Նախագծման Ձևանմուշներ: Singleton

Հրաչյա Թանդիլյան

2020

Singleton

Նպատակը

Ապահովել դասի միակ օրինակի գոյությունը և տալ նրան դիմելու գլոբալ կետ:

Նաև հայտնի է որպես

- Այլ լայնորեն կիրառվող անուններ չկան:

Մոտիվացիան

Դասի միակ օրինակի գոյության անհրաժեշտություն:

Օրինակ՝

- Printer Spooler
- File System
- Window Manager

Կիրառելիությունը

Այս Ն.Ձ. պետք է օգտագործել երբ.

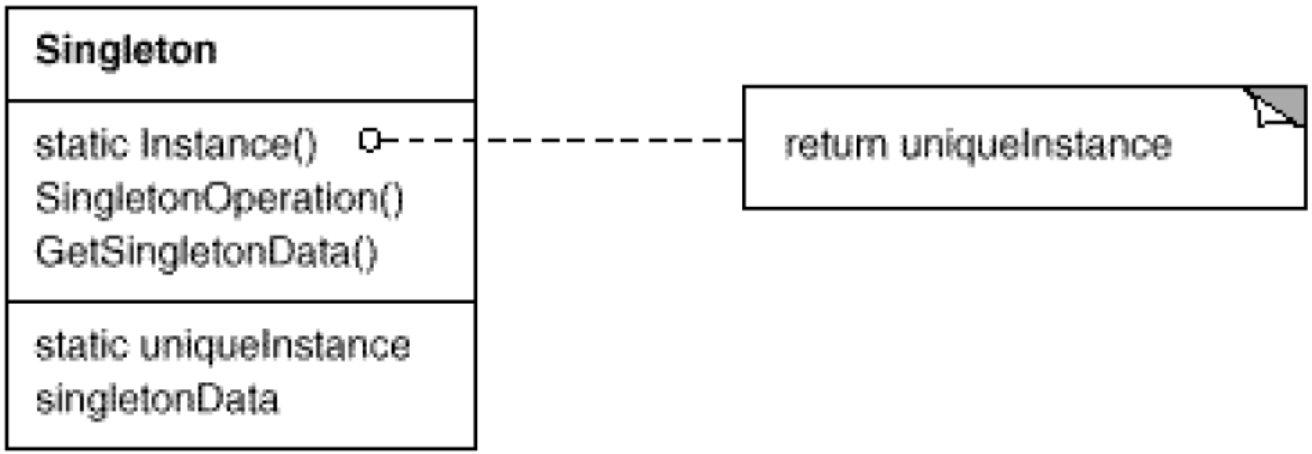
- Անհրաժեշտ է, որ դասի միայն մեկ օբյեկտ գոյություն ունենա և այն հեշտորեն հասանելի լինի օգտագործողներին:

Կիրառելիությունը

Այս Ն.Ձ. պետք է օգտագործել երբ.

- Ա Անհրաժեշտ է, որ դասի միայն մեկ օբյեկտ գոյություն ունենա և այն հեշտորեն հասանելի լինի օգտագործողներին:
- Բ Միակ օբյեկտը պետք է ընդլայնելի լինի ժառանգության միջոցով և օգտագործողները պետք է կարողանան օգտագործել ընդլայնված դասի օրինակ առանց իրենց կոդը փոփոխելու:

Կառուցվածքը



Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

- Վերահսկվող դիմում միակ օրինակին:

Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

Ա Վերահսկվող դիմում միակ օրինակին:

Բ Կրճատված անվանային տարածություն:

Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

Ա Վերահսկվող դիմում միակ օրինակին:

Բ Կրճատված անվանային տարածություն:

Գ Գործողությունների և իրականացման լավացման հնարավորություն:

Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

Ա Վերահսկվող դիմում միակ օրինակին:

Բ Կրճատված անվանային տարածություն:

Գ Գործողությունների և իրականացման լավացման հնարավորություն:

Դ Գոյություն ունեցող օրինակների մաքսիմալ քանակի՝ կամայական ֆիքսված թվով սահմանափակման հնարավորություն (Multiton pattern):

Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

Ա Վերահսկվող դիմում միակ օրինակին:

Բ Կրճատված անվանային տարածություն:

Գ Գործողությունների և իրականացման լավացման հնարավորություն:

Դ Գոյություն ունեցող օրինակների մաքսիմալ քանակի՝ կամայական ֆիքսված թվով սահմանափակման հնարավորություն (Multiton pattern):

Ե Ավելի ճկուն քան static ֆունկցիաների միջոցով իրականացումը:

Իրականացումը

- Օբյեկտի միակության ապահովում:

Դասական Իրականացում

```
class Singleton {

public:
    static Singleton* Instance();

protected:
    Singleton();

private:
    static Singleton* instance;
};

Singleton* Singleton::instance = NULL;

Singleton* Singleton::Instance() {
    if (instance == 0) {
        instance = new Singleton;
    }
    return instance;
}
```

Սխալ Իրականացում

```
class Singleton {

public:
    static Singleton* Instance();

protected:
    Singleton();

private:
    static Singleton instance;
};

Singleton Singleton::instance;

Singleton* Singleton::Instance() {
    return &instance;
}
```

Meyers' Singleton

```
static Singleton& Singleton::Instance()
{
    static Singleton instance;
    return instance;
}
```

Իրականացումը

Ա. Օբյեկտի միակության ապահովում:

Բ. Մեկից ավել փոխհամագործակցող Singleton դասերի առկայության դեպքում նրանց ոչնչացման հերթականության ղեկավարում:

Գ. Singleton դասից ժառանգում:

Լավացված Իրականացում

```
class Singleton {

public:
    static Singleton& Instance();

protected:
    Singleton();
    ~Singleton();

private:
    Singleton(const Singleton &) = delete;
    Singleton(Singleton &&) = delete;
    Singleton& operator=(const Singleton &) = delete;
    Singleton& operator=(Singleton &&) = delete;
};
```

Singleton Դասից ժառանգում

```
class Singleton {

public:
    static void Register(const char* name, Singleton*);
    static Singleton* Instance();

protected:
    static Singleton* Lookup(const char* name);

private:
    static Singleton* instance;
    static List<NameSingletonPair>* registry;
};
```

Singleton Դասից ժառանգում

```
Singleton* Singleton::Instance() {
    if (instance == 0) {
        const char* singletonName = getenv("SINGLETON");
        instance = Lookup(singletonName);
    }
    return instance;
}

MySingleton::MySingleton() {
    Singleton::Register("MySingleton", this);
}
```

Իրականացումը

Ա. Օբյեկտի միակության ապահովում:

Բ. Մեկից ավել փոխհամագործակցող Singleton դասերի առկայության դեպքում նրանց ոչնչացման հերթականության ղեկավարում:

Գ. Singleton դասից ժառանգում:

Դ. Thread Safety

Դասական Իրականացում

```
Singleton* Singleton::Instance () {
    if (instance == 0)  {          // 1
        instance = new Singleton; // 2
    }
    return instance;               // 3
}
```

Thread Safety

```
Singleton* Singleton::Instance () {
    Lock guard(mutex);
    if (instance == 0)  {          // 1
        instance = new Singleton; // 2
    }
    return instance;               // 3
}
```

Thread Safety – Սխալ Լուծում

```
Singleton* Singleton::Instance () {
    if (instance == 0)  {          // 1
        Lock guard(mutex);         // 2
        instance = new Singleton; // 3
    }
    return instance;               // 4
}
```

Thread Safety – Սասամբ Ճիշտ Լուծում

```
Singleton* Singleton::Instance () {
    if (instance == 0) {                    // 1
        Lock guard(mutex);                  // 2
        if (instance == 0) {                // 3
            instance = new Singleton;       // 4
        }
    }
    return instance;                        // 5
}
```

Thread Safety Java-ում

```
public class Singleton {

    private static Singleton uniqueInstance;

    private Singleton() {}

    /**
     * By adding the synchronized keyword to getInstance(), we force
     * every thread to wait its turn before it can enter the method.
     * That is, no two threads may enter the method at the same time.
     */
    public static synchronized Singleton getInstance() {

        if (uniqueInstance == null) {
            uniqueInstance = new Singleton();
        }
        return uniqueInstance;
    }
}
```

Thread Safety Java-ում

```
public class Singleton {

    /**
     * Creating an instance in a static initializer
     * This code is guaranteed to be thread safe!
     */
    private static Singleton uniqueInstance = new Singleton();

    private Singleton() {}

    public static Singleton getInstance() {

        // We already have an instance, so just return it
        return uniqueInstance;
    }
}
```

Thread Safety Java-ում

```
public class Singleton {

    /**
     * The volatile keyword ensures that multiple threads handle the uniqueInstance
     * variable correctly when it is being initialized to the Singleton instance.
     */
    private volatile static Singleton uniqueInstance;

    private Singleton() {}

    public static Singleton getInstance() {

        // If there is no instance, enter a synchronized block
        if (uniqueInstance == null) {

            // Note we only synchronize the first time through!
            synchronized (Singleton.class) {

                // Check again and if still null, create an instance
                if (uniqueInstance == null) {

                    uniqueInstance = new Singleton();
                }
            }
        }
        return uniqueInstance;
    }
}
```

Thread Safety Bill Pugh-ի Լուծումը

```
public class Singleton {

    private Singleton() { }

    /**
     * SingletonHolder is thread safely loaded on the first access to
     * SingletonHolder.instance which is the first call of getInstance
     */
    private static class SingletonHolder {
        public static final Singleton instance = new Singleton();
    }

    public static Singleton getInstance() {
        return SingletonHolder.instance;
    }
}
```

Օրինակ

```
class MazeFactory {

public:
    static MazeFactory* Instance();

    // existing interface goes here

protected:
    MazeFactory();

private:
    static MazeFactory* instance;
};
```

Օրինակ

```
MazeFactory* MazeFactory::instance = 0;

MazeFactory* MazeFactory::Instance() {

    if (instance == 0) {
        instance = new MazeFactory;
    }
    return instance;
}
```

Օրինակ

```
MazeFactory* MazeFactory::instance = 0;

MazeFactory* MazeFactory::Instance () {
    if (instance == 0) {
        const char* mazeStyle = getenv("MAZESTYLE");

        if (strcmp(mazeStyle, "bombed") == 0) {
            instance = new BombedMazeFactory;

        } else if (strcmp(mazeStyle, "enchanted") == 0) {
            instance = new EnchantedMazeFactory;

        } else {
            instance = new MazeFactory; // default
        }
    }
    return instance;
}
```

Առնչվող Նախագծման Ձևանմուշները

- Abstract Factory
- Facade
- Observer
- State