Singleton pattern i C++

Denne artikel beskriver Singleton pattern og implementation i C++.

Den forudsætter kendskab til C++ men ikke til Singleton.

Skrevet den **05. Feb 2009** af **arne_v** I kategorien **Programmering / C/C++** | ⭐⭐⭐☆☆

Historie:
V1.0 - 12/01/2004 - original
V1.1 - 31/01/2004 - forbedret formatering + tilføje private copy constructor

Teori

Singleton pattern løser problemet med at man kun vil have en enkelt instans af en given klasse.

Singleton pattern er en god objekt orienteret løsning på samme problem som løses ikke objekt orienteret via en klasse med kun static members og methods.

Singleton pattern er et såkaldt GoF pattern, hvilket refererer til bogen "Design Patterns" af Erich Gamma, Richard Helm, Ralph Johnson og John Vlissides (4 forfattere = Gang Of Four = GoF).

Kendetegnene ved en singleton klasse er:
- public static metode Instance
- private constructor

(den original GoF kode bruger protected constructor og det kan man også godt, men min erfaring er at man ikke kan arve fra en singleton klasse på fornuftig vis)

Eksempel

Her er et standard eksempel på en singleton klasse:

S.h

```
#include <vector>

using namespace std;

// singleton klasse
class S
{
   private:
      // normale attributter eksemplificeret ved en vector af string
      vector<string> *_list;
      // den eneste instas der eksisterer
      static S *_instance;
      // private constructor
      S();
```

```
      // private copy constructor to override default one
      S(const S& s) { };
   public:
      // public static metode til at hente instance
      static S *Instance();
      // normale metoder
      void Add(string s);
      vector<string> *List();
};
```

S.cpp

```

#include <vector>
#include <string>
#include <cstdlib>

using namespace std;

#include "S.h"

// initialiser instance til NULL
S *S::_instance = NULL;

// implementer constrctor
S::S()
{
   _list = new vector<string>();
}

// implementer getinstance
S *S::Instance()
{
   if(_instance == NULL)
   {
      _instance = new S();
   }
   return _instance;
}

// implementer normale metoder
void S::Add(string s)
{
   _list->push_back(s);
}

vector<string> *S::List()
{
   return _list;
}
```

Klassen kan bruges som følger:

```
#include <iostream>
#include <vector>
#include <cstdlib>

using namespace std;

#include "S.h"

int main()
{
   S *a = S::Instance();
   a->Add("A");
   S *b = S::Instance();
   b->Add("B");
   S *c = S::Instance();
   vector<string> *v = c->List();
   for(int i = 0; i < v->size(); i++)
   {
      cout << v->at(i).c_str() << endl;
   }
   return EXIT_SUCCESS;
}
```

Singleton i multithreaded kontekst
Bemærk at i en multithreaded kontekst bør man kode sin singleton klasse så man undgår samtidigheds problemer. Der er ingen standard for hvordan man gør det i C++ (Windows og POSIX har hver sin måde), så jeg vil undlade at vise eksempler på dette.

Kommentar af driis d. 23. Jan 2004 | 1

Udmærket artikel, men jeg vil gerne påpege at copy constructoren også bør erklæres private - ellers kan man stadig oprette 2 objekter af klassen, f.eks.:

```
int main()
{
   S * a = S::Instance();
   S b = *a;

   cout << "Object a addresse: " << a << endl
      <<  "Object b addresse: " << &b << endl;

   return 0;
}
```

Kommentar af mathiash d. 21. Apr 2006 | 2

Kommentar af baxos d. 14. Jan 2004 | 3