Laboratorul 1. TMPS

Șabloanele de proiectare creaționale

Să se implementeze la nivel de cod 5 șabloane de proiectare creaționale din cele 6 (cu Object Pool).

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, de exemplu Singleton cu Abstract Factory, Prototype cu Abstract Factory, Builder cu Singleton, Factory Method cu Abstract Factory, Object Pool cu Singleton. În așa caz într-o aplicație veți avea 2 șabloane implementate.

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

Întrebări la apărarea laboratorului:

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

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

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

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

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

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

- Principiile SOLID

- Diferența dintre clasă abstractă și interfață

- Tipurile de polimorfism și funcţiile virtuale

- Ciclul de dezvoltare a unui produs program

Link-uri utile:

- https://refactoring.guru/design-patterns/singleton
- https://refactoring.guru/design-patterns/factory-method
- https://refactoring.guru/design-patterns/abstract-factory
- https://refactoring.guru/design-patterns/builder
- https://refactoring.guru/design-patterns/prototype
- https://sourcemaking.com/design_patterns/object_pool
- https://www.interviewbit.com/design-patterns-interview-questions/
- https://www.baeldung.com/creational-design-patterns
- https://www.javatpoint.com/creational-design-patterns
- https://stackoverflow.com/questions/478773/how-is-oop-and-design-patterns-related
- https://stackoverflow.com/questions/2898366/is-design-pattern-only-for-object-oriented-design
- https://www.baeldung.com/solid-principles
- https://stackify.com/solid-design-principles/
- https://betterprogramming.pub/10-design-principles-in-software-engineering-f88647cf5a07
- https://blog.bitsrc.io/kiss-solid-yagni-and-other-fun-acronyms-b5d207530335
- https://workat.tech/machine-coding/tutorial/software-design-principles-dry-yagni-eytrxfhz1fla
- https://stackify.com/what-is-sdlc/
- https://stackoverflow.com/questions/1874049/explanation-of-the-uml-arrows
- https://howtodoinjava.com/gang-of-four-java-design-patterns/
- http://www.cs.utsa.edu/~cs3443/uml/uml.html
- https://taylorial.com/cs1021/UMLClass.htm