Singleton là một phần **Gang of Four design pattern** và nó được phân loại theo các **creational** design patterns. Trong bài viết này, chúng ta sẽ đi sâu hơn vào việc sử dụng Singleton pattern. Đây là một trong những Design Pattern đơn giản nhất về mặt mô hình nhưng mặt khác, đây là một trong những **pattern** gây tranh cãi nhiều nhất về độ phức tạp khi sử dụng.

Singleton pattern là một Design Pattern hạn chế khởi tạo lớp với nhiều đối tượng của nó. Nó không là gì khác ngoài một cách khai báo lớp. Lớp được định nghĩa theo cách mà chỉ một thể hiện của lớp được tạo ra trong quá trình thực thi hoàn chỉnh một chương trình hoặc dự án. Nó được sử dụng khi chỉ cần một thể hiện duy nhất của một lớp để kiểm soát hành động trong suốt quá trình thực thi. Một lớp singleton không nên có nhiều trường hợp trong mọi trường hợp và bằng mọi giá. Các lớp Singleton được sử dụng để ghi nhật ký, đối tượng trình điều khiển, bộ nhớ đệm và nhóm luồng, kết nối cơ sở dữ liệu.

Code ví dụ

#include <iostream>

#include <string>

#include <stdlib.h>

using namespace std;

class Number

{

public:

// 2. Define a public static accessor func

static Number \*instance();

static void setType(string t)

{

type = t;

delete inst;

inst = 0;

}

virtual void setValue(int in)

{

value = in;

}

virtual int getValue()

{

return value;

}

protected:

int value;

// 4. Define all ctors to be protected

Number()

{

cout << ":ctor: ";

}

// 1. Define a private static attribute

private:

static string type;

static Number \*inst;

};

string Number::type = "decimal";

Number \*Number::inst = 0;

class Octal: public Number

{

// 6. Inheritance can be supported

public:

friend class Number;

void setValue(int in)

{

char buf[10];

sprintf(buf, "%o", in);

sscanf(buf, "%d", &value);

}

protected:

Octal(){}

};

Number \*Number::instance()

{

if (!inst)

// 3. Do "lazy initialization" in the accessor function

if (type == "octal")

inst = new Octal();

else

inst = new Number();

return inst;

}

int main()

{

// Number myInstance; - error: cannot access protected constructor

// 5. Clients may only use the accessor function to manipulate the Singleton

Number::instance()->setValue(42);

cout << "value is " << Number::instance()->getValue() << endl;

Number::setType("octal");

Number::instance()->setValue(64);

cout << "value is " << Number::instance()->getValue() << endl;

}