La programmazione orientata agli oggetti in Java

1. Ambienti di sviluppo (JDK) e primi approcci al codice
2. Le basi della programmazione object oriented: classi e oggetti
3. Variabili, attributi, metodi e costruttori
4. Identificatori, tipi di dati e array
5. Operatori e gestione del flusso di esecuzione
6. Costrutti di programmazione semplice: if, operatore ternario, while
7. Costrutti di programmazione avanzati: for, do while, for migliorato, switch
8. Classi ed oggetti
9. Classi innestate, classi anonime

- Java è un linguaggio di alto livello e orientato agli oggetti, creato dalla S u n M i c r o s y s t e m 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 o r i e n t a t o a g l i o g g e t t i (ereditarietà, polimorfismo, ...)
- la g e s t i o n e d e l l a m e m o r i a 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 p o r t a b i l i t à , cioè la capacità di un programma di poter essere eseguito su piattaforme diverse senza dover essere e modificato e ricompilato

---

Caratteristiche di Java

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

- Distribuito e dinamico
- Multi-thread

Semplice e familiare

- Basato su C
- Sviluppato da zero
- Estremamente semplice: senza puntatori, macro, registri
- Apprendimento rapido
- Semplificazione della programmazione
- Riduzione del numero di errori

Orientato a oggetti

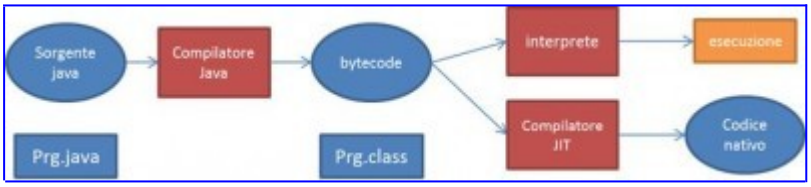
- Orientato a oggetti dalla base
- In Java tutto è un oggetto
- Incorpora le caratteristiche
- Incapsulamento
- Polimorfismo
- Ereditarietà
- Collegamento dinamico
- Non sono disponibili
- Ereditarietà multipla
- Overload degli operatori

Indipendente dalla piattaforma

- Più efficiente di altri linguaggi interpretati
- Soluzione: la macchina virtuale
- JVM (non è proprio la JVM)
- Linguaggio macchina bytecodes

Interpretato

- Il bytecode deve essere interpretato



- Vantaggi rispetto ad altri linguaggi interpretati
- Codice più compatto
- Efficiente
- Codice confidenziale (non esposto)

---

sicuro

- Supporta la sicurezza di tipo sandboxing
- Verifica del bytecode
- Altre misure di sicurezza
- Caricatore di classi
- Restrizioni nell'accesso alla rete

Robusto

- L'esecuzione nella JVM impedisce di bloccare il sistema
- L'assegnazione dei tipi è molto restrittiva
- La gestione della memoria è sempre a carico del sistema
- Il controllo del codice avviene sia a tempo di compilazione sia a tempo di esecuzione (runtime)

Distribuito e dinamico

- Disegnato per un'esecuzione remota e distribuita
- Sistema dinamico
- Classe collegata quando è richiesta
- Può essere caricata via rete
- Dinamicamente estensibile
- Disegnato per adattarsi ad ambienti in evoluzione

Multi-thread

- Soluzione semplice ed elegante per la multiprogrammazione
- Un programma può lanciare differenti processi

- Non si tratta di nuovi processi, condividono il codice e le variabili col processo principale
- Simultaneamente si possono svolgere vari compiti

Classi Java

Le classi estendono il concetto di "struttura" di altri linguaggi

3) Definiscono

- I dati (detti campi o attributi)
- Le azioni (metodi, comportamenti) che agiscono sui dati

4) Possono essere definite

- Dal programmatore (ex. Automobile)
- Dall'ambiente Java (ex. String, System, etc.)

5) La "gestione" di una classe avviene mediante

- Definizione della classe
- Instanziazione di Oggetti della classe

---

6) Creazione di classi in Java

- Definire i termini oggetto e classe
- Descrivere la forma nella quale possiamo creare nuove classi in Java
- mostrare come, una volta creata una classe possiamo creare oggetti di questa classe e utilizzarli

7) Struttura di una classe

```
public class HelloWorld {

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

}
```

---

8) Java è un linguaggio orientato agli oggetti

- In Java quasi tutto è un oggetto

- Come definire classi e oggetti in Java?
- Classe: codice che definisce un tipo concreto di oggetto, con proprietà e comportamenti in un unico file
- Oggetto: istanza, esemplare della classe, entità che dispone di alcune proprietà e comportamenti propri, come gli oggetti della realtà
- In Java quasi tutto è un oggetto, ci sono solo due eccezioni: i tipi di dato semplici (tipi primitivi) e gli array (un oggetto trattato in modo *particolare*)
- Le classi, in quanto tipi di dato strutturati, prevedono usi e regole più complessi rispetto ai tipi semplici

---

9) Le classi in Java

- Le classi, in quanto tipi di dato strutturati, prevedono usi e regole più complessi rispetto ai tipi semplici
- Il primo passo per definire una classe in Java è creare un file che deve chiamarsi esattamente come la classe e con estensione .java
- Java permette di definire solo una classe per ogni file
- Una classe in Java è formata da:
- Attributi: (o campi/proprietà) che immagazzinano alcune informazioni sull'oggetto. Definiscono lo stato dell'oggetto
- Costruttore: metodo che si utilizza per inizializzare un oggetto
- Metodi: sono utilizzati per modificare o consultare lo stato di un oggetto. Sono equivalenti alle funzioni o procedure di altri linguaggi di programmazione

---

Incapsulamento e visibilità in Java

- Quando disegniamo un software ci sono **due aspetti** che risultano fondamentali:
- Interfaccia: definita come gli **elementi che sono visibili dall'esterno**, come il sw può essere utilizzato
- Implementazione: definita definendo alcuni attributi e scrivendo il codice dei differenti metodi per leggere e/o scrivere gli attributi

---

10) Incapsulamento

- L'incapsulamento consiste nell'**occultamento degli attributi** di un oggetto in modo che possano essere **manipolati solo attraverso metodi** appositamente implementati. p.es la proprietà `saldo` di un oggetto `conto corrente`
- Bisogna fare in modo che l'interfaccia sia più indipendente possibile dall'implementazione

- In Java l'incapsulamento è strettamente relazionato con la visibilità

---

11) Visibilità

- Per indicare la visibilità di un elemento (attribuito o metodo) possiamo farlo precedere da una delle seguenti parole riservate
- `public`: accessibile da qualsiasi classe
- `private`: accessibile solo dalla classe attuale
- `protected`: solo dalla classe attuale, le discendenti e le classi del nostro package
- Se **non indichiamo la visibilità**: sono accessibili **solo dalle classi del nostro package**

---

12) Accesso agli attributi della classe

- Gli attributi di una classe sono strettamente relazionati con la sua implementazione.
- Conviene contrassegnarli come `private` e impedirne l'accesso dall'esterno
- In futuro potremo cambiare la rappresentazione interna dell'oggetto senza alterare l'interfaccia
- Quindi non permettiamo di accedere agli attributi!
- per consultarli e modificarli aggiungiamo i metodi accessori e mutatori: `getters` e `setters`

---

13) Modifica di rappresentazione interna di una classe

- Uno dei maggiori vantaggi di occultare gli attributi è che in futuro potremo cambiarli senza la necessità di cambiare l'interfaccia
- Un linguaggio di programmazione **ORIENTATO AGLI OGGETTI** fornisce meccanismi per definire nuovi tipi di dato basati sul concetto di classe
- Una classe definisce un insieme di oggetti (conti bancari, dipendenti, automobili, rettangoli, ecc...).
- Un oggetto è una struttura dotata di proprie **variabili** (che rappresentano il suo stato) propri **metodi** (che realizzano le sue funzionalità)

Classi e documentazione

- Come la maggior parte dei linguaggi di programmazione, Java è dotato di una libreria di classi "pronte all'uso" che coprono molte esigenze
- Usare classi già definite da altri è la norma per non sprecare tempo a risolvere problemi già risolti o a reinventare la ruota (DRY)

- La libreria Java standard è accompagnata da documentazione che illustra lo scopo e l'utilizzo di ciascuna classe presente,
- Dalla versione 9 di Java la libreria è stata divisa in moduli
- Documentazione Java 8
- Documentazione Java 9
- Documentazione Java 11
- Documentazione Java 13

Definizione di una Classe

14) Definizione

```
class <nomeClasse) {
<campi>
<metodi>
}
```

15) Esempi

- Classe contenente dati ma non azioni

```
class DataOnly {
boolean b;
char c;
int i;
float f;
}
```

- Classe contenente dati e azioni

```
class Automobile {

    //Attributi
    String colore;
    String marca;
    boolean accesa;

    //metti i in moto
    void mettiInMoto() {
        accesa = true;
    }

    //vernicia
    void vernicia (String nuovoCol) {
        colore = nuovoCol;
    }

    // stampaStato
    void stampaStato () {
        System.out.println("Questa automobile è una "+ marca + " " + colore);
        if (accesa)
        System.out.println("Il motore è acceso");
        else
        System.out.println("Il motore è spento");
    }
```

}

16) Dati & Metodi

- Public: visibili all'esterno della classe
- Private: visibili solo dall'interno della classe
- Protected: ...
- Nessuna specifica (amichevole): ...

La definizione di classe non rappresenta alcun oggetto.

ESPRESSIONI ARITMETICHE

```
public class Triangolo {
    public static void main ( String [] args ) {
        System.out.println (5*10/2);
    }
}
```

Il programma risolve l'espressione 5*10/2 e stampa il risultato a video

---

18) Espressioni aritmetiche e precedenza

singoli "letterali"

- Letterali interi: 3425, 12, -34, 0, -4, 34, -1234, ...
- Letterali frazionari: 3.4, 5.2, -0.1, 0.0, -12.45, 1235.3423, ...

operatori aritmetici

- moltiplicazione *
- divisione /
- modulo % (resto della divisione tra interi)
- addizione +
- sottrazione -

Le operazioni sono elencate in **ordine decrescente di priorità** ossia 3+2*5 fa 13, non 25

Le parentesi tonde cambiano l'ordine di valutazione degli operatori ossia (3+2)*5 fa 25

Inoltre, tutti gli operatori sono associativi a sinistra ossia 3+2+5 corrisponde a (3+2)+5 quindi 18/6/3 fa 1, non 9

---

19) operazione di divisione

- L'operazione di divisione / si comporta diversamente a seconda che sia applicato a letterali interi o frazionari
- 25/2 = 12 (divisione intera)

- 25%2 = 1 (resto della divisione intera)
- 25.0/2.0 = 12.5 (divisione reale)
- 25.0%2.0 = 1.0 (resto della divisione intera)
- Una operazione tra un letterale intero e un frazionario viene eseguita come tra due frazionari
- 25/2.0 = 12.5
- 1.5 + (25/2) = 13.5 (attenzione all'ordine di esecuzione delle operazioni)
- 2 + (25.0/2.0) = 14.5

OPERATORI ARITMETICI, RELAZIONALI, DI ASSEGNAZIONE

- Di assegnazione: = += -= *= /= &= |= ^=
- Di assegnazione/incremento: ++ -- %=

Operatore	Significato
=	assignment
+=	addition assignment
-=	subtraction assignment
*=	multiplication assignment
/=	division assignment
%=	remainder assignment

- Operatori Aritmetici: `+ - * / %`

Operatore	Significato
+	addition
-	subtraction
*	multiplication
/	division
%	remainder
++var	preincrement
--var	predecrement
var++	postincrement
var--	postdecrement

- Relazionali: == != > < >= <=

Operatore	Significato
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal

---

21) Operatori per Booleani

- Bitwise (interi): & | ^ << >> ~

Operatore	Significato
&&	short circuit AND
||	short circuit OR
!	NOT
^	exclusive OR

Attenzione:

- Gli operatori logici agiscono solo su booleani
  - Un intero NON viene considerato un booleano
  - Gli operatori relazionali forniscono valori booleani

Operatori su reference

22) Per i puntatori/reference, sono definiti:

- Gli operatori relazionali == e !=
  - N.B. test sul puntatore NON sull'oggetto
- Le assegnazioni
- L'operatore "punto"
- NON è prevista l'aritmetica dei puntatori

Operatori matematici

23) Operazioni matematiche complesse sono permesse dalla classe Math (package java.lang)

- `Math.sin (x) calcola sin(x)`
- `Math.sqrt (x) calcola x^(1/2)`
- `Math.PI ritorna pi`
- `Math.abs (x) calcola |x|`
- `Math.exp (x) calcola e^x`
- `Math.pow (x, y) calcola x^y`

Esempio

- `z = Math.sin (x) – Math.PI / Math.sqrt(y)`

Caratteri speciali

Literal	Represents
`\n`	New line
`\t`	Horizontal tab
`\b`	Backspace
`\r`	Carriage return
`\f`	Form feed
`\\`	Backslash
`\"`	Double quote
`\ddd`	Octal character
`\xdd`	Hexadecimal character
`\udddd`	Unicode character

LE VARIABILI

- Una variabile è un'area di memoria identificata da un nome
- Il suo scopo è di contenere un valore di un certo tipo
- Serve per memorizzare dati durante l'esecuzione di un programma
- Il nome di una variabile è un identificatore
- può essere costituito da lettere, numeri e underscore
- non deve coincidere con una parola chiave del linguaggio
- è meglio scegliere un identificatore che sia significativo per il programma

25) esempio

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

        int base , altezza ;
        int area ;

        base = 5;
        altezza = 10;
        area = base * altezza / 2;

        System.out.println ( area );
    }
}
```

Usando le variabili il programma risulta essere più chiaro:

- Si capisce meglio quali siano la base e l'altezza del triangolo
- Si capisce meglio che cosa calcola il programma

---

26) Dichiarazione

- In Java ogni variabile deve essere dichiarata prima del suo uso
- Nella dichiarazione di una variabile se ne specifica il nome e il tipo
- Nell'esempio, abbiamo dichiarato tre variabili con nomi base, altezza e area, tutte di tipo int (numeri interi)
    - int base , altezza ;
    - int area ;

ATTENZIONE! Ogni variabile deve essere dichiarata UNA SOLA VOLTA (la prima volta che compare nel programma)

```
base =5;
altezza =10;
area = base * altezza /2;
```

---

27) Assegnazione

- Si può memorizzare un valore in una variabile tramite l'operazione di assegnazione
- Il valore da assegnare a una variabile può essere un letterale o il risultato della valutazione di un'espressione
- Esempi:

```
base =5;
altezza =10;
area = base * altezza /2;
```

- I valori di base e altezza vengono letti e usati nell'espressione
- Il risultato dell'espressione viene scritto nella variabile area

28) Dichiarazione + Assegnazione

Prima di poter essere usata in un'espressione una variabile deve:

- essere stata dichiarata
- essere stata assegnata almeno una volta (inizializzata)
- NB: si possono combinare dichiarazione e assegnazione.

Ad esempio:

```
int base = 5;
int altezza = 10;
int area = base * altezza / 2;
```

Costanti

Nella dichiarazione delle variabili che **NON DEVONO** mai cambiare valore si può utilizzare il modificatore **final**

```
final double IVA = 0.22;
```

- Il modificatore **final** trasforma la variabile in una costante
- Il compilatore si occuperà di controllare che il valore delle costanti non venga mai modificato (ri-assegnato) dopo essere stato inizializzato.
- Aggiungere il modificatore **final** non cambia funzionamento programma, ma serve a prevenire errori di programmazione
- Si chiede al compilatore di controllare che una variabile non venga ri-assegnata per sbaglio
- Sapendo che una variabile non cambierà mai valore, il compilatore può anche eseguire delle ottimizzazioni sull'uso di tale variabile.

---

29) Input dall'utente

- Per ricevere valori in input dall'utente si può usare la classe Scanner, contenuta nel package **java.util**
- La classe Scanner deve essere richiamata usando la direttiva import prima dell'inizio del corpo della classe

TIPI DI DATO PRIMITIVI

- In un linguaggio ad oggetti puro, vi sono solo classi e istanze di classi:
- i dati dovrebbero essere definiti sotto forma di oggetti

Java definisce alcuni tipi primitivi

- Per efficienza Java definisce dati primitivi
- La dichiarazione di una istanza alloca spazio in memoria
- Un valore è associato direttamente alla variabile

- (e.g, i == 0)
- Ne vengono definiti dimensioni e codifica
- Rappresentazione indipendente dalla piattaforma

Tabelle riassuntive: tipi di dato

Primitive Data Types

type	bits
byte	8 bit
short	16 bit
int	32 bit
long	64 bit
float	32 bit
double	64 bit
char	16 bit
boolean	true/false

I caratteri sono considerati interi

---

31) I tipi numerici, i char

- Esempi
- `123` (int)
- `256789L` (L o l = long)
- `0567` (ottale) `0xff34` (hex)
- `123.75 0.12375e+3` (float o double)
- `'a' '%' '\n'` (char)
- `'\123'` (\ introduce codice ASCII)

32) Tipo boolean

- `true`
- `false`

Esempi

```
int i = 15;
long longValue = 10000000000001;
byte b = (byte)254;

float f = 26.012f;
double d = 123.567;
```

```
boolean isDone = true;
boolean isGood = false;
char ch = 'a';
char ch2 = ';';
```

```
public class Applicazione {

	public static void main(String[] args) {

		int mioNumero;
		mioNumero = 100;
		System.out.println(mioNumero);

		short mioShort = 851;
		System.out.println(mioShort);

		long mioLong = 34093;
		System.out.println(mioLong);

		double mioDouble = 3.14159732;
		System.out.println(mioDouble);

		float mioFloat = 324.4f;
		System.out.println(mioFloat);

		char mioChar = 'y';
		System.out.println(mioChar);

		boolean mioBoolean = true;
		System.out.println(mioBoolean);

		byte mioByte = 127;
		System.out.println(mioByte);
	}

}
```

Data Type	Bits	Minimum	Maximum
byte	8	-128	127
short	16	-32,768	32,767
int	32	-2,147,483,648	2,147,483,647
long	64	-9.22337E+18	9.22337E+18
float	32	See the docs	
double	64	See the docs	

Esempi gist

Selezione

if //Statements

```
if (condition) {

    //statements;

}

[optional]
else if (condition2) {

    //statements;

}



[optional]
else {

//statements;

}
```

---

switch Statements

```
switch (Expression) {

    case value1:

    //statements;

    break;

    ...

    case valuen:

    //statements;

    break;

    default:

    //statements;

}
```

---

loop Statements

```
while (condition) {

//statements;

}
```

```
do {

//statements;

} while (condition);


for (init; condition; adjustment) {

//statements;

}
```

Cicli definiti

Se il numero di iterazioni è prevedibile dal contenuto delle variabili all'inizio del ciclo.

Esempio: prima di entrare nel ciclo so già che verrà ripetuto 10 volte

```
int n=10;
for (int i=0; i<n; ++i) {
    ...
}
```

Cicli indefiniti

Se il numero di iterazioni non è noto all'inizio del ciclo.

Esempio: il numero di iterazioni dipende dai valori immessi dall'utente.

```
while(true) {
    x = Integer.parseInt(JOptionPane.showInputDialog("Immetti numero
positivo"));
    if (x > 0) break;
}
```

Cicli annidati

Se un ciclo appare nel corpo di un altro ciclo.

Esempio: stampa quadrato di asterischi di lato n

```
for (int i=0; i<n; i++) {
    for (int j=0; j<n; j++) System.out.print("*");
    System.out.println();
}
```

Cicli con filtro

Vengono passati in rassegna un insieme di valori e per ognuno di essi viene fatto un test per verificare se il valore ha o meno una certa proprietà in base alla quale decideremo se prenderlo in considerazione o meno.

Esempio: stampa tutti i numeri pari fino a 100

```
for (int i=1; i<100; ++i) { // passa in rassegna tutti i numeri fra 1 e 100
```

```
    if (i % 2 == 0) // filtra quelli pari
        System.out.println(i);
}
```

Cicli con filtro e interruzione

Se il ciclo viene interrotto dopo aver filtrato un valore con una data proprietà.

Esempio: verifica se un array contiene o meno numeri negativi

```
boolean trovato = false;
for (int i=0; i<v.length; ++i) // passa in rassegna tutti gli indici dell'array
v
    if (v[i]<0) { // filtra le celle che contengono valori negativi
        trovato = true;
        break; // interrompe ciclo
    }
// qui trovato vale true se e solo se vi sono numeri negativi in v
```

Cicli con accumulatore

Vengono passati in rassegna un insieme di valori e ne viene tenuta una traccia cumulativa usando una opportuna variabile.

Esempio: somma i primi 100 numeri interi.

```
int somma = 0; // variabile accumulatore di tipo int
for (int i=1; i<100; ++i) { // passa in rassegna tutti i numeri fra 1 e 100
    somma = somma + i; // accumula i valori nella variabile accumulatore
}
```

Esempio: data una stringa s, ottieni la stringa rovesciata

```
String rovesciata = ""; // variabile accumulatore di tipo String
for (int i=0; i<s.length(); ++i) { // passa in rassegna tutti gli indici dei
caratteri di s
    rovesciata = s.substring(i, i+1) + rovesciata; // accumula i caratteri in
testa all'accumulatore
}
```

Cicli misti

Esempio di ciclo definito con filtro e accumulatore: calcola la somma dei soli valori positivi di un array

```
int somma = 0;
for (int i=0; i<v.length; ++i) // passa in rassegna tutti gli indici dell'array
v
    if (v[i]>0) // filtra le celle che contengono valori positivi
        somma = somma + v[i]; // accumula valore nella variabile accumulatore
```