

# Programmazione ad Oggetti

Alex Narder

December 1, 2022

## 1 Definizioni

- **Variabili:** Aree di memoria a cui viene assegnato un valore
- **Classi:** Una classe è la definizione di un tipo di oggetto, quindi al suo interno contiene metodi, campi e variabili
- **Oggetto:** Un oggetto di una classe è un'istanza della classe, e ha tutti gli attributi di quella classe
- **Costruttori:** Il costruttore è quel metodo di una classe il cui compito è proprio quello di creare nuove istanze. Il costruttore è dichiarato con il nome della classe e serve soprattutto per inizializzare i campi con un valore.
- **Metodi:** Essenzialmente sono le funzioni contenute nella classe
- **Campi:** Anche definiti fields, sono le variabili della classe che vengono dichiarate al suo interno ma fuori dai costruttori e dai metodi

```

class Appunti{
    int n;           //Campo (field)
    int res;         //Campo (field)

    Appunti(int n){  //Costruttore con valore
        this.n = n;  //Inizializzo n della classe assegnandogli valore di n
    }

    void calcRes(){  //Questo è un metodo per calcolare il risultato finale
        res = n*n;
    }

    void print(){    //Metodo, e questa è la sua intestazione
        System.out.println("The result is:"); //Corpo del metodo
    }

    int returnRes(){ //Metodo
        print();     //Invocazione del metodo print()
        return res;
    }
}

class Nuova{
    public static void main(String[] args){
        int n = 5;
        Appunti newObj = new Appunti(n);    //Nuovo oggetto
    }
}

```

## 2 Javadoc

Ci sono diversi modi per commentare in Java, e uno dei pregi di questo linguaggio è quello di poter integrare la documentazione con il codice stesso:

Formato dei commenti:

- */\* commenti \*/*
- *// commenti*
- */\*\* commenti documentazione \*/*

Quest'ultimo genera automaticamente la documentazione in formato HTML utilizzando il programma javadoc (scritto in Java).

Il commento ha due parti:

una descrizione a parole, seguita da alcuni **block tags**, ovvero delle etichette standard che discutono alcuni aspetti.

```
/**
 * Accede un elenco di studenti di un url e ritorna uno
 * Studente che ha come matricola il numero passato come
 * parametro. L'url deve essere un {@link URL} assoluto.
 * <p> <!-- -->
 * Il metodo ritorna sempre un valore. Se lo studente
 * non esiste o l'elenco è vuoto, ritorna null.
 * @param url    URL assoluto dove si trova l'elenco
 * @param matr   la matricola dello studente da trovare
 * @returns      il link allo Studente
 * @see          Studente
 */
public Studente getStudente(URL url, int matricola)
{ try {...}
  catch(MalformedURLException) {return null;}
  catch(ElencoVuotoException) {return null;} ...
}
```

Una volta scritta la documentazione, questa viene generata con il programma **javadoc**. Ad esempio se abbiamo definito una classe **Dado** nel file **Dado.java** possiamo generare la documentazione con **javadoc Dado.java**. Vengono generati i file di documentazione della classe a partire da un **index.html** che possono essere visualizzati con un browser.

## 2.1 Block Tag comuni

- **@author**: specifica il nome dell'autore, viene considerato solo se javadoc viene eseguito con l'opzione -author
- **@version**: indica un numero di versione, viene considerato solo se javadoc viene eseguito con l'opzione -version
- **@param**: descrive uno dei parametri passati
- **@returns**: descrive il valore di ritorno restituito al chiamante
- **@throws**: (per ogni eccezione si può verificare) descrive il tipo di eccezione e la sua descrizione
- **@see**: rimanda a un'altra voce di documentazione
- **@deprecated**: indica che non andrebbe più usato

## 2.2 Documentazione iniziale

```
/**
 * This class contains various methods for manipulating arrays (such as
 * sorting and searching). This class also contains a static factory
 * that allows arrays to be viewed as lists.
 *
 * @author Josh Bloch
 * @author Neal Gafter
 * @author John Rose
 * @since 1.2
 */
```

## 2.3 Documentazione di un field

Nulla di che, solo un commento con la sua descrizione:

```
/**
 * The minimum array length below which a parallel sorting
 * algorithm will not further partition the sorting task. Using
 * smaller sizes typically results in memory contention across
 * tasks that makes parallel speedups unlikely.
 */
private static final int MIN_ARRAY_SORT_GRAN = 1 << 13;
```

## 2.4 Documentazione di un metodo

Un metodo viene definito con i tag `@return`, `@param` ;nome;, `@throw`, `@deprecated`:

```
/**
 * Finds and returns the index of the first mismatch
 * between two
 * {@code double} arrays, otherwise return -1 if no
 * mismatch is found.
 * (...)
 * @param a the first array to be tested for a mismatch
 * @param b the second array to be tested for a
 * mismatch
 * @return the index of the first mismatch between the
 * two arrays,
 * otherwise {@code -1}.
 * @throws NullPointerException
 * if either array is {@code null}
 * @since 9
 */
public static int mismatch(double[] a, double[] b)
```

## 3 Object

Gli Object sono superclassi di referenza universali:

Object ha dei metodi:

```
boolean equals(Object obj){    //determina quando due oggetti sono uguali.

}

int hashCode(){    //ritorna un valore hashCode dell'oggetto.

    return hashCode;
}

String toString(){    //ritorna la rappresentazione dell'oggetto sottoforma di stringa.

}
```

- **equals()**: indica quando due oggetti sono uguali. Di norma richiamata con `x.equals(y)`. Il metodo ritorna `true` quando `(this == obj)`.

Esempio di implementazione:

```
class ClassName{
    public String field;

    ClassName(String field){
        this.field = field;
    }

    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (!super.equals(obj)) //super in questo caso è come scrivere this.field,
                                //viene usato equals qui perchè le stringe si
                                //comparano così.
            return false;
        if (!(obj instanceof ClassName))
            return false;
        return true;
    }
}
```

- **hashCode()**: Ritorna una hashCode value per l'oggetto. Di norma richiamato con `x.hashCode()==y.hashCode()`.

Esempio di implementazione:

```
class ClassName{
    public String field;

    ClassName(String field){
        this.field = field;
    }

    public int hashCode(){ //questa implementazione è default quasi
                           //al 100% ogni volta
        final int prime = 31; //default value
        int hash = 1; //settato a 1 perchè vado a fare una moltiplicazione
                     //sotto
        hash = prime * hash + ((name == null) ? 0 : name.hashCode());
        //qui sopra va ad applicare la funzione dell'hashCode per
        //ottenere il risultato.
        return hash;
    }
}
```

- `toString()`: Esempio di implementazione:

```
class Person{
    private String name;

    public Person(String name){
        this.name = name;
    }
    public String toString(){
        return this.name;
    }
}

class Professor extends Person{
    public Professor(String name){
        super(name);
    }
    public String toString(){
        return "Prof. "+super.toString(); //super.toString() ritorna qualsiasi
                                           //valore ritornato da super.
    }
}
```

## 4 Set

Un set può essere definito come una raccolta di oggetti non ordinata che non contiene duplicati.

```
public class IterateOverHashSet {
    public static void main (String[] args){

        //Un set viene dichiarato in questo modo:
        Set<String> mySet = new HashSet<>();

        mySet.add("Apple"); //viene aggiunta la stringa Apple al set
        mySet.add("Banana");
        mySet.add("Mango");

        for(String s : mySet){ //iterazione di un set tramite variabile dello stesso tipo
            System.out.println("s is : "+s);
        }
    }
}
```

## 5 String