Un Linguaggio di Programmazione Orientato a Ogetti

La Programmazione Orientata agli Oggetti è un paradigma di programmazione che organizza il software attorno a "oggetti", che sono istanze di "classi". Ogni oggetto rappresenta un'entità concreta o astratta, e ha:

Stato → rappresentato dai dati (proprietà/attributi) Comportamento → definito dai metodi

CLASSE –> è un tipo di dato definito dall'Utente.

E' un modello o prototipo che descrive le caratteristiche comuni di un tipo di oggetto. Definisce attributi (ovvero proprietà) e metodi (funzionalità).

OGGETTO –> un valore concreto di quel tipo

E' un'istanza concreta di una classe. Ha un proprio stato e può eseguire delle azioni.

ISTANZA –> un singolo oggetto creato da una classe

esempio in JAVA: class Cane { String nome; int eta;

```
void abbaia() {
    System.out.println(nome + " dice: Bau!");
}
```

}

public class Main { public static void main(String[] args) { Cane c1 = new Cane(); c1.nome = "Fido"; c1.eta = 3; c1.abbaia(); // Output: Fido dice: Bau! } }

EREDITARIETA'

Permette a una classe di ereditare attrinuti e metodi da un'altra classe. Favorisce il riutilizzo del codice e la specializzazione.

INCAPSULAMENTO Consiste nel proteggere i dati interni di un oggetto nascondendoli dall'esterno, e fornendo metodi controllati per accedervi o modificarli. Usa solitamente: -attributi privati -getter e setter pubblici.

POLIFORMISMO

Significa "molte forme". Permette di chiamare lo stesso metodo su oggetti diversi ottenendo comportamenti differenti. Avere metodi con lo stesso nome ma funzionalità diverse a seconda del contesto (overloading / overriding).

Concetto	Descrizione
Costruttori	Metodi speciali per inizializzare oggetti
Interfacce	Contratti di metodi che una classe deve implementare
Classi astratte	Classi base non istanziabili, con metodi da implementare

Concetto	Descrizione
Composizione	Un oggetto può "contenere" altri oggetti (es. un'auto ha un motore)
Overloading	Metodi con lo stesso nome ma parametri diversi
Overriding	Una sottoclasse ridefinisce un metodo della superclasse

EX: class Animale { constructor(nome) { this.nome = nome; }

```
parla() {
    console.log(`${this.nome} fa un verso.`);
}
```

}

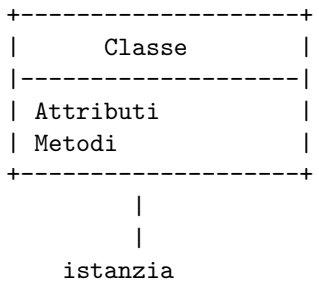
class Cane extends Animale { parla() { console.log(`${this.nome} abbaia.`); } }

const fido = new Cane("Fido"); fido.parla();

Qui Animale è una classe base. Cane estende Animale e sovrascrive il metodo 'parla' Questo permette di modellare sistemi complessi in modo più vicino alla realtà. Favorisce il riuso, organizzazione e manutenzione del codice.

I PRINCIPI SOLID. Le 5 regole dell'OOP

1. Single Responsability - S Ogni classe deve avere una sola responsabilità(un solo motivo per cambiare)
2. Open/Closed Principle - O Le classi devono essere aperte all'estensione, ma chiuse alla modifica. Cioè si può estendere una classe, ma non modificarne il comportamento originale.
3. Liskov Substitution - L Le classi derivate devono poter essere usate al posto di quelle base senza rompere la logica del programma
4. Interface Segregation - I Meglio avere molte interfacce speficifiche piuttosto che una generale e pesante
5. Dependency Inversion - D Le classi devono dipendere da astrazioni, non da classi concrete.



```
          |
+---------------------+
|     Oggetto         |
|---------------------|
| Stato attuale       |
| Comportamento       |
+---------------------+
```