In questa lezione applichiamo i principi di un linguaggio orientato agli oggetti al Java.

1 - Everything is an object

Non esiste, in Java, un modo per accedere ai dati per valore o per puntatore; quindi le variabili non si riferiranno a cose presenze in memoria, ma direttamente agli oggetti - quindi non sono altri che nomi locali per denotare gli oggetti.

La dichiarazione di una variabile avviene tramite la seguente istruzione

```
String s = new String();
```

- `String` è il tipo di variabile di cui stiamo dichiarando l'oggetto.
- `s` è il nome della variabile stringa. Le variabili si indicano con lettere minuscole.
- `new` è la keyword usata per inizializzare una variabile.
- `String()` è la classe di cui stiamo creando l'oggetto. Le classi si indicano con lettere maiuscole. La classe deve esistere nel codice o deve essere implementata tramite libreria.

```
String s2 = new String("Ciao!");
```

In questo caso stiamo passando come parametro la stringa "Ciao!", con cui il nostro oggetto verrà inizializzato.

```
Object o = new Object();
```

C'è anche una classe `Object`, che vedremo in seguito.

Solo per le stringhe è possibile utilizzare la stessa sintassi del C per l'assegnazione e l'inizializzazione:

```
// Assegnazione e inizializzazione come prima.
String s2 = new String("Ciao!");

// Assegnazione e inizializzazione come il C.
String s = "Ciao!"
```

Nonostante la filosofia "Everything is an object", ci sono delle eccezioni. Esistono anche tipi primitivi che funzionano esattamente come il C. Essi sono `int, boolean, char, byte,`

`short, long, float, double` - le dimensioni di questi tipi primitivi sono fissate e non variano in base all'architettura, a differenza del C; inoltre non esistono gli unsigned.

Quindi i tipi di Java sono le classi e i tipi primitivi; questi possono cooperare.

```
int i = 5; // Assegnamo tipi primitivi.
double d = 145e-20;
boolean b = true;
int other = i;

Object o = new Object();
String s = "hello sono una stringa";
Point2D p = new Point2D(10.4, 23.2);
Point2D q = p; // Assegno a q una variabile p, dello stesso tipo.

Object on = null; // Assegno a null.
```

In Java, come in C, la memoria viene suddivisa nella heap e nella stack. All'interno della stack vengono salvate tutte le variabili, con i loro relativi valori assegnati. All'interno della heap vengono contenuti gli oggetti creati con `new`; l'istruzione `Object o = new Object()` fa la seguente cosa:

- Crea la variabile `o` nella stack.
- Crea l'oggetto `Object` nella heap.
- Associa `o` al nuovo `Object` creato.

```
Point2D q = p; // Assegno a q una variabile p, dello stesso tipo.
```

In questa istruzione, q si riferirà allo stesso oggetto assegnato a p. Manca la parola `new` quindi non viene creato un nuovo oggetto.
Da Java 10, è stato introdotto il costrutto `var`; quando viene dichiarata una variabile tramite `var` viene effettuata un'inferenza al fine di capire quale tipo di dato associare alla variabile sulla base di ciò che gli è stato assegnato. Ovviamente bisogna che ci siano abbastanza informazioni per rendere possibile l'inferenza, altrimenti il compilatore non sa che tipo di variabile associare e restituisce un errore di compilazione.

Se un oggetto, presente nella heap, non è associato più ad una variabile, un programma, detto "garbage collector", dealloca automaticamente gli oggetti non più utilizzati, liberando memoria.

Lo scope degli oggetti è molto simile a quella del C.

2 - Le classi

Le classi sono le unità fondamentali della programmazione orientata agli oggetti. È un template per generare oggetti tutti della stessa forma, come uno "stampino" che genera tutte cose simili. Una classe è formata da:

- Il nome della classe
- I campi della classe (fields), ossia lo stato che gli oggetti di tale classe mantengono in uno specifico momento
- I metodi della classe (methods), ossia le operazioni che i vari oggetti possono effettuare
  Un programma nella OOP è fatto da tante classi, ma gli oggetti, istanze di classi, sono gli elementi tangibili.

La dichiarazione in Java è la seguente:

```
class Name{ // Name è il nome della classe.
    ... // Contenuto della classe.
}

class AnotherClass{
    ...
}

Name obj1 = new Name();
```

Al posto dei "..." potrebbero esserci i metodi, i campi, oppure nessuno dei due.
Due classi, anche costruite nello stesso modo, non sono mai compatibili:

```
Class1 obj1;
obj1 = new Class1(); // Corretta
obj1 = new Class2(); // ERRORE SEMANTICO.
```

Campi

Assomigliano molto ai campi di una struct.
Possiamo inserire quanti campi vogliamo in una classe, anche nessuno; ragionevolmente i campi in una classe vanno dagli 1 ai 5-6.

```
Class MyClass{
    int i;
    int j = 2;
    Object o;
}
```

Nei campi non è possibile usare `var`. Se non vengono assegnati valori iniziali, come nel caso di `int i`, viene assegnato un valore in base al tipo di dato:

- I tipi numerici (`int, float`...) vengono inizializzati a 0
- I `boolean` sono inizializzati a `false`
- Agli oggetti vengono inizializzati a `null`
  I campi definiscono lo stato iniziale degli oggetti di quella classe in memoria. È possibile ricavare gli attuali valori di uno specifico oggetto nello stesso modo in cui si fa in C con le classi, tramite la "dot notation", ossia utilizzando il punto.

```
MyClass obj = new MyClass();
int a = obj.i; // a vale 0
int b = obj.j; // b vale 2
obj.i = 10; // Il campo i dell'oggetto obj vale 10.
int d = obj.i; // d vale 10
obj.o = new Object(); // Il campo o dell'oggetto obj punta a un nuovo
oggetto creato in memoria, di tipo Object.
MyClass obj2 = new MyClass();
obj.i = obj2.i; // Il campo i dell'oggetto obj vale 0, siccome obj2.i vale
0.
```

Metodi

```
class MyClass {
    int i;
    void add(int a){ // Parametro int a, passato in input.
        i = i + a;
    }
    int getValue(){ // Intestazione funzione
        return i; // Corpo della funzione
    }
}
```

I metodi sono funzioni presenti all'interno di una specifica classe. L'utente, tramite messaggi inviati agli oggetti di tali classi può eseguire tali funzioni su tali oggetti, i quali le elaborano e le eseguono (risposta).

```
MyClass obj = new MyClass();
int v = obj.i; // v = 0
obj.add(10); // obj.i ora vale 10
obj.add(20); // obj.i ora vale 30
```

```
int v2 = obj.i; // v2 = 30
int v3 = obj.getValue(); // v3 = 30
```

Un messaggio può avere o meno parametri da passare al metodo. Quando viene invocato un metodo ad uno specifico oggetto, la sua risposta dipenderà dal corpo del metodo stesso all'interno della classe.

All'interno del corpo dei metodi, il prefisso `this` indica la variabile presente all'interno della classe (o dell'oggetto quando istanziata). Nel corso sarà importante utilizzarla quando necessaria.

```
class MyClass {
    int i; // i locale alla classe
    void add(int a){
        this.i = this.i + a; // this.i è i a riga 2, mentre la a è quella
passata come parametro.
    }
    int getValue(){
        return this.i; // questa è la i presente in riga 2.
    }
}
```

Com'è fatto un programma Java

Un programma di Java è formato da diverse parti, tutte formate da classi:

- Le librerie di Java, che contengono più di 4000 classi raggruppati in più di 200 package; sono organizzati in una struttura gerarchica ad albero: il pacchetto `java` è suddiviso in `java.base`, `java.lang`, ecc...
- Le classi realizzate dall'utente; è necessario sviluppare una classe avente un metodo chiamato `main`, realizzata nel seguente modo:

```
class Hello{
    public static void main(String[] args){
        System.out.println("Hello World!");
    }
}
```

Se un programma Java contiene una sola classe, il nome del file va rinominato come il nome della classe.