*Andrea Giuseppe Cavallaro*

# Test

# Java

**Che differenza c’è tra classe e oggetto in Java?**

- ☐ Una classe è una singola istanza di un oggetto
- ☑ La classe è il modello, mentre l'oggetto è l'istanza concreta basata su quel modello
- ☐ Una classe è un'istanza di un oggetto.

**Qual è lo scopo principale del costruttore in Java?**

- ☐ Permette di creare nuove variabili all'interno di una classe.
- ☐ Definisce le variabili di istanza di una classe.
- ☑ Inizializza gli oggetti della classe con valori predefiniti o personalizzati.
- ☐ Fornisce metodi per modificare le variabili di istanza di una classe.

**Qual è lo scopo principale dell'ereditarietà in Java?**

- ☐ Permette di creare più istanze di una classe.
- ☐ Consente di definire le variabili di istanza di una classe.
- ☑ Fornisce un modo per condividere attributi e comportamenti tra classi, quindi consente il riutilizzo del codice
- ☐ Definisce metodi per modificare le variabili di istanza di una classe.

**Qual è uno degli scopi principali del polimorfismo in Java?**

- ☐ Consentire agli oggetti di essere trattati come istanze di loro superclassi.
- ☐ Consentire la definizione di variabili di istanza di una classe.
- ☐ Fornire un modo per condividere attributi e comportamenti tra classi.
- ☑ Scrivere codice generico che lavora con superclassi o interfacce senza la necessità di conoscere il tipo specifico dell'oggetto al momento della compilazione.

**Quale modificatore di visibilità consente di accedere direttamente sia dalla classe in questione che dalle classi figlie?**

- ☐ public
- ☑ protected
- ☐ private
- ☐ shared

**Spiega cosa sono gli array in Java e in quali casi è utile utilizzarli. Fai anche un esempio pratico.**
Un array è una collezione di dati che contiene un determinato numero di dati massimo. Ad ogni dato corrisponde un indice che ne identifica univocamente la posizione, cosa molto utile in quanto semplifica la ricerca di un determinato dato.
Come esempio di dato prendiamo l'altezza di n persone, creiamo un array chiamato altezze e al suo interno mettiamo, una ad una, le altezze che abbiamo ottenuto. Come risultato finale avremo una collezione di altezze, tutto indicizzate da 0 a n-1

**Cos'è un metodo static in Java? Quando è opportuno usarlo?**
Un metodo static è un metodo di una classe che possiamo invocare anche quando non viene istanziato un oggetto relativo alla classe che contiene il metodo. Ciò viene fatto usando direttamente la classe per invocare il metodo nel main. Il metodo static va usato solo quando il metodo stesso non fa riferimento ad attributi della classe in quanto questi non sono stati inizializzati

**Cos'è final e quando è opportuno utilizzarlo?**
Final è una parola chiave che va usata quando sfruttiamo l'ereditarietà e serve a specificare che una classe non può ulteriormente essere estesa e avere dei figli

**Che differenza c'è tra ArrayList e un array semplice ([]) in Java?**
Come scritto in precedenza, un array ha una dimensione massima e contiene degli indici. L'ArrayList non ha queste caratteristiche, quindi possiamo usarlo quando non sappiamo il numero di elementi massimo che dovrà contenere, perchè non dobbiamo specificare un tetto massimo ed avremo sempre la possibilità di aggiungere nuovi elementi. Di contro però non abbiamo la possibilità di accedere agli indici e quindi per trovare un elemento siamo costretti a scorrere l'arrayList con un foreach per trovare ciò che vogliamo

**Spiega che differenza c'è tra una coda (Queue) e una pila (Stack).**
La coda e la pila sono due strutture dati molto simili all'ArrayList, ma con alcune differenze.
La coda è un tipo di struttura detta FIFO(first in, first out), ciò vuol dire che il primo elemento che viene inserito in coda sarà il primo ad uscire e tutti gli altri elementi che si accodano devono aspettare il loro turno per poter raggiungere la fine della coda e uscire a loro volta, ciò simula effettivamente il comportamento di una coda nella vita di tutti i giorni.
La pila è una struttura detta LIFO(last in, first out), l'ultimo elemento che viene inserito sarà il primo ad essere estratto e se altri elementi vengono inseriti dopo esso, l'elemento in questione dovrà attendere il pop degli elementi sopra di esso

**Cos'è il concetto di incapsulamento in Java e perché è importante nella programmazione ad oggetti?**
L'incapsulamento serve ad aumentare la sicurezza del nostro codice dando dei criteri di accesso agli attributi e ai metodi delle classi. Serve ad aumentare la manutenzione del codice e ad evitare di apportare modifiche indesiderate

**Che ruolo ha il modificatore private in una classe? Perché è utile usarlo insieme ai metodi getter e setter?**
Private serve a rendere visibile l'elemento in questione soltanto all'interno della classe stessa. I metodi getter e setter sono essenziali quando abbiamo un attributo private perchè questi metodi ci permettono di interagire con l'attributo quando siamo fuori dalla classe in modo controllato

**Scrivi una classe Java che rappresenta un "Libro" con almeno 3 attributi e un metodo per stampare le informazioni del libro.**

```
public class Libro{
        private String titolo;
        private String autore;
        private int numPagine;

        public Libro(String titolo, String autore, int numPagine){
                this.titolo = titolo;
                this.autore = autore;
                this.numPagine = numPagine;
        }

        public void stampa(){
                System.out.printl("Titolo: " + titolo + "\nAutore: " + autore + "\nNumero di
        Pagine: " + numPagine);
        }
}
```

**Cosa si intende per overriding di un metodo? Quando e perché lo si usa?**
Overriding è un modo per applicare il polimorfismo e sovrascrive un metodo con lo stesso nome che appartiene alla classe genitore. L'overriding viene usato quando è presente anche l'ereditarietà e si usa quando vogliamo che l'oggetto della classe figlia specializzata abbia un comportamento diverso rispetto a quello definito nella classe genitore

**Spiega cosa fa il costrutto super e quando è utile utilizzarlo.**
Il costrutto super viene usato dalle classi figlie per passare i parametri degli attributi in comune alla classe genitore, perchè tali attributi sono presenti solo nella classe genitore

**In molti linguaggi di programmazione una stringa è trattata semplicemente come un array di caratteri. In Java, invece, le stringhe sono oggetti della classe `String`. Spiega cosa significa che una stringa è un oggetto in Java e quali vantaggi comporta.**
Che una stringa sia un oggetto vuol dire che è un'istanza della classe Stringa, ciò vuol dire che incapsula sia i dati che il comportamente della classe stringa e rende il codice più pulito e riutilizzabile

**Parlami di un argomento a piacere tra tutti quelli che abbiamo trattato, specificando in quali contesti è opportuno utilizzarlo e descrivi anche un esempio.**
Le interfacce sono l'argomento di cui voglio parlare. Abbiamo visto che in java quando usiamo l'ereditarietà possiamo avere una classe genitore che ha più classi figlie, ma non il contrario. Le interfacce servono proprio per risolvere questo problema. Un esempio che abbiamo fatto per definire le interfacce è quello della classe genitore veicolo di terra, tale classe può avere come classi figlie diversi veicoli terrestri, ma se volessimo aggiungere un veicolo anfibio? tale veicolo può camminare sia sulla terra che navigare sull'acqua, per poterlo rappresentare al meglio usiamo un interfaccia, che sostanzialmente è come una classe astratta che definirà dei metodi che saranno ereditati dal veicolo anfibio e ne definiranno il comportamento

## Esericizio 1 - Classi e Oggetti

Crea una classe `Studente` con i seguenti attributi:

- `nome` (String)
- `cognome` (String)
- `annoNascita` (int)

Crea un costruttore per inizializzare questi attributi e un metodo `stampaScheda()` che stampi una frase come:

> "Mario Rossi, nato nel 2004"

Poi, nel `main`, crea 2 oggetti `Studente` e chiama il metodo `stampaScheda()`.

## Esericizio 2 - Ereditarietà e Override

Crea una gerarchia di classi per rappresentare le persone che fanno parte di una scuola.

Consegna anche il diagramma UML che rappresenta lo schema delle classi.

**1. Classe `Persona`**

- Attributi:
  - `nome` (String)
  - `cognome` (String)
- Costruttore che inizializza nome e cognome
- Metodo `presentati()` che stampa:
  "Ciao, sono [nome] [cognome]"

**2. Classe `Studente` (estende `Persona`)**

- Attributo aggiuntivo: `matricola` (String)
- Costruttore che riceve anche la matricola
- Override del metodo `presentati()` per stampare:
  "Sono lo studente [nome] [cognome], matricola: [matricola]"

**3. Classe `Professore` (estende `Persona`)**

- Attributo aggiuntivo: `materia` (String)
- Costruttore che riceve anche la materia
- Override del metodo `presentati()` per stampare:
  "Sono il prof. [nome] [cognome], insegno [materia]"

---

### 🧪 Nel `main`

- Crea 3 oggetti:
  - 1 `Studente`
  - 1 `Professore`
  - 1 `Persona` generica
- Inseriscili in un array di tipo `Persona[]`
- Scorri l'array con un ciclo `for` e chiama `presentati()` su ciascuno

## Esericizio 3 - Sistema di autenticazione utenti

Il programma gestisce diverse tipologie di utenti (Studenti, Professori, Utenti generici) e consente l'autenticazione solo agli utenti che implementano l'interfaccia Autenticabile. Se la password inserita non è corretta, viene sollevata un'eccezione personalizzata.

Consegna anche il diagramma UML che rappresenta lo schema delle classi.

**1. Classe base `Utente`**

- Attributi: `username`, `email`
- Costruttore
- Metodo `presentati()` che stampa:
  "Utente generico: [username], email: [email]"

---

**2. Interfaccia `Autenticabile`**

- Metodo: `autentica(String password)`
- Lancia `AutenticazioneException` se la password è errata

---

**3. Classi derivate:**

**`Studente` estende `Utente` e implementa `Autenticabile`**

- Attributo: `matricola`
- `autentica()`: accetta solo la password `"studente123"`
- `presentati()` stampa:
  "Studente [username], matricola: [matricola]"

---

**`Professore` estende `Utente` e implementa `Autenticabile`**

- Attributo: `materia`
- `autentica()`: accetta solo la password `"prof2024"`
- `presentati()` stampa:
  "Professore [username], insegna: [materia]"

---

**`Segreteria` estende `Utente`**

- Attributo: `ufficio`
- Non implementa: `Autenticabile`
- `presentati()` stampa:
  "Personale di segreteria [username], ufficio: [ufficio]"

---

**4. Eccezione personalizzata `AutenticazioneException`**

- Estende `Exception`
- Costruttore con messaggio

---

### 🧪 Nel `main`

- Crea una `ArrayList<Utente>` con:
  - 1 `Studente`
  - 1 `Professore`
  - 1 `Segreterio` (non autenticabile)
- Per ogni utente nella lista:
  - Chiama `presentati()`
  - Gestisci eventuali eccezioni con `try-catch`
  - Se l'oggetto è Autenticabile, allora **invocare autentica()** e gestire l'eccezione se la password è errata