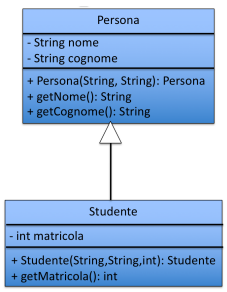
~ by Katsu Curry

Ereditarietà

Definizione | processo attraverso cui una classe (classe derivata) viene creata a partire dal codice di un'altra classe (classe base), rispetto alla quale esiste una relazione generalizzazione - specializzazione. Le sottoclassi ereditano tutte le caratteristiche della superelasse.



OSS CLASSE DERIVATA = CLASSE BASE + metodi/variabili addizionali

Keyword extends

Sintassi ClasseDerivata extends ClasseBase

Definizione | l'ereditarietà multipla é un processo secondo il quale una sottoclasse é specializzazione di due classi base. Questo non é permesso in Java.

OSS | Perché usare l'ereditarietà?
- Riuso: la superelasse si può riutilizzare in contesti diversi.
- Economia: le caratteristiche in comune a un certo insieme di classi vengono scritte 1 v.
- Consistenza: la modifica alla superelasse si ripercuote su tutte le sottoclassi.
- Estendibilità: é sempre possibile aggiungere nuove sottoclassi senza riscrivere gli aspetti comuni alla superelasse.

Es:

```
public class Persona {
    private String nome;
    private String cognome;

    public Persona(String nome, String cognome) {
        this.nome = nome;
        this.cognome = cognome;
    }

    public String getNome() {
        return nome;
    }

    public String getCognome() {
        return cognome;
    }
}
```

```
public class Studente extends Persona {
    private int matricola;
    private String nome;
    private String cognome;

    public Persona(String nome, String cognome) {
        this.nome = nome;
        this.cognome = cognome;
    }

    public Studente(String nome, String cognome,
        int matricola) {
            super(nome, cognome)
            this.matricola = matricola;
    }

    public String getNome() {
        return nome;
```

```
    }

    public String getCognome(){
        return cognome;
    }

    public int getMatricola() {
        return matricola;
    }
}
```

Overriding dei metodi

OSS: Se negli esempi di prima volessi aggiungere il metodo toString()?

ES: CLASSE PADRE (Persona)

```
public String toString(){
    return "nome =" + nome + " cognome = " + cognome;
}
```

CLASSE FIGLIO (Studente)

```
@Override → non strettamente necessario, ma permette al compilatore di vedere se é overriding e
public String toString(){      migliora la documentazione del codice
    return super.toString()  + "matricola = " + matricola;
}
```

modificatore final

Definizione | il modificatore final non permette specializzazioni in caso di una classe o l'overriding nel caso di un metodo.

ES:
```
public final int nomeMetodo()
public final class NomeClasse
```

costruttore

La costruzione di una sottoclasse richiede la definizione sia degli attributi della superclasse che della sottoclasse. Per questo i costruttori della classe derivata devono richiamare un costruttore della classe base.

REGOLE FONDAMENTALI

1) I costruttori devono invocare un costruttore della classe base con super()

Se omessa, si considera una invocazione implicita al costruttore senza parametri.

2) Se non esiste il costruttore senza parametri bisogna invocare un costruttore esplicitamente.

Es:
```
public Studente(String nome, String cognome, int matricola){
  super(nome, cognome);
  this.matricola = matricola;
}
```

classe Object

Definizione | È il tipo più generale. Ogni classe is a Object e implicitamente estende Object.
Definizioni e metodi della classe Object sono ereditati in tutte le classi Java.

Alcuni metodi di Object → public boolean equals(Object other)
ritorna true se this == other

→ public String toString()
ritorna "nomeClasse@hashcode"

OSS: la loro implementazione è troppo generica e solitamente vengono ridefiniti nelle sottoclassi.

Java packages

Definizione | un package è una collezione di classi correlate a cui viene assegnato un nome.
Evita i conflitti tra i nomi delle classi (due classi con lo stesso nome possono coesistere in package differenti).

Visibilità delle classi una classe ha visibilità di tutte le classi che si trovano nello stesso package. La visibilità può essere estesa tramite il comando import.

Il nome completo diventa nomePackage.nomeClasse

visibilità degli attributi

Definizione |
- public → nessuna restrizione
- private → accessibili solo dalla classe di definizione
- protected → accessibili solo dalle classi derivate e le classi nello stesso package
- package-wide → accessibili solo dalle classi nello stesso package

OSS: le regole di visibilità si applicano anche ai metodi solo dalla classe di definizione.
gli attributi private sono accessibili

casting

Perché il polimorfismo sia possibile dobbiamo poter assegnare ad una variabile un dato di tipo diverso da quello dichiarato.

Definizione | casting → coercizione di tipo
up-casting → polimorfismo type-safe
down-casting → polimorfismo type-unsafe

upcasting

Definizione coercizione di un tipo specializzato in un tipo più generico

OSS up-casting è sempre corretto (type-safe) e garantito dal compilatore
dopo l'up-casting si possono invocare solo i metodi del tipo statico ma i metodi che vengono eseguiti sono del tipo dinamico (overriding)

ES
```
Studente s = new Studente()
Persona p = s
```

p ha solo i metodi in comune con s e seguono le firme dei metodi di s (in caso di override)

downcasting

Definizione | coercizione di un tipo generico in un tipo più specializzato (down, più in basso in gerarchia).

OSS il downcasting deve essere dichiarato esplicitamente dal programmatore.

ES (importanti)

1.
```
Poligono p1 = new Rettangolo();
double b = ((Rettangolo) p1).getBase();
```
→ solo metodi in comune, ma di tipo dinamico
→ per poter accedere ai metodi che sono SOLO della sottoclasse, serve il cast esplicito

2.
```
Persona p = new Studente();
Studente s = (Studente) p;
```

errori generati con DownCasting

runtime → java.lang.ClassCastException
↳ casting tra classi di gerarchie diverse, ma non posso riconoscerlo immediatamente.

compile time → java.lang.ClassCastException
↳ uso di tipi incompatibili riconosciuti immediatamente dal compilatore.

instanceof

Definizione l'operatore instanceof permette di controllare il tipo dinamico associato ad un referenee.

Sintassi | <reference> instanceof <Nome classe>
↳ true : tipo dinamico del referenze è dello stesso tipo o è un sottotipo.

Object

Tutte le classi in Java sono in relazione di ereditarietà con java.lang.Object

Oss | upcast a Object è sempre possibile
downcast a una qualsiasi classe è accettato dal compilatore, ma attenzione alle classi.

getClass()

Definizione | getClass() è un metodo final della classe Object e restituisce una rappresentazione del tipo dinamico dell'oggetto.