

L'ereditarietà

Daniel Falbo — danielfalbo@engineer.com

22 Gennaio 2021

Introduzione

Quando il prof ci spiegò cosa fosse la **programmazione orientata agli oggetti** per la prima volta, nominò le 3 caratteristiche fondamentali di questo paradigma:

- l'incapsulamento,
- il polimorfismo,
- e l'**ereditarietà**.

Questo articolo introdurrà la terza di quelle caratteristiche: l'**ereditarietà**. *È consigliato scrivere ed eseguire il codice mentre si legge l'articolo.*

Cosa non è

In poche parole, quando una classe eredita un'altra classe, non sta diventando una **collezione** di oggetti della classe che sta ereditando, come si potrebbe erroneamente pensare.

L'esempio delle **classi** come **insiemi di alunni** non funziona in informatica! Per quello utilizziamo semplicemente array o liste.

Se questo riassunto ti soddisfa, puoi saltare al prossimo paragrafo, altrimenti di seguito è presente una spiegazione più dettagliata e con più esempi.

Spiegazione completa - Spesso e volentieri quando si sente parlare di classi ed **ereditarietà** in informatica per la prima volta, le si paragona erroneamente alle classi scolastiche.

Cos'è una classe a scuola? Un insieme di alunni.

Allora di sicuro anche in informatica le classi e l'ereditarietà funzioneranno come nella vita reale ... le avranno chiamate "*classi*" per un motivo, giusto? **Sbagliato!**

Per evidenziare ancor di più cosa l'ereditarietà **non** è, ecco un altro controesempio:

- questo **articolo** è un insieme di **paragrafi**,
 - ogni **paragrafo** è un insieme di **frasi**,
 - * ogni **frase** è un insieme di **parole**,
 - ogni **parola** è un insieme di **lettere**;

si potrebbe pensare che per implementare una struttura del genere in un linguaggio di programmazione orientato agli oggetti, come *Java*, potremmo utilizzare una classe *Lettera* e poi da quella derivare

- una classe Parola che eredita da Lettera,
 - una classe Frase che eredita da Parola,
 - * una classe Paragrafo che eredita da Frase,
 - ed infine una classe Articolo che eredita da Paragrafo.

Non è così! Se volessimo implementare una struttura come quella di questo articolo, utilizzeremo semplici collezioni di dati, come array o liste, non abbiamo assolutamente bisogno dell’ereditarietà e non è questo quello che l’ereditarietà è disegnata per fare.

Sappiamo che in *Java* tra i **tipi primitivi** abbiamo i **char**, caratteri Unicode. Loro potrebbero essere le nostre “lettere”, poi procederemmo semplicemente utilizzando un **array di char** per rappresentare le parole, un **array di array di char** per le lettere, e così via.

```
char        lettera;
char []     parola;
char [] []  frase;
char [] [] [] paragrafo;
char [] [] [] [] articolo;
```

Cos’è

In poche parole, ereditare una **classe** significa ereditare tutti i suoi metodi e tutte le sue proprietà.

Potremmo utilizzare le **persone** ed i **programmatori** come esempio. I **programmatori**, prima di essere tali, sono **persone**: hanno tutte le proprietà che le persone hanno, perchè anche loro sono persone. In un linguaggio orientato agli oggetti potremmo disegnare una **classe Persona** per rappresentare tutti i tipi di persona, ed un’altra **classe Programmatore** che **eredita da Persona**, che letteralmente eredita tutte le proprietà ed i metodi che la **classe Persona** offre, ma che potrebbe aggiungerne, modificarne o sovrascriverne alcuni.

Se questo riassunto ti soddisfa, puoi saltare al prossimo paragrafo, altrimenti di seguito è presente una spiegazione più dettagliata e con più esempi.

Spiegazione completa - Nella programmazione orientata agli oggetti, l’ereditarietà è un meccanismo che consiste nel basare una classe su un’altra classe con un’implementazione simile. Quando descriviamo una **classe B** che **eredita da A**, stiamo descrivendo una nuova **classe B** che, di suo, senza ulteriori modifiche, avrà tutti i metodi e le proprietà della **classe A** e che in più potrebbe aggiungerne, modificarne o sovrascriverne alcuni. In gergo tecnico si parla di “*classi derivate da superclassi*”, infatti, nel nostro esempio, **classe A** sarebbe la cosiddetta “*superclasse*”, e da essa staremmo “*derivando*” una nuova **classe B**.

Come si traduce tutta questa teoria in *Java*?

1. Già dovremmo sapere bene come definire una nuova classe ed istanziarne oggetti.

```
public class A {  
  
    public int a;  
  
    public A() {  
        a = 0;  
    }  
  
}
```

2. Per creare la nostra nuova classe B, che eredita dalla nostra classe A, utilizzeremo la parolina **extends**, che in italiano si tradurrebbe in **estende** e infatti, pensandoci, potremmo dire che una classe letteralmente **estende** la sua superclasse.

```
public class B extends A {  
  
}
```

A questo punto, istanziando un oggetto di classe B e provando a stampare a video il valore del suo attributo **a**

```
B oggetto = new B();  
System.out.println(oggetto.a);
```

, possiamo notare che il nostro oggetto effettivamente ha ereditato l'attributo **a** dalla classe **A** anche se noi non l'abbiamo esplicitamente dichiarato nella definizione della classe **B** (abbiamo solo dichiarato che la classe **B** avrebbe esteso **A**).

3. In *Java*, tutti gli oggetti hanno un metodo `getClass()` che ritorna, come potremmo intuire dal nome, la classe dell'oggetto. Seguendo il nostro esempio,

```
System.out.println(oggetto.getClass());
```

stampa `class B`. La classe dell'oggetto ritornato dal metodo `getClass`, è `java.lang.Class` (esercizio per il lettore: con che linea di codice potremmo verificarlo?) e tutti gli oggetti di classe `Class` a loro volta hanno un metodo `getSuperclass()`, come `getClass`, ma ritorna la superclasse. Ad esempio,

```
System.out.println(oggetto.getClass().getSuperclass());
```

stampa `class A`, come previsto (esercizio per il lettore: utilizzando i metodi `getClass` e `getSuperclass`, cerca di capire:

- C'è una classe primitiva in *Java* che è superclasse di tutte le classi?
- C'è una classe che non ha alcuna superclasse?

