Programmazione II

Esercitazione 03: Array di figure

Alessandro Mazzei

Slides Credits: dispensa 'Sviluppo di Interfacce Grafiche in Java. Concetti di Base ed Esempi', prodotta dai Proff. M. de Leoni, M. Mecella, S. Saltarelli

Slides Credits: Attilio Fiandrotti
Daniele Radicioni

breve sintesi dei temi oggetto dell’esercitazione

- AWT
- Swing
- JFrame
- Static vs. dynamic binding
- AWT + Swing

- AWT
- Swing
- JFrame
- Static vs. dynamic binding
- AWT + Swing

AWT: abstract window toolkit

- Windowing toolkit grafico originale Java (1995)
- Contiene funzioni per (elenco non esaustivo):
  1. Disegnare primitive grafiche a schermo (graphical primitives);
  2. Reagire ad azioni sulla finestra (resize, drag, ...) di una finestra o all'interno della finestra stessa (mouseclick, keystroke, ...) (event handling);
  3. Costruire un'interfaccia grafica con elementi annidati gerarchicamente (toolkit / widgets).

AWT: abstract window toolkit

1. Disegno di primitive grafiche via *java.awt.Graphics*

- Primitive grafiche come metodi
  - `drawLine (int x1, int y1, int x2, int y2)`
  - `drawRect (int x, int y, int width, int height)`
  - `drawString (String str, int x, int y)`
  - `drawOval (int x, int y,  int a_x, int a_y);`
  - `setColor (Color col)`
  - `...`
- Tipicamente invocate nel metodo paint(Graphics g)

java.awt

Class Graphics

java.lang.Object
    java.awt.Graphics

https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html

AWT: abstract window toolkit

Oggi utilizziamo principalmente questi metodi

1. Disegno di primitive grafiche via *java.awt.Graphics*

- Primitive grafiche come metodi
  - `drawLine (int x1, int y1, int x2, int y2)`
  - `drawRect (int x, int y, int width, int height)`
  - `drawString (String str, int x, int y)`
  - `drawOval (int x, int y,  int a_x, int a_y);`
  - `setColor (Color col)`
  - `...`
- Tipicamente invocate nel metodo paint(Graphics g)

java.awt

Class Graphics

java.lang.Object
    java.awt.Graphics

https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html

AWT: abstract window toolkit

2. Gestione eventi finestre via callback functions

- L'interfaccia *WindowListener* consiste in:
  - `windowActivated(WindowEvent e)`
  - `windowDeactivated(WindowEvent e)`
  - `windowOpened(WindowEvent e)`
  - `windowClosing(WindowEvent e)`
  - `windowClosed(WindowEvent e)`
  - `windowIconified(WindowEvent e)`
  - `windowDeiconified(WindowEvent e)`

https://docs.oracle.com/javase/7/docs/api/java/awt/event/WindowListener.html

AWT: abstract window toolkit

Oggi utilizziamo anche questi metodi

2. Gestione eventi finestre via callback functions

- L'interfaccia *WindowListener* consiste in:
  - `windowActivated(WindowEvent e)`
  - `windowDeactivated(WindowEvent e)`
  - `windowOpened(WindowEvent e)`
  - `windowClosing(WindowEvent e)`
  - `windowClosed(WindowEvent e)`
  - `windowIconified(WindowEvent e)`
  - `windowDeiconified(WindowEvent e)`


java.awt.event

Interface WindowListener

All Superinterfaces:

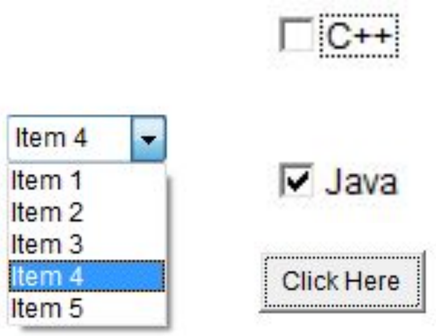
EventListener

https://docs.oracle.com/javase/7/docs/api/java/awt/event/WindowListener.html

AWT: abstract window toolkit

3. Fornisce i widgets per costruire GUIs

- Alcune estensioni di *java.awt.Component*
  - java.awt.Button
  - java.awt.Checkbox
  - java.awt.Label
  - java.awt.Choice



java.awt

Class Component

java.lang.Object
    java.awt.Component

All Implemented Interfaces:

ImageObserver, MenuContainer, Serializable

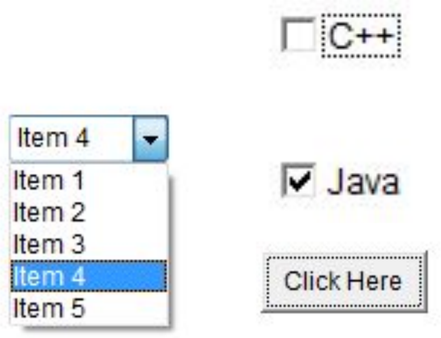
https://docs.oracle.com/javase/7/docs/api/java/awt/event/WindowListener.html

AWT: abstract window toolkit

Oggi utilizziamo anche questi metodi

3. Fornisce i widgets per costruire GUIs

- Alcune estensioni di *java.awt.Component*
  - java.awt.Button
  - java.awt.Checkbox
  - java.awt.Label
  - java.awt.Choice



java.awt

Class Component

java.lang.Object
    java.awt.Component

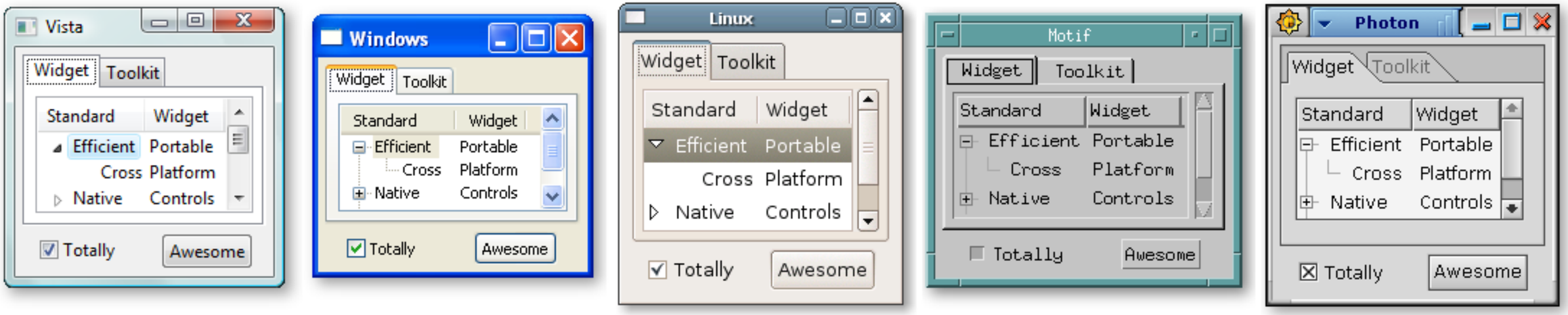
All Implemented Interfaces:

ImageObserver, MenuContainer, Serializable

https://docs.oracle.com/javase/7/docs/api/java/awt/event/WindowListener.html

AWT: abstract window toolkit

- AWT utilizza il toolkit del sistema host per implementare i widgets
  - lo stesso codice produrrà risultati diversi a seconda del sistema host (significa che le classi che mappano i componenti grafici si appoggiavano alle chiamate del sistema operativo)
- scarsa portabilità. appoggiandosi al sistema operativo, l'aspetto grafico può variare sensibilmente in funzione della piattaforma su cui gira la JVM.
  - limitazione legata al fatto che il set di componenti grafici AWT comprende solamente quel limitato insieme di controlli grafici che costituiscono il minimo comune denominatore tra tutti i sistemi a finestre esistenti



- AWT
- Swing
- JFrame
- Static vs. dynamic binding
- AWT + Swing

JAVA Swing

- Introdotto in Java SE 1.2 (1998 circa)
- Implementazione dei widget nativa in linguaggio JAVA
  - stesso risultato su piattaforme diverse (o quasi)
  - altre opzioni oggi possibili per implementare GUIs in Java

https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html

swing

- Gli oggetti grafici Swing derivano da quelli AWT
- JComponent eredita da Container, un contenitore che offre la possibilità di disporre altri componenti all'interno mediante il metodo *void add(Component)*.

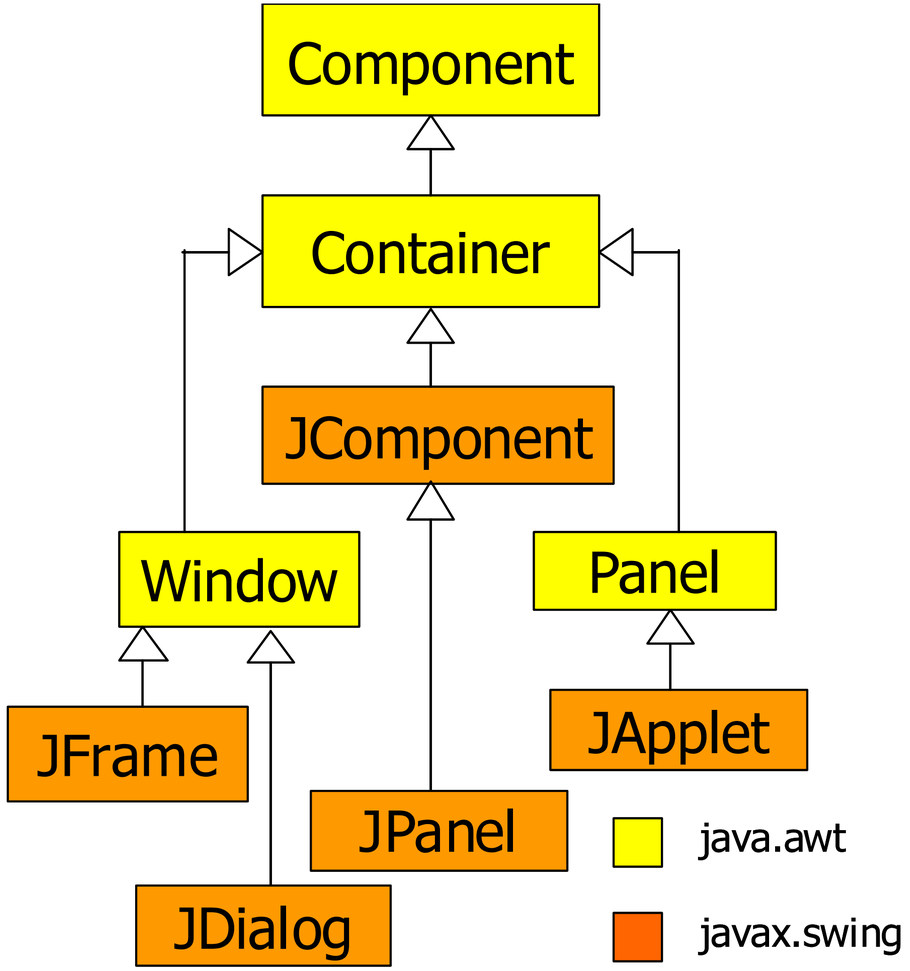


Diagramma UML di base del package Swing.

- AWT
- Swing
- JFrame
- Static vs. dynamic binding
- AWT + Swing

JFrame

- Due importanti proprietà dell'oggetto sono la dimensione e la posizione, che possono essere impostate sia specificando le singole componenti sia mediante oggetti Dimension e Point del package AWT
  - `public void setSize(Dimension d);`
  - `public void setSize(int width,int height);`
  - `public void setLocation(Point p);`
  - `public void setLocation(int x,int y);`

JFrame

- tre metodi importanti di *JFrame* sono:
  - `public void pack();` ridimensiona la finestra tenendo conto delle dimensioni ottimali di ciascuno dei componenti presenti all'interno.
  - `public void setVisible(boolean b);` permette di visualizzare o di nascondere la finestra.
  - `public void setDefaultCloseOperation(int operation);` imposta l'azione da eseguire alla pressione del bottone close, con quattro impostazioni disponibili: JFrame.DO NOTHING ON CLOSE (nessun effetto), JFrame.HIDE ON CLOSE (nasconde la finestra), JFrame.DISPOSE ON CLOSE (chiude la finestra e libera le risorse di sistema) e JFrame.EXIT ON CLOSE (chiude la finestra e conclude l'esecuzione del programma).

primo swing

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


public class PrimaGui {
  public static void main(String args[]) {
    JFrame window = new JFrame("prima finestra"); // nome finestra

    Container c = window.getContentPane();
    c.add(new JLabel("Esercitaz di Prog2")); // testo contenuto nella finestra
    window.setSize(200, 200); // settiamo dimensione

    // comportamento alla chiuisura della finestra;: il processo deve terminare
    window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    window.setVisible(true);
  }
}
```

gestore di layout

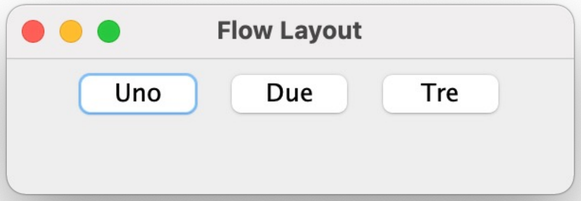
- per rendere interattiva la finestra di una applicazione è necessario predisporre vari tipi di elementi per l'interazione: questi elementi sono gestiti tramite layout manager.

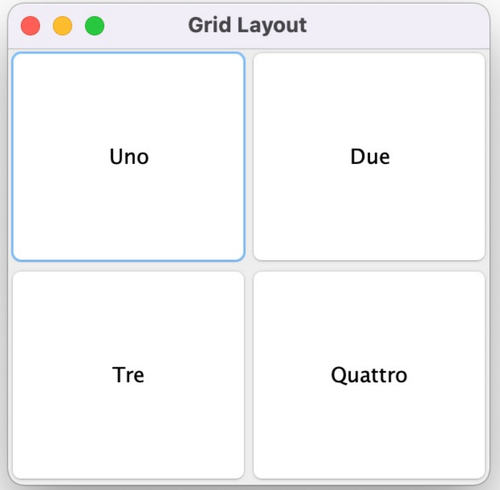
gestore di layout

- in molti Container i controlli sono posizionati linearmente da sinistra verso destra, ma questa modalità può non essere soddisfacente in alcun casi.
- *Layout Manager* è una politica di posizionamento dei componenti in un *Container*.
  - Un gestore di layout è una qualsiasi classe che implementa *LayoutManager* (interfaccia definita in *java.awt*)
  - il Gestore di Layout viene chiamato quando bisogna dimensionare un Container la prima volta, e successivamente quando si cerca di ridimensionare il Container.
- esistono finestre (la maggior parte) che richiedono di comporre più layout e Container:
  - in questo caso si montano più Container, ciascuno dotato del proprio layout (della propria politica di posizionamento degli elementi interni)

esplorazione dei layout

- vediamo i layout seguenti:
  - *FlowLayout*
  - *GridLayout*
  - un layout misto, contenente 3 pannelli (*JPanel*) in cui posizioniamo diversi elementi:
    - *JLabel*
    - *JCheckBox*
    - *JButton*





primo swing

- gli esempi di codice per confrontare diversi gestori di layout sono nella directory `primoSwing/`, fra il materiale della lezione

- AWT
- Swing
- JFrame
- Static vs. dynamic binding
- AWT + Swing

static vs. dynamic binding

- binding = associazione di chiamata di metodo alla definizione del metodo stesso
  - se questa associazione avviene durante la compilazione, allora parliamo di static binding (o early binding)
  - se questa associazione avviene a tempo di esecuzione, allora parliamo di dynamic binding (o late binding)

static binding

- il binding di metodi static, private, e final è condotto a tempo di compilazione
  - perché il compilatore è in grado di determinare il tipo della classe
  - a runtime non possiamo quindi sovrascrivere (in inglese: *override*) questi metodi oggetto di static binding
  - in generale static binding garantisce performance migliori

static binding: esempio

```
class Persona {
  public static void parla() {System.out.println("Persona parla");}
}
class Docente extends Persona {
  public static void parla() {System.out.println("Docente parla");}
}

public class StaticBinding {
  public static void main(String args[]) {
    Persona andrea = new Persona();
    Persona daniele = new Docente();

    andrea.parla();  // tipo Persona
    daniele.parla(); // tipo Docente
  }
}
```

```
$ java StaticBinding
Persona parla
Persona parla
$
```

dynamic binding

- overriding del metodo avviene a runtime (se super/sottoclasse hanno un metodo con stessa signature, ovviamente)
- avviene quando i metodi non sono dichiarati static, private, e final

dynamic binding: esempio

```
class Persona {
  public void parla() {System.out.println("Persona parla");}
}
class Docente extends Persona {
  public void parla() {System.out.println("Docente parla");}
}

public class StaticBinding {
  public static void main(String args[]) {
    Persona andrea = new Persona();
    Persona daniele = new Docente();

    andrea.parla();  // tipo Persona
    daniele.parla(); // tipo Docente
  }
}
```

```
$ java DynamicBinding
Persona parla
Docente parla
$
```

staticDynamicBinding

- gli esempi di codice per testare gli esempi illustrati si trovano nella directory `staticDynamicBinding/`, fra il materiale della lezione

dalla lezione 13 (teoria)

- AWT
- Swing
- JFrame
- Static vs. dynamic binding
- AWT + Swing

AWT + Swing

- È possibile combinare AWT e Swing
- A lezione è stato introdotto *javax.swing.JFrame*
  - estende java.awt.Frame
- È una 'tavolozza' per elementi grafici
- Il render a schermo è effettuato tramite
  public void paint(Graphics g)

Lezione 13. Ereditarietà e binding dinamico

- Java contiene la classe *Graphics* degli oggetti grafici:
  - un oggetto grafico è un'area rettangolare dello schermo in cui sono disponibili dei metodi per disegnare.
- a partire da Graphics, definiamo una classe `Figura` di figure, ciascuna con un suo metodo *draw(Graphics g)* che disegna la figura in un oggetto grafico *g*.
  - definiamo un metodo *draw* vuoto per una figura generica, quindi usando l'ereditarietà ri-definiamo *draw* ogni volta che definiamo una sottoclasse di Figure.

Lezione 13. Ereditarietà e binding dinamico

- A partire da `Figura`, definiamo la classe `Disegno` delle finestre (cioè oggetti della classe *JFrame*) con incluso un array di Figure.
  - Utilizziamo un metodo void *paint(Graphics g)* che prende in ingresso in un oggetto grafico *g* posto nella finestra, e disegna tutte le figure dell'array in *g* usando *draw*.
  - Alla fine invochiamo su un disegno un metodo di libreria *setVisible(true)*, che fornisce un oggetto grafico *g* al metodo *paint* e genera il disegno.
  - Non è possibile definire *g* noi stessi: il costruttore della classe Graphics è protected, dunque inaccessibile in Disegno e Figure, che non sono sottoclassi di Graphics e non fanno parte della stessa cartella.

Figura

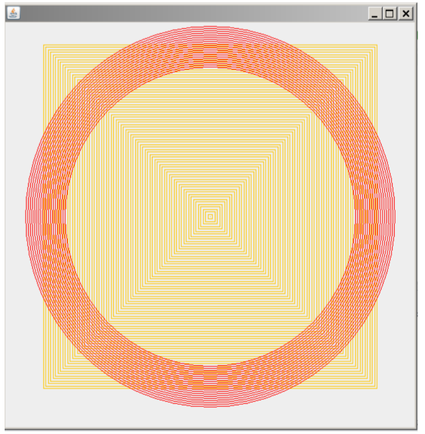
Cerchio

Quadrato

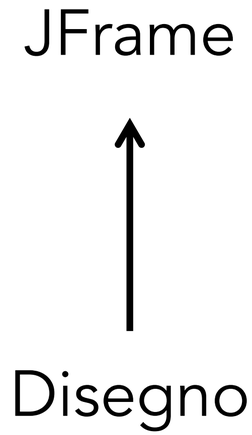
```
public void draw(Graphics g){ /* vuoto */ }
```

```
public void draw(Graphics g) {
  g.setColor(Color.red);
  g.drawOval(-raggio,-raggio,  2*raggio,2*raggio);
}
```

```
public void draw(Graphics g) {
   g.setColor(Color.orange);
   int m = lato / 2;
   g.drawLine( m,  m, -m,  m);
   g.drawLine(-m,  m, -m, -m);
   g.drawLine(-m, -m,  m, -m);
   g.drawLine( m, -m,  m,  m);
}
```



```
public class Disegno extends JFrame{
    private Figura[] figure;
    [...]
}
```

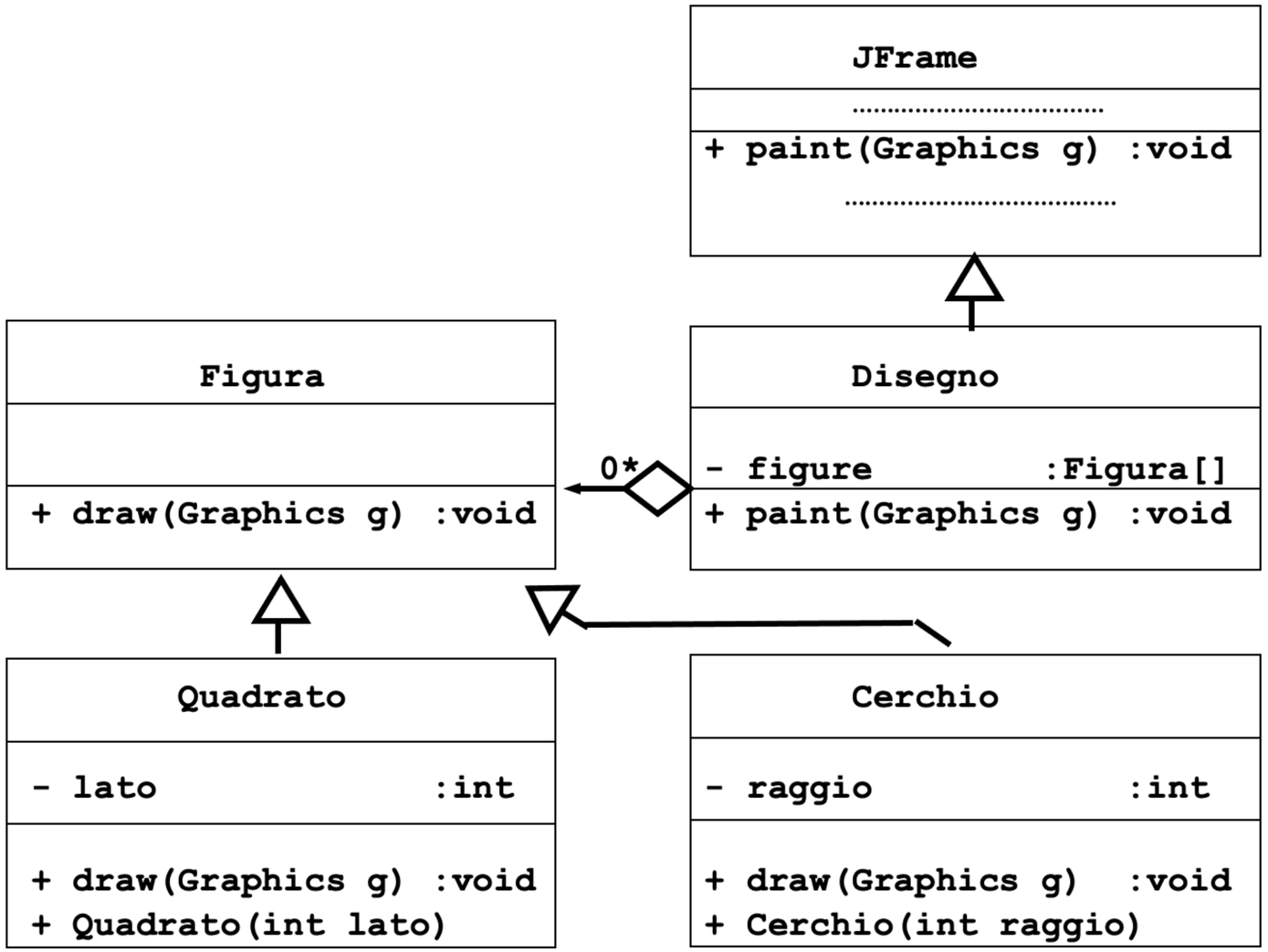



```
public void paint(Graphics g) {
    [...]
    //DISEGNO tutte le figure dell'array figure
    for(int i=0;i<figure.length;++i) {
        figure[i].draw(g);
    }
}

public static void main(String[] args){
  Figura[] figure = new Figura[n];
  [...]
  Disegno frame = new Disegno(figure)
  [...]
  frame.setVisible(true);
}
```

Diagramma UML della classe Disegno

(omettiamo di indicare il main di Disegno)



Fine