Lezione 4

Organizzare le migliaia di classi (con relativi metodi) in Java sarebbe estremamente difficili se prese singolarmente. Queste classi sono organizzate in packages, i quali possono essere inclusi nel codice tramite `import`:

```
import java.util.Arrays
```

aggiungerà le classi con relativi metodi definiti nel package `java.util.Arrays`.

`for-each`

`for-each` è una variante del `for` che può essere letta per leggere tutti gli elementi di una data struttura dati.

```
for (<type> <name>: <structure>){
    ...
}
```

- `type` è il tipo di elemento considerato
- `name` è il nome che l'elemento preso in considerazione ha dentro al ciclo
- `structure` è il riferimento alla struttura dati considerata
  Questa tipologia di ciclo non può essere usata per modificare i valori interni alla struttura dati, ma solo per leggerli.

Codice statico vs. non statico

Tutto ciò che viene scritto nella OOP pura NON è statico. Tuttavia, numerosi metodi utilizzati in Java sono dichiarati con la keyword `static`. Il codice cosiddetto "statico" è un codice che ammette input e restituisce output: un esempio è il codice C-like.

- Codice non statico è formato da campi che possono differire o possono essere modificate in runtime e metodi che modificano alcuni specifici campi.
- Codice statico è formato da campi o metodi che sono accessibili da ogni oggetto e che sono comuni a tutti gli oggetti di una data classe. Ciò include costanti oppure oggetti particolarmente utili di quella classe.
  Quando un campo/metodo non è statico esso verrà richiamato su un oggetto specifico; quando invece è statico possiamo richiamarlo a piacimento al fine di eseguire assegnazioni o operazioni.

Alcuni metodi non sono vuoti, ma ritornano se stessi. Questa variante di build si dice "fluente" e permette ulteriori compressioni.
È inoltre utile dividere ciò che è statico da ciò che non lo è all'interno di un programma; spesso addirittura mettendo ciò che è statico in una classe a parte con un nome "plurale" di quello utilizzato.

```
class Point3D { // Qui metodi e campi non statici
    int x;
    int y;
    int z;

    void buildPoint(int a, int b, int c){
        this.x = a;
        this.y = b;
        this.z = c;
    }
}

class Points3D { // Qui metto tutto lo static
    static Point3D zeropoint = new Point3D();
}
```

Costruttori

Un costruttore è un metodo avente lo stesso nome della classe, il quale serve per impostare i campi iniziali di un oggetto alla sua dichiarazione.
Un costruttore è così formato:

- Non ha tipo
- Ha lo stesso nome della classe
- Può avere o meno parametri

```
class Point3D {
    int x;
    int y;
    int z;

    Point3D(int a, int b, int c){
        this.x = a;
        this.y = b;
        this.z = c;
    }
}
```

```
...

Point3D newPunto = new Point3D(1, 2, 3); // nuovoPunto = (1, 2, 3)
```

La keyword `new`, quando usata, richiama il costruttore della classe che viene specificata. Se non è presente un costruttore all'interno della classe, verrà usato il costruttore "default", ossia un costruttore vuoto privo di parametri (ossia che imposta i campi ai loro valori di default).

Un costruttore può essere soggetto a "overloading": possiamo dichiarare più metodi costruttori al fine di associare parametri di diverso tipo ai campi della classe. Quando verrà dichiarato un nuovo oggetto della classe, a patto di non ricadere in casi di ambiguità, verrà usato il costruttore adeguato ai parametri passati.

```
class Point3D {
    int x;
    int y;
    int z;

    Point3D(int a, int b, int c){ // 3 params
        this.x = a;
        this.y = b;
        this.z = c;
    }

    Point3D(int a, int b) { // 2 params
        this.x = 0;
        this.y = a;
        this.z = b;
    }
}

...

Point3D newPunto = new Point3D(1, 2, 3); // nuovoPunto = (1, 2, 3)
Point3D newPunto2 = new Point3D(1, 2); // newPunto2 = (0, 1, 2)
Point3D newPunto3 = new Point3D(); // newPunto3 = (0, 0, 0)
```

Packages e accesso alle variabili

Tutti i packages di Java contengono le classi utilizzabili nel linguaggio. Tutte le classi di un package sono contenuti tutti all'interno della stessa directory. I package sono inoltre disposti in maniera gerarchica, ossia ci sono dei package che controllano altri package.

La visibilità di metodi e campi di una certa classe è determinata dal tipo di visibilità, ossia se sono pubbliche (`public`) o private (`private`).

- Se una classe/metodo/campo è `public` (e va specificato), allora essa/o è accessibile se il package viene importato tramite `import`, quindi utilizzabile direttamente.
- Se una classe/metodo/campo è `private` (e lo è di default), allora essa/o non è accessibile in maniera diretta se il package in cui è contenuto è importato tramite `import`, ma possono essere utilizzate all'interno del package stesso.
  Spesso, le classi, i metodi o i campi `public` sono utilizzabili direttamente in un altro codice, mentre se `private` sono utilizzabili solo da altre classi/metodi presenti direttamente all'interno del package.

È possibile inoltre dichiarare valori costanti tramite la keyword `final`. In combinazione con `static`, permette di creare delle costanti, presenti all'interno della classe e quindi utilizzabili in maniera chiara e facile.

```
class Numbers {
final static PI = 3.141;
...
}

...

int circonferenza = 2 * raggio * Numbers.PI;
```

Il garbage collector

La memoria, in Java, viene automaticamente gestito da un algoritmo chiamato "garbage collector". Questo automaticamente dealloca dalla memoria oggetti che non sono utilizzati dal programma, andando a liberare progressivamente memoria per altri oggetti utilizzabili.