Lezione 19/03/2024

Insegnante Mauro Bogliaccino

Corso: linguaggio java

i **pilastri** della **programmazione ad oggetti** sono**:**

**Astrazione** ()

**Incapsulamento** (modificatori d’accesso:privato -; pubblico +; protetto #; package/default)

**Ereditarietà** ()

**Polimorfismo**()

i getter “prendono gli elementi su cui lavorare”, è un metodo che restituirà qualcosa

i setter “ modificano gli elementi su cui lavorare”, è un metodo che non restituirà qualcosa

# Classi Java

* Una classe è uno dei concetti fondamentali di OOP.
* Una classe è un modello o un progetto per la creazione di oggetti.
* Una classe non consuma memoria.
* Una classe può essere istanziata più volte.
* Una classe fa una, e solo una, cosa.

## La classe

Una classe è uno dei concetti fondamentali di OOP. Una classe è un insieme di istruzioni necessarie per costruire un tipo specifico di oggetto. Possiamo pensare a una classe come a un modello, un progetto o una ricetta che ci dice come creare oggetti di quella classe.

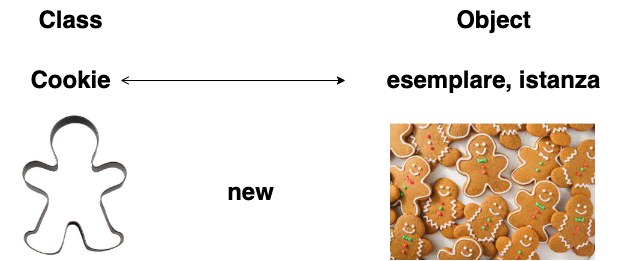
La creazione di un oggetto di quella classe è un processo chiamato istanziazione e di solito viene eseguito tramite la parola chiave new.

Possiamo istanziare tutti gli oggetti che vogliamo. Una definizione di classe non consuma memoria salvata come file sul disco rigido. Una delle migliori pratiche che una classe dovrebbe seguire è il principio di responsabilità singola (SRP): una classe dovrebbe essere progettata e scritta per fare una, e solo una, cosa.

## Java è un linguaggio orientato agli oggetti

* 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

## La classe è lo 'stampo' per gli oggetti



## Le classi definiscono

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

Possono essere definite

* Dal programmatore (p.es. Automobile, Topo, Studente, ...)
* Dall'ambiente Java (p.es. String, System, Scanner, ...)

## La "gestione" di una classe avviene mediante

* Definizione della classe
* Instanziazione di Oggetti della classe

### Struttura di una classe

package model;

public class Persona {

//proprietà private - vedi incapsulamento

private String nome;

private String cognome;

private int eta;

//metodo costruttore

public Persona (String nome, String cognome, int eta) {

this.nome = nome;

this.cognome = cognome;

this.eta = eta;

}

//per gestire le proprietà vedi metodi getters and setters

//metodi...

@Override

public String toString () {

return this.nome + " " + this.cognome + " " + this.eta;

}

}

### Le classi in Java

* 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

## Classi e documentazione

* Java è dotato di una libreria di classi "pronte all'uso" che coprono molte esigenze
* Usare classi già definite da altri è la norma (principio 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](https://docs.oracle.com/javase/8/docs/api/overview-summary.html)
* [Documentazione Java 9](https://docs.oracle.com/javase/9/docs/api/overview-summary.html)
* [Documentazione Java 11](https://docs.oracle.com/en/java/javase/11/)
* [Documentazione Java 14](https://docs.oracle.com/en/java/javase/14/)
* [Documentazione Java 17](https://docs.oracle.com/en/java/javase/17/)

[esempi classi](https://github.com/maboglia/CorsoJava/blob/master/esempi/05_OOP/)

# Tipologie di classi

* All'interno della libreria standard, le classi possono essere suddivise principalmente in due categorie:
  + Classi istanziabili
  + Classi non istanziabili
* Questa stessa distinzione si applica alle nostre classi.
* La separazione tra classi istanziabili e non istanziabili riflette il significato logico del loro utilizzo.
* Il termine "classe non istanziabile" sarà utilizzato per indicare una classe che non dovrebbe essere istanziata, in base alle sue caratteristiche.

## Non ci sono solo oggetti

* Tecnicamente, è possibile utilizzare l'operatore new su classi "non istanziabili" (composte solo da metodi e attributi statici), ma ciò non avrebbe senso pratico.
* Alcune classi, come quelle astratte, non consentono l'utilizzo dell'operatore new.
* La stragrande maggioranza delle classi è istanziabile, ma la presenza di alcune classi non istanziabili è necessaria.
* La classe (essenziale) che contiene il metodo main è normalmente non istanziabile.
* Poiché i numeri non sono oggetti, i metodi numerici appartengono a classi non istanziabili.

## Classi istanziabili

* Una classe istanziabile fornisce il prototipo di una famiglia di oggetti (istanze della classe) che condividono una struttura simile, ma hanno proprietà distinte a livello individuale (valori diversi degli attributi e, di conseguenza, risultati differenti prodotti dai metodi).
* L'uso tipico è la creazione di istanze (tramite new) e quindi l'invocazione di metodi su di esse.
* Nel caso di una classe istanziabile, attributi e metodi rappresentano proprietà possedute da tutti gli oggetti istanza della classe.
* Ogni oggetto istanza di una classe possiede la sua identità e "contiene" individualmente gli attributi e i metodi definiti nella classe.
* Ogni volta che si crea un'istanza con new, si genera un nuovo insieme di attributi e metodi individuali.

## Classi istanziabili II

* In una classe non istanziabile, attributi e metodi sono condivisi a livello della classe, senza istanze distinte che li differenziano.
* Una classe istanziabile rappresenta un concetto che esiste in molteplici versioni individuali, con una struttura comune ma ciascuna con una propria identità:
  + Ad esempio, ci sono molte sequenze di caratteri (la classe String è istanziabile).
  + Esistono diverse valute (la classe Valuta è istanziabile).
  + Ci sono molte persone (ipoteticamente, una classe Persona è istanziabile).

## una classe istanziabile

* Di solito, una classe istanziabile dispone di costruttori.
* Gli attributi e i metodi sono tutti (o quasi tutti) non statici.
* Quando penso all'esecuzione dei suoi metodi, devo immaginare un'istanza individuale a cui applicarli, anche senza argomenti esterni, poiché utilizzano attributi interni.
* Nel caso di classi istanziabili, attributi e metodi sono definiti a livello di istanza.
* Nel caso di classi non istanziabili, attributi e metodi sono definiti a livello di classe.

## Classi non istanziabili

* Una classe non istanziabile contiene un insieme di metodi (e eventualmente attributi) di natura generale, non legati alle proprietà di oggetti individuali specifici.
* La nozione di istanza della classe non ha senso in questo contesto poiché non ci sono caratteristiche differenziabili tra oggetti distinti.
* Una classe non istanziabile rappresenta un concetto concettualmente unico, che non esiste e non può esistere in versioni separate, ognuna con una propria identità:
  + Ad esempio, esiste una sola matematica (la classe Math non è istanziabile).
  + Esiste un unico sistema su cui un programma è eseguito (la classe System non è istanziabile).
  + Esiste un solo punto di inizio per un programma (le classi contenenti il metodo main non sono istanziabili).

## una classe non istanziabile

* Non ha costruttori
* Attributi e metodi sono tutti static
* I suoi metodi sono applicabili direttamente alla classe con almeno un argomento, non è necessaria un'istanza individuale

Math . sqrt (2)

Math . abs ( - 3)

// In memoria ...

Math.E //2.7182

MATH.PI //3.1415

## Classi istanziabili “ibride”

Titolo: Esplorando la Dualità delle Classi Istanziabili: Statiche e Dinamiche

Le classi istanziabili svolgono un ruolo fondamentale nella programmazione orientata agli oggetti, rappresentando concetti che possono esistere in molteplici versioni individuali. Tuttavia, in alcuni casi, queste classi mostrano una dualità intrigante, incorporando sia aspetti statici che dinamici.

Ibridi: Classi Istanziabili con Natura Statica e Dinamica

Un esempio comune di questa dualità si trova in classi come String della libreria standard di Java. Sebbene siano classificate come istanziabili, contengono attributi o metodi statici, creando una sorta di ibridismo nella loro natura.

È come se queste classi avessero due sottoparti distintive: una statica e una dinamica, ciascuna operante secondo le proprie regole. La parte dinamica rappresenta le istanze specifiche di quella classe, ciascuna con la propria identità e dati unici. D'altra parte, la parte statica è condivisa tra tutte le istanze e può essere utilizzata senza dover istanziare un oggetto specifico.

Consigli sulla Creazione di Classi Ibride

Sebbene questa dualità possa sembrare interessante, è importante notare che, in generale, la creazione di classi istanziabili ibride è sconsigliata, salvo rari casi. Mentre l'uso di attributi costanti definiti come statici può essere accettabile, combinare aspetti statici e dinamici all'interno di una stessa classe può complicare la progettazione e la comprensione del codice.

Le classi istanziabili sono progettate per rappresentare concetti che esistono in molteplici forme, ciascuna con le proprie caratteristiche dinamiche. Introdurre elementi statici in questo contesto può portare a una confusione concettuale e potenziali problemi di manutenzione del codice.

In conclusione, mentre alcune classi istanziabili possono sfidare le convenzioni includendo elementi statici, è essenziale valutare attentamente la necessità di questa dualità nella progettazione del software. In genere, è consigliabile mantenere la distinzione tra classi istanziabili e non istanziabili, preservando la chiarezza e la coerenza nella struttura del codice.

[esempi classi](https://github.com/maboglia/CorsoJava/blob/master/esempi/05_OOP/)

# 1. Instanze di una Classe: gli oggetti

In programmazione orientata agli oggetti (OOP), un oggetto è uno dei concetti centrali, rappresentando un'entità del mondo reale dotata di stato e comportamenti. Gli oggetti hanno attributi, chiamati campi, che definiscono il loro stato, e metodi, che determinano il loro comportamento. Ogni oggetto rappresenta un'istanza di una classe, fungendo da esempio concreto dei concetti astratti definiti dalla classe stessa. Gli oggetti occupano spazio nella memoria del sistema, consentendo loro di mantenere uno stato persistente. Un aspetto fondamentale degli oggetti è la loro capacità di comunicare tra loro, permettendo lo scambio di informazioni e il coordinamento delle attività all'interno di un programma. Questa interazione tra oggetti è uno dei pilastri della programmazione orientata agli oggetti, consentendo la creazione di sistemi complessi e dinamici.

## 1.1. Cos'è un oggetto?

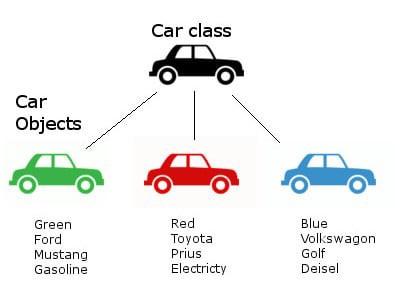
* Un oggetto è uno dei concetti fondamentali di OOP.
* Un oggetto è un'entità del mondo reale.
* Un oggetto ha stato (campi) e comportamenti (metodi: modi per compiere azioni).
* Un oggetto rappresenta un'istanza di una classe.
* Un oggetto occupa un po' di spazio nella memoria.
* Un oggetto può comunicare con altri oggetti.

Gli oggetti sono istanze (esemplari) delle classi

Gli oggetti sono caratterizzati da

* Classe di appartenenza - tipo (ne descrive attributi e metodi)
* Stato (valore attuale degli attributi)
* Identificatore univoco (reference - handle - puntatore)

Leggi tutto



## 1.2. Per creare un oggetto occorre

* Dichiarare una istanza
* La dichiarazione non alloca spazio ma solo una riferimento (puntatore) che per default vale null
* Allocazione e inizializzazione
* Riservano lo spazio necessario creando effettivamente l'oggetto appartenente a quella classe

## 1.3. Notazioni Puntate

* Le notazioni puntate possono essere combinate
* System.out.println("Hello world!");
* System è una classe del package java.lang
* out è una variabile di classe contenente il riferimento ad un oggetto della classe PrintStream che punta allo standard output
* println() è un metodo della classe PrintStream che stampa una linea di testo

## 1.4. Operazioni su reference == e !=

* Attenzione: il test di uguaglianza viene fatto sul puntatore (reference) e NON sull'oggetto
* gli operatori relazionali == e != stabiliscono se i reference si riferiscono allo stesso oggetto
* È definita l'assegnazione con l'operatore =
* È definito l'operatore punto (notazione puntata)
* In java NON è prevista l'aritmetica dei puntatori

## 1.5. Operazioni su istanze

Le principali operazioni che si possono effettuare sulle variabili che riferiscono istanze di una classe sono:

* assegnamento
* confronto
* invocazione di metodi
* Il valore di una variabile di tipo reference è il riferimento ad un oggetto (istanza di una classe)
* Una stessa variabile può riferire oggetti diversi in tempi diversi a seguito di operazioni di assegnazione sul suo valore
* Se la variabile contiene il valore null non riferisce nessun oggetto in quel momento

## 1.6. Accesso a metodi e attributi non static

* All'interno del corpo di un metodo che non è dichiarato static, è possibile accedere a qualsiasi attributo e metodo della stessa classe.
* Nel contesto di un metodo, è consentito riferirsi in modo abbreviato agli attributi e ai metodi definiti all'interno della stessa classe.
* Quando, all'interno di un metodo non static, compare il nome di un metodo o attributo anch'esso non static, è implicito che si stia facendo riferimento all'istanza specifica su cui è stato invocato il metodo.

## 1.7. Oggetti e riferimenti

* Mentre le variabili sono dotate di un nome distintivo, gli oggetti non lo sono.
* Per interagire con un oggetto, è necessario passare attraverso una variabile che contiene il suo riferimento.
* Un singolo oggetto può essere accessibile tramite diverse variabili, consentendo l'utilizzo di nomi variabili per accedervi.
* Il rapporto tra variabili e gli oggetti a cui fanno riferimento è dinamico; il riferimento iniziale non è necessariamente vincolato all'oggetto per l'intera durata della sua esistenza.
* Quando nessuna variabile fa più riferimento a un oggetto, quest'ultimo diventa irraggiungibile e il garbage collector interviene per liberare le risorse associate.

## 1.8. Confronti tra variabili di tipo strutturato

* Gli operatori di confronto == e != possono essere impiegati con variabili di tipo strutturato.
* Quando uno dei due termini del confronto è il valore null, l'analisi verifica se una specifica variabile fa riferimento a un oggetto o se è priva di un riferimento, ad esempio, nell'espressione saluto3 != null.
* Quando entrambi i termini del confronto sono variabili, il confronto determina se possiedono lo stesso valore, indicando se fanno riferimento esattamente allo stesso oggetto.

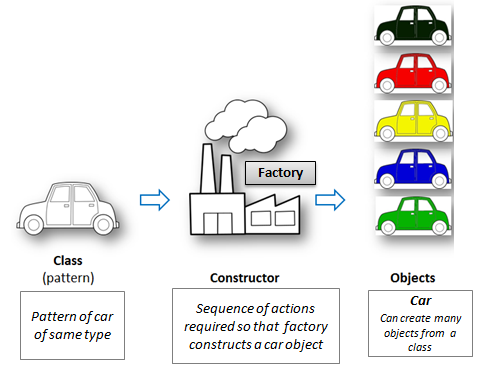
## 1.9. Confronto tra riferimenti vs. confronto tra oggetti

* L'operatore == confronta i riferimenti, non i valori, contenuti negli oggetti.
* Tipicamente, si desidera confrontare i contenuti, non i riferimenti, ed è per questo che si utilizza il metodo equals().
* Il metodo booleano equals della classe String richiede come argomento il riferimento a un altro oggetto e restituisce true se le stringhe contenute sono uguali, altrimenti restituisce false.
* In modo simile, il metodo booleano equalsIgnoreCase effettua la stessa operazione senza fare distinzione tra maiuscole e minuscole.

[esempi classi](https://github.com/maboglia/CorsoJava/blob/master/esempi/05_OOP/)

# Il Metodo Costruttore

Specifica le operazioni di inizializzazione (attributi, etc.) che vogliamo vengano eseguite su ogni oggetto della classe appena viene creato



## Caratteristiche del costruttore

Il metodo costruttore ha

* Lo stesso nome della classe: inizia quindi con lettera Maiuscola
* Tipo di ritorno non specificato

Non possono esistere attributi non inizializzati

* Gli attributi vengono inizializzati comunque con valori di default

## il costruttore è sempre presente

Se *non viene dichiarato* un costruttore, ne viene creato uno di default *vuoto e senza parametri*

Spesso si usa l'overloading dei metodi definendo diversi costruttori per gli oggetti di un certo tipo

La distruzione di oggetti non è a carico del programmatore (garbage-collection)

### Il costrutto new

* Crea una nuova istanza della classe specificata, allocandone la memoria
* Restituisce il riferimento all'oggetto creato
* Chiama il costruttore del nuovo oggetto

Automobile a = new Automobile ();

Motorcycle m = new Motorcycle ();

String s = new String ("ABC");

### Per "gestire" una classe occorre

* Accedere mediante il reference ai metodi della classe
* Accedere mediante il reference agli attributi della classe

### inviare Messaggi

* L'invio di un messaggio provoca l'esecuzione del metodo

Inviare un messaggio ad un oggetto

* Usare la notazione "puntata" oggetto.messaggio(parametri)
* I metodi definiscono l'implementazione delle operazioni
* I messaggi che un oggetto può accettare coincidono con i nomi dei metodi
* p.es mettiInMoto(), frena(), accelera(100), etc.
* Spesso i messaggi includono uno o più parametri
* .vernicia("Rosso")

### Esempi

Automobile a = new Automobile();

a.mettiInMoto();

a.accelera(10);

a.frena(5);

a.spegni();

### All'interno della classe

* I metodi che devono inviare messaggi allo stesso oggetto cui appartengono non devono obbligatoriamente utilizzare la notazione puntata: è sottinteso il riferimento

public class Libro {

int nPagine;

public void leggiPagina (int nPagina) {

// ...

}

public void leggiTutto () {

for (int i=0; i<nPagine; i++)

leggiPagina (i);

}

}

## Attributi

* Stessa notazione "puntata" dei messaggi oggetto.attributo
* Il riferimento viene usato come una qualunque variabile

Automobile a=new Automobile();

a.colore = "Blu";

boolean x = a.accesa;

I metodi che fanno riferimento ad attributi dello stesso oggetto possono tralasciare il riferimento all'oggetto

public class Automobile {

String colore;

void vernicia(){

colore = "Verde";// colore si riferisce all'oggetto corrente

}

}

## Operatore this (Puntatore Auto-referenziante)

La parola riservata this e' utilizzata quale puntatore auto-referenziante

* this riferisce l'oggetto (e.g., classe) corrente

Utilizzato per:

* Referenziare la classe appena istanziata
* Evitare il conflitto tra nomi

### Esempio (costruttori con overloading e this)

class Automobile {

String marca;

String modello;

// Automobile senza marca nè modello

Automobile () {

//...

}

// Automobile con marca senza modello

Automobile (String marca) {

//...

this.marca = marca;

}

// Automobile con marca e modello

Automobile (String marca, String modello) {

//...

this.marca = marca;

this.modello = modello;

}

}

[esempi classi](https://github.com/maboglia/CorsoJava/blob/master/esempi/05_OOP/)

# Metodi getter e setter

I cosiddetti metodi getters e setters, cioè accessori e mutatori, regolano l'accesso alle proprietà ed ai metodi delle classi

## Principi di encapsulation e visibilità

* Modularità = diminuire le interazioni
* Information Hiding = delegare responsabilità
* private attributo/metodo visibile solo da istanze della stessa classe
* public attributo/metodo visibile ovunque

### Ad esempio

class NomeClasse{

private double nome1, nome2;

public double getNome1(){

return nome1;

}

public void setNome1(double nome1){

this.nome1 = nome1;

}

* Vantaggi:
* possiamo cambiare la rappresentazione interna,
* verificare che i valori siano corretti,
* modificare altri aspetti dell'oggetto

## Getters e setters

Metodi accessori e mutatori per leggere scrivere un attributo privato

String getColore() {

return colore;

}

void setColore(String nuovoColore) {

this.colore = nuovoColore;

}

## Proprietà visibili

class Automobile {

public String colore;

}

//creazione oggetto

Automobile a = new Automobile();

a.colore = "bianco"; // ok

## Classe incapsulata

class Automobile {

private String colore;

public void vernicia(String colore) {

this.colore = colore;

}

}

//creazione oggetto

Automobile a = new Automobile();

a.colore = "bianco"; // errore

a.vernicia("verde"); // ok

[esempi classi](https://github.com/maboglia/CorsoJava/blob/master/esempi/05_OOP/)

# Incapsulamento e visibilità in Java

* due aspetti che risultano fondamentali in un software:
  + Interfaccia: definita come gli elementi che sono visibili dall'esterno, cioè come il sw può essere utilizzato
  + Implementazione: la realizzazione pratica interna dei metodi e la loro interazione con le proprietà degli oggetti
* L'incapsulamento è uno dei concetti fondamentali di OOP.
* L'incapsulamento è la tecnica mediante la quale lo stato dell'oggetto è nascosto dal mondo esterno e viene esposto un insieme di metodi pubblici per l'accesso a questo stato.
* L'incapsulamento si ottiene quando ogni oggetto mantiene il proprio stato privato, all'interno di una classe.
* L'incapsulamento è noto come meccanismo di occultamento dei dati.
* L'incapsulamento ha una serie di importanti vantaggi ad esso associati, come codice debolmente accoppiato, riutilizzabile, sicuro e facile da testare.
* In Java, l'incapsulamento viene implementato tramite i modificatori di accesso: public, private e protected.

## 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

### 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
* package: se non indichiamo la visibilità: sono accessibili solo dalle classi del nostro package

### 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
* per consultarli e modificarli aggiungiamo i metodi accessori e mutatori: getters e setters

### 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)

[esempi classi](https://github.com/maboglia/CorsoJava/blob/master/esempi/05_OOP/)

# Astrazione

* L'astrazione è uno dei concetti fondamentali di OOP.
* L'astrazione è il concetto di esporre all'utente solo quelle cose che sono rilevanti per lui e nascondere il resto dei dettagli.
* L'astrazione consente all'utente di concentrarsi su ciò che fa l'applicazione invece che su come lo fa.
* L'astrazione si ottiene in Java tramite classi e interfacce astratte.

L'astrazione è uno dei principali concetti OOP che si sforzano di rendere le cose il più semplici possibile per l'utente. In altre parole, l'astrazione espone l'utente solo alle cose che sono rilevanti per lui e nasconde il resto dei dettagli. In termini OOP, diciamo che un oggetto dovrebbe esporre ai suoi utenti solo un insieme di operazioni di alto livello, mentre l'implementazione interna di tali operazioni è nascosta.

Quindi, l'astrazione consente all'utente di concentrarsi su ciò che fa l'applicazione invece che su come lo fa. In questo modo, l'astrazione riduce la complessità dell'esposizione delle cose, aumenta la riusabilità del codice, evita le duplicazioni del codice e sostiene un basso accoppiamento e un'elevata coesione. Inoltre, mantiene la sicurezza e la discrezione dell'applicazione esponendo solo i dettagli importanti.

## In ingegneria del software

Nell'ingegneria del software e nell'informatica, l'astrazione è:

Il processo di rimozione o generalizzazione di dettagli o attributi fisici, spaziali o temporali nello studio di oggetti o sistemi per focalizzare l'attenzione su dettagli di maggiore importanza; è di natura simile al processo di generalizzazione;

la creazione di oggetti-concetto astratti rispecchiando caratteristiche o attributi comuni di vari oggetti o sistemi di studio non astratti – il risultato del processo di astrazione.

## Modellare la realtà

L'astrazione, in generale, è un concetto fondamentale nell'informatica e nello sviluppo del software. Il processo di astrazione può anche essere definito modellazione ed è strettamente correlato ai concetti di teoria e progettazione. I modelli possono anche essere considerati tipi di astrazioni per la loro generalizzazione di aspetti della realtà.

L'astrazione in informatica è strettamente correlata all'astrazione in matematica a causa della loro comune attenzione alla costruzione di astrazioni come oggetti, ma è anche correlata ad altre nozioni di astrazione utilizzate in altri campi come l'arte.

La pratica consiste nel presentare il sistema, ad esempio un pezzo di codice sorgente o uno scambio di trasmissioni di dati, in maniera ridotta ai soli dettagli considerati essenziali all'interesse specifico, ad esempio raggruppando il codice in una funzione o formalizzando un protocollo di comunicazione.

## Codice astratto

Astrazione indica anche quanto il codice scritto in un linguaggio di programmazione si distacca dalle istruzioni in linguaggio macchina che ad esso corrisponderanno dopo l'operazione di compilazione.

Delle istruzioni scritte in Java, per esempio, sono molto più vicine al linguaggio comprensibile all'uomo piuttosto che a quello comprensibile dalla macchina (alto livello di astrazione).

Viceversa delle istruzioni scritte in Assembly sono abbastanza vicine (ma non uguali) alle istruzioni in formato comprensibile alla macchina (basso livello di astrazione).

La seguente definizione di astrazione aiuta a capire come questo termine viene applicato all'informatica:

«Astrazione - Un concetto o un'idea non associata a nessuna istanza specifica.»



## Astrazione matematica

Il concetto è originato mediante l'analogia con l'astrazione in matematica. La tecnica matematica dell'astrazione comincia con definizioni matematiche; Ad esempio, sia in Informatica che in Matematica i numeri sono concetti nei linguaggi di programmazione. I dettagli dell'implementazione dipendono dall'hardware e dal software, ma non è una restrizione, poiché il concetto di numero in Informatica è ancora basato sul concetto matematico.

Le astrazioni possono anche riferirsi a oggetti e sistemi del mondo reale, regole di sistemi computazionali o regole di linguaggi di programmazione che portano o utilizzano caratteristiche dell'astrazione stessa, come:

l'utilizzo di tipi di dati per eseguire l'astrazione dei dati per separare l'utilizzo dalle rappresentazioni funzionanti delle strutture di dati all'interno dei programmi;

il concetto di procedure, funzioni o subroutine che rappresentano una specifica dell'implementazione del flusso di controllo nei programmi; le regole comunemente denominate "astrazione" che generalizzano le espressioni utilizzando variabili libere e vincolate nelle varie versioni del calcolo lambda;

il processo di riorganizzazione del comportamento comune da classi non astratte in "classi astratte" utilizzando l'ereditarietà per astrarre su sottoclassi come si vede nei linguaggi di programmazione C++ e Java orientati agli oggetti.

## Strutture di controllo e strutture dati

In programmazione, l'astrazione può essere applicata a strutture di controllo o a strutture di dati.

* L'astrazione di strutture di controllo implica l'uso di sottoprogrammi e di altri concetti relativi a queste.
* Le strutture di dati astratte permettono di maneggiare i dati in maniere significanti.

Il fatto che i programmatori utilizzino le astrazioni quando possibile per evitare duplicazioni (solitamente di codice) è noto come principio di astrazione.

## Astrazione nella programmazione orientata agli oggetti

Nella teoria della programmazione orientata agli oggetti, l'astrazione implica la possibilità di definire oggetti che rappresentano "attori" astratti che possono eseguire lavori, riferire e cambiare il loro stato e "comunicare" con altri oggetti nel sistema. Il termine incapsulamento si riferisce all'occultamento dei dettagli dello stato, ma l'estensione del concetto di tipo di dati dai linguaggi di programmazione precedenti per associare il comportamento in modo più forte ai dati e standardizzare il modo in cui interagiscono diversi tipi di dati è l'inizio dell'astrazione.

Quando l'astrazione procede nelle operazioni definite, consentendo la sostituzione di oggetti di tipo diverso, si parla di polimorfismo. Quando si procede in direzione opposta, all'interno dei tipi o delle classi, strutturandoli per semplificare un insieme complesso di relazioni, si parla di delega o ereditarietà.

Vari linguaggi di programmazione orientati agli oggetti offrono strutture simili per l'astrazione, il tutto per supportare una strategia generale di polimorfismo nella programmazione orientata agli oggetti, che include la sostituzione di un tipo con un altro nello stesso ruolo o simile.

## ADT

Un Tipo di Dato Astratto o Abstract Data Type - ADT è, per definizione, un nuovo tipo di dato che estende i tipi nativi forniti dal linguaggio di programmazione. Un ADT è caratterizzato da un insieme di:

* dati;
* operazioni che agiscono sui dati, leggengoli/scrivendoli;

Anche i linguaggi procedurali, come per esempio C, consentono di definire un ADT. Ma, mentre per tali linguaggi chiunque può avere accesso ai dati e modificarli, i linguaggi Object Oriented ne garantiscono la loro riservatezza.

La classe consente di implementare gli ADT attraverso il meccanismo di incapsulamento: i dati devono rimanere privati insieme all’implementazione e solo l’interfaccia delle operazioni è resa pubblica all’esterno della classe.

Questo approccio è fondamentale per garantire che nessuno possa accedere alle informazioni della classe e quindi, dal punto di vista del programmatore, è una garanzia per non fare errori nella stesura del codice Quindi una classe implementa un ADT (un sinonimo di classe è proprio tipo) attraverso il meccanismo di incapsulamento.

## La classe

La descrizione di una classe deve elencare:

* i dati (o attributi): contengono le informazioni di un oggetto;
* le operazioni (o metodi): consentono di leggere/scrivere gli attributi di un oggetto;

Quando si scrive una applicazione è buona norma iniziare con la progettazione dell’applicazione stessa:

* identificare le classi;
* identificare le funzionalità di queste classi;
* trovare le relazioni fra le classi;

## L’oggetto

Per definizione, diciamo che un oggetto è una istanza di una classe. Quindi un oggetto deve essere conforme alla descrizione di una classe.

Un oggetto pertanto è contraddistinto da:

1. attributi;
2. metodi;
3. identità;

[esempi classi](https://github.com/maboglia/CorsoJava/blob/master/esempi/05_OOP/)

# Associazione

* L'associazione è uno dei concetti fondamentali dell'OOP.
* L'associazione definisce la relazione tra due classi indipendenti l'una dall'altra.
* L'associazione non ha proprietario.
* L'associazione può essere uno a uno, uno a molti, molti a uno e molti a molti.

L'associazione è uno dei concetti fondamentali dell'OOP.

L'obiettivo dell'associazione è definire la relazione tra due classi indipendenti l'una dall'altra ed è anche indicata come relazione di molteplicità tra oggetti.

## L'associazione non ha proprietario

Non c'è nessun titolare dell'associazione.

Gli oggetti coinvolti in un'associazione possono usarsi l'un l'altro (associazione bidirezionale), oppure uno solo usa l'altro (associazione unidirezionale), ma hanno una loro durata di vita.

L'associazione può essere unidirezionale/bidirezionale, uno-a-uno, uno-a-molti, molti-a-uno e molti-a-molti.

# Aggregazione

* L'aggregazione è uno dei concetti fondamentali di OOP.
* L'aggregazione è un caso particolare di associazione unidirezionale.
* L'aggregazione rappresenta una relazione HAS-A.
* Due oggetti aggregati hanno il proprio ciclo di vita, ma uno degli oggetti è il proprietario della relazione HAS-A.

L'aggregazione è uno dei concetti fondamentali di OOP. Principalmente, l'aggregazione è un caso speciale di associazione unidirezionale.

## HAS-A

Mentre un'associazione definisce la relazione tra due classi indipendenti l'una dall'altra, l'aggregazione rappresenta una relazione HAS-A tra queste due classi.

In altre parole, due oggetti aggregati hanno il proprio ciclo di vita, ma uno degli oggetti è il proprietario della relazione HAS-A.

Avere il proprio ciclo di vita significa che la fine di un oggetto non influirà sull'altro oggetto.

# Composizione

* La composizione è uno dei concetti chiave di OOP.
* La composizione è un caso più restrittivo di aggregazione.
* La composizione rappresenta una relazione HAS-A che contiene un oggetto che non può esistere da solo.
* La composizione sostiene il riutilizzo del codice e il controllo della visibilità degli oggetti.

La composizione è uno dei concetti chiave di OOP. In primo luogo, la composizione è un caso più restrittivo di aggregazione. Mentre l'aggregazione rappresenta una relazione HAS-A tra due oggetti che hanno il proprio ciclo di vita, la composizione rappresenta una relazione HAS-A che contiene un oggetto che non può esistere da solo.

## PART-OF

Per evidenziare questo accoppiamento, la relazione HAS-A può anche essere chiamata PART-OF. Ad esempio, un'auto ha un motore. In altre parole, il motore è PARTE DELL'auto. Se l'auto viene distrutta, anche il motore viene distrutto. Si dice che la composizione sia migliore dell'ereditarietà perché sostiene il riutilizzo del codice e il controllo della visibilità degli oggetti.

## Rappresentazione UML

