Finanziato dall'Unione europea
NextGenerationEU

MINISTERO del LAVORO e delle POLITICHE SOCIALI

REGIONE PIEMONTE

GOL Garanzia Occupabilità Lavoratori

DISPENSA DIDATTICA

Corso: B22-418-2025

Tecnico dello sviluppo e progettazione dei programmi informatici

IMMAGINAZIONE E LAVORO DAL 1978

COESIONE ITALIA 21-27 PIEMONTE

Cofinanziato dall'Unione europea

REGIONE PIEMONTE

Java™

Cos'è JAVA

Java è un linguaggio di alto livello e orientato agli oggetti, creato dalla SunMicrosystem nel 1995.

Le motivazioni, che guidarono lo sviluppo di Java, erano quelle di creare un linguaggio semplice e familiare.

Le caratteristiche del linguaggio di programmazione Java sono:

- La tipologia di linguaggio **orientato agli oggetti**(ereditarietà, polimorfismo, incapsulamento...)
- la **gestione della memoria** effettuata automaticamente dal sistema, il quale si preoccupa dell'allocazione e della successiva deallocazione della memoria (il programmatore viene liberato dagli obblighi di gestione della memoria
- la **portabilità**, cioè la capacità di un programma di poter essere eseguito su piattaforme diverse senza dover essere e modificato e ricompilato

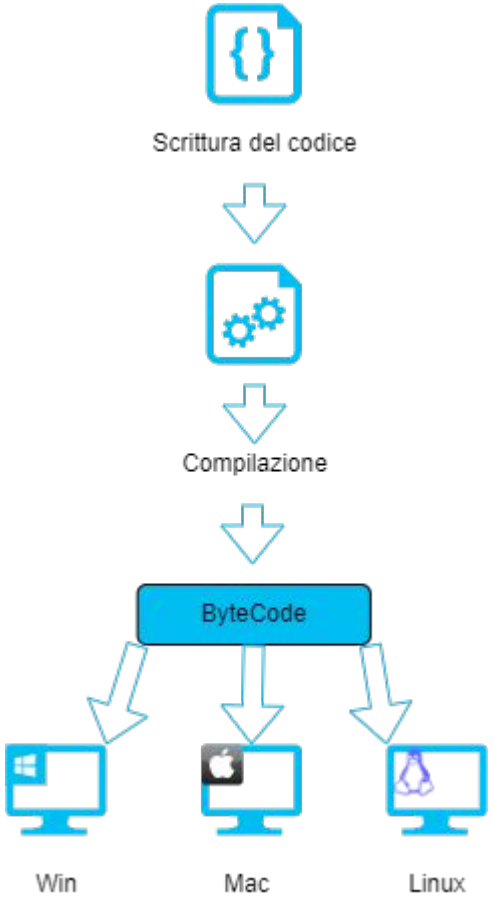
Caratteristiche Principali

- Semplice e familiare
- Orientato a oggetti
- Indipendente dalla piattaforma
- Interpretato
- Sicuro
- Robusto

Indipendente dalla piattaforma

JVM: Java Virtual Machine

Interprete disponibile per la maggior parte dei sistemi operativi Desktop e Mobile. Permette la “portabilità”



Vantaggi

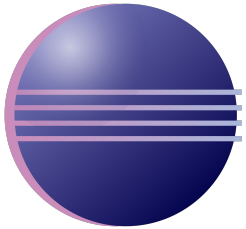
- Completamente gratuito.
- Possibilità di scrivere codice su qualsiasi piattaforma per poter essere eseguito su qualunque piattaforma (scrivo su Win, eseguo su Android).
- I software possono essere eseguiti anche sui browser web, i browser più utilizzati possiedono una JVM interna.
- Possibilità di scrivere per qualsiasi dispositivo, dalle lavatrici agli ultimi smartphone in commercio.

Cosa Ci Serve

- JDK - Java Development Kit. Strumenti di sviluppo per esecuzione, compilazione e debug di un’applicazione

OpenJDK

- IDE - Integrated Development Environment



Il Linguaggio

Identificatori, convenzioni, regole e Tipi

Identificatori

L’identificatore è il nome che forniamo ad un package, una classe, un metodo, una variabile.

Caratteri consentiti:

- Lettere alfabeto
- Numeri (ATT: non possono cominciare con un numero)
- Underscore (_)
- Dollaro ($)

Parole vietate per denominare un identificatore: tutte le keyword di Java, inclusi true false

abstarct	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Convenzioni per **classi**:

- iniziale Maiuscola (es: `class Studente`)
- lettere successive Maiuscola
- PascalCase

Convenzioni per **metodi**:

- camelCase (`public void studiaJava()`)
- solitamente è un verbo

Convenzioni per **variabili**

- camelCase
- snake_case

Tutto deve essere “parlante” affinché qualsiasi altro sviluppatore, nel leggere il codice, capirà di cosa si tratta.

Variabili e Tipi

Compito delle variabili è conservare temporaneamente delle informazioni che servono allo script per eseguire il proprio lavoro. Il nome “variabile” è perfetto poiché il valore può variare ad ogni esecuzione del codice.

Caratterizzata da:

Identificativo:
nome della
variabile

Tipo:
tipologia di dato
che la variabile può
contenere

Tipi Primitivi

Interi:
- byte
- short
- int
- long

Virgola mobile:
- float
- double

Testuali:
- char

Logici:
- true
- false

Tipi per Oggetti

Array

Object:
- **String**

I tipi di dati ***interi*** sono quattro: **byte, short, int, long**.

Tutti possono immagazzinare numeri interi, positivi o negativi, la differenza è nell'intervallo di rappresentazione, ovvero la quantità di byte immagazzinabili.

- 1 byte può immagazzinare un intero utilizzando otto bit (2^8 = 256 informazioni diverse)
- 1 short due byte,
- 1 int quattro byte
- 1 un long otto byte

byte 8 bit -128,..,+127

short 16 bit -32.768,..,+32.767

int 32 bit -2.147.483.648,..,+2.147.483.647

long 64 bit -9.223.372.036.854.775.808,..,9.223.372.036.854.775.807

Perché tutte queste suddivisioni ?

I tipi di dati a **virgola mobile** sono due: **float**, **double**

float che gestisce valori frazionari fino a 32 bit con precisione singola e Range da 3 .4e−038 a 3.4e+038 e 7 cifre decimali

double che, come si può intuire facilmente dal nome, gestisce invece valori frazionari fino a 64 bit con precisione doppia e Range da 1 .7e−308 a 1.7e+308 con 15 cifre decimali.

Il tipo di dato **boolean**, utilizza solo un bit per memorizzare un valore, e gli unici valori che può immagazzinare sono true e false.

Array

Per memorizzare un intero elenco di valori correlati. Questi valori dovranno essere simili tra loro, per cui anche in questo caso bisogna specificare il tipo di dato che conterrà.

```
String[] colors = new String[3];
colors[0] = “bianco”;
colors[1] = “rosso”;
colors[2] = “verde”;
```

Dichiarazione alternativa

```
String[] colors = {“bianco”, “rosso”, “verde”}
```

Control Flow

If, For, Switch, While

Operatori di Confronto

==	→	è uguale a
!=	→	non è uguale a
>	→	maggiore di
<	→	minore di
>=	→	maggiore uguale
<=	→	minore uguale

Operatori Logici

&& → AND logico

|| → OR logico

! → NOT logico

Operatori Matematici

+ * - / ++ -- %

IF …ELSE statement

L’istruzione if valuta una condizione. Se questa condizione è true allora viene eseguito il blocco di codice successivo, altrimenti viene eseguito il blocco di codice successivo all’ Else

```
if (ore > 10  ){
      fai qualcosa..
}else{
       fai altro…
}
```

Switch Statement

L’istruzione switch inizia con una variabile di controllo, seguita dai vari casi.

```
switch(condizione){
   case 1:
      faiqualcosa;
   break;
   case 2:
       faialtro;
   break;
   case 3:
       faialtroancora
   break;
}
```

Cicli

Anche i cicli si basano su una condizione. Se questa condizione è verificata allora viene eseguito il blocco di codice compreso tra le parentesi graffe. Successivamente la condizione viene ricontrollata e se è ancora true viene ripetuto il codice, questo finché la condizione non diventa false.

FOR

Esegue un blocco di codice per un numero determinato di volte

WHILE

Il codice continuerà ad essere eseguito finché la condizione resterà true

DO...WHILE

simile al while il codice nel do viene eseguito almeno una volta, successivamente viene verificata la condizione

FOR

```
for (var i = 0; i < 10; i++) {
    document.write(i);
}
```

var i = 0 → **Inizializzazione**

i < 10 → **Condizione**

i++ → **Aggiornamento**

WHILE

```
while (condizione){

    …questo codice viene eseguito
fino a quando la condizione
risulta essere true


}
```

Programmazione OOP

Object Oriented Programming

OOP

La programmazione orientata agli oggetti (in inglese Object Oriented Programming o OOP) è un paradigma di programmazione nato nella seconda metà degli anni '50 e l'inizio degli anni '60 nei laboratori del **M.I.T.**

La programmazione a oggetti permette di:

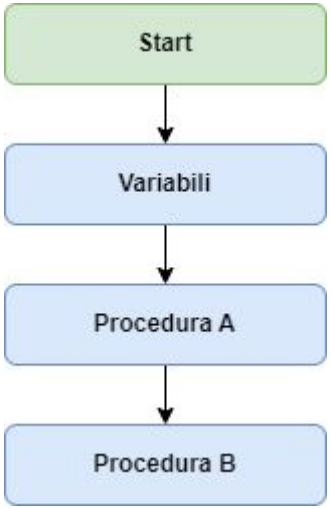
- rappresentare un problema o delle entità reali attraverso oggetti software
- stabilire le relazioni che intercorrono tra le entità

Smartphone
Display
Cpu
Memoria

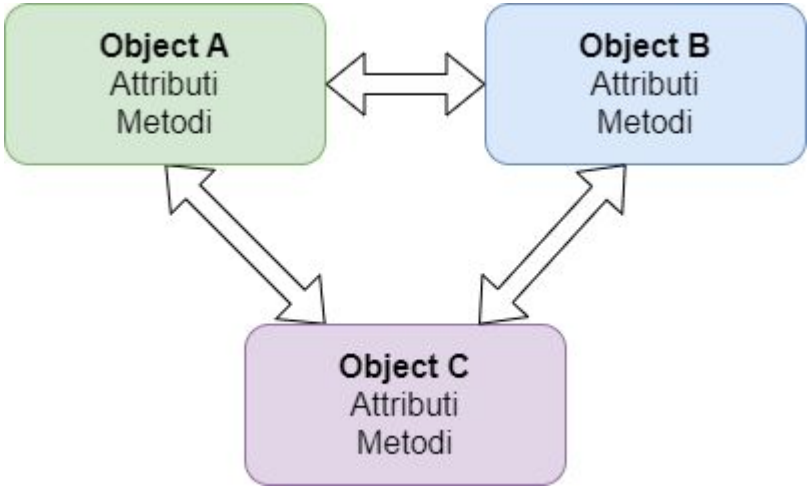
Software Gestionale
Clienti
Articoli
Ordini

Dal Procedurale alla OOP

Procedurale:



Programmazione ad oggetti:



Questo approccio permette di semplificare la risoluzione di problemi andando a suddividere i compiti assegnandoli ai singoli **Oggetti.**

Alcuni Vantaggi

- Supporto alla modellazione: rappresentazione di oggetti del mondo reale
- Alta manutenibilità
- Sviluppo modulare

Oss: Java utilizza la programmazione procedurale all'interno degli oggetti stessi. L'oggetto può essere inteso come una micro applicazione che interagisce con altri oggetti a sua volta.

L'utilizzo di questo paradigma è stato fondamentale per la crescita dell'utilizzo di Java in tantissimi ambiti.

Garbage Collector

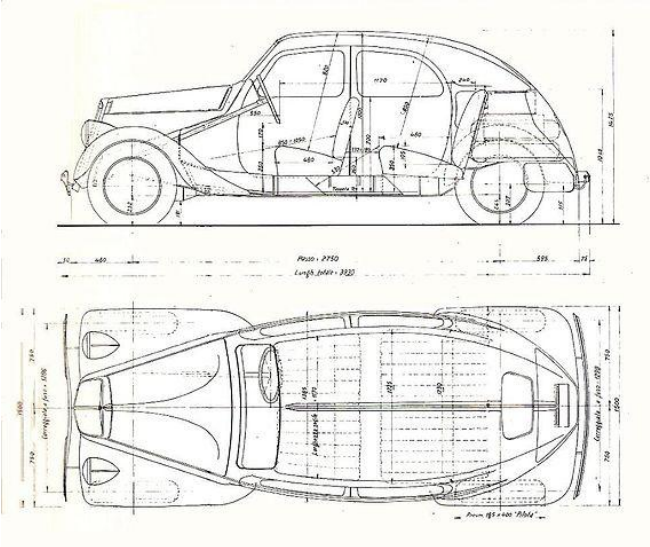
Strumento utilizzato per la gestione della memoria.

Il garbage collector traccia le variabili inutilizzate e le distrugge in quanto non più necessarie.

Le Classi

Per poter creare e quindi utilizzare un oggetto ho bisogno di un progetto, del blueprint della mia Auto. Proprio questo è ciò che rappresenta una Classe.

La classe quindi è un **tipo di dato complesso**



La **classe** definisce le caratteristiche di un oggetto proprio come il progetto di un'auto, il progetto di una casa, lo stampo per i biscotti.

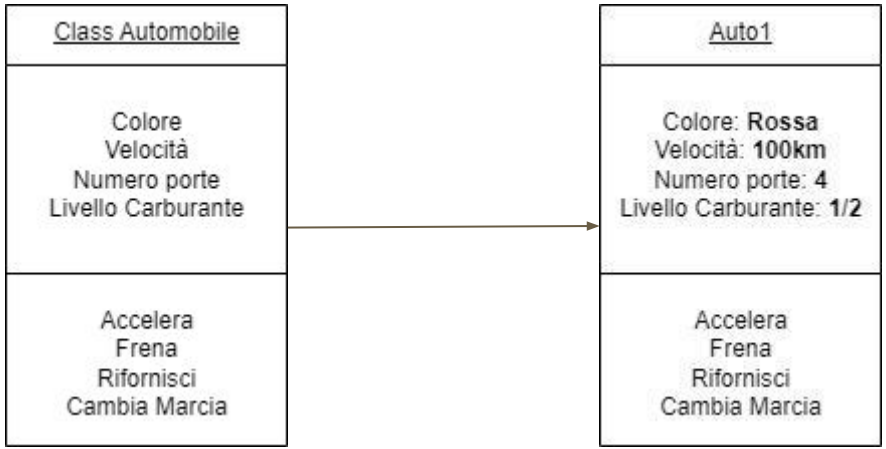
Con lo stesso progetto realizzo migliaia di auto dello stesso tipo ma tutte con caratteristiche diverse.

Per cui posso avere più oggetti dello stesso tipo, tutti diversi tra loro (dal punto di vista delle caratteristiche) ma tutti saranno **istanze della stessa classe**.

“Una **classe** è un’astrazione indicante un insieme di oggetti che condividono le stesse caratteristiche e le stesse funzionalità”

Un oggetto in Java può esistere solo se esiste la relativa classe. La creazione dell’oggetto è chiamata: **istanza della classe.**

“Un **oggetto** è un’istanza (ovvero una creazione fisica) di una classe”



Gli Oggetti

Un oggetto è un modello di entità presente nel mondo reale.

Attributi →

Colore
Cilindrata
Num Porte (const)
Livello Carburante

Metodi →

Accelera
Frena
Cambia Marcia
Rifornisci

OSS: alcuni metodi vanno direttamente a modificare alcuni attributi. Es (Rifornisci())

I Metodi

I metodi sono un insieme di istruzioni racchiuse in un unico blocco: rendono i programmi più leggibili e di più facile manutenzione, lo sviluppo più veloce e stabile, evitano le duplicazioni e favoriscono il riuso del codice.

Dichiarazione di un metodo

```
[modificatori] tipo_di_ritorno nome_metodo ([parametri]){

    corpo_del_metodo

}
```

OSS: in altri linguaggi i “metodi” di Java sono paragonabili alle “funzioni” o alle “subroutine”

- **Modificatori**: parole chiave di Java utilizzate per modificare le funzionalità, le caratteristiche, la visibilità di un metodo.
- **Tipo di ritorno**: è il tipo del dato che un metodo può restituire al termine della sua esecuzione. Può essere un tipo primitivo (int, boolean, ecc) oppure un tipo complesso come un oggetto. Quando il metodo non restituisce nulla il tipo è dichiarato **void** come nel caso del metodo main.
- **Nome del metodo**: identificatore.
- **Parametri**: variabili che potranno essere passate al metodo in input e sfruttate nel corpo del metodo. OSS: nome metodo e parametri costituiscono la **firma del metodo** o **signature.**
- **Corpo del metodo**: l'insieme di istruzioni che verranno eseguite all'invocazione del metodo.

Il metodo costruttore

Particolare metodo con le seguenti caratteristiche:

- ha lo stesso nome della classe;
- non ha tipo di ritorno;
- è chiamato automaticamente (e solamente) ogni volta che è istanziato un oggetto, relativamente a quell'oggetto;
- è presente in ogni classe;
- solitamente un metodo costruttore viene definito allo scopo di inizializzare le variabili d'istanza.

```
public class Punto{
    //metodo costruttore
    public Punto(){
        System.out.println("Costruito un Punto!");
    }
    public int x;
    public int y;
}
```

This

Utilizzato all'interno di metodi o costruttori fa riferimento all'oggetto corrente, andando a risolvere problemi di ambiguità nella definizione dei parametri.

Esempio

```
public class Cliente {
private String nome, indirizzo, numeroDiTelefono;

public void setDati (String nome, String indirizzo,String numeroDiTelefono){

        this.nome = nome;
        this.indirizzo = indirizzo;
        this.numeroDiTelefono = numeroDiTelefono;

    }
}
```

Modificatori

I modificatori sono keyword di Java. Scritto prima di un qualsiasi componente (variabile, classe, metodo) è in grado di cambiare la visibilità e l'accesso.

- **public**: può essere utilizzato su classi, metodi e variabili d'istanza (le variabili definite all'interno di una classe). Visibilità:
  - le classi sono visibili in qualsiasi package del progetto;
  - i metodi e le variabili sono visibili da qualsiasi classe in qualsiasi package

- **protected**: può essere utilizzato su metodi e variabili d'istanza, leggermente più restrittivo del public. Visibilità:
  - i metodi o le variabili definite protected sono visibili da qualsiasi classe all'interno dello stesso package e da tutte le sottoclassi

- **default (senza modificatore)** Visibilità:
  - una classe senza modificatore è visibile solo dalle classi dello stesso package;
  - I metodi e le variabili sono visibili solo dalle classi dello stesso package della classe che li definisce

- **private**: può essere utilizzato su metodi e variabili di istanza. Rappresenta il modificatore più restrittivo. Visibilità:
  - i metodi o le variabili definite private sono visibili solo all'interno della classe che li definisce -> **metodi Getters & Setters**

	Ovunque	Stessa Classe	Stesso Package	Classe derivata
public				
protected				
default				
private				

Altri Modificatori

- **final**: può essere utilizzato su classi, metodi e variabili. Questo modificatore indica che la caratteristica non può essere modificata. Comportamento:
  - una **classe** definita final non può essere estesa
  - un **metodo** final non può essere sovrascritto (**no override**)
  - un **attributo** è una costante (valore immutabile)

- **static**: può essere utilizzato su metodi e variabili. Gli elementi static **appartengono solo alla classe che li definisce**, non appartengono all'oggetto (cioè la singola istanza).
  - un elemento static può essere utilizzato anche senza l'istanza dell'oggetto che lo contiene (il metodo main ne è un esempio)
  - un metodo static non può accedere a elementi non static, potrebbero non esistere in memoria.

Paradigmi Fondamentali

Incapsulamento

Questa tecnica consente di nascondere il funzionamento interno di una porzione di programma. Nella OOP l'incapsulamento prevede che:

- gli attributi siano visibili solo internamente all'oggetto (visibilità **private**);
- l'accesso agli attributi deve avvenire solo attraverso dei metodi public (**Getters & Setters**);

Principio dell' *Information Hiding* ti dico cosa ma non ti dico come. L'aspetto importante è come assemblare i vari elementi, dandoci delle interfacce per permettere l'assemblaggio ma non come è fatto un determinato oggetto.

Ereditarietà

Questa tecnica consente di ***estendere*** le caratteristiche di una classe ad un'altra classe. Questi attributi sono quindi utilizzabili dalla nuova classe che ***eredita***. Si parla di attributi e metodi definiti public o protected, quelli private non possono essere ereditati.

Quando una classe eredita può:

- definire i suoi metodi e attributi
- ridefinire quelli della classe da cui eredita (meccanismo dell' **overriding**)

keyword: **extends**

Esempio:

```
public class ClasseB extends ClasseA{
    …
}
```

Polimorfismo

Questa tecnica consente di ridefinire dei metodi già definiti in altre classi. Il polimorfismo è valido **solo su classi in cui è implementata l'ereditarietà**.

Con l'annotation @Override comunico alla JVM che sto sovrascrivendo il metodo della classe genitore da cui lo sto ereditando.