SOLID

I **principi SOLID** sono linee guida di progettazione del software orientato agli oggetti. Servono a scrivere codice più **robusto, flessibile e mantenibile**.

---

1. **Single Responsibility Principle (SRP)**

Ogni classe dovrebbe avere una sola responsabilità.

➡️ Una classe = un motivo per cambiare.

```
# NON SRP: la classe gestisce sia i dati dell'utente sia il salvataggio su file
class User:
    def __init__(self, name, email):
        self.name = name
        self.email = email

    def save_to_file(self):
        with open("users.txt", "a") as f:
            f.write(f"{self.name}, {self.email}\n")

# SRP: separo i compiti in due classi
class User:
    def __init__(self, name, email):
        self.name = name
        self.email = email

class UserRepository:
    def save_to_file(self, user: User):
        with open("users.txt", "a") as f:
            f.write(f"{user.name}, {user.email}\n")
```

---

2. **Open/Closed Principle (OCP)**

Il codice dovrebbe essere aperto all'estensione ma chiuso alla modifica.

➡️ Aggiungere funzionalità senza toccare il codice esistente.

```
# Base: classe astratta
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self): pass

# Estensioni senza modificare Shape
class Square(Shape):
    def __init__(self, side):
        self.side = side
    def area(self):
        return self.side ** 2

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * self.radius ** 2
```

---

3. **Liskov Substitution Principle (LSP)**

Le sottoclassi devono poter sostituire le superclassi senza rompere il codice.

➡️ Una `subclass` deve comportarsi come la `superclass`.

```
class Bird:
    def fly(self): return "Sto volando"

class Sparrow(Bird):  # OK
    pass

class Penguin(Bird):  # Violazione LSP: non può volare!
    def fly(self):
        raise Exception("I pinguini non volano")
```

Soluzione → cambiare gerarchia:

```
class Bird: pass
class FlyingBird(Bird):
    def fly(self): return "Sto volando"
class Penguin(Bird): pass
```

---

4. **Interface Segregation Principle (ISP)**

Un client non dovrebbe dipendere da metodi che non usa.

➡️ Meglio interfacce piccole e specifiche.

```
# Violazione ISP: troppi metodi inutili
class Machine:
    def print(self): pass
    def scan(self): pass
    def fax(self): pass

# Segregazione
class Printer:
    def print(self): pass

class Scanner:
    def scan(self): pass
```

---

5. **Dependency Inversion Principle (DIP)**

Dipendere da astrazioni, non da implementazioni.

➡️ Le classi alte non devono dipendere da classi basse, ma da interfacce.

```
# Violazione DIP: Report dipende da FileSaver
class FileSaver:
    def save(self, data): print("Saving to file")

class Report:
    def __init__(self):
        self.saver = FileSaver()
    def save(self, data):
        self.saver.save(data)
```

```
# Rispetto DIP: dipendo da un'interfaccia
class Saver:
    def save(self, data): pass

class FileSaver(Saver):
    def save(self, data): print("Saving to file")

class Report:
    def __init__(self, saver: Saver):
        self.saver = saver
    def save(self, data):
        self.saver.save(data)
```