

Programmazione Orientata agli Oggetti

Esercitazione:
Interfacce, Polimorfismo

Esercitazione **FormeGeometriche**

(TRATTO DALL'ESAME DEL GIUGNO 2003)

- Una software house sta sviluppando una libreria per la gestione di forme geometriche. Allo stato attuale nella libreria ci sono le classi **Punto**, **Cerchio** e **Rettangolo** (vedi codice)
- Sono già date le seguenti classi:
 - **Punto**
 - **Cerchio**
 - **Rettangolo**

La Classe Punto

```
public class Punto {  
    private int x,y;  
  
    public Punto (int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public void setX(int x){  
        this.x = x;  
    }  
  
    public void setY(int y){  
        this.y = y;  
    }  
  
    public int getX(){  
        return this.x;  
    }  
  
    public int getY(){  
        return this.y;  
    }  
}
```

La Classe Cerchio

```
public class Cerchio {  
    private int raggio;  
    private Punto centro;  
  
    public Cerchio(Punto centro, int raggio) {  
        this.raggio = raggio;  
        this.centro = new Punto(centro.getX(), centro.getY());  
    }  
  
    public void trasla(int deltaX, int deltaY) {  
        this.centro.setX(this.centro.getX() + deltaX);  
        this.centro.setY(this.centro.getY() + deltaY);  
    }  
  
    public Punto getCentro() { return this.centro; }  
  
    public int getRaggio() { return this.raggio; }  
  
}
```

La Classe Rettangolo

```
public class Rettangolo {  
  
    private int altezza, base;  
    private Punto vertice;  
  
    public Rettangolo(Punto vertice, int altezza, int base) {  
        this.altezza = altezza;  
        this.base = base;  
        this.vertice = new Punto(vertice.getX(), vertice.getY());  
    }  
  
    public void trasla(int deltaX, int deltaY) {  
        this.vertice.setX(this.vertice.getX() + deltaX);  
        this.vertice.setY(this.vertice.getY() + deltaY);  
    }  
  
    public Punto getVertice() { return this.vertice; }  
  
    public int getBase()      { return this.base;    }  
  
    public int getAltezza()   { return this.altezza; }  
  
}
```

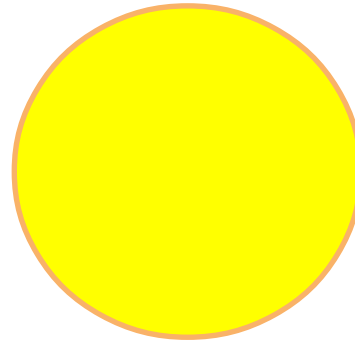
Esercizio 1: Polimorfismo

- Si vuole introdurre una classe **GruppoDiForme** che rappresenta un raggruppamento di forme. In particolare, le forme di un raggruppamento possono essere rettangoli, cerchi e ***altri raggruppamenti***. Un gruppo di forme può essere traslato (vengono traslate tutte le forme che lo compongono)
- La classe **GruppoDiForme** deve offrire i metodi:
 - **`void trasla(int deltaX, int deltaY)`**
trasla tutto il raggruppamento (cioè tutti gli oggetti che compongono il raggruppamento)
 - **`void aggiungiForma(Forma forma)`**
aggiunge una forma al raggruppamento

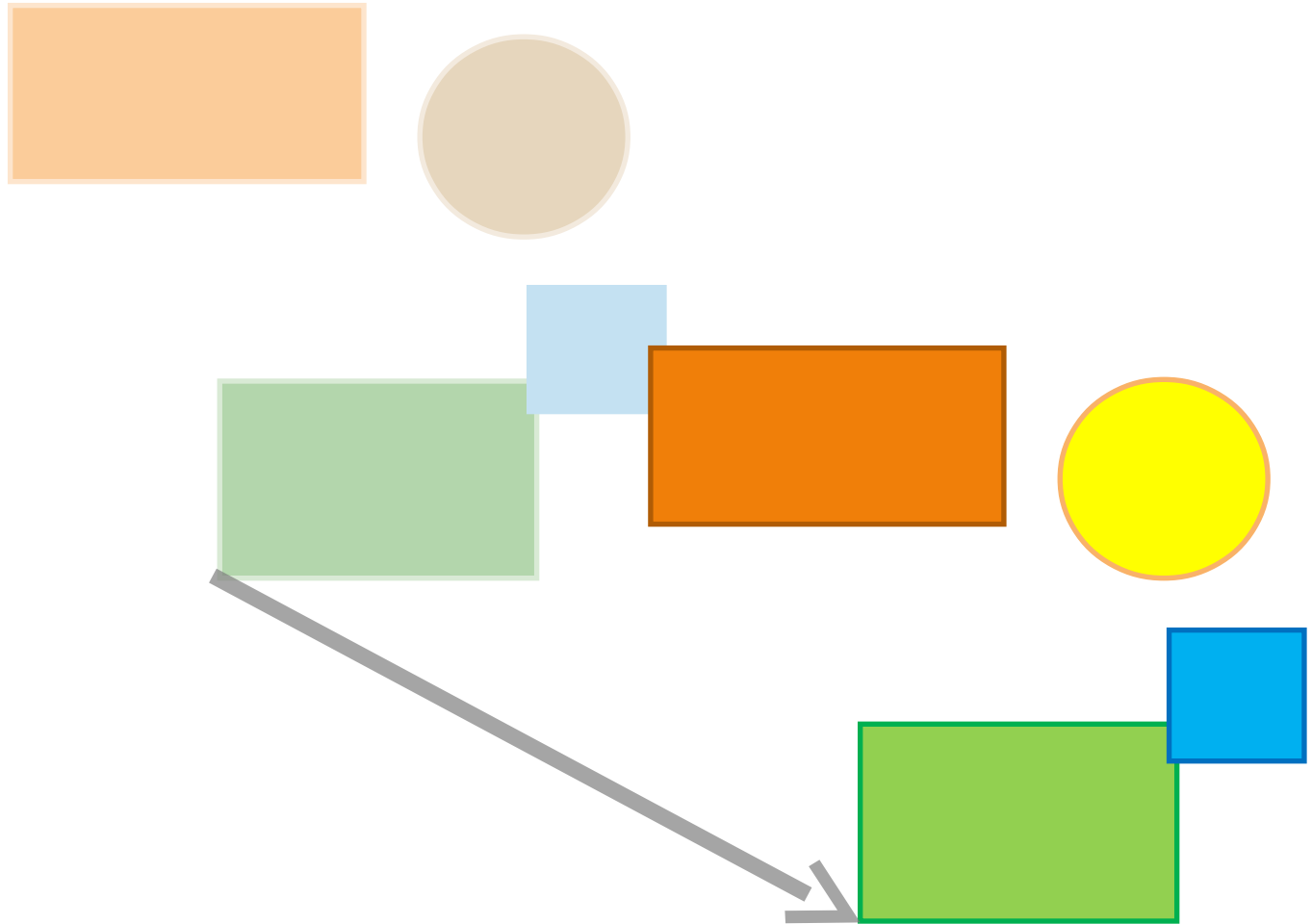
Esercizio 1: Polimorfismo (Continua)

- **Suggerimento:** astrarre i concetti di forma geometrica (rettangolo, cerchio, gruppo) in una interfaccia **Forma**. Quindi, nell'ordine:
 1. Scrivere l'interfaccia **Forma**
 2. Rendere **Cerchio** e **Rettangolo** specializzazioni di **Forma**
 3. Scrivere le classi di test **CerchioTest** e **RettangoloTest**
 4. Scrivere la classe **GruppoDiForme**: Un gruppo di forme è composto da un array di riferimenti a oggetti che implementano **Forma**. Per semplicità si supponga che un gruppo di forme possa essere composto al massimo da 10 forme
 5. Scrivere la classe **GruppoDiFormeTest**

Un GruppoDiForme



Un GruppoDiForme *traslato()*



Esercizio 2: Testing sulle Forme Semplici

- Utilizzando JUnit creare classi di test per **Cerchio** e **Rettangolo**.
- Aggiungere una serie di test-case *minimali* relativi al metodo **trasla()**
- Ad esempio, un primo test-case **testTrasla()** di **Cerchio**:
 - istanzia un cerchio unitario ($r=1$) di centro sull'origine (0,0)
 - trasla di (+0, +0)
 - asserisce che dopo la traslazione il cerchio non si è spostato
- Un secondo test-case potrebbe traslare di (+1,+0)
- Un terzo test-case potrebbe ...

Esercizio 3: Testing su GruppoDiForme

- Creare una classe di test per **GruppoDiForme** e scrivere test-case *minimali* del metodo `trasla()` che eseguano le seguenti istruzioni
- Distinguere almeno questi scenari di testing a complessità crescente, nell'ordine:
 - un gruppo *vuoto*
 - un gruppo *semplice*, con una sola forma non ulteriormente decomponibile
 - un gruppo *composito*, ovvero di un gruppo contenente un gruppo semplice
 - un gruppo *complesso*, ovvero di un gruppo contenente un gruppo composito
- Scrivere diversi test-case *minimali* per ciascun scenario

Esercizio 4: Downcasting per `assertEquals()` nel Metodo `equals()`

- Il metodo (di JUnit) `assertEquals()` si basa sull'esecuzione del metodo `equals()` sugli oggetti passati come argomento
- La segnatura esatta del metodo `equals` DEVE essere:
@Override

```
public boolean equals(Object o)
```
- Per accorciare i test-case già prodotti:
 - munire la classe `Punto` di un metodo `equals()`
 - Oggetti istanza della classe `Punto` distinti ma di pari coordinate uguali sono considerati equivalenti
 - assicurarsi che i test usino `assertEquals()` passandogli oggetti istanza di `Punto`
 - controllare che risulti effettivamente invocato il metodo `Punto.equals(Object o)`