Objcted-Oriented

Cos'è un oggetto?

Gli oggetti sono delle entità del programma che rappresentano la realtà circostante. Per ogni oggetto ci interessa gli **stati interni in cui esso** si può trovare e i **comportamenti** che esso può avere.

Un oggetto può memorizzare i suoi stati in *campi* e può esporre i comportamenti tramite *metodi*.

> I campi sono variabili - mentre i metodi sono funzioni.

I metodi permettono di operare sugli stati interni dell'oggetto e sono il meccanismo primario per far comunicare due oggetti.

> Nascondere lo stato interno e richiedere che tutte le interazioni siano performati tramite metodi dell'oggetto è noto come **data encapsulation.**

Unire i codici in software oggetti individuali provede diversi vantaggi:

- **Modularità**: Il codice di un oggetto è mantenuto in maniera indipendente rispetto agli altri oggetti.
- **Information-Hiding:** Interagendo solo tramite i metodi - i dettagli interni degli oggetti rimangono nascosti.
- **Codice ri-utilizzabile:** Se il codice di un oggetto già esiste - allora possiamo usarlo per diversi software.
- **Semplicità di aggiunta e debug:** Se un oggetto crea delle problematiche nel programma possiamo semplicemente eliminare quell'oggetto e sostituirlo con un altro.

Cos'è una classe?

Nel mondo reale - ci sono oggetti individuali che sono dello stesso tipo.

> Ad esempio esistoni miglaia di biciclette dello stesso modello. Ogni bicicletta è costruit dallo stesso *blueprint*.

Nella programmazione orientata ad oggetti diciamo che il nostro *oggetto* bicicletta sarà un'**istanza** della *classe do oggetti* nota come bicicletta.

> La classe è il blueprint.

```
class Bicycle {

    int cadence = 0;
    int speed = 0;
    int gear = 1;

    void changeCadence(int newValue) {
         cadence = newValue;
    }

    void changeGear(int newValue) {
         gear = newValue;
    }

    void speedUp(int increment) {
         speed = speed + increment;
    }

    void applyBrakes(int decrement) {
         speed = speed - decrement;
    }

    void printStates() {
         System.out.println("cadence:" +
             cadence + " speed:" +
             speed + " gear:" + gear);
    }
}
```

Notiamo come il codice fornito non ha un metodo main - perché non è un'applicazione completa e solo il blueprint di un oggetto che potrebbe essere usato in un'applicazione. Il compito di creare un nuovo oggetto bicicletta spetta a qualche altra classe nella tua applicazione:

```
class BicycleDemo {
    public static void main(String[] args) {

        // Create two different
        // Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        // Invoke methods on
```

```
        // those objects
        bike1.changeCadence(50);
        bike1.speedUp(10);
        bike1.changeGear(2);
        bike1.printStates();

        bike2.changeCadence(50);
        bike2.speedUp(10);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.speedUp(10);
        bike2.changeGear(3);
        bike2.printStates();
    }
}

The output of this test prints the ending pedal cadence, speed, and gear for
the two bicycles:

cadence:50 speed:10 gear:2
cadence:40 speed:20 gear:3
```

Cos'è l'Inheritance?

Diversi tipi do oggetti potrebbero avere in comune alcune caratteristiche fra di loro. Per esempio esistono diversi tipi di bicilette - da passeggio - tandem - tricicli e così via. Ovviamente questi oggetti hanno anche delle caratteristiche 'personali'.

La programmazione ad oggetti permette alle classi di **ereditare** stati e comportamenti da altre classi. Nell'esempio che abbiamo fatto fino ad ora possiamo aggiungere una *sottoclasse* per tipi particolare di bicicletta:

```
class MountainBike extends Bicycle {

    // new fields and methods defining
    // a mountain bike would go here

}
```

La clausola **extends** permette di estendere tutti i campi e metodi della classe bicicletta.

La classe da cui si ereditano le caratteristiche si chiama **superclasse** - ogni classe può avere una sola superclasse diretta. Ogni superclasse ha potenzialmente infinite sottoclassi.

Cos'è un'Interfaccia?

Gli oggetti definiscono il modo in cui interagiscono con il mondo tramite dei metodi. L'insieme dei metodi forma **l'interfaccia dell'oggetto** - ossia un gruppo di metodi correlati fra loro con i corpi vuoti.

L'interfaccia di una bicicletta potrebbere apparire come segue:

```
interface Bicycle {
    //  wheel revolutions per minute
    void changeCadence(int newValue);

    void changeGear(int newValue);

    void speedUp(int increment);

    void applyBrakes(int decrement);
}
```

Quando una classe **implementa** un'interfaccia **è obbligata a fornire una propria implementazione concreta** di tutti i metodi dichiarati nell'interfaccia. Per esempio:

```
interface Animale {
    void faiVerso();
}

class Cane implements Animale {
    public void faiVerso() {
        System.out.println("Bau!");
    }
}
```

> La classe cane implementa l'interfaccia.

Cos'è un pacchetto?

Un *package* è uno spazio dei nomi che organizza classi e interfacce correlate, simile a una cartella sul computer. Serve a tenere ordinato il codice, soprattutto nei progetti grandi con molte classi.

La piattaforma Java include una vasta libreria di classi suddivisa in package, chiamata **API (Application Programming Interface)**. Questa libreria offre strumenti per operazioni comuni, come gestire stringhe (`String`), file (`File`), connessioni di rete (`Socket`) e interfacce grafiche (GUI).