Le Interfacce

Programmazione

Corso di Laurea in Informatica
Corso di Laurea in Informatica per la Comunicazione Digitale

Università di Camerino

Prof. Michele Loreti

Anno Accademico 2024/25

Motivazione

Problema

L'ereditarietà (in Java) impone una organizzazione gerarchica delle classi ad **albero**, per cui ogni classe può avere un numero arbitrario di sotto classi ma **al massimo una super classe** (`Object` è l'unica classe senza super classe). Questa organizzazione non sempre (quasi mai...) rispecchia tutte le relazioni che si vogliono esprimere tra classi diverse

Soluzione

Si introduce il concetto di **interfaccia**: una interfaccia definisce un **tipo** e un **insieme di metodi**. Ogni classe che **implementa** l'interfaccia fornisce (almeno) quell'insieme di metodi e dunque è sottotipo dell'interfaccia. L'implementazione di interfacce garantisce maggiore flessibilità rispetto all'estensione tra classi perché non deve necessariamente rispettare l'organizzazione gerarchica

Esempio: gerarchia di figure geometriche

Command Not Found: mmdc

Data la gerarchia di figure geometriche mostrata in figura scrivere un metodo che, date due figure dotate di raggio, stabilisca il raggio maggiore tra i due

Calcolare il raggio massimo

```
double raggio_massimo(Cerchio a, Cerchio b) {
    return Math.max(a.get_raggio(), b.get_raggio());
}

double raggio_massimo(Cerchio a, PoligonoRegolare b) {
    return Math.max(a.get_raggio(), b.get_raggio());
}

double raggio_massimo(PoligonoRegolare a, PoligonoRegolare b) {
    return Math.max(a.get_raggio(), b.get_raggio());
}
```

Note (dolenti)

- servono **tre versioni** del metodo `raggio_massimo` perché le classi che rappresentano figure dotate di raggio, ovvero `Cerchio` e `PoligonoRegolare`, non sono direttamente in relazione tra loro
  (`Cerchio` non è sotto classe di `PoligonoRegolare` né `PoligonoRegolare` è sotto classe di `Cerchio`)
- non c'è un tipo che rappresenti le figure dotate di raggio

(Non) soluzione con gerarchia alternativa

Command Not Found: mmdc

- introduciamo una **classe intermedia** `FiguraConRaggio` e dunque un tipo che rappresenta la famiglia di figure dotate di raggio
- c'è una sola classe in cui è definito il campo `raggio` ed il relativo metodo getter
- è possibile scrivere un solo metodo `raggio_massimo`
- **`Cerchio` non è più in relazione con `Ellisse`**

```
double raggio_massimo(FiguraConRaggio a, FiguraConRaggio b)
{ return Math.max(a.get_raggio(), b.get_raggio()); }
```

Morale: non è possibile avere una gerarchia in cui `Cerchio <: Ellisse` e `Cerchio` e `PoligonoRegolare` hanno un antenato comune con raggio

Soluzione con interfaccia

Command Not Found: mmdc

- introduciamo una **interfaccia** `FiguraConRaggio` e dunque un tipo che rappresenta la famiglia di figure dotate di raggio
- il metodo `get_raggio` è definito sia in `Cerchio` che in `PoligonoRegolare`
- `Cerchio` e `PoligonoRegolare` sono entrambe sottotipo di `FiguraConRaggio`, dunque è possibile sfruttare il principio di sostituzione per scrivere un solo metodo `raggio_massimo`
- `Cerchio` è sottotipo di `Ellisse`, come nella prima gerarchia

```
double raggio_massimo(FiguraConRaggio a, FiguraConRaggio b) {
    return Math.max(a.get_raggio(), b.get_raggio());
}
```

Interfaccia `FiguraConRaggio`

`FiguraConRaggio.java`

```java
public interface FiguraConRaggio {
    public double get_raggio(); // non c'è il corpo del metodo!
}
```

Note

- Una **interfaccia** è un elenco di metodi (operazioni)
- Ogni classe che **implementa** l'interfaccia deve fornire una realizzazione di quei metodi

Classe `PoligonoRegolare`

```
public class PoligonoRegolare extends Figura
                              implements FiguraConRaggio {
    private int lati;
    private double raggio;

    public PoligonoRegolare(double x, double y, int lati,
          double raggio) {
  // lati > 2, 0 <= raggio
  super(x, y);
  this.lati = lati;
  this.raggio = raggio;
    }
    public double get_raggio() { return raggio; }
}
```

Note

- `PoligonoRegolare` **estende** `Figura`, dunque ne eredita campi e metodi
- `PoligonoRegolare` **implementa** `FiguraConRaggio`, dunque deve includere una realizzazione di *tutti* i metodi elencati in `FiguraConRaggio`

Classe `Cerchio`

```
public class Cerchio extends Ellisse
                     implements FiguraConRaggio {
    public Cerchio(double x, double y, double raggio) {
        super(x, y, raggio, raggio);
    }

    public double get_raggio() { return get_asse_minore(); }
}
```

Note

- `Cerchio` **estende** `Figura`, dunque ne eredita campi e metodi
- `Cerchio` **implementa** `FiguraConRaggio`, dunque deve includere una realizzazione di *tutti* i metodi elencati in `FiguraConRaggio`
- L'implementazione di `get_raggio` in `Cerchio` è *diversa* dall'implementazione di `get_raggio` in `PoligonoRegolare`
- Ogni classe che implementa `FiguraConRaggio` può fornire la propria realizzazione del metodo `get_raggio`

La relazione di sottotipo rivista

Definizione

La relazione di **sottotipo** `<:` è il più piccolo ordine parziale tale che X `<:` Y se

- X e Y sono classi e X **estende** Y
- X è una classe, Y è una interfaccia e X **implementa** Y
- X è una interfaccia e Y è `Object`

Esempi

- `Cerchio <: Ellisse`
- `Cerchio <: FiguraConRaggio`
- `PoligonoRegolare <: FiguraConRaggio`

Soluzione del problema

```
public class Main {
    public static double raggio_massimo(FiguraConRaggio a,
          FiguraConRaggio b) {
  return Math.max(a.get_raggio(), b.get_raggio());
    }

    public static void main(String[] args) {
  Cerchio a = new Cerchio(10, 5, 60);
  PoligonoRegolare b = new PoligonoRegolare(0, 3, 5, 50);
  System.out.println(raggio_massimo(a, b));
    }
}
```

Nota

- è possibile usare un `Cerchio` **oppure** un `PoligonoRegolare` laddove è attesa una `FiguraConRaggio`

Differenze tra classi e interfacce

- Una classe può **estendere al massimo un'altra classe**, ma può **implementare un numero arbitrario di interfacce**
- Se una classe dichiara di implementare un'interfaccia ma non realizza tutti i metodi dell'interfaccia, la classe è incompleta e dunque deve essere dichiarata astratta

In questo corso

- Una classe può contenere campi e metodi (possibilmente astratti)
- Un'interfaccia può contenere solo metodi (sempre astratti)

In Java

- È possibile avere metodi con implementazione in un'interfaccia, sono detti metodi "di default"
- È possibile dichiarare alcuni campi all'interno delle interfacce, ma questi sono soggetti a restrizioni
- C'è un meccanismo per **estendere interfacce**


TO
BE
CONTINUED!