Tutoriat 6
Singleton

1. **Design patterns**
   Sunt soluții tipice pentru probleme comune în dezvoltarea de software. Acestea sunt independente de limbaj, pentru că se referă mai mult la proiectarea unor clase, decât la modul în care acestea sunt implementate. Adică, un design pattern din C++ este același și în Java, cu diferențe minore de implementare.

   Tipuri:
   - Creational Patterns
   - Structural Patterns
   - Behavioral Patterns

2. **Creational Patterns**
   Oferă diverse mecanisme de creare a obiectelor. Astfel, este mai ușoară întreținerea codului care poate fi refolosit mai târziu.

3. **Singleton**
   a. Ce este?
      - design pattern creațional
      - permite crearea unei clase care are în *permanență* o **singură instanță**

   b. La ce folosește în practică?
      - menține o singură conexiune activă la o bază de date
      - menține un singur flux deschis la un fișier comun mai multor procese
      - menținerea unei singure instanțe pentru **meniul** unui joc/unei aplicații

   c. Cum se implementează?
      1. Se creează un **constructor privat** pentru a împiedica crearea de noi instanțe prin keyword-ul *new.*
      2. Se creează un câmp de date **static** care va fi un **pointe**r către un obiect de **tipul clasei** și care va reprezenta instanța unică a acelei clase.
      3. Se creează o metoda **statică** care apelează **constructorul privat** dacă nu a fost creată deja o instanță a clasei.

```
class Singleton {
private:
   static Singleton* instance;
   Singleton() {
      cout << "Constructor called";
   }
public:
   static Singleton* getInstance() {
      if (instance == NULL) {
         instance = new Singleton;
      }
      return instance;
   }
};
Singleton* Singleton:: instance;

int main () {
   Singleton *s1;
   s1 = Singleton::getInstance(); // Constructor called
   Singleton *s2; //pointer, nu are o zona de memorie catre care arata la
linia asta
   s2 = Singleton::getInstance(); // nu intra in constructor
//    Singleton s3; // constructorul e privat => eroare
//    Singleton *s4 = new Singleton();  // constructorul e privat => eroare
}
```

d. Particularizare
O clasă care permite crearea a maxim 3 instanțe.

```
class Singleton3{
private:

   static Singleton3* _instance[3]; // declaring 3 pointers to Singleton3
   static int count; // the current number of instances created

   Singleton3(){cout<<"Constructor called"<<endl;}

public:
   static Singleton3* getInstance(){
      if (count < 3){
         _instance[count] = (Singleton3*) new Singleton3();
         count++;
         return (Singleton3*)(_instance[count - 1]);
      }
      else {
         return nullptr;
      }
```

```
    }

    ~Singleton3(){
        delete  [] *_instance;
    }
};
Singleton3* Singleton3 :: _instance[] = { nullptr,nullptr,nullptr };
int Singleton3 :: count = 0;

int main() {
    Singleton3* objArray[3];

    for (int i = 0; i < 3; i++){
        // Create an instance
        objArray[i] = Singleton3::getInstance();
    }

    // Attempt to create object no 4
    Singleton3* obj4 = Singleton3::getInstance();
    if (obj4 == nullptr) {
        cout << "Error of creating obj4." << endl;
    }

    return 0;
}
```