Exercitii Tutoriat 7

**Exercitiul 1**

Scrieți o clasă care nu poate avea la un moment dat mai mult de 3 obiecte instanțiate + explicatii.

**Rezolvare:**

Implementarile sunt variate, in functie de necesitatile programului. Cateva exemple simpliste:

Aici este varianta mai scurta (dar evident, nu foarte completa). Dupa un pas, nu vom mai avea acces la toate obiectele create - se va returna doar singleton[0]. Pentru a avea acces la orice instanta creata ne mai trebuie logica suplimentara (o functie gen getInstanceByIndex).

#include <iostream>

using namespace std;

class Singleton

{

private:

Singleton() = default;

Singleton(const Singleton&) = delete;

Singleton& operator=(const Singleton&) = delete;

inline static Singleton\* singleton[3] = { nullptr, nullptr, nullptr };

inline static int counter = 0;

public:

static Singleton\* getInstance()

{

if (counter < 3)

{

singleton[counter] = new Singleton();

return singleton[counter++];

}

else

return singleton[0];

}

};

int main()

{

Singleton\* s1 = Singleton::getInstance();

Singleton\* s2 = Singleton::getInstance();

Singleton\* s3 = Singleton::getInstance();

Singleton\* s4 = Singleton::getInstance(); // acesta va returna tot s1

cout << "s1: " << s1 << endl;

cout << "s2: " << s2 << endl;

cout << "s3: " << s3 << endl;

cout << "s4: " << s4 << endl;

delete s1;

delete s2;

delete s3;

//delete s4; - nu facem delete la s4 deoarece este s1 pe care deja l-am sters

return 0;

}

Aici folosim un index dat ca parametru in getInstance pentru a crea/manipula obiectele.

#include<iostream>

#include<vector>

using namespace std;

class Singleton

{

private:

Singleton() = default;

Singleton(const Singleton&) = delete;

Singleton& operator=(const Singleton&) = delete;

inline static Singleton\* singleton[3] = { nullptr, nullptr, nullptr };

inline static int counter = 0;

public:

static Singleton\* getInstance(int index)

{

if (index < 0 || index > 2)

return nullptr;

if (singleton[index] == nullptr)

singleton[index] = new Singleton();

return singleton[index];

}

};

int main()

{

Singleton\* s1 = Singleton::getInstance(0);

Singleton\* s2 = Singleton::getInstance(1);

Singleton\* s3 = Singleton::getInstance(2);

Singleton\* s4 = Singleton::getInstance(3); //return null

cout << "s1: " << s1 << endl;

cout << "s2: " << s2 << endl;

cout << "s3: " << s3 << endl;

cout << "s4: " << s4 << endl; // 0

cout << "s3: " << Singleton::getInstance(2) << endl;

delete s1;

delete s2;

delete s3;

delete s4;

return 0;

}