ООП

> ООП, на відміну від функціонального програмування, спирається більше на об'єкти, аніж на функції

Основі поняття ООП:

> **Клас** - це засіб опису сутності, що визначає стан, поведінку та правила взаємодії з цією сутністю

C#:

```
public class Person
{
    // Constructor that takes no arguments:
    public Person()
    {
        Name = "unknown";
    }

    // Constructor that takes one argument:
    public Person(string name)
    {
        Name = name;
    }

    // Auto-implemented readonly property:
    public string Name { get; }

    // Method that overrides the base class (System.Object) implementation.
    public override string ToString()
    {
        return Name;
    }
}
```

JavaScript: (ES6, засновується на наслідуванні прототипів - Prototypal Inheritance)

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
  // гетер
  get area() {
    return this.calcArea();
  }
  // метод
  calcArea() {
    return this.height * this.width;

```

```
  }
}
```

Swift:

```
class Person {
    var name: String
    var age: Int

    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }

    func getName() -> String {
        return "Your name is \(name)"
    }
}
```

> **Об’єкт** - це окремий представник класу, що має конкретний стан та поведінку, що повністю визначається класом.

> **Інтерфейс** - це набір методів класу, що є доступним для використання іншими класами.

Основні принципи ООП:

- **Інкапсуляція**
- **Абстракція**
- **Наслідування**
- **Поліморфізм**

> **Інкапсуляція** - об’єкт містить в собі не тільки дані (поля та змінні), але і правила їх обробки (сетери, гетери, методи).
>
> Цей принцип дозволяє об’єднати дані та методи в класі та приховати деталі реалізації від користувача.

> **Абстракція** - це засіб виділити набір значних характеристик об’єкту, виключаючи незначні

> **Поліморфізм** - це можливість використовувати об’єкти з однаковим інтерфейсом без без іформації про їхній тип та внутрішню структуру.

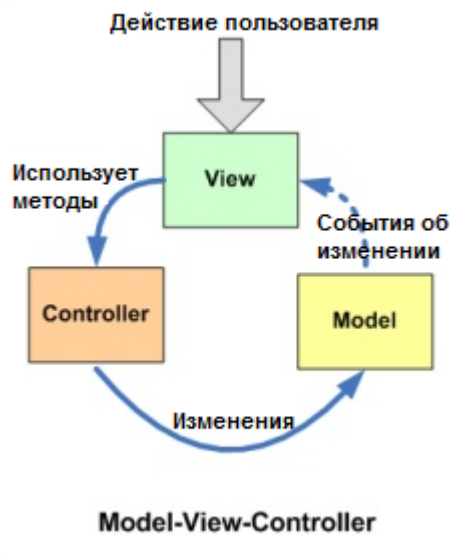
> **Наслідування** - це властивість системи, що дозволяє описати новий клас на основі вже існуючого. Клас, від якого наслідують, називається *батьківським*, а клас, який наслідує - *потомком/дитиною*.

Принципи SOLID:

Буква	Абревіатура	Назва
S	SPR	**Принцип єдиного обов'язку** - об'єкт має виконувати лише один обов'язок
O	OCP	**Принцип відкритості/закритості** - Програмні сутності повинні бути відкритими для розширення, але закритими для змін. Розширення певного класу/інтерфейсу може здійснюватись через його успадкування
L	LSP	**Принцип підстановки Лісков** - Об'єкти в програмі можуть бути заміненими їх нащадками без зміни коду програми.
I	ISP	**Принцип розділення інтерфейсу** - Багато спеціалізованихінтерфейсів краще за один універсальний. Інтерфейс може бути поділений на спеціалізовані ще на стадії проектування, заради майбутньої гнучкості програмних компонентів.
D	DIP	**Принцип інверсії залежностей** - Залежності всередині системи будуються на основіабстракцій, що не повинні залежати від деталей; навпаки, деталі мають залежати від абстракцій. Модулі вищих рівнів не мають залежати від модулів нижчих рівнів.

MVC Pattern/Concepts - Model View Controller
Всі об'єкти поділяються на три типи:

- **Models** - функціональна бізнес-логіка (операції з даними, саме тут приміняється ООП). Модельний шар не знає, як буде відображатися, нічого не знає про дизайн.
- **Views** - відображення даних, що отримуються з моделі.
- **Controllers**



Model-View-Controller

MVP Pattern/Concepts - Model View Presenter

- **Model**
- **View**

- **Presenter** - реалізує взаємодію між моделлю та видом

Recources

- ООП с примерами (часть 1) / Habr
- ООП с примерами (часть 2) / Хабр
- Object-oriented Programming in JavaScript: Made Super Simple
- S.O.L.I.D. Principles of Object-Oriented Design - A Tutorial on Object-Oriented Design