Laboratorul 2. TMPS

Șabloanele de proiectare structurale

Să se implementeze la nivel de cod 5 șabloane de proiectare structurale din cele 7.

Atenție:

- Elaborați cîte o aplicație pentru fiecare șablon de proiectare ales în parte. Să fiți capabili să identificați participanții șabloanelor implementate pe baza claselor/interefețelor. Aplicațiile elaboratre trebuie **să posede un sens**, să nu fie doar structura șablonului de proiectare.

- Puteți combina șabloanele de proiectare între ele. În așa caz într-o aplicație veți avea 2 sau mai multe șabloane implementate.

- Limbajul de programare în care veți efectua laboratorul trebuie să fie orientat pe obiecte.

- Pentru fiecare laboratorul trebuie să faceți **raport**

- În codul sursă **să nu aveți comentarii**

- La apărarea laboratorului **nu se permite cu conspect,** doar cu o foaie/caiet curat pentru UML

- Studenții la care **laboratoarele sunt asemănatoare** vor fi nevoiți să răspundă din nou cu altă versiune a aplicației și raportului

- Pe baza baremului de notare **se formează nota finală**

- Lucrarea de laborator nu este susținută dacă studentul **nu corespunde cerințelor** pentru Nota 5 din baremul de notare (codul nu-i aparține, nu știe răspunsul corect la întrebările date de profesor, nu poate face modificări în codul elaborat)

Barem de notare:

Nota 5

1) Codul sursa să fie scris în totalmente de voi (se verifica la plagiat prin intermediul la Github, Bitbucket, Google, Yandex, Yahoo, Rapoartele studenților din anii anteriori)

2) Să puteți explica corect orice rînd scris de cod

3) Să puteți explica corect pentru fiecare șablon de proiectare diagramele UML(clasele, interfețele, relațiile dintre clase și sensul lor)

4) Să puteți face schimbări corecte în codul elaborat (adăugare de atribute, metode, schimbare de nume a unor clase, utilizarea clasei abstracte în loc de interfață și invers)

Nota 6

1) Să puteți adăuga participați noi în aplicație la nivel de cod (în contextul aplicației create și șablonului de proiectare)

2) Să puteți șterge participați din aplicație la nivel de cod (în contextul aplicației create și șablonului de proiectare)

Nota 7

1) Să puteți adăuga participați noi în aplicație la nivel de diagramă UML (în contextul aplicației create și șablonului de proiectare)

2) Să puteți șterge participați din aplicație la nivel de diagramă UML (în contextul aplicației create și șablonului de proiectare)

Nota 8

1) Să puteți identifica corect participanții șabloanelor de proiectare pe baza codului elaborat la nivel de UML

Nota 9

1) Să puteți explica relațiile șablonului pe care-l răspundeți cu alte șabloane de proiectare (pentru laboratorul 1 doar cu cele creaționale, laboratorul 2 – cu creaționale și structurale)

Nota 10

1) Pentru șabloanele de proiectare elaborate să puteți argumenta cînd utilizarea acestora va fi drept un antipattern și dacă în cadrul aplicației elaborate utilizarea a fost una corectă

Întrebări la apărarea laboratorului:

- **Adapter** - scopul, problema care o rezolvă, structura(diagrama de clase), aplicabilitatea, modul de implementare, argumente pro şi contra, relațiile cu alte șabloane de proiectare
- **Bridge** - scopul, problema care o rezolvă, structura(diagrama de clase), aplicabilitatea, modul de implementare, argumente pro şi contra, relațiile cu alte șabloane de proiectare
- **Composite** - scopul, problema care o rezolvă, structura(diagrama de clase), aplicabilitatea, modul de implementare, argumente pro şi contra, relațiile cu alte șabloane de proiectare
- **Decorator** - scopul, problema care o rezolvă, structura(diagrama de clase), aplicabilitatea, modul de implementare, argumente pro şi contra, relațiile cu alte șabloane de proiectare

- **Facade** - scopul, problema care o rezolvă, structura(diagrama de clase), aplicabilitatea, modul de implementare, argumente pro şi contra, relațiile cu alte șabloane de proiectare

- **Flyweight** - scopul, problema care o rezolvă, structura(diagrama de clase), aplicabilitatea, modul de implementare, argumente pro şi contra, relațiile cu alte șabloane de proiectare

- **Proxy** - scopul, problema care o rezolvă, structura(diagrama de clase), aplicabilitatea, modul de implementare, argumente pro şi contra, relațiile cu alte șabloane de proiectare

- Diferența dintre șabloane de proiectare și principii de proiectare

- Diferența dintre Inversion of Control și Dependency injection

- Diferența dintre șabloane arhitecturale și șabloane de proiectare

- Diferența dintre Upcasting și Downcasting

Link-uri utile:

- https://refactoring.guru/design-patterns/adapter
- https://refactoring.guru/design-patterns/bridge
- https://refactoring.guru/design-patterns/composite
- https://refactoring.guru/design-patterns/decorator
- https://refactoring.guru/design-patterns/facade
- https://refactoring.guru/design-patterns/flyweight
- https://refactoring.guru/design-patterns/proxy
- https://stackoverflow.com/questions/31317141/whats-the-difference-between-design-patterns-and-design-principles
- https://stackoverflow.com/questions/6550700/inversion-of-control-vs-dependency-injection
- https://martinfowler.com/articles/injection.html#InversionOfControl
- https://stackoverflow.com/questions/4243187/whats-the-difference-between-design-patterns-and-architectural-patterns
- https://stackoverflow.com/questions/4192887/software-architecture-design-patterns/46419722
- https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013
- https://circleci.com/blog/soa-vs-microservices/
- https://www.talend.com/resources/microservices-vs-soa/
- https://stackoverflow.com/questions/23414090/what-is-the-difference-between-up-casting-and-down-casting-with-respect-to-class
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/casting-and-type-conversions
- https://www.mygreatlearning.com/blog/type-casting-in-java/
- https://www.interviewbit.com/design-patterns-interview-questions/