**TMPS Lab 1**

**Singleton**

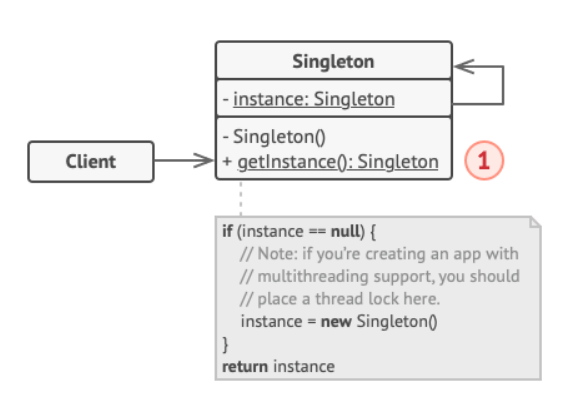
1. **Scopul**

**-** este utilizat pentru a restricționa numărul de instanțieri ale unei clase la un singur obiect, deci reprezintă o metodă de a folosi o singură instanță a unui obiect în aplicație.

1. **Problema care o rezolvă**

**-** Modelul Singleton rezolvă două probleme în același timp, încălcând Principiul responsabilității unice

1. **Structura(diagrama de clase)**



1. **Aplicabilitatea**

* Utilizăm Singleton atunci când o clasă din program ar trebui să aibă doar o singură instanță disponibilă pentru toți clienții; de exemplu Guvernul.
* Utilizăm aunti cînd avem nevoie de un control mai strict asupra variabilelor globale

1. **Modul de implementare**

* Adăugați un câmp static privat la clasă pentru stocarea instanței singleton.
* Declarați o metodă publică de creare statică pentru obținerea instanței singleton.
* Implementați „inițializarea leneșă” în cadrul metodei statice. Ar trebui să creeze un nou obiect la primul apel și să-l pună în câmpul static. Metoda ar trebui să returneze întotdeauna acea instanță la toate apelurile ulterioare.
* Faceți privat constructorul clasei. Metoda statică a clasei va putea apela în continuare constructorul, dar nu și celelalte obiecte.
* Treceți peste codul clientului și înlocuiți toate apelurile directe către constructorul singleton-ului cu apeluri către metoda sa de creare statică.

1. **Argumente pro şi contra**

|  |  |
| --- | --- |
| **PRO** | **CONTRA** |
| * Clasa are doar o singură instanță | * Încalcă *principiul responsabilității unice* . Modelul rezolvă două probleme în același timp. |
| * Obținem un punct de acces global la acea instanță | * Modelul Singleton poate masca un design prost, de exemplu, atunci când componentele programului știu prea multe unele despre altele. |
| * Obiectul singleton este inițializat numai atunci când este solicitat pentru prima dată. | * Modelul necesită un tratament special într-un mediu cu mai multe fire, astfel încât firele multiple să nu creeze un obiect singleton de mai multe ori. |
|  | * Poate fi dificil să testați codul client al Singleton, deoarece multe cadre de testare se bazează pe moștenire atunci când produc obiecte simulate. |

1. **Relațiile cu alte șabloane de proiectare**

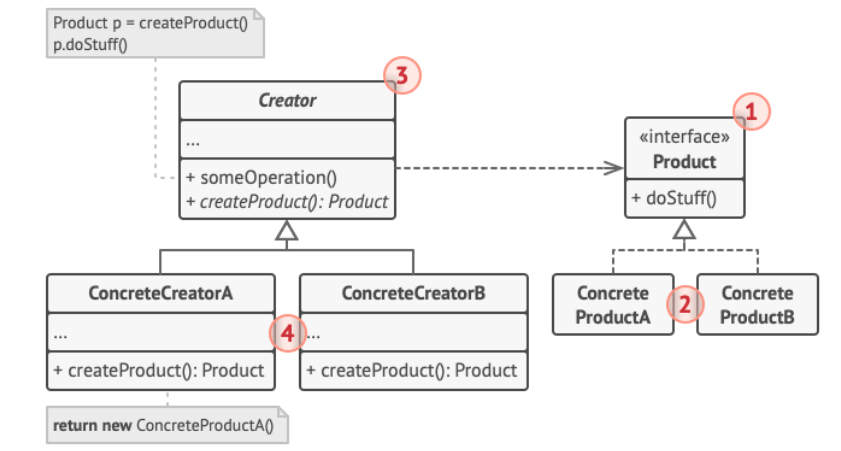
* O clasă *Facade* poate fi adesea transformată într-un [**Singleton**](https://refactoring.guru/design-patterns/singleton) , deoarece un singur obiect de fațadă este suficient în majoritatea cazurilor.
* Abstract Factories, Builders si Prototypes toate pot fi implementate ca Singleton.

**Factory Method**

1. **Scopul**

Model de design creațional care oferă o interfață pentru crearea de obiecte într-o superclasă, dar permite subclaselor să modifice tipul de obiecte care vor fi create.

1. **Problema care o rezolvă**
2. **Structura(diagrama de clase)**



1. **Aplicabilitatea**
2. **Modul de implementare**
3. **Argumente pro şi contra**
4. **Relațiile cu alte șabloane de proiectare**
5. **Scopul**
6. **Problema care o rezolvă**
7. **Structura(diagrama de clase)**
8. **Aplicabilitatea**
9. **Modul de implementare**
10. **Argumente pro şi contra**
11. **Relațiile cu alte șabloane de proiectare**