**SOLID**

SOLID – загальні принципи написання коду і взаємодії між системами. SOLID розглядається в рамках ООП в осносному, окрім моделей даних, функціонального програмування.

S (**Single Responsibility Principle**) – одинична відповідальність

Ідея принципу – при розширені функціоналу не варто добавляти функціонал в уже існуючі класи, які змішуватимуть (робитимуть не конкретним) його поведінку / першопочатковий задум. Замість цього новий функціонал задати в новому класі. Під кожен новий тип функціоналу задаємо окремий клас .

O (**Open Close Principle**)

Ідея принципу – наші класи повинні бути відкриті для розширення, але закриті для модифікації. Ми повинні проектувати наші класи таки способом, щоб навіть, коли в нас появляться нові кейси – старий код не модифікувався, а розширявся (навіть при добавлені чогось нового – ми не міняємо старий код )

L (**Liskov Subtitution Principle**)

Ідея принципу – функція, яка використовує базовий тип(клас), повинна із ним вміти взаємодіяти і взаємодіяти із усіма його підтипами (дочірніми класами), не знаючи про це. Якщо у нас є функція, що працює із наслідниками базового класу і його методами, то в разі виникнення поліморфізму і переопреділення якогось із методів базового класу наслідниками, то наша функція не зможе коректного працювати із усіма наслідниками базового класу, що порушує даний принцип. Як рішення створюємо прослойку абстракції – 2 класи наслідними від базового, із яким працювала наша функція, один стане новий базовим класом для нашої функції і вона працюватиме із ним і його дочірніми класами, тому що методи попереднього базового класу в них не переопреділятимуться, а отже функція зможе коректно працювати із даним класом і його наслідниками, а в іншому класі з абстрактної прослойки і в його наслідниках фукнціонал базової функції буде переписуватись, але жоден із класів даної гілки не матиме відношення до нашої функції (що не порушить даний принцип)

I (**Interface Segrigation Principle**)

Ідея принципу - ті класи, які наслідуються від базового класу, але не використовують його методи, не повинні від нього залежати

D (**Dependency Inversion Principle**)

Ідея принципу – верхній рівень модулів не повинен залежати він абстракцій нижнього рівня. Будь-які рівні повинні залежати від абстракцій, а не від конкретики. Створення інтерфейс-клас для реалізації конкретного функціоналу і передачі його в наступну сутність, без переписувань її вручну.