SOLID

In programarea orientată pe obiecte **, SOLID** reprezintă acronimul pentru cele cinci princpii de proiectare menite să facă designurile software mai inteligibile , flexibile si întreținute.

Single Responsibility Principle (Principiul Responsabilității Unice) ,reprezintă principul care spune că o clasa ar trebui să aibă o singură responsabilitate si că această responsabilitate să fie în întregime încapsulată de clasă. Când cerințele se schimbă, codul trebuie să fie supus unei reconstrucții , imiplicit si clasele vor fi modificiate, așadar , cu cât clasa are mai multe responsabilități cu atât mai greu vor fi făcute schimbările necesare noilor cerințe.

Open-Closed Principle(Principiul deschis/închis) , entitățiile software ar trebui să fie deschise pentru extindere , dar închise pentru modificare. Modulul este deschis dacă este disponibil pentru o extensie , de exemplu adăugarerea de noi elemente , câmpuri unei clase. Un modul este închis pentru modificare atunci când codul sursă al unei astfel de clase este "blocat" ,nimănui nu i se permite să modifice codul. Acest principiu se face prin abstractizare.

Liskov Substitution Principle(Principiul de substituție Liskov) spune despre clasele derivate că trebuie să fie substituibile pentru clasele de bază ale acestora. Dacă S este un subtip de T, atunci obiectele de tipul T pot fi înlocuite cu obiecte de tipul S , un obiect de tip T poate fi înlocuit cu orice obiect al unui subtip S, fără alterarea proprietătilor programului.De exemplu avem o clasă dreptunghi si avem o clasă care o extinde , pătrat. Clasa dreptunghi are ca metode setarea lungimii si a lățimii , comportamentul celor două metode vor fi diferite , deoarce pătratul prin definiție matematică spune că este un dreptunghi cu lățime si lungime egală , pe cand dreptunghiul are lățimea si lungimea diferită. Fiecare metodă trebuie să aibă precondiții si postcondiții definite.

Interface-segregation principle(Principiul de Segregare a Interfeței) prevede că niciun client nu ar trebui să fie obligat să depindă de metodele pe care nu le utilizează.Interfețele ar trebui să fie subțiri și să nu aibă metode neutilizate , adică e bine să existe mai multe interfețe mai mici decât interfețe mai puține, mai mari ,care au metode în plus.

Dependency Inversion Principle(Principiul inversării dependenței) se referă la o formă specifică de separare a modulelor, principiul prevede că modulele de nivel înalt nu ar trebui să depindă de modulele de nivel inferior , ambele ar trebui să depindă de abstractizare. Acest lucru reduce fragilitatea cauzată de modificările aduse modulelor de nivel scăzut care introduc bug-uri în straturile superioare.

Folosirea principilor **SOLID** oferă ca avantaje : , reducerea erorilor de implementarea ,reducerea complexității codului , creșterea lizibilității , mentenanței si extensibilitatea codului