Balintoni Dumitru-Beniamin
An 4 IS

Principii de proiectare orientată pe obiecte

S.O.L.I.D este un acronim pentru primele cinci principii de proiectare orientate pe obiecte :

- **S**—ingle Responsibility Principle, un obiect ar trebui să aibă o singură responsabilitate și nu sarcini diferite sau multe, acesta poate avea multe comportamente și metode, dar toate acestea sunt relevante pentru unica sa responsabilitatea. Deci, ori de câte ori există o schimbare care trebuie să se întâmple, va exista o singură clasă care va fi modificată, această clasă are o responsabilitate primordială.
- **O**—pen/Closed Principle, entitățile software (clase, module, funcții etc.) ar trebui să fie deschise pentru extensie, dar închise pentru modificare. Ori de câte ori trebuie să adaugi comportamente suplimentare sau metode, nu ar trebui modificate cele existente, ci scrise metode noi, pentru că, dacă ai schimba comportamentul unui obiect, de el ar putea depinde și alte părți ale sistemului, care ar trebui modificate și retestare.
- **L**—iskov Substitution Principle, o super clasă poate fi înlocuită de oricare dintre subclasele care o moștenesc în orice parte a sistemului fără nicio modificare a codului, aceasta înseamnă că subclasele ar trebui să extindă funcționalitatea super clasei, fără a o suprascrie.
- **I**—nterface Segregation Principle, interfețele ar trebui să fie mai degrabă specifice decât să facă multe lucruri diferite, Acest lucru se datorează faptului că o clasă de implementare va implementa doar interfețele specifice necesare, în loc să fie nevoită să implementeze metode de care nu are nevoie. Deci, interfețele mari ar trebui să fie descompuse în unele mai mici, mai specifice.
- **D**—ependency Inversion Principle, încearcă să minimalizeze dependența dintre obiecte utilizând abstractizarea. Dacă, de exemplu, avem o clasă App care depinde de clase foarte specializate, baze de date sau mail, în schimb, am putea avea obiect App care se ocupă cu clasa Service, care este mai abstractă, decât ceva foarte specific. Deci, acum clasa App nu depinde de clasele concrete, ci de abstractizare, iar avantajul este că putem înlocui și extinde funcționalitatea serviciului fără a schimba deloc clasa App.

General Responsibility Assignment Software Patterns (**GRASP**), este un alt set de principii de proiectare, principiile de aici iau o perspectivă puțin diferită de principiile din SOLID, deși există cu siguranță o intersectare.

GRASP tinde să-și asume un accent de responsabilitate, cum ar fi: *cine creează acest obiect, cine se ocupă de modul în care aceste obiecte comunică unul cu celălalt, cine are grijă să treacă toate mesajele primite de la o interfață utilizator?*