

# **Programmazione ad Oggetti**

## **01 - Introduzione**

Martedí, Marzo 01, 2023

*Claudio Menghi*

## Contents

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Le caratteristiche di Java . . . . .	3
1.1.1	Java è portabile e compilato . . . . .	3
1.1.2	Java è orientato agli oggetti . . . . .	4
1.1.3	Java è staticamente tipizzato . . . . .	5
1.2	Variabili e tipi di riferimento . . . . .	5
1.2.1	Dichiarazione di una variabile . . . . .	6
1.2.2	Convenzione di notazione . . . . .	6
1.2.3	Inizializzazione di una variabile . . . . .	6
1.2.4	Creazione di un nuovo oggetto . . . . .	7
1.3	Costruttori . . . . .	7
1.4	Array . . . . .	7
1.5	Semantic versioning . . . . .	8
1.6	Java versions history . . . . .	9
<b>2</b>	<b>Esercizi</b>	<b>9</b>
2.1	Java è compilato ed è portabile . . . . .	9
2.1.1	Esercizio . . . . .	9
2.1.2	Esercizio . . . . .	10
2.2	Java è orientato agli oggetti e staticamente tipizzato . . . . .	11
2.2.1	Esercizio . . . . .	11
2.2.2	Esercizio . . . . .	12
2.2.3	Esercizio . . . . .	13
2.2.4	Esercizio . . . . .	13
2.2.5	Esercizio . . . . .	14
2.3	Esercizio . . . . .	15
2.4	Eclipse Integrated Development Environment . . . . .	15
2.4.1	Esercizio . . . . .	15
2.5	Java è orientato agli oggetti e staticamente tipizzato . . . . .	16
2.5.1	Esercizio . . . . .	16
2.5.2	Esercizio . . . . .	18
2.5.3	Esercizio . . . . .	20
2.6	Variabili e tipi di riferimento . . . . .	22
2.6.1	Esercizio . . . . .	22
2.7	Costruttori . . . . .	23
2.7.1	Esercizio . . . . .	23
2.7.2	Esercizio . . . . .	25
2.7.3	Esercizio . . . . .	26
2.8	Array . . . . .	28
2.8.1	Esercizio . . . . .	28
<b>3</b>	<b>Curiosity</b>	<b>29</b>

# 1 Introduzione

Questa lezione copre le slides “Classi come astrazioni” o in alternativa i capitoli 1, 2, 3, 4 e 5 del libro Pellegrino Principe – “Java 8”.

## 1.1 Le caratteristiche di Java

Questa esercitazione ha come scopo quella di fornire una panoramica su Java e sulle sue caratteristiche fondamentali.

In particolare Java è:

- **Portabile:** è la caratteristica principale di Java. L’obiettivo di Java è quello di consentire allo sviluppatore di scrivere il programma una volta sola avendo la certezza che sarà possibile eseguirlo ovunque indipendentemente dall’architettura della macchina su cui viene eseguito. La filosofia di Java è quindi “Write Once, Run Anywhere”.
- **Compilato:** Java è un linguaggio compilato, ovvero viene “tradotto” dal linguaggio di programmazione Java al linguaggio “oggetto” bytecode.
- **Orientato agli oggetti:** in un linguaggio orientato agli oggetti lo sviluppatore ragiona in termini di oggetti ovvero astrazioni dei concetti del mondo reale che lo sviluppatore vuole modellare<sup>1</sup>. In realtà Java è un linguaggio multi-paradigma dal momento che è anche procedurale, e se consideriamo Java 8 è anche funzionale (parzialmente).
- **Staticamente tipizzato:** un linguaggio è staticamente tipizzato quando è necessario associare ad ogni variabile un tipo.

### 1.1.1 Java è portabile e compilato

La caratteristica principale di Java è il fatto di essere portabile/platform independent. Se prendiamo per esempio C il compilatore genera un linguaggio “oggetto” che è dipendente dalla macchina nel quale il compilatore viene eseguito. Il codice compilato può solo essere eseguito sulla piattaforma per il quale il codice è stato compilato.

In Java la fase di compilazione produce un codice intermedio chiamato *byte-code* che può essere eseguito su macchine differenti supposto che ci sia installato sulla macchina un *interprete* (Java Virtual Machine JVM) capace di capire il byte-code. In altre parole il byte-code è un codice intermedio prodotto dopo la compilazione. Il Byte-code differisce dal codice eseguibile (per esempio dai file “.exe”) dal momento che deve essere interpretato da una Java Virtual Machine per essere eseguito.

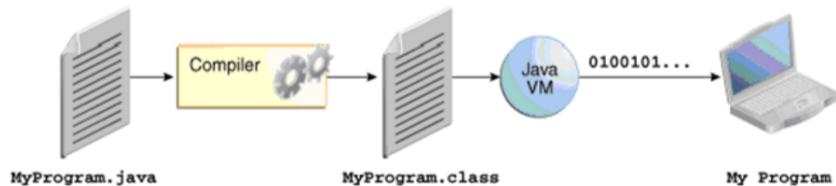


Figure 1: Java architecture

L’architettura del linguaggio Java è mostrata in Figura 1. Più precisamente:

- MyProgram.java: è il codice sorgente nativo dell’applicazione Java (Il file che contiene il nostro codice)

<sup>1</sup>La filosofia è diversa da quella utilizzata, per esempio, quando si scrive in C, dove gli ingredienti fondamentali sono le procedure. Infatti in un linguaggio procedurale le componenti fondamentali sono le funzioni (“procedure”) che manipolano i dati del programma.

- Compiler: (Compilatore) prende in input il nostro codice Java (i nostri files .java), e produce dei file intermedi (.class) ovvero i file contenenti il byte-code.
- Java Virtual Machine (JVM): è l'interprete che deve essere installato sulla nostra macchina locale al fine di eseguire il byte-code. Il byte-code viene interpretato dalla Java Virtual Machine. Per questo motivo alcuni testi considerano Java anche come un linguaggio interpretato.

Per eseguire il vostro programma su una macchina è sufficiente che l'utente abbia installato la Java Virtual Machine. In particolare, il **Java Run-time Environment** (JRE) contiene la macchina virtuale che consente di eseguire i programmi Java. JREs differenti sono associati a sistemi operativi differenti. Una volta installata la JRE è possibile eseguire i file .class generati.

Nota che, una volta generati, i file .class possono essere eseguiti su ogni sistema che abbia una Macchina Virtuale Java installata, indipendentemente dal sistema operativo su cui la macchina virtuale è eseguita.

Per compilare i file .java è necessario avere un tool di sviluppo Java: un **Java Development Kit** (JDK). Il JDK include la JRE e contiene un set di tool di sviluppo che consentono di scrivere e compilare i tuoi programmi Java.

### 1.1.2 Java è orientato agli oggetti

La programmazione orientata agli oggetti implica un approccio alla programmazione completamente diverso rispetto ai normali linguaggi procedurali. Gli oggetti sono entità che hanno una “*vita propria*” ed esistono indipendentemente dal come vengono utilizzate all'interno della nostra applicazione.

**Definizione 1.** Oggetto è un astrazione di un oggetto del mondo reale e ne descrive le caratteristiche che sono di interesse allo sviluppatore.

In altre parole in un paradigma orientato agli oggetti lo sviluppatore descrive il mondo e i suoi oggetti e come il mondo evolve nel corso del tempo. In un paradigma procedurale lo sviluppatore specifica come risolvere un problema, mentre in un linguaggio orientato agli oggetti lo sviluppatore prima descrive “il problema” e poi cerca una strategia per risolvere il problema utilizzando il modello del mondo costruito.

Gli oggetti vengono creati partendo da modelli più generali chiamati classi.

**Definizione 2.** Classi sono dei *tipi* definiti dall'utente che descrivono degli oggetti. In particolare descrivono l'oggetto, il suo *stato* e come lo *stato dell'oggetto* cambia in risposta a delle operazioni eseguite sull'oggetto.

Utilizzando un esempio automobilistico possiamo pensare alle classi come ai documenti di progetto di un'autovettura, e agli oggetti come alle macchine realizzate partendo dai quei documenti. In altre parole le classi sono la teoria, gli oggetti sono particolari istanze di una specifica classe.

Gli oggetti vengono creati col costruttore **new**. Per esempio l'istruzione seguente crea un nuovo oggetto di tipo Bike.

---

```
Bike myBike=new Bike();
```

---

La descrizione del tipo bike, ovvero la classe Bike, è presentata nel seguito.

---

```
public class Bike {  
    public Bike() {  
    }  
}
```

---

Le classi sono definite mediante la parola chiave (keyword) **class** e descrivono le caratteristiche dello stato di un oggetto e come lo stato può cambiare nel tempo.

### 1.1.3 Java è staticamente tipizzato

**Definizione 3.** **Staticamente tipizzato** un linguaggio è staticamente tipizzato se ogni variabile è associata a un tipo: non è possibile creare una variabile senza un tipo.

---

```
Bike bike1=new Bike();
```

---

Per esempio, l'istruzione precedente dichiara una variabile di tipo **Bike** con **identificatore** **bike1**, che è associata a un oggetto di classe **Bike**. **In Java ogni variabile deve essere associata a un tipo**.

## 1.2 Variabili e tipi di riferimento

Prima di discutere le variabili e i tipi di riferimento discutiamo per l'organizzazione della memoria in Java

- **Stack e Heap:** Lo stack e lo heap sono due aree di memoria che sono utilizzate per contenere le nostre variabili e i nostri oggetti.
  - Lo stack è un area di memoria dove è possibile memorizzare informazioni. È molto rapido e basato su un **LIFO pattern**.
  - Lo heap è uno spazio di memoria assegnabile al progetto e facilmente estendibile.
  - Lo stack ha una dimensione fissa assegnata prima di eseguire il programma “durante la fase di startup”. Lo heap non viene utilizzato seguendo uno specifico pattern, e ciò lo rende molto flessibile (è estendibile) ma anche più lento.

**Definizione 4.** Una variabile è uno spazio di memoria che contiene un “valore”.

Java è fortemente tipizzato (*strongly typed*), quindi *ogni variabile ha un tipo*. Il compilatore durante la compilazione controlla che le variabili siano utilizzate in maniera corretta in relazione al loro tipo.

- **Variabili primitive:** le variabili primitive definiscono una “cella di memoria” che contiene direttamente il valore della variabile. Le variabili primitive di Java sono:
  - *byte*: 8 bit, *short*: 16 bit, *int*: 32 bit, *long*: 64 bit, *float*: 32 bit, *double*: 64 bit, *char*: 16 bit *boolean*: true/false
- **Variabili riferimento:** le altre variabili contengono un riferimento alle aree di memoria che contengono il “valore”: l'**oggetto**. I tipi di riferimento 1) possono essere definiti dall'utente mediante classi e interfacce, 2) includono gli array e le enumerazioni.
- variabili primitive e di riferimento possono essere utilizzate nella stesso modo: come attributi, restituiti o passati a metodi.

**Le variabili:**

- sono allocate sullo *stack* a run-time quando si chiama il metodo su cui si sono dichiarate
- si trovano nello *heap* quando rappresentano un attributo di un oggetto (vedremo in seguito).
- se non sono attributi di un oggetto sono deallocate quando il sottoprogramma ritorna al chiamante

**Gli oggetti:**

- sono allocati sullo *heap*.

**Dove vengono allocati gli oggetti Java?**

Gli oggetti Java risiedono in una zona detta *heap*. L'heap viene creato all'avvio della JVM e può cambiare dimensione durante l'esecuzione dell'applicazione. A discrezione della JVM viene eseguita l'operazione di "garbage collection", che consiste nella rimozione degli oggetti non più utilizzabili dal programma per fare posto ad altri. Il componente che effettua tale operazione prende il nome di "garbage collector".

**Quali sono le differenze tra i puntatori di C e le variabili di riferimento di Java?**

- un riferimento non può essere de-allocato dall'utente, viene rimosso dal garbage collector
- un riferimento non permette l'acceso all'indirizzo di memoria relativo ad un oggetto
- non è possibile effettuare operazioni aritmetiche sul riferimento, come è possibile sui puntatori nel linguaggio C

**1.2.1 Dichiarazione di una variabile**

Per dichiarare una variabile è sufficiente specificare il *tipo* e un *identificatore*, ovvero un nome simbolico utilizzato per riferirsi alla variabile<sup>2</sup>.

Per esempio, l'istruzione:

---

```
int number;
```

---

dichiara una variabile di tipo primitivo "int" con identificatore "number", mentre l'istruzione

---

```
Car mycar;
```

---

dichiara una variabile di tipo riferimento "Car" e con identificatore "mycar". La dichiarazione non alloca spazio per l'oggetto ma solo per il riferimento all'oggetto.

*Qual è la differenza tra tipi riferimenti e puntatori (e.g., i puntatori di C)?*

Ci sono varie differenze e similitudini tra riferimenti e puntatori. In genere un riferimento può essere interpretato come un puntatore "ad alto livello", mentre la differenza fondamentale è il fatto che nei riferimenti l'indirizzo di memoria non è noto e non interessa. Per questo motivo, ad esempio, i riferimenti non consentono l'utilizzo dell'aritmetica dei puntatori.

**1.2.2 Convenzione di notazione**

In Java i nomi di classi, variabili e metodi fanno uso della notazione a cammello (*CamelCase*). La notazione a cammello è costituita dalla giustapposizione delle parole che costituiscono l'identificativo, unite con l'iniziale di ogni parola maiuscola. La prima lettera dell'identificativo è maiuscola nel caso di classi e minuscola nel caso di metodi e attributi/variabili.

**1.2.3 Inizializzazione di una variabile**

Una variabile non può essere usata senza essere inizializzata. Quando un attributo viene dichiarato gli viene assegnato un valore di default.

Il riferimento è assegnato inizialmente il valore *null*, per indicare che il riferimento non è ancora associato a un oggetto. Le variabili vengono inizializzate tramite l'operatore "*=*". Questo operatore assegna un valore ad una variabile. Il tipo della variabile deve essere compatibile col tipo del valore.

Dichiarazione e inizializzazione possono essere effettuate in una o più istruzioni, a discrezione del programmatore.

---

<sup>2</sup>Java è case sensitive quindi caratteri in upper e lower case sono interpretati come caratteri differenti.

```
// This characters are used to start a command
// declares the variable
int number;
// initializes a variable
number=0
```

---

Nell'esempio precedente la dichiarazione e l'assegnamento della variabile `number` sono effettuati in istruzioni differenti.

---

```
// This characters are used to start a command
// declares and initializes the variable
int number=0
```

---

In quest'ultimo esempio nella medesima istruzione `number` viene dichiarata e inizializzata.

#### 1.2.4 Creazione di un nuovo oggetto

La creazione di un nuovo oggetto si effettua con l'operatore `new`. Il metodo `new` costruisce un nuovo oggetto del tipo specificato e ritorna il suo riferimento => `NEW=indica che è un oggetto`

---

```
Car auto=new Car();
```

---

Effettuando `new Car()` viene creato un nuovo oggetto di tipo `auto` e viene ritornato il corrispettivo `reference`, che nel caso specifico viene assegnato alla variabile di riferimento `auto` mediante l'operatore di assegnamento `=`.

### 1.3 Costruttori

**Definizione 5.** **Costruttore** un costruttore è un metodo della classe che ha il *suo stesso nome* e non ha tipi o valori di ritorno. Lo scopo del costruttore è quello di creare un nuovo oggetto relativo a una classe.

- se in una classe non viene definito un costruttore ne viene automaticamente generato ed utilizzato uno di default senza argomenti che, all'atto della creazione dell'oggetto, richiama il costruttore di default senza argomenti della super classe.
- se però vengono creati dei costruttori il costruttore di default non risulta più disponibile a meno che definito dall'utente.
- l'uso di costruttori diversi permette di creare un tipo di oggetto passando argomenti diversi

Un costruttore alloca e inizializza

- alloca lo spazio per gli attributi di tipo *primitivo*
- alloca lo spazio per i *riferimenti* agli attributi definiti dall'utente
- inizializza i riferimenti e gli attributi. Se un riferimento non è inizializzato gli viene associato il valore `null`. Se una variabile numerica non è inizializzata gli viene associato il valore zero. Ai `boolean` viene assegnato il valore `false`

### 1.4 Array

- Un array è una variabile di tipo *reference*: contiene un reference a un area di memoria dove ci sono un insieme di variabili dello stesso tipo.

- un array ha una natura *statica*: esso mantiene la propria dimensione, ovvero il numero di elementi che contiene.

Vari tipi di array possono essere dichiarati:

- array monodimensionali (anche noti come vettori)
  - possono essere dichiarati come `type nomeVariabile[]` o come `type[] nomeVariabile` dove `type` indica il tipo di dato contenuti negli elementi dell'array mentre le parentesi poste dopo l'identificatore o dopo il tipo indicano che la variabile è di tipo array.
  - ovviamente in mancanza di *inizializzazione* la dichiarazione di un array non alloca spazio per gli elementi dell'array
  - l'allocazione si realizza come `int[] i=new int[10]`; dove 10 è il numero di elementi dell'array
- Per quanto riguarda gli array multidimensionali la dichiarazione e l'inizializzazione avviene in maniera analoga  
`float[][] f=new float[10][10];`

## 1.5 Semantic versioning

Solitamente, ogni applicazione è associata a più versioni. Il numero di versione tiene traccia di come l'applicazione evolve nel tempo. Per esempio Java 1.8 è una evoluzione di Java 1.7.

Solitamente a una applicazione è associato un numero di versione nella forma MAJOR.MINOR.PATCH, dove i campi hanno il seguente significato:

- MAJOR ogni volta che viene rilasciata una nuova versione con un cambiamento MAJOR significa che un cambiamento nelle API che rende l'applicazione incompatibile con la versione precedente è stata rilasciata.
- MINOR viene modificato ogni volta che vengono rilasciate nuove funzionalità in una maniera backward compatible
- PATCH viene eseguita un attività di bug fixing in maniera backward compatibile.

Regole per l'uso di semantic versioning:

- il software DEVE dichiarare delle API pubbliche, precise e comprensibili;
- MAJOR, MINOR e PATCH devono essere dei numeri NON negativi, e devono sempre incrementare, con la seguente eccezione: quando la versione MAJOR (MINOR) viene incrementata il valore di MINOR e PATCH (PATCH) viene settato a zero;
- una volta che una verione è rilasciata il contenuto della versione NON può essere modificato
- la versione 0.y.z è la versione iniziale del software. In questa versione le API pubbliche devono essere considerate non stabili. Tutto può cambiare nel tempo.
- la versione 1.0.0 è la prima versione del software che definisce le API pubbliche, da questo momento l'applicazione dipende dalle API pubbliche
- il valore di PATCH deve essere incrementato solo se backward compatible e bug fixes sono eseguiti. (Il bug fix corregge un comportamento scorretto)
- il valore di MINOR deve essere incrementato se sono aggiunte funzionalità introdotte nelle API pubbliche o se sono rimosse funzionalità deprecate. Dopo aver incrementato MINOR è necessario settare PATCH a zero
- il valore di MAJOR deve essere incrementato quando cambiamenti incompatibili sono introdotti nelle API pubbliche. PATCH e MINOR devono essere resettati a zero

- una pre-release viene indicata utilizzando un trattino seguita da un identificatore che specifica la patch version. Per esempio, 1.0.0-alpha.1, 1.0.0-0.3.7, 1.0.0-x.7.z.92.

La versione iniziale del software è in genere la 0.1.0. Da lì si inizia a utilizzare le regole sopra descritte. È tempo di passare alla versione 1.0.0 quando:

- il software viene utilizzato in produzione;
- si hanno delle API stabili che possono essere usati da altri.

Il lettore interessato può trovare informazioni addizionali in <http://semver.org/>.

## 1.6 Java versions history

JDK	improvements
JDK1.1	RMI, inner classes
JDK1.2	Swing, Collections
JDK1.3	CORBA, JNDI
JDK1.4	Regular expressions, logging, assert
JDK1.5	Generics, Autoboxing, Unboxing, Enumerations, concurrency utilities
JDK1.6	JDBC, GUI, compiler optimization IO
JDK1.7	String switch, type inference improvements, new IO libraries, new network protocols, and cryptography algorithms
JDK1.8	lambda expressions, map-reduce, annotations, static library linking with JNI

## 2 Esercizi

L'obiettivo di questi esercizi è di chiarire le caratteristiche di Java prima descritte.

### 2.1 Java è compilato ed è portabile

Questi esercizi hanno come fine quello di mostrare il fatto che Java è un linguaggio compilato ed è portabile.

#### 2.1.1 Esercizio

**Esercizio 1:** Scrivere un programma Java che stampa a video: “Benvenuto al corso di Programmazione ad Oggetti”

- **Creare un file Java**

- Aprire TextEdit (per gli utenti mac), Blocco Note (per gli utenti Windows) o vi (per gli utenti Linux).
- (Per gli utenti Mac da TextEdit selezionare *preferenze > Solo Testo > File > New*)
- Scrivere il codice Java seguente

---

```
public class Welcome{
    public static void main(String[] args){
        System.out.println("Benvenuto al corso di ingegneria del software");
    }
}
```

---

Se fai `System.out.println(args[0]);` ti da errore di esecuzione perché all'interno del vettore non c'è nulla.

- **Descrizione del file creato**

- **il codice definisce una classe** (class) chiamata `Welcome` con un singolo metodo chiamato `main`
- `public` è utilizzato per specificare che la classe è pubblica: è possibile accedervi dall'esterno, per esempio da altre classi.
- `la classe ha solo un metodo statico main`
- `void` specifica che il metodo non ritorna alcun valore
- `static` specifica che il metodo è statico: può essere invocato senza instanziare l'oggetto di classe (`Welcome`)
- `main` è il “metodo di partenza” che deve essere presente in almeno una classe affinché l'applicazione sia eseguibile.
- `args` contiene il parametro (anche chiamato argomento) del metodo `main`. Il parametro `args` contiene i valori dei parametri passati al metodo `main` quando il programma è eseguito da command line.
- `String[]` (array of String) è il tipo della variabile `args`
- `System.out.println` scrive il testo specificato sullo schermo.

- **Salvare il file** con nome `Welcome.java` (attenzione: non scegliere un nome differente e controllare che l'estensione del file sia `.java`).

- **Compilare il file Java:**

- aprire il Terminale (per gli utenti Mac e Linux) o il Prompt dei Comandi (per gli utenti Windows)
- portarsi nella cartella dove si trova il file creato mediante i seguenti comandi
  - \* per aprire una sottocartella e tornare alla cartella padre è possibile utilizzare i comandi `cd`, e `cd ..`, rispettivamente. Per mostrare il contenuto della cartella utilizzare il comando `ls` (`dir` per gli utenti Windows).
- Digitare `javac Welcome.java`
- il comando `javac` **compila** il file `Welcome.java` file e genera un file `.class` (o un `.jar`) che contiene il bytecode file che è possibile eseguire su qualunque piattaforma<sup>3</sup>. Questo processo supporta la **portabilità**.
- *Per eseguire con successo il comando javac è necessario avere una JDK (Java Development Kit) installata sul proprio pc<sup>4</sup>.* La JDK permette quindi di *compilare e eseguire* il programma creato.

- **Eseguire il file Java**

- il java bytecode (`.class`) può essere eseguito in qualsiasi architettura dove è installata una Virtual Machine (la quale fornisce un **interprete** per il Java bytecode). In particolare la Java JRE fornisce questa Virtual Machine.
- per eseguire il file bytecodefile eseguire il comando `java Welcome`
- questo comando esegue il `main` method della classe `Welcome`

## 2.1.2 Esercizio

**Esercizio 2:** Richiedere all'utente di inserire il proprio nome (per esempio Carlo) e stampare “Ciao Carlo, Benvenuto al corso di Programmazione ad Oggetti”

<sup>3</sup>Ricordiamo che il bytecode è una rappresentazione intermedia che può essere interpretata da una macchina virtuale Virtual Machine.

<sup>4</sup>Nota che la JDK include anche il Java Run-Time Environment

- **Creare un file Java**

- crea un file Java con nome Welcome
- scrivere il seguente codice Java → **N.B.: È CASE SENSITIVE**

---

```
import java.util.Scanner;

public class Welcome{
    public static void main(String[] args){
        Scanner scanner=new Scanner(System.in);
        System.out.println("Inserisci il tuo nome");
        String nome=scanner.nextLine();
        scanner.close();
        String welcome="Benvenuto al corso di Ingegneria del Software";
        System.out.println("Ciao "+nome+", "+welcome);
    }
}
```

**CONCATENAZIONE** => cioè stampa "Ciao" seguito dal nome inserito

---

- **Descrizione del file creato**

- `import` è utilizzata per importare la classe `Scanner` contenuta all'interno del package `java.util`
- `new` crea (instantiates) un nuovo oggetto di classe `Scanner`
- `scanner.nextLine()` legge la prossima riga inserita
- `scanner.close()` chiude lo scanner

↪ = DIRECTORY

- **Compilare ed eseguire il file creato**

- `javac Welcome.java`
- `java Welcome`

## 2.2 Java è orientato agli oggetti e staticamente tipizzato

Questa sezione presenta un insieme di esercizi che mostrano che Java è orientato agli oggetti e staticamente tipizzato.

### 2.2.1 Esercizio

**Esercizio 3:** Modellizzare in Java una bicicletta, una bicicletta (in un particolare istante temporale) ha una determinata velocità, che dipende dalla marcia inserita e dal ritmo di pedalata

Per progettare le vostre classi in Java è necessario rispondere a due quesiti:

- Come è possibile descrivere lo *stato* degli oggetti relativi a una determinata classe?
- Quali sono le *funzionalità* che gli oggetti di una determinata classe devono fornire?

---

```
// definisce una classe pubblica chiamata Bike
// Il nome della classe DEVE iniziare con una lettera maiuscola
public class Bike{
```

IMPORT = importa una classe scritta da qualcun'altro

ex. Scanner

"Ciao" + nome => CONCATENAZIONE => cioè stampa "Ciao" seguito dal nome inserito

JAVA: - è CASE SENSITIVE

- REFACTOR = Cambio nome del costruttore, automaticamente se l'ho usato da qualche parte, me lo modifica automaticamente.

Dichiarazione String:

1° metodo) String messaggioBenvenuto = new String ("Benvenuti");

2° metodo) String messaggioBenvenuto = "Benvenuti";

```
// gli attributi della classe sono utilizzati per descrivere lo stato
// dell'oggetto che verra' istanziato partendo dalla classe corrente e sono
// solitamente dichiarati privati o protetti.
// il nome degli attributi inizia con una lettera minuscola
private int gear=1; // default 1;
private int cadence; // default 0;
private int speed; // default 0;

// il costruttore della classe permette di creare una nuova bicicletta
// il costruttore della classe e' un metodo con lo stesso nome della classe che
// non ha un valore di ritorno
public Bike(){
}
}
```

---

## 2.2.2 Esercizio

**Esercizio 4:** Implementare un client per la classe bicicletta

Un Java si utilizza l'espressione client per indicare una generica classe, che utilizza la classe da noi progettata (nel nostro caso la classe Bicicletta).

```
public class Client{

    public static void main(String[] args){
        Bike bike1; //definisce un nuovo referece (puntatore) alla classe Bike
        // con un valore predefinito null
        bike1=new Bike(); //istanzia un oggetto
        // l'istruzione new Bike() crea un nuovo oggetto di tipo Bike e ritorna
        // il referece a questo oggetto che e' ritornato e assegnato alla
        // variabile bike1
        // quando il metodo new e' invocato la Virtual Machine alloca
        // dinamicamente la quantita' di memoria sufficiente a contenere
        // l'oggetto Bike
        Bike bike2=new Bike();
        // definisce e istanzia un oggetto di classe Bike

    }
}
```

---

- compilare ed eseguire il programma

- **Quesiti:**

- Qual è lo stato dell'oggetto dopo l'istruzione `Bike bike1;`?
   
*l'oggetto non ha uno stato, visto che non esiste esiste solamente il suo reference.*
- Qual è lo stato dell'oggetto `bike1` dopo l'istruzione `bike1 = new Bike ()`?
   
`gear=1, cadence=0, speed=0`

In genere gli attributi sono privati tranne qualche eccezione

- Qual è lo stato dell'oggetto bike1 dopo l'istruzione bike2 = new Bike () ?  
gear=1, cadence=0, speed=0
- Qual è lo stato dell'oggetto bike1 dopo l'istruzione bike2 = new Bike () ?  
gear=1, cadence=0, speed=0

### 2.2.3 Esercizio

**Esercizio 5:** Dire che cosa viene stampato quando viene eseguita la seguente classe

```
public class Client{

    public static void main(String[] args) {
        Bike bike1; //definisce un nuovo referece (puntatore) alla classe Bike
        // con un valore predefinito null
        bike1=new Bike(); //istanzia un oggetto
        // l'istruzione new Bike() crea un nuovo oggetto di tipo Bike e ritorna
        // il referece a questo oggetto che e' ritornato e assegnato alla
        // variabile bike1
        // quando il metodo new e' invocato la Virtual Machine alloca
        // dinamicamente la quantita' di memoria sufficiente a contenere
        // l'oggetto Bike
        Bike bike2=new Bike();
        // definisce e istanzia un oggetto di classe Bike
        System.out.println(bike1==bike2);
        // == confronta i reference bike1 e bike2
    }
}
```

---

- Che cosa stampa l'istruzione bike1==bike2?  
*false, visto che i references ai due oggetti bike1 e bike2 si riferiscono a due oggetti diversi*

### 2.2.4 Esercizio

**Esercizio 6:** Aggiungere alla bicicletta una funzionalità che permette all'oggetto di stampare il suo stato

```
//definisce una classe pubblica chiamata Bike
//Il nome della classe DEVE iniziare con una lettera maiuscola
public class Bike{

    // gli attributi della classe sono utilizzati per descrivere lo stato
    // dell'oggetto che verra' istanziato partendo dalla classe corrente e sono
    // solitamente dichiarati privati o protetti.
    // il nome degli attributi inizia con una lettera minuscola
    private int gear=1; // default 1;
    private int cadence; // default 0;
    private int speed; // default 0;
```

Private int gear;  $\Rightarrow$  Attributi generalmente la si indica con la lettera minuscola.

Private = sta a indicare che solo questa classe possono modificarli perché vogliamo evitare che le altre classi possano vederli e farli fare controlli.

Public = ogni classe può modificarli e vederli.

Protected = possono modificare tutte le classi solo quelli interni al PACKAGE MAIN.

```

// il costruttore della classe permette di creare una nuova bicicletta
// il costruttore della classe e' un metodo con lo stesso nome della classe che
// non ha un valore di ritorno
public Bike(){
}
    ↗ VALORE DI RITORNO, in questo caso non ritorna nulla.
public void printState(){
    // + is the String concatenation operator
    // the int gear, speed and cadence are automatically converted into a
    // String
    System.out.println("gear: "+gear+", speed: "+speed+", cadence: "+cadence);
}
}

```

## 2.2.5 Esercizio

**Esercizio 7:** Mettere la classe Bike nel package transport

- creare la cartella transport mkdir transport

```

package transport;

// definisce una classe pubblica chiamata Bike
// Il nome della classe DEVE iniziare con una lettera maiuscola
public class Bike{

    // gli attributi della classe sono utilizzati per descrivere lo stato
    // dell'oggetto che verra' istanziato partendo dalla classe corrente e sono
    // solitamente dichiarati privati o protetti.
    // il nome degli attributi inizia con una lettera minuscola
    private int gear=1; // default 1;
    private int cadence; // default 0;
    private int speed; // default 0;

    // il costruttore della classe permette di creare una nuova bicicletta
    // il costruttore della classe e' un metodo con lo stesso nome della classe che
    // non ha un valore di ritorno

    public Bike(){
    }

    public void printState(){
        // + is the String concatenation operator
        // the int gear, speed and cadence are automatically converted into a
        // String
        System.out.println("gear: "+gear+", speed: "+speed+", cadence: "+cadence);
    }
}

```

Public Int IncrementGear() {  
    ↑  
    VALORE DI  
    RITORNO

Public Bike() { = COSTRUTTORE → è un metodo, un qualcosa che costruisce, è un'insieme d'istruzioni e viene eseguito quando viene costruita una nuova bici.

## 2.3 Esercizio

**Esercizio 8:** Modificare il client affinche' chiami il metodo `printState()`

```
import transport.Bike2;

public class Client{

    public static void main(String[] args){
        Bike2 bike1; //definisce un nuovo referece (puntatore) alla classe Bike
        // con un valore predefinito null
        bike1=new Bike2(); //istanzia un oggetto
        // l'istruzione new Bike() crea un nuovo oggetto di tipo Bike e ritorna
        // il referece a questo oggetto che e' ritornato e assegnato alla
        // variabile bike1
        // quando il metodo new e' invocato la Virtual Machine alloca
        // dinamicamente la quantita' di memoria sufficiente a contenere
        // l'oggetto Bike
        Bike2 bike2=new Bike2();
        // definisce e istanzia un oggetto di classe Bike
        System.out.println(bike1==bike2);
        // == confronta i reference bike1 e bike2
        bike1.printState();
        // invoca il metodo printState sull'oggetto bike1
    }
}
```

Se cambiamo la cartella della classe `bike` dobbiamo cambiarne anche il package.

## 2.4 Eclipse Integrated Development Environment

La gestione di un progetto man mano che la complessità aumenta diventa sempre più difficile. Per questo, nello sviluppo software Integrated Development Environment (*IDE*) sono comunemente utilizzati. Gli *IDEs* rimuovono alcune delle difficoltà solitamente incontrate nel processo di sviluppo e forniscono funzionalità aggiuntive come il completamento automatico del codice. In questo corso e in particolare nel laboratorio useremo *Eclipse IDE* come ambiente di sviluppo<sup>5</sup>.

### 2.4.1 Esercizio

**Esercizio 9:** Crea il tuo primo progetto in Eclipse

- **Creare un progetto in eclipse**

- apri *Eclipse*
- non appena *Eclipse* si avvia richiede all'utente di selezionare il *Workspace*. Il *Workspace* è l'area di lavoro dell'utente, contiene i file creati etc. L'utente può definire vari *Workspaces* ognuno dei quali può contenere più progetti.

<sup>5</sup>*Netbeans* è uno dei maggiori competitor di *Eclipse*

- clicca su *Workbench*
- File > New > Project > Maven Project
- seleziona Create a simple project
- digita ProgrammazioneAdOggetti come Group Id e Esercitazione1 come Artifact Id
- scegli finish
- Quando create il vostro nuovo progetto Maven<sup>6</sup> in Eclipse, Eclipse crea automaticamente la struttura del vostro progetto. In particolare,
  - \* *src/main/java* contiene il codice sorgente Java dell'applicazione in via di sviluppo.
  - \* *src/main/resources* contiene le "risorse" utilizzate dalla vostra applicazione, file di configurazione, etc...
  - \* *src/test/java* contiene le classi usate per testare la vostra applicazione
  - \* *src/test/resources* contiene le risorse necessarie a testare la vostra applicazione

- **Creare un package**

- tasto destro su *src/main/java* > New > package
- scegliere il nome del package (*esercizio10*)
- premere finish

- **Creare una nuova classe in Eclipse**

- tasto destro sul package dove si desidera creare la classe > New > class
- scegliere il nome della classe (*Bike*)
- scegliere i modificatori di accesso della classe etc...
- premere finish

Come noti Eclipse automaticamente aggiunge il package, gli identificatori etc...

- **Running a project in Eclipse**

- cliccare con il tasto destro sulla classe che si desidera eseguire (*deve contenere il metodo main*)
- Run As > Java Application

- **Che cosa succede?**

- se apriamo il workspace notiamo che il nostro progetto contiene varie cartelle:
  - \* *src*: contiene i file sorgenti delle nostre applicazioni (.java files)
  - \* *target*: contiene i file che contengono il bytecode (.class files) che sono generati dopo aver eseguito la procedura appena descritta. La compilazione ed esecuzione è resa trasparente all'utente

## 2.5 Java è orientato agli oggetti e staticamente tipizzato

### 2.5.1 Esercizio

**Esercizio 10:** Deve essere possibile monitorare la velocità la marcia inserita e il ritmo di pedalata di una bicicletta

<sup>6</sup>Maven sarà spiegato nel corso del laboratorio.

```

package esercizio10;

// definisce una classe pubblica chiamata Bike
// Il nome della classe DEVE iniziare con una lettera maiuscola
public class Bike {
    // gli attributi della classe sono utilizzati per descrivere lo stato
    // dell'oggetto che verra' istanziato partendo dalla classe corrente e sono
    // solitamente dichiarati privati o protetti.
    // il nome degli attributi inizia con una lettera minuscola
    private int gear = 1; // default 1;
    private int cadence; // default 0;
    private int speed; // default 0;

    // il costruttore della classe permette di creare una nuova bicicletta
    // il costruttore della classe e' un metodo con lo stesso nome della classe
    // che non ha un valore di ritorno

    public Bike() {
    }

    public void printState() {
        // + is the String concatenation operator
        // the int gear, speed and cadence are automatically converted into a
        // String
        System.out.println("gear: " + gear + ", speed: " + speed + ", cadence: "
                           + cadence);
    }

    // returns the gear of the bike
    public int getGear(){
        return this.gear;
    }

    // returns the cadence of the bike
    public int getCadence(){
        return this.cadence;
    }

    // returns the speed of the bike
    public int getSpeed(){
        return this.speed;
    }
}

```

---

Il “Monitoring” di un oggetto è eseguito mediante i metodi getters.

Command tricks:

- il comando Control + Space permette di completare automaticamente il testo e fornisce suggerimenti al riguardo

---

```

package esercizio10;

public class Client {

```

N. B: Metodi che è sempre meglio fare SEMPRE!

GET = Accede allo stato della cadenza

Public getCadence () { // serve per vedere lo stato della bicicletta

    return Cadence;

}

SET = Modifica lo stato della cadenza

Public setCadence () {

    this.cadence = cadence;

}

```
public static void main(String[] args) {  
  
    Bike bike1;  
  
    bike1=new Bike();  
  
    Bike bike2=new Bike();  
  
    // == compare the reference of the Bike1 with the reference of Bike2  
    System.out.println(bike1==bike2);  
  
    // invoca il metodo print state su Bike1  
    bike1.printState();  
  
    // invoca il metodo print state su Bike2  
    bike2.printState();  
  
    System.out.println(bike1.getCadence());  
  
}  
}
```

---

## 2.5.2 Esercizio

**Esercizio 11:** Deve essere possibile incrementare la marcia inserita e il ritmo di pedalata, in tal caso la velocità deve aumentare di conseguenza

```
package esercizioli;  
  
//definisce una classe pubblica chiamata Bike  
//Il nome della classe DEVE iniziare con una lettera maiuscola  
public class Bike {  
    // gli attributi della classe sono utilizzati per descrivere lo stato  
    // dell'oggetto che verra' istanziato partendo dalla classe corrente e sono  
    // solitamente dichiarati privati o protetti.  
    // il nome degli attributi inizia con una lettera minuscola  
    private int gear = 1; // default 1;  
    private int cadence; // default 0;  
    private int speed; // default 0;  
  
    // il costruttore della classe permette di creare una nuova bicicletta  
    // il costruttore della classe e' un metodo con lo stesso nome della classe  
    // che non ha un valore di ritorno  
  
    public Bike() {  
    }  
  
    public void printState() {  
        // + is the String concatenation operator
```

```

        // the int gear, speed and cadence are automatically converted into a
        // String
        System.out.println("gear: " + gear + ", speed: " + speed + ", cadence: "
            + cadence);
    }

    // returns the gear of the bike
    public int getGear(){
        return this.gear;
    }

    // returns the cadence of the bike
    public int getCadence(){
        return this.cadence;
    }

    // returns the speed of the bike
    public int getSpeed(){
        return this.speed;
    }

    // changes the cadence of the bike
    public void changeCadence(int cadence){
        this.cadence=cadence;
        this.updateSpeed();
    }
    // increments the gear
    public void incrementGear(){
        this.gear++;
        this.updateSpeed();
    }

    // decrements the gear
    public void decrementGear(){
        this.gear--;
        this.updateSpeed();
    }

    // updates the speed of the Bike
    private void updateSpeed(){
        this.speed=this.gear*this.cadence;
    }
}

```

---

```

package esercizio11;

public class Client {

    public static void main(String[] args) {

        Bike bike1;

```

Public Int IncrementGear() {  
    ↑  
    VALORE DI  
    RITORNO

this = attributo GEAR di questa classe è uguale alla variabile che gli passiamo tra parentesi.

```

bikel=new Bike();

Bike bike2=new Bike();

// == compare the reference of the Bikel with the reference of Bike2
System.out.println(bikel==bike2);

// invoca il metodo print state su Bikel
bikel.printState();

// invoca il metodo print state su Bike2
bike2.printState();

System.out.println(bikel.getCadence());

}

}

```

---

- **Quesiti**

- Qual è lo stato dell'oggetto bike1 dopo che l'istruzione bike1.incrementGear(); è eseguita?  
gear=2, cadence=0, speed=0.
- Qual è lo stato dell'oggetto bike1 dopo che l'istruzione bike1.changeCadence(10); è eseguita?  
gear=2, cadence=10, speed=0.

### 2.5.3 Esercizio

**Esercizio 12:** La classe di biciclette considerate ha 6 marce

```

package esercizio12;

//definisce una classe pubblica chiamata Bike
//Il nome della classe DEVE iniziare con una lettera maiuscola
public class Bike {
    // gli attributi della classe sono utilizzati per descrivere lo stato
    // dell'oggetto che verra' istanziato partendo dalla classe corrente e sono
    // solitamente dichiarati privati o protetti.
    // il nome degli attributi inizia con una lettera minuscola
    private int gear = 1; // default 1;
    private int cadence; // default 0;
    private int speed; // default 0;

    // static fields specify that this variable is shared from all the objects of
    // the class
    // the final modifier indicates that the value of this field cannot be change
    // the static modifier in combination with the final modifier is also used to
    // define constants
    private static final int MAX_GEAR=6;
    private static final int MIN_GEAR=1;
}

```

Private static Bike bike = null;  $\Rightarrow$  Variabile STATICÀ esiste sempre.

```
// il costruttore della classe permette di creare una nuova bicicletta
// il costruttore della classe e' un metodo con lo stesso nome della classe
// che non ha un valore di ritorno

public Bike() {
}

public void printState() {
    // + is the String concatenation operator
    // the int gear, speed and cadence are automatically converted into a
    // String
    System.out.println("gear: " + gear + ", speed: " + speed + ", cadence: "
        + cadence);
}

// returns the gear of the bike
public int getGear(){
    return this.gear;
}

// returns the cadence of the bike
public int getCadence(){
    return this.cadence;
}

// returns the speed of the bike
public int getSpeed(){
    return this.speed;
}

// changes the cadence of the bike
public void changeCadence(int cadence){
    this.cadence=cadence;
    this.updateSpeed();
}

// increments the gear
public void incrementGear(){
    if(gear<MAX_GEAR) {
        this.gear++;
        this.updateSpeed();
    }
}

// decrements the gear
public void decrementGear(){
    if(gear>MIN_GEAR) {
        this.gear--;
        this.updateSpeed();
    }
}

// updates the speed of the Bike
```

```
    private void updateSpeed() {
        this.speed=this.gear*this.cadence;
    }
}
```

---

## 2.6 Variabili e tipi di riferimento

### 2.6.1 Esercizio

**Esercizio 13:** Analizzare che cosa succede eseguendo il seguente codice

```
package esercizio13;

public class Client {

    public static void main(String[] args) {

        // dichiaro una variabile di tipo reference con identificativo bike1
        Bike bike1;
        // creo un nuovo oggetto di tipo Bike e assegno il reference ritornato alla
        // variabile bike1
        bike1 = new Bike();
        // creo un nuovo oggetto di tipo Bike e assegno il reference ritornato alla
        // variabile bike2
        Bike bike2 = new Bike();

        // vengono stampati due reference diversi relativi alle variabili bike1 e bike2
        System.out.println(bike1);
        System.out.println(bike2);

        // copia il reference contenuto nella variabile bike2 nella variabile bike1
        bike1 = bike2;

        // vengono stampati i due reference
        System.out.println(bike1);
        System.out.println(bike2);
    }
}
```

---

Per avere una visualizzazione migliore di quanto accade utilizzeremo il debugger di eclipse. Il debugger vi consente di eseguire passo passo il vostro codice e vi assiste nell'individuazione e nella correzioni di errore in un programma.

- cliccare con il tasto destro del mouse sulla barra vertical a fianco a `public static void main`
- cliccare su `toggle breakpoint`
- cliccare con il tasto destro sulla nostra classe (`Client.java`)
- cliccare su `debug as Java Application`
- se viene chiesto se aprire la perspective di debug rispondere Yes

Nella View variables è possibile vedere passo passo le variabili sullo stack. Se una variabile è di tipo reference è possibile esplorare il corrispondente oggetto (e di conseguenza lo heap)

È possibile proseguire nell'esecuzione del codice (un passo alla volta) premendo sul pulsante step over (la freccia gialla posta al centro delle le tre in alto a sinistra)<sup>7</sup>.

## 2.7 Costruttori

### 2.7.1 Esercizio

**Esercizio 14:** Modificare la classe bicicletta affinchè sia possibile creare una bicicletta con un numero di marce un dato ritmo di pedalata

```
package esercizio14;

//defines a public class called Bike
//the name of the class usually starts with an upper case letter
public class Bike {

    // static specifies that this variable is shared from all the objects
    // The final modifier indicates that the value of this field cannot change.
    // The static modifier, in combination with the final modifier, is also used to
    // define constants.
    private static final int MAX_GEAR = 6;
    private static final int MIN_GEAR = 1;

    // The attributes of the class are used to describe the state of the class
    // and are usually private or protected
    // The attributes of the class usually start with lower case letters
    private int gear = 1; // default 1
    private int cadence; // default 0
    private int speed; // default 0

    // is the constructor of the class which allows to create a new Bike
    // the constructor has the same name of the class and does not have a return
    // type
    // gear and cadence are the intial values of the gear and the cadence
    public Bike(int gear, int cadence) {

        // this allows to refer to the variable gear of THIS class, otherwise
        // gear refers to the variable passed as parameter
        this.gear=gear;
        this.cadence=cadence;
        this.updateSpeed();
    }

    public void printState() {
        // + is the String concatenation operator
        // the int gear, speed and cadence are automatically converted into
        // String
        System.out.println("gear: " + gear + ", speed: " + speed

```

---

<sup>7</sup>Un utilizzo più dettagliato del debugger verrà descritto nel corso del laboratorio

```

        + ", cadence: " + cadence);
    }

// changes the cadence of the bike
public void changeCadence(int cadence) {
    this.cadence = cadence;
    updateSpeed();
}

// increment the gear
public void incrementGear() {
    if (gear < MAX_GEAR) {
        gear++;
        updateSpeed();
    }
}

// decrement the gear
public void decrementGear() {
    if (gear > MIN_GEAR) {
        gear--;
        updateSpeed();
    }
}

// every time the cadence or the gear changes the speed is updated
// the method is private and cannot be invoked from other external classes
private void updateSpeed() {
    this.speed = this.gear * this.cadence;
}

// returns the gear of the bike
public int getGear() {
    return this.gear;
}

// returns the cadence of the bike
public int getCadence() {
    return this.cadence;
}

// returns the speed ok the bike
public int getSpeed() {
    return this.speed;
}
}



---


package esercizio14;

public class Client {

    public static void main(String[] args) {

```

```
Bike bike1;
bike1 = new Bike(3, 2);
Bike bike2 = new Bike(1, 2);

// == compare the reference of the Bike1 with the reference of the Bike2
System.out.println(bike1 == bike2);
// invokes the method print state of the object bike1
bike1.printState();
// invokes the method print state of the object bike2
bike2.printState();

bike1.printState();
bike1.incrementGear();
bike1.printState();
bike1.changeCadence(10);
bike1.printState();
}
}
```

---

**Quesiti:**

- Qual è lo stato dell'oggetto bike1 dopo che l'istruzione bike1=new Bike(3, 2); è eseguita? gear=3, cadence=2, speed=0.
- Qual è lo stato dell'oggetto bike2 dopo che l'istruzione Bike bike2=new Bike(1, 2); è eseguita? gear=1, cadence=2, speed=0.
- Qual è lo stato dell'oggetto bike1 dopo che l'istruzione bike1.incrementGear(); è eseguita? gear=3, cadence=2, speed=0.
- Qual è lo stato dell'oggetto bike1 dopo che l'istruzione bike1.changeCadence(10); è eseguita? gear=3, cadence=10, speed=0.

**2.7.2 Esercizio**

**Esercizio 15:** Analizzare che cosa succede eseguendo il seguente codice

---

```
package esercizio15;

public class Client {

    public static void creatingANewBike() {
        // dichiaro una variabile di tipo reference con identificativo bike1
        Bike bike1;
        // creo un nuovo oggetto di tipo Bike e assegno il reference ritornato
        // alla variabile bike1
        bike1 = new Bike(4, 5);
    }

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

Con Public Static non c'è bisogno di dichiarare l'oggetto.

```

// dichiaro una variabile di tipo reference con identificativo bike1
Bike bike1;
// creo un nuovo oggetto di tipo Bike e assegno il reference ritornato
// alla variabile bike1
bike1 = new Bike(4, 5);
// creo un nuovo oggetto di tipo Bike e assegno il reference ritornato
// alla variabile bike2
Bike bike2 = new Bike(2, 3);

// vengono stampati due reference diversi relativi alle variabili bike1
// e bike2
System.out.println(bike1);
System.out.println(bike2);

// copia il reference contenuto nella variabile bike2 nella variabile
// bike1
bike1 = bike2;

// vengono stampati i due reference
System.out.println(bike1);
System.out.println(bike2);

creatingANewBike();
}
}

```

---

### 2.7.3 Esercizio

**Esercizio 16:** Modificare la classe bicicletta affinchè sia possibile creare una bicicletta con un dato numero di marce

```

package esercizio16;

//defines a public class called Bike
//the name of the class usually starts with an upper case letter
public class Bike {

    // static specifies that this variable is shared from all the objects
    // The final modifier indicates that the value of this field cannot change.
    // The static modifier, in combination with the final modifier, is also used to
    // define constants.
    private static final int MAX_GEAR = 6;
    private static final int MIN_GEAR = 1;

    // The attributes of the class are used to describe the state of the class
    // and are usually private or protected
    // The attributes of the class usually start with lower case letters
    private int gear = 1; // default 1
    private int cadence; // default 0
    private int speed; // default 0
}

```

```
// is the constructor of the class which allows to create a new Bike
// the constructor has the same name of the class and does not have a return
// type
// gear and cadence are the intial values of the gear and the cadence
public Bike(int gear, int cadence) {

    // this allows to refer to the variable gear of THIS class, otherwise
    // gear refers to the variable passed as parameter
    this.gear=gear;
    this.cadence=cadence;
    this.updateSpeed();
}

public Bike(int gear) {

    // this allows to refer to the variable gear of THIS class, otherwise
    // gear refers to the variable passed as parameter
    this.gear=gear;
    this.cadence=0;
    this.speed=0;
}

public void printState() {
    // + is the String concatenation operator
    // the int gear, speed and cadence are automatically converted into
    // String
    System.out.println("gear: " + gear + ", speed: " + speed
        + ", cadence: " + cadence);
}

// changes the cadence of the bike
public void changeCadence(int cadence) {
    this.cadence = cadence;
    updateSpeed();
}

// increment the gear
public void incrementGear() {
    if (gear < MAX_GEAR) {
        gear++;
        updateSpeed();
    }
}

// decrement the gear
public void decrementGear() {
    if (gear > MIN_GEAR) {
        gear--;
        updateSpeed();
    }
}
```

```
// every time the cadence or the gear changes the speed is updated
// the method is private and cannot be invoked from other external classes
private void updateSpeed() {
    this.speed = this.gear * this.cadence;
}

// returns the gear of the bike
public int getGear() {
    return this.gear;
}

// returns the cadence of the bike
public int getCadence() {
    return this.cadence;
}

// returns the speed ok the bike
public int getSpeed() {
    return this.speed;
}
}
```

---

## 2.8 Array

### 2.8.1 Esercizio

**Esercizio 17:** Creare un array di bike, riempirlo e stampare la più veloce

---

```
package esercizio17;

public class ArrayOfBikes {
    private static void printFastBikes(Bike [] bikes) {
        for (Bike bike: bikes) {
            if (bike != null && bike.getSpeed() > 10)
                bike.printState();
        }
    }
    public static void main(String [] args) {
        // definiamo una dimensione fissa
        // tutti gli elementi degli array hanno valore null
        Bike [] arr1 = new Bike[10];
        Bike [] arr2 = new Bike[10];

        arr1[0] = new Bike(3, 20);
        arr1[1] = new Bike(3, 10);
        arr1[2] = new Bike(1, 1);
        System.out.println("Print fast bikes in arr1");
        printFastBikes(arr1);
        int k = 0;
        for (Bike b: arr1){
```

```
    if (b!= null) {
        Bike b2 = new Bike(b.getGear(),b.getSpeed());
        b2.changeCadence(b2.getCadence() + 5);
        arr2[k++] = b2;
    }
}
System.out.println("Print fast bikes in arr2");
printFastBikes(arr2);
}
}
```

---

### 3 Curiosity

Il nome Java riflette la passione dei teams di sviluppatori per il caffè. Java è un particolare tipo di chicco di caffè prodotto nell'isola Indonesiana di Java.