**OOP**

**La OOP è un tipo di programmazione non procedurale (quindi non esegue istruzione per istruzione) in cui vengono creati degli oggetti che interagiscono tra loro. Ha 4 proprietà fondamentali:**

1. **INCAPSULAMENTO:**

**E’ il modo che abbiamo per raggruppare le informazioni e le funzionalità di un oggetto, all’interno dell’oggetto stesso (O CLASSE)**

**INFORMATION HIDING: con l’IH possiamo rendere privati, pubblici e protetti attributi e metodi.**

**Rendendoli privati ci assicuriamo che siano inaccessibili dall’esterno.**

**Quelli protetti sono inaccessibili dall’esterno ma visibili dalle classi che ereditano.**

**Quelli pubblici sono accessibili da ogni parte del programma.**

1. **POLIMORFISMO:**

**\*\* E’ la capacità di un oggetto di assumere più forme. Con il polimorfismo possiamo dichiarare degli oggetti con la loro superclasse e in automatico ognuno riconsocerà il proprio metodo da utilizzare\*\***

1. **EREDITARIETA’:**

**CLASSE**

Una classe è un tipo (o modello base) di un oggetto, che ha degli attributi (ovvero delle caratteristiche) e dei metodi (delle funzionalità).

**ISTANZA:**

Oggetto concreto creato a partire da una classe

**A COSA SERVONO I DUE PUNTI QUANDO CREIAMO UNA CLASSE? Studente : Persona**

Significano EXTENDS ( estendere ) perché con la sottoclasse estendiamo attributi e metodi della superclasse.

Quindi Studente estende Persona ed eredita attributi e metodi

**PROPERTY**

Le property servono per dichiarare un attributo di una classe che sarà privato ma che avrà i metodi get e set (che serviranno per leggerlo e modificarlo) che saranno pubblici e quindi accessibili dall’esterno.

**CLASSE OBJECT**

La classe Object è la classe genesi da cui derivano ed ereditano tutti i tipi

**CLASSE ASTRATTA**

E’ una classe non permette la sua implementazione. Può contenere attributi e metodi che possono essere estesi dalle sue sottoclassi. Può avere dei metodi astratti:

**METODI ASTRATTI:**

sono dei metodi senza corpo, che devono essere ridefiniti nelle sottoclassi.

**VIRTUAL:**

La keyowrd virtual si utilizza per permettere ai metodi una superclasse di essere sovrascritti in una sottoclasse.

**OVERRIDE:**

La keyword Override si utilizza per sovrascrivere (o ridefinire) dei metodi

**COSTRUTTORE:**

Il **costruttore** è un tipo speciale di funzione in una classe che viene eseguita automaticamente ogni volta che crei un nuovo oggetto di quella classe

**COSTRUTTORE DI DEFAULT:**

Un **costruttore di default** non ha parametri (o utilizza solo valori predefiniti) e viene eseguito senza passare argomenti durante la creazione dell'oggetto.

Permette di allocare in memoria gli attributi della superclasse quando genero un’istanza della sottoclasse

**SPIEGA BASE E THIS?**

Base si riferisce agli attributi della classe genitore (superclasse) mentre this si riferisce a questo oggetto (sicuramente sottoclasse)

**SIGNATURE:**

E’ la descrizione completa delle informazioni per per richiamare una funzione, quali nome, numero , tipo di parametri e valori di ritorno.

**OVERLOAD:**

I metodi possono avere lo stesso nome ma se hanno parametri o tipi diversi , appunto la signature, possono avere diverse funzioni

**PATTERN:**

Soluzione strategica per problemi noti e ricorrenti

**INTERFACCIA:**

E’ un costrutto che ha solamente metodi ed obbliga le classi che la ereditano ad implementare i suoi metodi

**INTERFACCIA<T>:**

L’interfaccia con il tipo T è un’interfaccia generica che grazie a questo parametro <T> mi consente di richiamare al suo interno uno dei tipi che implementano la data interfaccia. (Interface<Automobile>)

**SOLID:**

I principi SOLID sono intesi come linee guida per lo sviluppo di software leggibile, estendibile e [manutenibile](https://it.wikipedia.org/wiki/Manutenzione_(software)).

5 principi fondamentali della OOP:

* **S: principio di singola responsabilità**
  + ogni classe dovrebbe avere una ed una sola responsabilità incapsulate al suo interno
* **O: principio aperto chiuso**
  + Le entità devono essere aperte alle estensioni (devono essere ereditabili) ma chiuse alle modifiche
* **L: principio di sostituzione di liskov**
  + Gli oggetti devono poter essere sostituiti con dei loro sottotipi ( polimorfismo)
* **I: principio di segregazione delle interfacce**
  + È preferibile avere più interfacce specifiche che una singola generica (una per la lettura, e una per la scrittura ad esempio)
* **D: principio di inversione delle dipendenze**

**DAO**

**Il pattern DAO o DATA ACCESS OBJECT si utilizza per separare la logica di accesso ai dati del database, ad esempio le operazioni CRUD dalla logica di business.**

**SINGLETON**

**Il singleton è un pattern utilizzato per garantire che una data classe abbia una sola istanza. In pratica se si prova a creare più oggetti di questa classe il singleton ci restituirà sempre lo stesso oggetto.**

**Innanzitutto bisogna**

* **rendere impossibile l’uso del costruttore new rendendo private la funzione del costruttore in modo che non sia accessibile dall’esterno ma solo all’interno della classe stessa;**
* **creare un metodo pubblico che crei l’istanza della classe se già non esiste, oppure restituire l’istanza già esistente**

**Questo pattern è utile per risparmiare memoria e tempo di elaborazione.**