Singleton pattern

- هو عبارة عن طريقة معينة يتم تطبيقها على ال class كي لا يستطيع أحد الحصول على أكثر من object واحد منه.

- طريقة القيام بهذا من خلال

1- جعل ال constructor يكون private or protected
2- القيام بعمل method تتحقق هل يوجد object تم إنشاءه من هذا ال class قبل ذلك ام لا
3- إذا كانت الإجابة بلا بعمل call constructor ثم بعمل return ل address of new object
4- إذا كانت الإجابة بنعم أي يوجد object تم إنشاءه قبل ذلك بعمل return address of this object

- مثال ل class معمول بال singleton using pointers وطريقة التعامل معه:

```
//Singleton.h

#include <iostream>
#include <string>

using namespace std;

class Singleton {

private:

	string name;

	static Singleton* s1;
	// why static? عشان يحافظ على قيمته القديمة
	Singleton(string n);

	/*override the default copy constructor and assignment operator
	so that you can not make a copy of the singleton*/

	Singleton(Singleton const* s); //don't implement
	Singleton operator=(Singleton const* s); //don't implement

public:

	static Singleton* getInstance(string n);
	// why static? عشان اقدر اناديها من غير معرف اوبجكت لانى مش هعرف اصلا
	void print();
```

```cpp
//Singleton.cpp

#include “Singleton.h”

Singleton* Singleton::s1 = NULL;

Singleton::Singleton(string n)
{
	name = n;
}

Singleton* Singleton::getInstance(string n)
{
	if (!s1)
		s1 = new Singleton(n);

	return s1;
}

void Singleton::print()
{
	cout << s1->name << endl;
}
```

```cpp
// Usage
#include “Singleton.h”

void main()
{

	Singleton* s1;
	s1 = Singleton::getInstance("Akram");
	s1->print();

	s1 = Singleton::getInstance("Mahmoud");
	s1->print();

	Singleton* s2;
	s2 = Singleton::getInstance("Ahmed");
	s2->print();
}
```

إيه هو ال static member and static method

static member as `static Singleton* s1;`

هو انى بعرف member ال member ده بيحصله creation مرة واحدة بس لما بعمل creation لأول object من ال class ده في البرنامج بتاعى وبتكون قيمته ال default ب zero إلا لو انت عملتله initialization زي (`Singleton* Singleton::s1 = NULL;`) بغض النظر عن عدد ال objects اللى عملتها بعد كده من نفس ال class مش هيحصل ال creation لل member ده تانى وهيكون shared بينهم ليه بعمله initialization بالطريقة ديه مش بعمله داخل ال file.h لانه غير مسموحلى بده لو كان ال static member بس :D .

Static method as `static Singleton* getInstance(string n);`

By declaring a function member as static, you make it independent of any particular object of the class. A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator.::

A static member function can only access static data member, other static member functions and any other functions from outside the class.

Static member functions have a class scope and they do not have access to the this pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not**.**

(وده اللى احنا بنعمله في ال singleton)

- النتائج المترتبة على تصميم ال class بهذه الطريقة

اصبح ال object اللي هنشأه من ال class --> Global object .....ليه ؟؟
لانى من اى مكان (class or main) اقدر اعمل pointer واعمل `getInstance()`
فيرجعلى نفس ال object (وده عيب وميزه حسب استخدامك واحتياجاتك)

- سلبيات singleton pattern

  - ال class بتاعك بيكون dependent على ال singleton pattern فالتغيير فيه بيكون اعقد.

  - عملية ال Testing بتكون أرخم لأنك مضطر تتعامل مع ال constrains بتاعت ال class اللى هي انى مقدرش اعمل اكتر من one instance فقط

  - عمليه ال inheritance اعقد فلو خليت ال constructor <-- private فمش هتعرف تعمل inheritance أصلا،
    ولو خليت ال constructor <-- protected هعرف اعمل inheritance بس أنا كده بسمح لل derived classes أنهم لا ينطبق عليهم ال singleton pattern.

من الاخر كده متستخدمش ال pattern ده إلا لو كنت عايز تعمل class تاخد منه One and only One General Object

- مثال لنفس class معمول بال singleton using reference وطريقة التعامل معه:

```
//Header file
#include <iostream>
#include <string>

using namespace std;

class Singleton
{
public:
	void print();
	static Singleton& getInstance(string n);
private:
	string name;
	Singleton(string n);                    // Constructor? (the {} brackets)
are needed here.
	// Dont forget to declare these two. You want to make sure they
	// are unaccessable otherwise you may accidently get copies of
	// your singleton appearing.
	Singleton(Singleton const&);              // Don't Implement
	void operator=(Singleton const&); // Don't implement
};
```

```cpp
//cpp file
Singleton::Singleton(string n)
{
	name = n;
}
Singleton& Singleton::getInstance(string n)
{
	static Singleton instance(n); // Guaranteed to be destroyed.
	// Instantiated on first use.
	return instance;
}

void Singleton::print()
{
	cout << name << endl;
}
```

```cpp
void main()
{

	Singleton& s1 = Singleton::getInstance("Akram");
	s1.print();

	Singleton& s2 = Singleton::getInstance("Ahmed");
	s2.print();
}
```