Tehnici Avansate de Programare

Principiile SOLID și modularizarea codului-sursă

Petru Rebeja, Marius Apetrii

18 Martie 2021

Facultatea de Matematică
Universitatea Alexandru Ioan Cuza, Iași

Introducere

Recapitulare

- Resurse gestionate automat vs resurse gestionate manual,
- Organizarea în 3 generații,
- Gestionarea ineficientă a resurselor are impact negativ,
- Folosim `NuGet` pentru adăugarea bibliotecilor terțe.

Despre ce vom discuta azi

- Principiile `SOLID`
- Cum organizăm codul-sursă în soluție

Principiile SOLID

SOLID

S Single Responsibility Principle

O Open-Closed Principle

L Liskov Substitution Principle

I Interface Segregation Principle

D Dependency Inversion Principle

Context

Vrem să simulăm tranzacții financiare modelând diverse tipuri de conturi bancare (`Debit`, `Credit` etc.) precum și operațiunile care se fac asupra acestora.

Single Responsibility Principle

Single Responsibility Principle
Fiecare modul trebuie să fie responsabil pentru un singur aspect legat de funcționalitatea oferită de sistemul software. Mai mult, acel aspect trebuie să fie încapsulat în întregime de modulul responsabil[1].

[1] https://en.wikipedia.org/wiki/Single_responsibility_principle

Open-Closed Principle

Open-Closed Principle
Părțile componente ale unui sistem software trebuie să fie ușor de extins dar greu de modificat[2].

[2] https://en.wikipedia.org/wiki/Open/closed_principle

Liskov Substitution Principle

Liskov Substitution Principle
Obiectele unui sistem software trebuie să fie substituibile de către instanțe ale unor subtipuri de obiecte, fără ca substituția să afecteze corectitudinea sistemului[3].

[3] https://en.wikipedia.org/wiki/Liskov_substitution_principle

Interface Segregation Principle

Interface Segregation Principle
Interfețele trebuie să fie mici și specifice contextului de utilizare; nu mari și generale[4].

[4] https://en.wikipedia.org/wiki/Interface_segregation_principle

Dependency Inversion Principle

Dependency Inversion Principle
Modulele unui sistem software trebuie să depindă de reprezentări abstracte și nu de implementări concrete[5].

[5] https://en.wikipedia.org/wiki/Dependency_inversion_principle

Concepte de bază în Visual Studio

Soluție

- O mulțime de resurse necesare pentru crearea unei aplicații.
- Resursele sunt grupate (de obicei) în proiecte.

Proiect

- O submulțime de resurse menite să fie compilate împreună în:
  - O bibliotecă software (`.dll`),
  - O aplicație executabilă (`.exe`),
  - O aplicație Web.

Spațiu de nume (`namespace`)

- Denotă un domeniu restrâns căruia îi aparține o mulțime de tipuri.
- Este folosit pentru a diferenția tipurile cu același nume.

Denumirea spațiilor de nume

Șablonul recomandat pentru denumirea spațiilor de nume noi este[6]:

```
<Company>.(<Product>|<Technology>)[.<Feature>][.<Subnamespace>]
```

[6] https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/names-of-namespaces

Modularizarea codului sursă

Recapitulare: Acuplarea

Acuplarea este o măsură a gradului de interdependență dintre modulele unui produs software[7] .

[7] https://en.wikipedia.org/wiki/Coupling_(computer_programming)

Recapitulare: Coeziunea

Coeziunea este măsura în care elementele unui modul aparțin unul de celălalt[8].

[8] https://en.wikipedia.org/wiki/Cohesion_(computer_science)

De ce modularizăm codul-sursă?

Separarea funcționalităților

- Fiecare modul este responsabil de o singură funcționalitate.
- Gradul ridicat de coeziune și gradul redus de acuplare rezultă în cod ușor de întreținut.

Reutilizarea codului-sursă

- Modulele pot fi combinate pentru a obține aplicații noi (ex: pachete `NuGet`).
- Aceleași module pot fi reutilizate pentru publicarea aplicației pe diferite platforme.

Încapsulare

- Modulul expune doar interfața publică și ascunde detaliile de implementare.
- Ulterior detaliile de implementare pot fi modificate fără să afecteze funcționalitatea consumatorilor.

Ușor de testat

- Este mai ușor să scrii teste pentru module cu acuplare mică și coeziune mare.
- La rândul lor și testele devin suficient de mici și modulare.

Încheiere

Recapitulare — SOLID

- Single Responsibility Principle
- Open-Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

Recapitulare — modularizare

Modularizarea codului-sursă ne permite să:

- Separăm funcționalitățile/responsabilitățile,
- Încapsulăm și reutilizăm codul-sursă și
- Să testăm mai codul mai ușor/rapid.

Vă mulțumesc!

Mulțumesc pentru atenție!