POO

Sabloane

Singleton

Cuprins

- sabloane de proiectare (software design patterns)
- clase cu o singura instanta (Singleton)

Sabloane de proiectare (Design Patterns)

- intai aplicate in proiectare urbanistica:

  C. Alexander. A Pattern Language. 1977
- prima contributie in software: 1987, Kent Beck (creatorul lui Extreme Programming) & Ward Cunningham (a scris primul wicki)
- contributia majora: Design Patterns:

  Gamma et al. Elements of Reusable Object-Oriented Software was published, 1994
  - cunoscuta ca **GoF** (Gang of Four)
  - in functie de scop, clasifica patternurile in
    - creationale
    - structurale
    - comportamentale
  - pot fi aplicate la nivel de clasa sau obiect

Ce este un sablon de proiectare?

- definitia originala a lui Alexander: *"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice“*
- Elementele esentiale ale unui pattern (GoF):
  - nume
  - problema (si context)
  - solutie
  - consecinte
- GoF include 23 de sabloane

Formatul (template) unui sablon

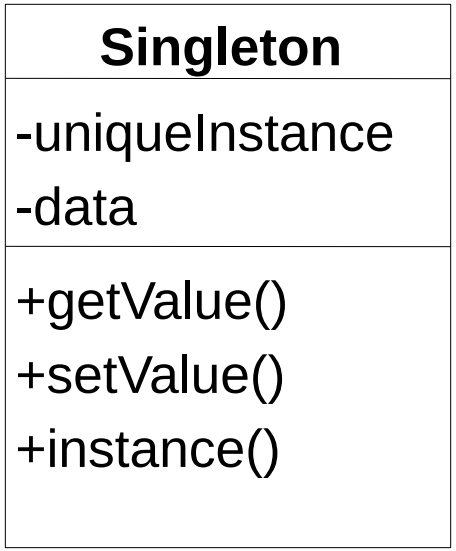
- nume si clasificare
- intentie
- cunoscut de asemenea ca
- motivatie
- aplicabilitate
- structura
- participanti
- colaborari
- consecinte
- implementare
- cod
- utilizari cunoscute
- sabloane cu care are legatura

Clase cu o singura instanta (Singleton)

- Intentia
  - proiectarea unei clase cu un singur obiect (o singura instanta)
- Motivatie
  - intr-un sistem de operare:
    - exista un sistem de fisiere
    - exista un singur manager de ferestre
- Aplicabilitate
  - cand trebuie sa existe exact o instanta
  - clientii clasei trebuie sa aiba acces la instanta din orice punct bine definit

Clase cu o singura instanta (Singleton)

- structura



- participant: Singleton
- colaborari: clientii clasei

Clase cu o singura instanta (Singleton)

- Consecinte
  - acces controlat la instanta unica
  - reducerea spatiului de nume (eliminarea variab. globale)
  - permite rafinarea operatiilor si reprezentarii
  - permite un numar variabil de instante
    - Doubleton
    - Tripleton
    - ...
  - mai flexibila decat operatiile la nivelde clasa (statice)
- Implementare

cum?

Clase cu o singura instanta

```
class Singleton
{
public:
   static Singleton& instance()
          {return uniqueInstance;}
   int getValue() { return i; }
   void setValue(int x) { i = x; }
private:
   static Singleton uniqueInstance;
   int i;
   Singleton(int x) : i(x) { }
   void operator=(Singleton&);
   Singleton(const Singleton&);
};
```

manerul cu care se are acces la instanta

constructor

operator atribuire

constructor de copiere

Clase cu o singura instanta

```
Singleton Singleton::uniqueInstance(47);


int main()
{
  Singleton& s1 = Singleton::instance();
  cout << s1.getValue() << endl;
  Singleton& s2 = Singleton::instance();
  s2.setValue(9);
  cout << s1.getValue() << endl;
  return 0;
}
```

initializare

refera aceeasi instanta (pe cea unica)

Clase cu o singura instanta

- daca se comenteaza constructorul de copiere, atunci se poate executa urmatorul cod:

```
Singleton s4 = s2;
s4.setValue(23);
cout << s4.getValue() << endl; // 23
cout << s2.getValue() << endl;  // 9
```

- daca se comenteaza operatorul de atribuire, atunci se poate executa urmatorul cod:

```
s4 = s2;
s4.setValue(43);
cout << s4.getValue() << endl;  // 43
cout << s2.getValue() << endl;  // 9
```

Demo

Instanta unica dinamica 1/2

```
class Singleton {
public:
  static Singleton* instance() {
    if (uniqueInstance == 0) {
      uniqueInstance = new Singleton();
  }
    return uniqueInstance;
  }
  int getValue() { return i; }
  void setValue(int x) { i = x; }
```

Diferenta dintre pointer si referinta 1/2

- urmatoarele instructiuni se executa, indiferent cum decalaram constructorul de copiere si/sau operatorul de atribuire

```
Singleton s4 = s2;
s4.setValue(23);
cout << s4.getValue() << endl; // 23
cout << s2.getValue() << endl;  // 23

s4 = s2;
s4.setValue(43);
cout << s4.getValue() << endl;  // 43
cout << s2.getValue() << endl;  // 43
```

- de ce?

Diferenta dintre pointer si referinta 2/2

```
s2->setValue(9);
```

- dar urmatoarele 4 instructiuni nu se compileaza daca se decomenteaza constructorul de copiere

```
Singleton s5 = (*s2);
s5.setValue(23);
cout << s5.getValue() << endl;  // 23
cout << s2->getValue() << endl; // 9
```

- si urmatoarele 4 instructiuni nu se compileaza daca se decomenteaza si operatorul de atribuire

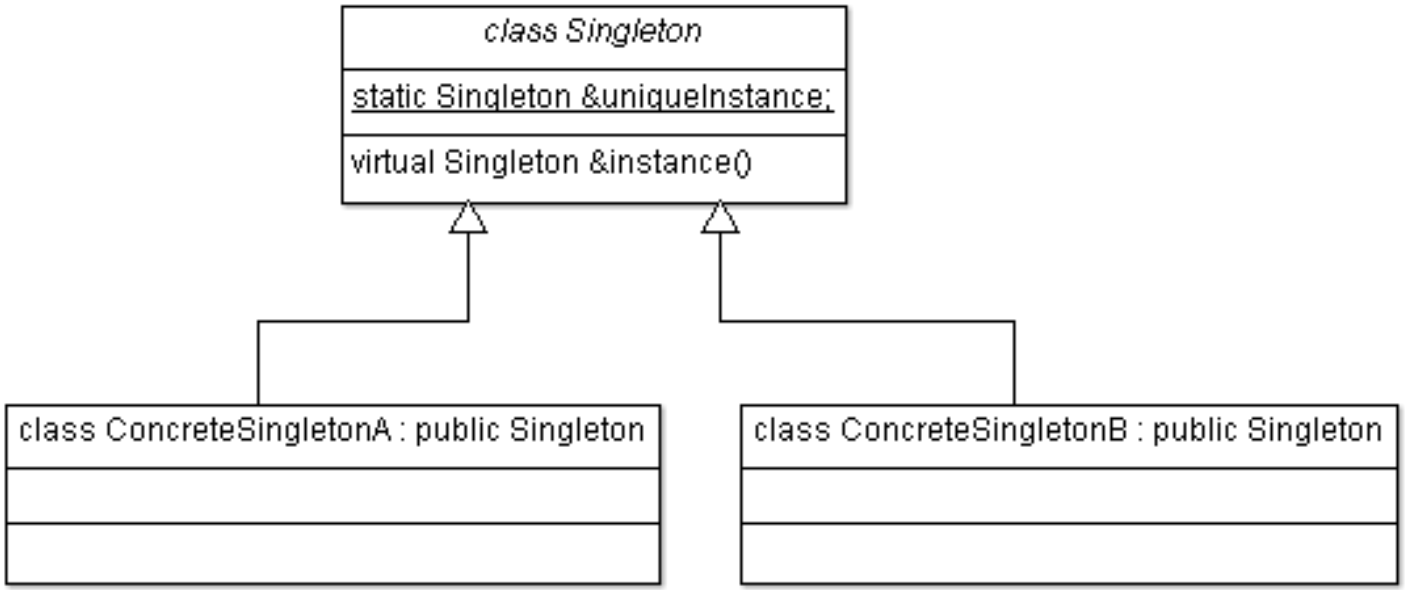
```
s5 = *s2;
s5.setValue(43);
cout << s5.getValue() << endl;   // 43
cout << s2->getValue() << endl;  // 9
```

Instanta unica dinamica 2/2

```
protected:
  int i;
  Singleton(int x = 0) : i(x) { }
 // void operator=(Singleton&);
     // nu mai e necesar (de ce?)
 // Singleton(const Singleton&);
     // nu mai e necesar (de ce?)
private:
  // pointer la instanta unica
  static Singleton* uniqueInstance;
};
```

Demo

Clase singleton derivate



- pot fi probleme la crearea instantelor a claselor concrete din ierarhie

Repozitoriu de clase singleton 1/3

```
class Singleton {
public:
  static void register(const char* name,
                       Singleton*);
  static Singleton* instance();
protected:
  static Singleton* lookup(const char* name);
private:
  static Singleton* uniqueInstance;
  static List<NameSingletonPair>* registry;
};
```

Repozitoriu de clase singleton 2/3

- metoda instance cauta in registru adresa instantei unice pentru o clasa concreta din ierarhie

```
Singleton* Singleton::instance () {
 if (uniqueInstance == 0) {
  const char* singletonName = getenv("SINGLETON");
   // furnizata la incarcarea aplicatiei
  uniqueInstance = lookup(singletonName);
   // lookup intoarce 0 daca nu gaseste
 }
 return _instance;
}
```

Repozitoriu de clase singleton 3/3

- o clasa singleton concreta din ierarhie trebuie sa inregistreze in registru adresa instantei unice

```
ConcreteSingletonA::Singleton() {
   // ...
   Singleton::register("ConcreteSingletonA",
                       this);
}
```

- in fisierul cu implementarea trebuie sa avem

```
static ConcreteSingletonA theSingletonA;
```