Classi astratte

Classi astratte

Una classe astratta è una classe si distingue da una classe ordinaria perché:

- È dichiarata con il modificatore abstract
- Non può essere istanziata
- **Può** dichiarare metodi astratti (ovvero metodi dichiarati con il modificatore abstract)

Classi astratte

Per il resto è una classe come le altre e può dichiarare costruttori, metodi concreti, variabili, costanti, membri statici, tipi innestati, può implementare interfacce, estendere un'altra classe, essere estesa da altre classi e così via. Un esempio di classe astratta potrebbe essere il seguente:

```
public abstract class Veicolo {
    private String descrizione;
    public abstract void accelera();
    public abstract void decelera();
    // costruttori e metodi setter e getter omessi
}
```

Classi astratte

Notare che i metodi accelera e decelera sono dichiarati astratti. Infatti, stiamo definendo un concetto generico (un veicolo che si muove) non implementabile in maniera concreta (ogni veicolo si muove in maniera diversa).

Abbiamo detto che una classe astratta che dichiara almeno un metodo astratto, deve essere dichiarata astratta anch'essa. Questo è il caso standard, la classe `Veicolo` per esempio, è astratta perché non sappiamo come definire i suoi metodi accelera e decelera dato che non sappiamo di che veicolo stiamo parlando. Vien da sé che tale classe è stata definita per essere estesa da sottoclassi concrete, per esempio la classe `Auto`, la classe `Nave`, e la classe `Aereo`, ognuna delle quali fornirà un'implementazione diversa ai metodi accelera e decelera con degli override.

Classi astratte

L' utilizzo delle classi astratte rispetto alle interfacce ci darà la possibilità di trattare gli oggetti delle sottoclassi Auto, Nave, e Aereo, come fossero veicoli, e di quindi poter utilizzare il polimorfismo.

Non ha senso dichiarare una classe astratta se non si ha intenzione di estenderla. Inoltre, è fondamentale aver presente che le classi astratte hanno il compito di rappresentare concetti troppo generici per il contesto in cui si definiscono. In particolare, il modificatore abstract impone un importante vincolo progettuale: **la classe non si può direttamente istanziare**, anche se ha tutti i suoi metodi definiti.

Interfacce

- Si dichiara usando la parola chiave interface
- Non può essere istanziata
- Può estendere altre interfacce
- Una classe può implementare più interfacce

- Può dichiarare:
  - Metodi astratti pubblici (non è necessario usare i modificatori public e abstract che verranno aggiunti implicitamente dal compilatore)
  - Metodi di default pubblici, ovvero metodi concreti marcati con il modificatore default (non è necessario usare il modificatore public che verrà aggiunto implicitamente dal compilatore)
  - Metodi privati concreti (possono essere invocati solamente da metodi di default)
  - Metodi statici pubblici o privati (un metodo statico senza specificatori d’accesso sarà implicitamente considerato public dal compilatore)
  - Costanti statiche e pubbliche (non è necessario usare i modificatori public, final e static e che verranno aggiunti implicitamente dal compilatore)

Non è possibilie dichiarare altro

Interfacce

Un esempio di interfaccia può essere il seguente

```
public interface Pesabile {
    public static final String UNITA_DI_MISURA = "kg";
    public abstract double getPeso();
}
```

Che è in realtà equivalente a

```
public interface Pesabile {
    String UNITA_DI_MISURA = "kg";
    double getPeso();
}
```

Differenze

Una delle differenze più importanti e troppo spesso ignorata, è una differenza concettuale. Abbiamo detto che una classe astratta, deve definire un'astrazione troppo generica per essere istanziata nel contesto in cui si dichiara. Un buon esempio è la classe `Veicolo`

```
public abstract class Veicolo {
    private String descrizione;
    public abstract void accelera();
    public abstract void decelera();
}
```

Possiamo quindi estendere la classe `Veicolo` per esempio con la classe `Aereo`, che ovviamente avrà una propria implementazione dei metodi accelera e decelera

Differenze

Un’interfaccia dovrebbe astrarre un comportamento che più classi potrebbero implementare, e un comportamento non si dovrebbe istanziare. Infatti non dovrebbero esistere oggetti che rappresentano un comportamento. Spesso infatti le interfacce hanno nomi che richiamano aggettivi e comportamenti (Pesabile, Comparable, Cloneable, Nameable, Collectable ...)

Per esempio potremmo introdurre un’interfaccia Volante che sarà implementata dalle classi che rappresentano oggetti che volano (si noti come il nome faccia pensare ad un comportamento più che ad un oggetto astratto):

```
public interface Volante {
    void atterra();
    void decolla();
}
```

Differenze

ogni classe che deve astrarre un concetto di oggetto volante (come un aereo, un drone o anche un uccello), deve implementare l'interfaccia Volante. Riscriviamo quindi la classe Aereo nel seguente modo:

```
public class Aereo extends Veicolo implements Volante {
    public void atterra() {
        // override del metodo di Volante
    }
    public void decolla() {
        // override del metodo di Volante
    }
    public void accelera() {
        // override del metodo di Veicolo
    }
    public void decelera() {
        // override del metodo di Veicolo
    }
}
```

Differenze

Potremmo quindi creare parametri polimorfi per sfruttare l'interfaccia Volante:

```
public class TorreDiControllo {
    public void autorizzaAtterraggio(Volante v) {
        v.atterra();
    }
    public void autorizzaDecollo(Volante v) {
        v.decolla();
    }
}
```

Così facendo, possiamo passare a questi metodi oggetti volanti creati da classi che implementano l'interfaccia Volante.

Ereditarietà Multipla

La più famosa e importante differenza però riguarda l’ereditarietà. Infatti, mentre è possibile estendere una sola classe alla volta, è invece possibile implementare un numero qualsiasi di interfacce.

In realtà con i metodi di default da Java 8 è possibile fare dei trick per avere una specie di ereditarietà multipla, ma permette di ereditare solo i metodi e non i dati, essendo quindi un'ereditarietà incompleta, a differenza di linguaggi come c++ che hanno un'ereditarietà multipla completa.