza, titolo, etc)

```
JFrame frame = new JFrame();  
frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);  
frame.setTitle("Two rectangles");  
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

2. Istanza una componente:

```
RectangleComponent component = new RectangleComponent();
```

3. Aggiungi la componente al frame

```
frame.add(component);
```

4. Rendi il frame visibile

```
frame.setVisible(true)
```

Applet

- Le applet sono programmi che vengono eseguiti in un web browser (analogo di JComponent)
- Per implementare una applet:

```
import javax.swing.JApplet;

public class MyApplet extends JApplet
{
    public void paint(Graphics g)
    {
        // Recover Graphics2D
        Graphics2D g2 = (Graphics2D) g;
        // Drawing instructions go here
        . . .
    }
}
```

Applet

- Rispetto alle componenti:
 1. Si estende `JApplet` invece di `JComponent`
 2. Il codice per tracciare il disegno viene messo nel metodo `paint` anziché `paintComponent`
 - Per eseguire una applet, si deve scrivere un file HTML con un tag `applet`
-

File RectangleApplet.java

```
01: import java.awt.Graphics;
02: import java.awt.Graphics2D;
03: import java.awt.Rectangle;
04: import javax.swing.JApplet;
05:
06: /**
07:     An applet that draws two rectangles.
08: */
09: public class RectangleApplet extends JApplet
10: {
11:     public void paint(Graphics g)
12:     {
13:         // Prepare for extended graphics
14:         Graphics2D g2 = (Graphics2D) g;
```

File RectangleApplet.java

```
15:
16:     // Construct a rectangle and draw it
17:     Rectangle box = new Rectangle(5, 10, 20, 30);
18:     g2.draw(box);
19:
20:     // Move rectangle 15 units to the right and 25 units
    // down
21:     box.translate(15, 25);
22:
23:     // Draw moved rectangle
24:     g2.draw(box);
25: }
26: }
27:
```

Applet

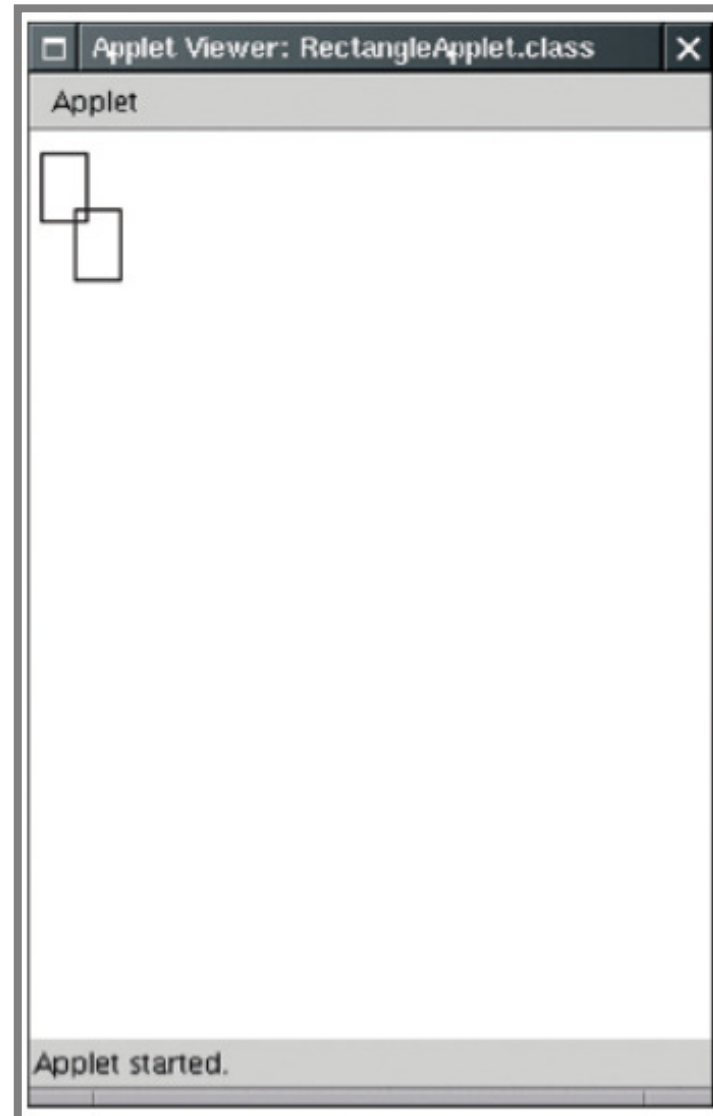
- Un file HTML può avere applet multiple
 - si aggiunge un tag `applet` per ogni applet
- Si possono visualizzare le applet con un `applet viewer` o con un browser con Java abilitato

```
appletviewer RectangleApplet.html
```

File `RectangleApplet.html`:

```
<applet code="RectangleApplet.class" width="300" height="400"  
> </applet>
```

Output con appletviewer

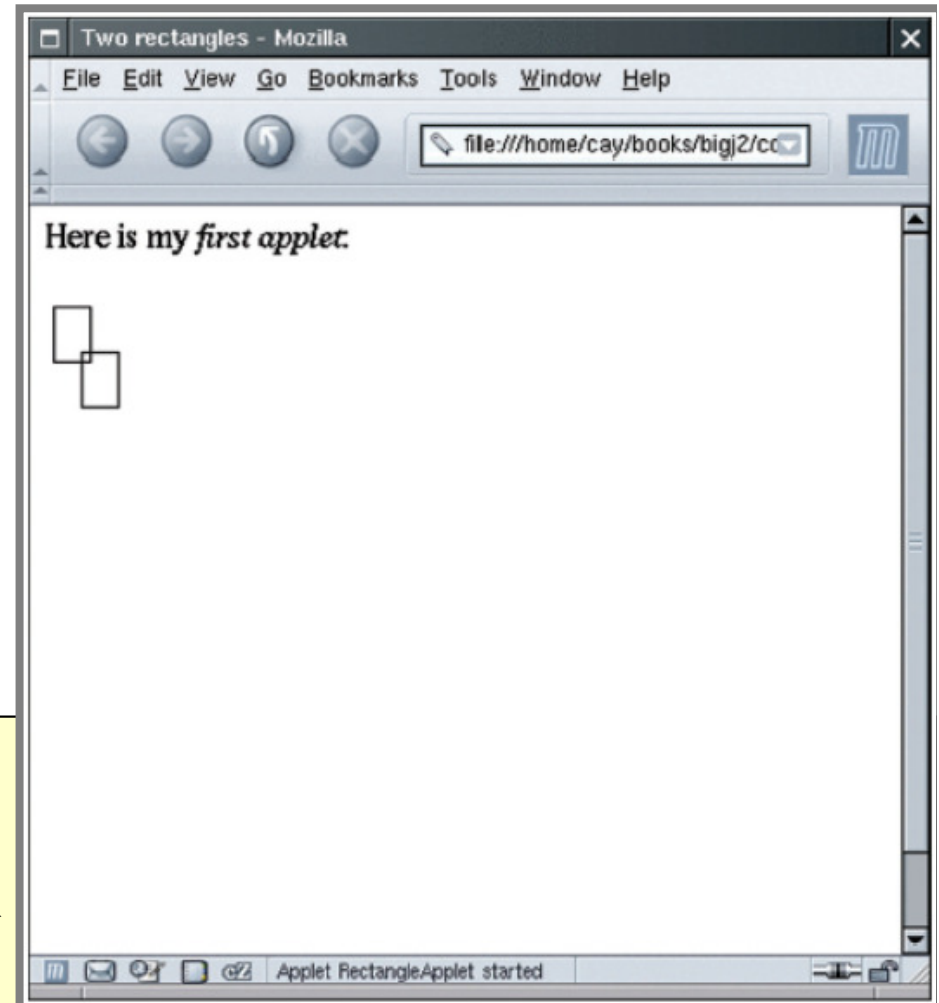


Applet con browser

File

RAppletExplained.html:

```
<html>
  <head>
    <title>Two rectangles</title>
  </head>
  <body>
    <p>Here is my <i>first applet</i>:</p>
    <applet code="RectangleApplet.class" width="300" height="400">
    </applet>
  </body>
</html>
```



Forme grafiche

- **Le classi** `Rectangle`, `Ellipse2D.Double`, e `Line2D.Double` **descrivono forme grafiche**
 - Classi `.Float` esprimono coordinate in float
- **Queste classi** `.Double` e `.Float` **sono classi interne** (`static`) di `Ellipse2D` e `Line2D`
 - fare attenzione con l'istruzione `import`:

```
import java.awt.geom.Ellipse2D; // no .Double
```

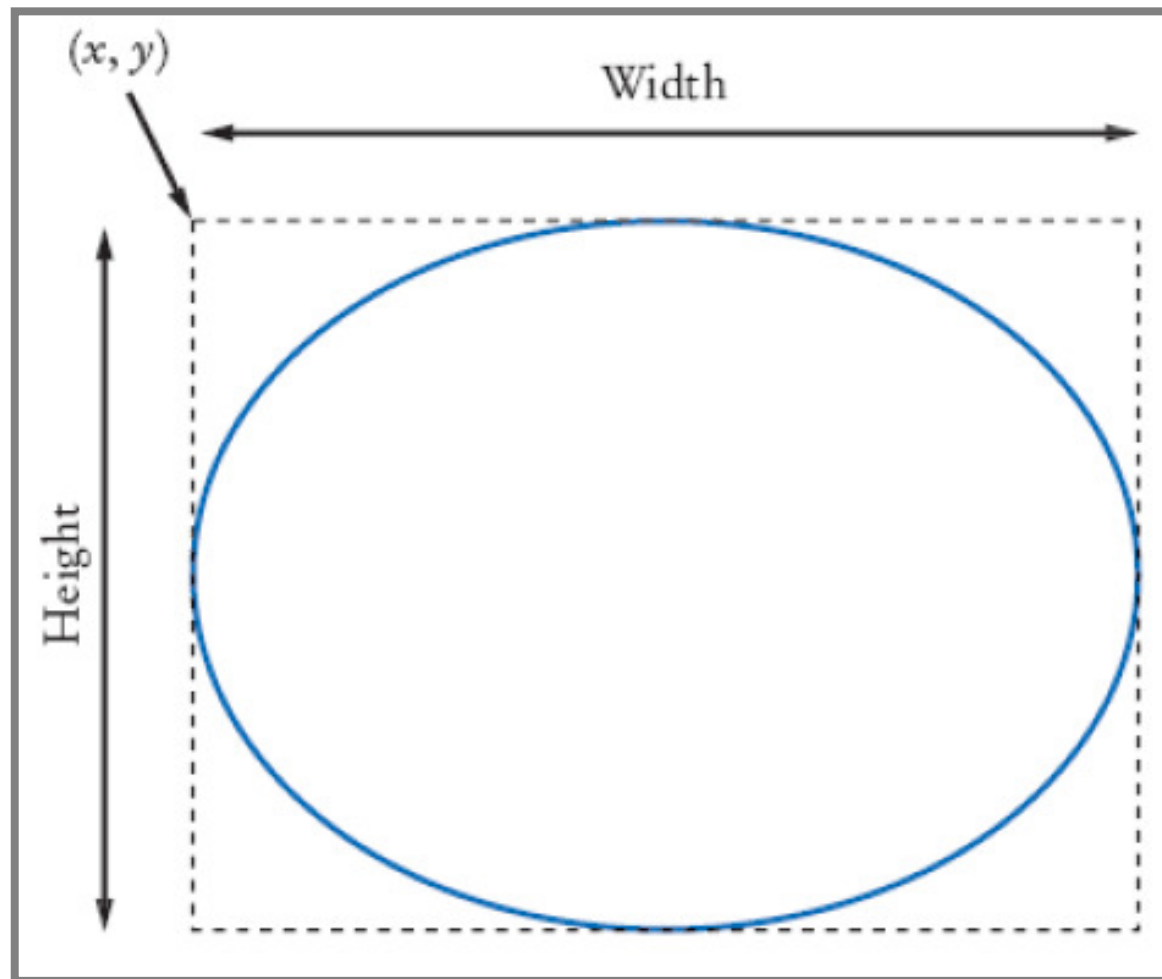
- **Una forma deve essere istanziata e tracciata**

```
Ellipse2D.Double ellipse = new Ellipse2D.Double(x, y, width, height);  
g2.draw(ellipse);
```

Classi interne static (nested top-level class)

- Sono definite come membri della classe (fuori dai metodi) e si usa lo specificatore **static**
 - Rispetto alle classi interne non static non devono essere istanziate necessariamente in un oggetto della classe ospitante
 - Possono essere istanziate in altre classi indipendentemente dalla classe in cui sono definite (da questo punto di vista sono classi a tutti gli effetti)
 - Sono un modo conveniente per raggruppare classi tra di loro collegate senza creare un nuovo package
 - Es.: `Ellipse2D.Double` e `Ellipse2D.Float`
-

Significato dei parametri di Ellipse



Istanziare oggetti Line2D

- Passiamo coordinate estremi:

```
Line2D.Double segment = new Line2D.Double(x1, y1, x2, y2);
```

oppure gli estremi stessi:

```
Point2D.Double from = new Point2D.Double(x1, y1);  
Point2D.Double to = new Point2D.Double(x2, y2);  
Line2D.Double segment = new Line2D.Double(from, to);
```

Tracciare oggetti di tipo String

```
g2.drawString("Message", 50, 100);
```

Coordinate punto base
(entrambe **int** oppure **float**)



Colori

- Colori standard `Color.BLUE`, `Color.RED`, `Color.PINK`, etc. (costanti)
- Altri colori si possono ottenere combinando **rosso**, **verde** e **blu** dando per ognuno dei valori compresi tra `0.0F` e `1.0F`

- Ad es.:

```
Color magenta = new Color(1.0F, 0.0F, 1.0F);
```

- Stabilire i colori in un contesto grafico

```
g2.setColor(magenta);
```

- `Color` è usato quando si tracciano e riempiono forme grafiche

```
g2.fill(rectangle); // riempie con il colore corrente
```

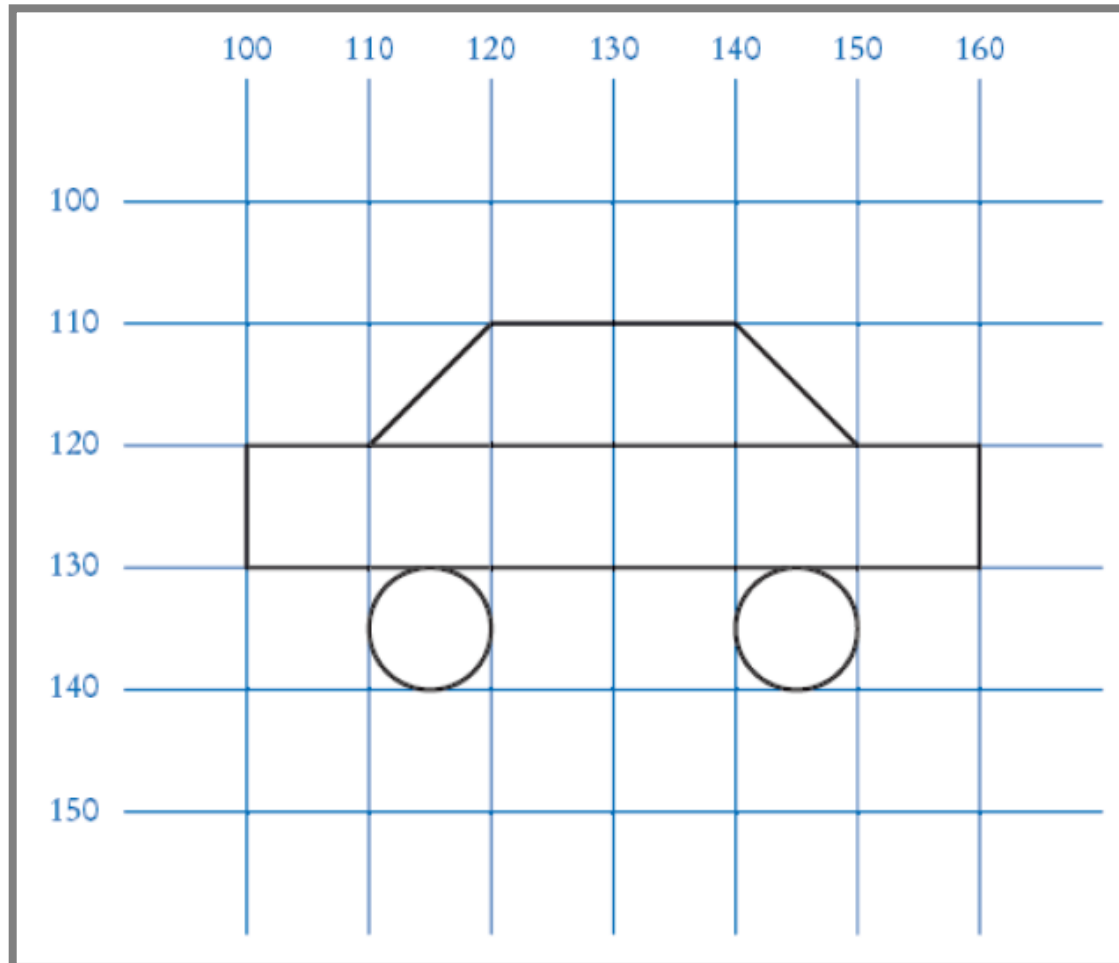
Disegnare figure complesse

- Consiglio: definire una classe per ogni forma

```
class Car
{
    . . .
    public void disegna(Graphics2D g2)
    {
        // Drawing instructions
        . . .
    }
}
```

- Per figure complesse determinare le coordinate con disegni su foglio quadrettato
-

Disegno auto



Applicazione grafica

- Disegna due auto:
 - una nell'angolo in alto a sinistra
 - l'altra nell'angolo in basso a destra
- Calcola la posizione in basso a destra nel metodo `paintComponent`:

```
int x = getWidth() - 60;  
int y = getHeight() - 30;  
Car car2 = new Car(x, y);
```

- `getWidth` e `getHeight` sono invocate sull'oggetto che esegue `paintComponent`
- se la taglia della finestra è modificata, `paintComponent` è invocata nuovamente e la posizione delle auto è ricalcolata

Output desiderato



File CarComponent.java

```
01: import java.awt.Graphics;
02: import java.awt.Graphics2D;
03: import javax.swing.JComponent;
04:
05: // This component draws two car shapes.
06: public class CarComponent extends JComponent
07: {
08:     public void paintComponent(Graphics g)
09:     {
10:         Graphics2D g2 = (Graphics2D) g;
11:         Car car1 = new Car(0, 0);
12:
13:         int x = getWidth() - Car.WIDTH;
14:         int y = getHeight() - Car.HEIGHT;
15:
16:         Car car2 = new Car(x, y);
17:
18:         car1.disegna(g2);
19:         car2.disegna(g2);
20:     }
21: }
```

File Car.java

```
01: import java.awt.Graphics2D;
02: import java.awt.Rectangle;
03: import java.awt.geom.Ellipse2D;
04: import java.awt.geom.Line2D;
05: import java.awt.geom.Point2D;
06:
07: /**
08:     A car shape that can be positioned anywhere on the screen.
09: */
10: public class Car
11: {
12:     /**
13:         Constructs a car with a given top left corner
14:         @param x the x coordinate of the top left corner
15:         @param y the y coordinate of the top left corner
16:     */
```

File Car.java

```
17:     public Car(int x, int y)
18:     {
19:         xLeft = x;
20:         yTop = y;
21:     }
22:
23:     /**
24:      * Draws the car.
25:      * @param g2 the graphics context
26:      */
27:     public void disegna(Graphics2D g2)
28:     {
29:         Rectangle body
30:             = new Rectangle(xLeft, yTop + 10, 60, 10);
31:         Ellipse2D.Double frontTire
32:             = new Ellipse2D.Double(xLeft + 10, yTop
33:                                     + 20, 10, 10);
33:         Ellipse2D.Double rearTire
```


File Car.java

```
34:         = new Ellipse2D.Double(xLeft + 40, yTop
           + 20, 10, 10);
35:
36:         // The bottom of the front windshield
37:         Point2D.Double r1
38:             = new Point2D.Double(xLeft + 10, yTop + 10);
39:         // The front of the roof
40:         Point2D.Double r2
41:             = new Point2D.Double(xLeft + 20, yTop);
42:         // The rear of the roof
43:         Point2D.Double r3
44:             = new Point2D.Double(xLeft + 40, yTop);
45:         // The bottom of the rear windshield
46:         Point2D.Double r4
47:             = new Point2D.Double(xLeft + 50, yTop + 10);
48:
49:         Line2D.Double frontWindshield
50:             = new Line2D.Double(r1, r2);
```

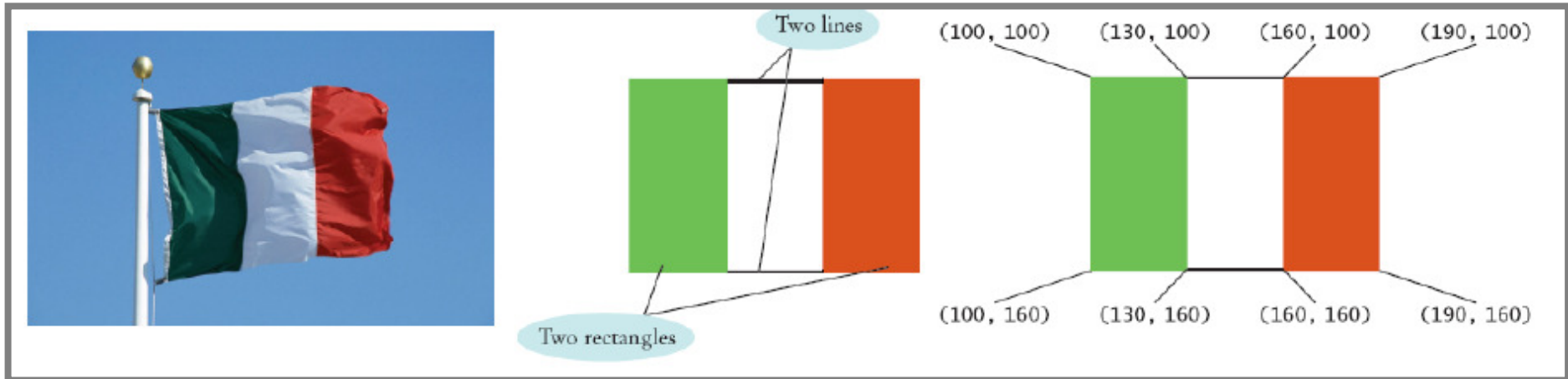
File Car.java

```
51:         Line2D.Double roofTop
52:             = new Line2D.Double(r2, r3);
53:         Line2D.Double rearWindshield
54:             = new Line2D.Double(r3, r4);
55:
56:         g2.draw(body);
57:         g2.draw(frontTire);
58:         g2.draw(rearTire);
59:         g2.draw(frontWindshield);
60:         g2.draw(roofTop);
61:         g2.draw(rearWindshield);
62:     }
63:
64:     public static int WIDTH = 60;
65:     public static int HEIGHT = 30;
66:     private int xLeft;
67:     private int yTop;
68: }
```

File CarViewer.java

```
01: import javax.swing.JFrame;
02:
03: public class CarViewer
04: {
05:     public static void main(String[] args)
06:     {
07:         JFrame frame = new JFrame();
08:
09:         final int FRAME_WIDTH = 300;
10:         final int FRAME_HEIGHT = 400;
11:
12:         frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
13:         frame.setTitle("Two cars");
14:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15:
16:         CarComponent component = new CarComponent();
17:         frame.add(component);
18:
19:         frame.setVisible(true);
20:     }
21: }
```

Disegnare forme grafiche

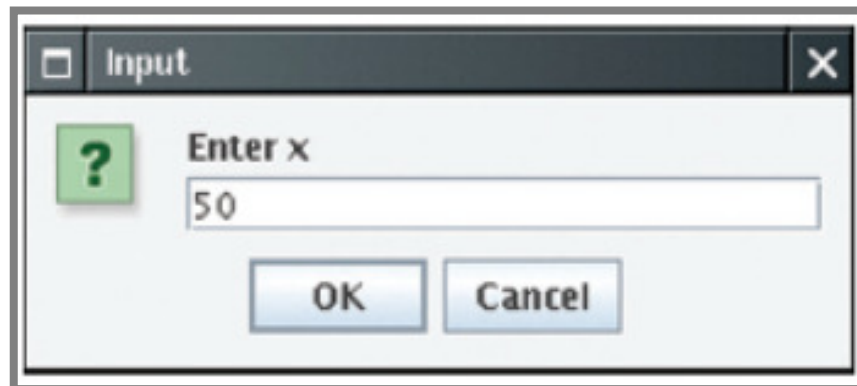


```
Rectangle leftRectangle = new Rectangle(100, 100, 30, 60);  
Rectangle rightRectangle = new Rectangle(160, 100, 30, 60);  
Line2D.Double topLine  
    = new Line2D.Double(130, 100, 160, 100);  
Line2D.Double bottomLine  
    = new Line2D.Double(130, 160, 160, 160);
```

Ricevere testo in input

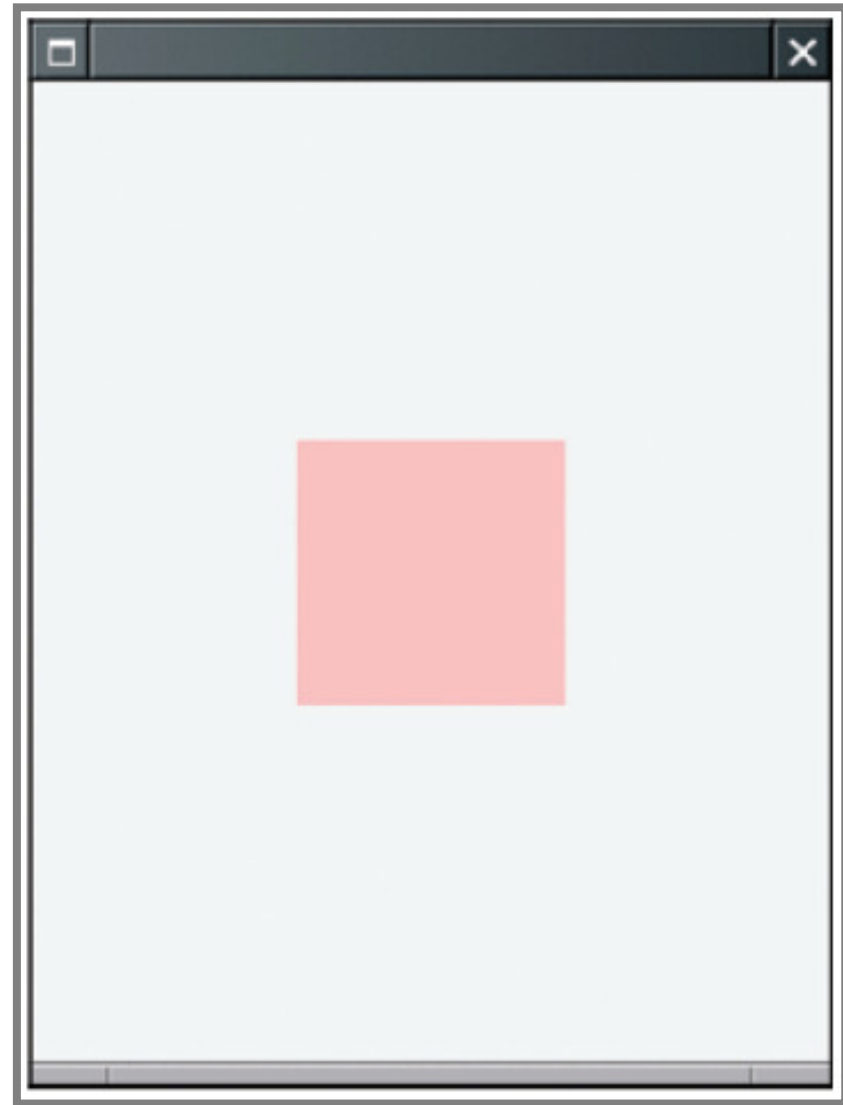
- Un'applicazione grafica può ricevere testo in input
 - classe `JOptionPane`
- Il metodo `showInputDialog`
 - visualizza un prompt e attende l'input dall'utente
 - restituisce la stringa digitata dall'utente

```
String input = JOptionPane.showInputDialog("Enter x");  
double x = Double.parseDouble(input);
```



Esercizio

**Si vuole generare
un quadrato del
colore specificato
dall'utente al
centro del frame**



File ColorViewer.java

```
01: import java.awt.Color;
02: import javax.swing.JFrame;
03: import javax.swing.JOptionPane;
04:
05: public class ColorViewer
06: {
07:     public static void main(String[] args)
08:     {
09:         JFrame frame = new JFrame();
10:
11:         final int FRAME_WIDTH = 300;
12:         final int FRAME_HEIGHT = 400;
13:
14:         frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
15:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16:
17:         String input;
18:
```

File ColorViewer.java

```
19:         // Ask the user for red, green, blue values
20:
21:         input = JOptionPane.showInputDialog("red:");
22:         double red = Double.parseDouble(input);
23:
24:         input = JOptionPane.showInputDialog("green:");
25:         double green = Double.parseDouble(input);
26:
27:         input = JOptionPane.showInputDialog("blue:");
28:         double blue = Double.parseDouble(input);
29:
30:         Color fillColor = new Color(
31:             (float) red, (float) green, (float) blue);
32:         ColoredSquareComponent component
33:             = new ColoredSquareComponent(fillColor);
34:         frame.add(component);
35:
36:         frame.setVisible(true);
37:     }
38: }
```

File ColoredSquareComponent.java

```
01: import java.awt.Color;
02: import java.awt.Graphics;
03: import java.awt.Graphics2D;
04: import java.awt.Rectangle;
05: import javax.swing.JComponent;
06:
07: /**
08:     A component that shows a colored square.
09: */
10: public class ColoredSquareComponent extends JComponent
11: {
12:     /**
13:         Constructs a component that shows a colored square.
14:         @param aColor the fill color for the square
15:     */
16:     public ColoredSquareComponent(Color aColor)
```

File ColoredSquareComponent.java

```
17:     {
18:         fillColor = aColor;
19:     }
20:
21:     public void paintComponent(Graphics g)
22:     {
23:         Graphics2D g2 = (Graphics2D) g;
24:
25:         // Select color into graphics context
26:
27:         g2.setColor(fillColor);
28:
29:         // Construct and fill a square whose center is
30:         // the center of the window
31:
```

File ColoredSquareComponent.java

```
32:         final int SQUARE_LENGTH = 100;
33:
34:         Rectangle square = new Rectangle(
35:             (getWidth() - SQUARE_LENGTH) / 2,
36:             (getHeight() - SQUARE_LENGTH) / 2,
37:             SQUARE_LENGTH,
38:             SQUARE_LENGTH);
39:
40:         g2.fill(square);
41:     }
42:
43:     private Color fillColor;
44: }
```

Informazione visuale e numerica

- Calcola l'intersezione tra un cerchio e una linea verticale
 - Il cerchio ha raggio $r = 100$ e centro $(a, b) = (100, 100)$
 - La linea ha un valore costante x
-

Informazione visuale e numerica

- Calcola i punti di intersezione:

Equazione di un cerchio di raggio r e centro (a, b)

$$(x - a)^2 + (y - b)^2 = r^2$$

Se x è nota, possiamo calcolare y :

$$(y - b)^2 = r^2 - (x - a)^2$$
$$y - b = \pm \sqrt{r^2 - (x - a)^2}$$
$$y = b \pm \sqrt{r^2 - (x - a)^2}$$

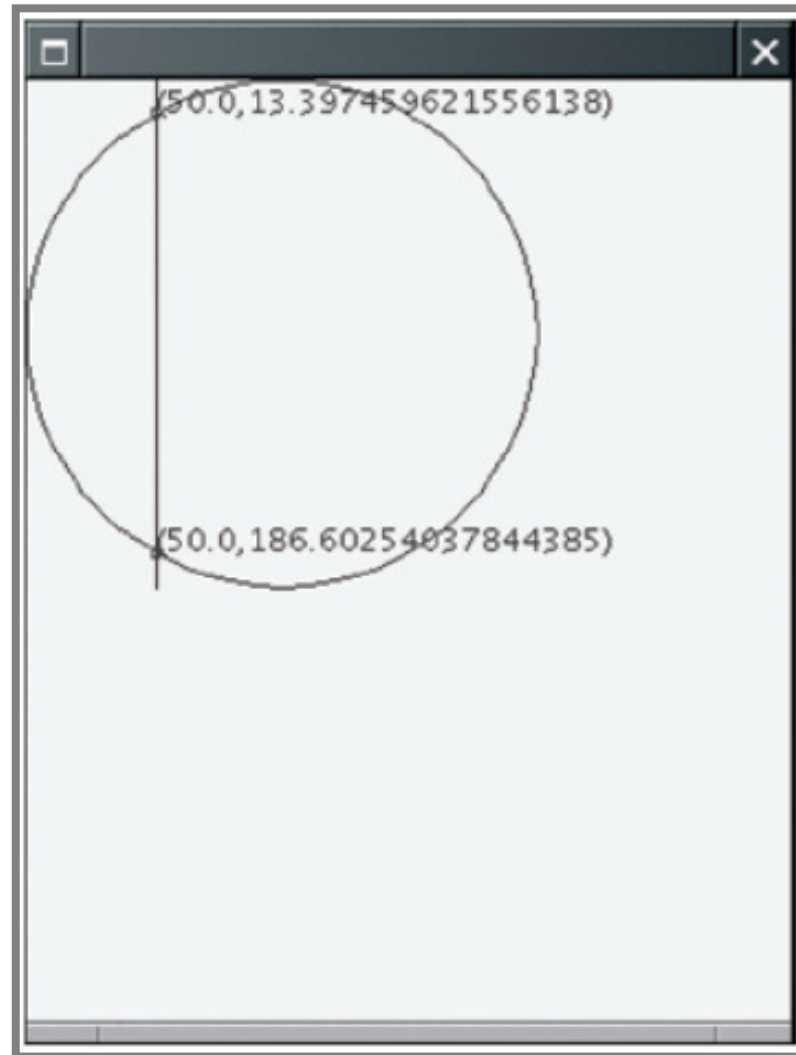
Informazione visuale e numerica

- Si può calcolare facilmente in Java:

```
double root = Math.sqrt(r * r - (x - a) * (x - a));  
double y1 = b + root;  
double y2 = b - root;
```

- Traccia il cerchio, la linea e i punti di intersezione calcolati
 - I risultati visuali e numerici dovrebbero essere consistenti
-

Intersezione di una linea e un cerchio



File IntersectionViewer.java

```
01: import javax.swing.JFrame;
02: import javax.swing.JOptionPane;
03:
04: public class IntersectionViewer
05: {
06:     public static void main(String[] args)
07:     {
08:         JFrame frame = new JFrame();
09:
10:         final int FRAME_WIDTH = 300;
11:         final int FRAME_HEIGHT = 400;
12:
13:         frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
14:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15:
```

File IntersectionViewer.java

```
16:     String input = JOptionPane.showInputDialog("Enter x");
17:     double x = Double.parseDouble(input);
18:     IntersectionComponent component
19:         = new IntersectionComponent(x);
20:     frame.add(component);
21:
22:     frame.setVisible(true);
23: }
24: }
```

File IntersectionComponent.java

```
01: import java.awt.Graphics;
02: import java.awt.Graphics2D;
03: import java.awt.geom.Ellipse2D;
04: import java.awt.geom.Line2D;
05: import javax.swing.JComponent;
06:
07: /**
08:     A component that computes and draws the intersection points
09:     of a circle and a line.
10: */
11: public class IntersectionComponent extends JComponent
12: {
13:     /**
14:         Constructs the component from a given x-value for the line
15:         @param anX the x-value for the line (between 0 and 200)
16:     */
```

File IntersectionComponent.java

```
17:     public IntersectionComponent (double anX)
18:     {
19:         x = anX;
20:     }
21:
22:     public void paintComponent (Graphics g)
23:     {
24:         Graphics2D g2 = (Graphics2D) g;
25:
26:         // Draw the circle
27:
28:         final double RADIUS = 100;
29:
30:         Ellipse2D.Double circle
31:             = new Ellipse2D.Double (0, 0, 2 * RADIUS, 2 * RADIUS);
32:         g2.draw(circle);
33:
34:         // Draw the vertical line
35:
```

File IntersectionComponent.java

```
36:         Line2D.Double line
37:             = new Line2D.Double(x, 0, x, 2 * RADIUS);
38:         g2.draw(line);
39:
40:         // Compute the intersection points
41:         double a = RADIUS;
42:         double b = RADIUS;
43:
44:         double root = Math.sqrt(RADIUS * RADIUS - (x - * (x - a));
45:         double y1 = b + root;
46:         double y2 = b - root;
47:
48:         // Draw the intersection points
49:         LabeledPoint p1 = new LabeledPoint(x, y1);
50:         LabeledPoint p2 = new LabeledPoint(x, y2);
51:
52:         p1.disegna(g2);
53:         p2.disegna(g2);
54:     }
55:
56:     private double x;
57: }
```

File LabeledPoint.java

```
01: import java.awt.Graphics2D;
02: import java.awt.geom.Ellipse2D;
03:
04: /**
05:     A point with a label showing the point's coordinates.
06: */
07: public class LabeledPoint
08: {
09:     /**
10:         Construct a labeled point.
11:         @param anX the x coordinate
12:         @param aY the y coordinate
13:     */
14:     public LabeledPoint(double anX, double aY)
15:     {
16:         x = anX;
17:         y = aY;
18:     }
```

File LabeledPoint.java

```
19:
20:     /**
21:         Draws the point as a small circle with a coordinate label.
22:         @param g2 the graphics context
23:     */
24:     public void disegna(Graphics2D g2)
25:     {
26:         // Draw a small circle centered around (x, y)
27:
28:         Ellipse2D.Double circle = new Ellipse2D.Double(
29:             x - SMALL_CIRCLE_RADIUS,
30:             y - SMALL_CIRCLE_RADIUS,
31:             2 * SMALL_CIRCLE_RADIUS,
32:             2 * SMALL_CIRCLE_RADIUS);
33:
34:         g2.draw(circle);
35:
36:         // Draw the label
```

File LabeledPoint.java

```
37:
38:     String label = "(" + x + "," + y + ")";
39:
40:     g2.drawString(label, (float) x, (float) y);
41: }
42:
43:     private static final double SMALL_CIRCLE_RADIUS = 2;
44:
45:     private double x;
46:     private double y;
47: }
```
