MODULO 6

Cosa sono?
Sono i *mattoni fondamentali* del linguaggio: i tipi di dato più semplici, da cui poi si costruisce tutto il resto (array, oggetti, strutture più complesse). Non contengono altri dati al loro interno.

Quali sono?
L'elenco che vedi (byte, short, int, long, float, double, boolean, char) è tipico di **Java**.
In breve:

- byte, short, int, long → numeri interi (con dimensioni diverse)
- float, double → numeri con la virgola
- boolean → vero o falso
- char → un singolo carattere

Come li riconosci?
Il nome del tipo è scritto in **minuscolo**.
Questo è importante soprattutto in Java, perché:

- int ≠ Integer
- boolean ≠ Boolean
  Quelli con la maiuscola non sono primitivi, ma classi.

A cosa servono?
Servono a rappresentare **valori semplici**: numeri, caratteri o condizioni logiche.
Sono veloci, occupano poca memoria e sono usati continuamente nelle operazioni di base.

Possono avere valore "null"? - No.
Un tipo primitivo **deve sempre avere un valore**. Anche se non lo assegni esplicitamente, il linguaggio gli dà un valore di default (per esempio 0, false, '\0').

i tipi primitivi sono semplici, efficienti, sempre valorizzati e fondamentali per capire come funziona davvero un programma. Se vuoi, nel prossimo passo possiamo confrontarli con i **tipi wrapper** o fare una tabella con dimensioni e range di valori.

Modellazione di un oggetto

Creazione di un modello su cui lavorare

La **modellazione di un oggetto** è il processo con cui si rappresenta un'entità reale o astratta all'interno di un programma. L'idea di base è semplice: prendere qualcosa del mondo reale (o logico) e descriverlo in codice attraverso **caratteristiche** e **comportamenti**.

Un oggetto nasce sempre da una **classe**, che puoi immaginare come un progetto o uno stampo. La classe definisce *come è fatto* l'oggetto e *cosa può fare*.

Durante la modellazione ci si chiede prima di tutto:

- che **cos'è** l'oggetto
- quali **dati** lo descrivono
- quali **azioni** può compiere

Le **caratteristiche** dell'oggetto diventano gli **attributi** (o campi). Sono le informazioni che descrivono lo stato dell'oggetto. Esempio: se stai modellando una *Persona*, attributi tipici possono essere nome, età, altezza.

I **comportamenti** diventano i **metodi**.
Sono le azioni che l'oggetto può eseguire o che possono modificarne lo stato.
Sempre con l'esempio della Persona: camminare, parlare, cambiare età.

Un punto fondamentale della modellazione è la **scelta del livello di dettaglio**.
Non si modella mai tutto: si inseriscono solo gli attributi e i metodi che servono allo scopo del programma. Se stai scrivendo un gestionale, magari l'altezza non serve; se stai facendo un'app sportiva, sì.

La modellazione segue spesso questi passi logici:

1. Identificare l'oggetto
2. Individuare gli attributi rilevanti
3. Individuare i metodi necessari
4. Stabilire le relazioni con altri oggetti

Un altro aspetto importante è l'**incapsulamento**:
i dati dell'oggetto non dovrebbero essere accessibili liberamente dall'esterno. L'oggetto espone solo ciò che serve, tramite metodi pubblici, proteggendo il proprio stato interno.

In pratica, modellare bene un oggetto significa creare una rappresentazione chiara, coerente e utile di una parte del problema che stai risolvendo. Una buona modellazione rende il codice più leggibile, più manu-tenibile e molto più facile da estendere in futuro. La **classe Person** serve a rappresentare una persona nel programma. Non è una persona reale, ma una **rappresentazione astratta** costruita scegliendo solo le informazioni e i comportamenti che ci interessano.

Quando modelli Person, ti fai domande molto pratiche:
che dati descrivono una persona? cosa può fare una persona *nel contesto del programma*?

Attributi della classe Person

Gli **attributi** descrivono lo stato della persona. Tipicamente:

- name → il nome della persona
- age → l'età
- height → l'altezza

Questi attributi usano **tipi primitivi** o tipi semplici:

- String name
- int age
- double height

Sono privati perché seguono il principio di incapsulamento:
dall'esterno non si deve poter modificare tutto liberamente.

Metodi della classe Person

I **metodi** rappresentano i comportamenti:

- ottenere informazioni (getter)
- modificare lo stato in modo controllato (setter)
- azioni logiche della persona

Esempi concettuali:

- getName() → restituisce il nome
- getAge() → restituisce l'età
- birthday() → aumenta l'età di 1

Costruttore

Il **costruttore** serve a creare l'oggetto Person inizializzando i suoi attributi.
Senza costruttore l'oggetto esisterebbe, ma sarebbe incompleto o poco significativo.

Nella **programmazione orientata agli oggetti**, **attributi** e **proprietà** servono entrambi a descrivere lo **stato di una classe o di un oggetto**, ma non sono esattamente la stessa cosa in tutti i linguaggi.

Attributi di una classe → Gli attributi sono le variabili dichiarate all'interno della classe.
Rappresentano i dati interni dell'oggetto.

Caratteristiche principali:

- descrivono lo stato dell'oggetto
- hanno un tipo di dato
- sono spesso dichiarati come private
- non dovrebbero essere modificati direttamente dall'esterno

Proprietà → Le proprietà sono il modo controllato con cui si accede agli attributi.
Non sono il dato in sé, ma l'interfaccia che la classe espone verso l'esterno.

Nella pratica, in Java una proprietà è composta da:

- un **getter** → per leggere il valore
- un **setter** → per modificarlo (se consentito)

Perché non si accede direttamente agli attributi

Separare attributi e proprietà permette di:

- proteggere i dati interni
- aggiungere controlli (es. età non negativa)
- cambiare l'implementazione senza rompere il codice esterno

DIFFERENZA TRA ISTANZA E OGGETTO

Un'**istanza** è un **oggetto concreto creato a partire da una classe**.
Classe e istanza non sono la stessa cosa.

La **classe** è il modello, lo schema, la definizione.
L'**istanza** (detta anche **oggetto**) è la realizzazione concreta di quel modello, con valori reali assegnati agli attributi.

Pensala così:

- la classe è il progetto
- l'istanza è qualcosa che esiste davvero in memoria

*p2.name significa **accedere all'attributo name dell'oggetto p2.***

"prendi l'oggetto p2 e vai a leggere (o modificare) il valore del suo attributo name".

Il **punto (.)** è l'operatore di accesso: serve per entrare dentro l'oggetto e usare ciò che contiene.

RIEPILOGO DEFINIZIONI

Classe tipo (Class)

È lo "stampo" da cui creare un oggetto. Possiamo definirla come una "istanza" di una classe. La classe indica la categoria generale, l'istanza è il caso singolo. La classe regola la struttura dell'oggetto, essendo l'oggetto creato dallo stampo della classe.

Oggetto (Object)

È un'area di memoria che contiene i dati atti a descrivere un individuo della categoria indicata dalla classe. Può sempre essere creato di new, e con l'uso di un metodo particolare che viene detto "costruttore".

Variabile

È un collegamento all'area di memoria che contiene i dati dell'oggetto. Ad esempio, Person george è una variabile di tipo non primitivo che può "puntare" a un oggetto (e non "contenere" un oggetto).

NULL (parola chiave)

NULL in Java (scritto null) rappresenta un riferimento che non punta a nessun oggetto

Caratteristiche di null in Java:

1. **Valore di default**: Tutte le variabili di tipo oggetto non inizializzate sono automaticamente null
2. Tentare di usare un oggetto null causa un NullPointerException
3. **Controllo**: Si deve sempre verificare se un oggetto è null prima di usarlo

Costruttori Impliciti ed Espliciti in Java

Costruttore Implicito (Default Constructor)

Il **costruttore implicito** è un costruttore che Java crea automaticamente se NON hai definito nessun costruttore nella tua classe.

Caratteristiche:

- **Non ha parametri**
- **Non fa nulla** (corpo vuoto)
- Viene generato automaticamente dal compilatore

- Inizializza le variabili ai valori di default (null per oggetti, 0 per numeri, false per boolean)

```
public class Persona {
    String nome;
    int eta;

    // NON ho scritto nessun costruttore
    // Java crea automaticamente questo:
    // public Persona() { }
}
// Uso:
Persona p = new Persona();  // Funziona! Usa il costruttore implicito
// p.nome sarà null, p.eta sarà 0
```

Quando scriviamo un costruttore, quello di default viene rimpiazzato da quello che scriviamo noi

Costruttore Esplicito (Explicit Constructor)

Il **costruttore esplicito** è un costruttore che **tu scrivi manualmente** nella classe.

Caratteristiche:

- Lo definisci tu
- Può avere parametri o essere senza parametri
- Può contenere logica personalizzata
- **ATTENZIONE**: Se crei un costruttore esplicito, Java NON genera più quello implicito!

Puoi avere **più costruttori** nella stessa classe con parametri diversi

THIS

this è una parola chiave in Java che rappresenta un **riferimento all'oggetto corrente**, cioè l'istanza della classe su cui stiamo lavorando.

Puoi pensare a this come "**me stesso**" o "**questo oggetto**".

Usi di this

1. Disambiguare tra Variabili di Istanza e Parametri

Quando un parametro ha lo **stesso nome** di una variabile di istanza, usiamo this per distinguerli.

```
public class Persona
{
    String nome;  // Variabile di istanza
    int eta;      // Variabile di istanza

    // Costruttore con parametri che hanno lo stesso nome
    public Persona(String nome, int eta) {
        this.nome = nome;  // this.nome = variabile di istanza
                           // nome = parametro
        this.eta = eta;    // this.eta = variabile di istanza
                           // eta = parametro
    }
}
```

2. Chiamare un Costruttore da un Altro Costruttore

Usando this() puoi chiamare un altro costruttore della stessa classe. Questo evita la duplicazione del codice.

```
public class Persona
{
    String nome;
    int eta;
    String citta;
    // Costruttore completo
    public Persona(String nome, int eta, String citta) {
        this.nome = nome;
        this.eta = eta;
        this.citta = citta;
    }
    // Costruttore con 2 parametri - chiama quello completo
    public Persona(String nome, int eta) {
```

```
        this(nome, eta, "Sconosciuta");  // Chiama Persona(String, int, String)
    }
    // Costruttore con 1 parametro - chiama quello con 2 parametri
    public Persona(String nome) {
        this(nome, 0);  // Chiama Persona(String, int)
    }
    // Costruttore senza parametri - chiama quello con 1 parametro
    public Persona() {
        this("Anonimo");  // Chiama Persona(String)
    }
}
**Catena di chiamate:**
new Persona()
    → chiama Persona(String) con "Anonimo"
        → chiama Persona(String, int) con "Anonimo", 0
            → chiama Persona(String, int, String) con "Anonimo", 0, "Sconosciuta"
```

3. Passare l'Oggetto Corrente come Parametro

Puoi passare l'oggetto corrente (this) ad altri metodi:

```
public class Studente {
    String nome;

    public Studente(String nome) {
        this.nome = nome;
    }

    public void registrati(Registro registro) {
        registro.aggiungiStudente(this);  // Passa "questo" studente
    }
}

public class Registro {
    public void aggiungiStudente(Studente s) {
        System.out.println("Aggiunto: " + s.nome);
```

```
    }
}
// Uso:
Studente s = new Studente("Mario");
Registro r = new Registro();
s.registrati(r);  // Passa se stesso al registro
```

4. Restituire l'Oggetto Corrente (Method Chaining)

Puoi restituire this per creare un'interfaccia "fluente":

```
public class StringBuilder {
    private String testo = "";

    public StringBuilder aggiungi(String s) {
        testo += s;
        return this;  // Restituisce l'oggetto corrente
    }

    public void stampa() {
        System.out.println(testo);
    }
}
// Uso (method chaining):
StringBuilder sb = new StringBuilder();
sb.aggiungi("Ciao ")
  .aggiungi("mondo")
  .aggiungi("!")
  .stampa();  // Output: Ciao mondo!
```

LO STATO DI UN OGGETTO

Cos'è lo Stato?

Lo **stato di un oggetto** è l'insieme dei **valori correnti** di tutte le sue **variabili di istanza** (chiamate anche attributi o campi) in un determinato momento.

Lo stato rappresenta le **caratteristiche** e le **informazioni** che l'oggetto contiene. Può essere mutevole.

VARIABILI DI ISTANZA VS VARIABILI DI CLASSE

Variabili di Istanza (Stato dell'Oggetto)

Ogni oggetto ha la **propria copia** di queste variabili. Lo stato è **unico per ogni istanza**.

Variabili di Classe (Stato Condiviso)

Le variabili dichiarate con static sono **condivise** da tutte le istanze. Non fanno parte dello stato dell'oggetto, ma dello **stato della classe**.

Modificare lo Stato

Lo stato di un oggetto può cambiare nel tempo attraverso i **metodi**.

Stato Valido vs Stato Non Valido

Un oggetto dovrebbe sempre essere in uno **stato valido** (coerente). È responsabilità della classe garantire questo.

Incapsulamento dello Stato

Per proteggere lo stato, usiamo l'**incapsulamento**: le variabili di istanza sono private e si accede tramite metodi public.

Oggetti Immutabili

Un oggetto è **immutabile** se il suo stato **non può cambiare** dopo la creazione.

Esempi di classi immutabili in Java: String, Integer, Double, LocalDate

I TRE PILASTRI DELL'AUTONOMIA

1. Incapsulamento (Encapsulation)

L'oggetto **nasconde** i suoi dati interni e **espone** solo ciò che è necessario attraverso metodi pubblici.

Vantaggi:

- Chi usa l'oggetto non deve conoscere i dettagli interni
- Puoi cambiare l'implementazione interna senza rompere il codice esterno
- Controllo totale sull'accesso ai dati

2. Responsabilità (Responsibility)

Ogni oggetto ha **responsabilità specifiche** e si occupa di gestire autonomamente le sue operazioni.

Principio: Ogni oggetto sa come svolgere i propri compiti senza bisogno di controllo esterno.

3. Interfaccia Definita (Well-defined Interface)

L'oggetto comunica con l'esterno attraverso un'**interfaccia chiara e stabile**.

Comunicazione tra Oggetti Autonomi

Gli oggetti autonomi collaborano **scambiando messaggi** (chiamate a metodi).

METODO DI CLASSE E METODO DI OGGETTO

Differenza Fondamentale

Metodi di Istanza (Metodi di Oggetto)

- Appartengono a un **singolo oggetto**
- Operano sullo **stato dell'oggetto** (variabili di istanza)
- Si chiamano **attraverso un oggetto specifico**
- Hanno accesso a `this`

Metodi di Classe (Metodi Static)

- Appartengono alla **classe stessa**, non agli oggetti
- **NON** operano sullo stato di un oggetto specifico
- Si chiamano **attraverso il nome della classe**
- **NON** hanno accesso a `this`