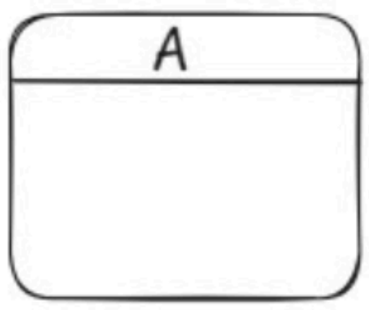
Singleton Design Pattern

Singleton is a special class which can create only one object/instance.
Even when you try to create other instance it gives that one already created object only.

* Understanding Object Creation Logic



In System we have 2 type of memories:
1.Heap(Non-Primitive DataTypes like objects get stored)
2.Stack(Primitive DataTypes like int,char,bool gets stored)

```
A* a = new A();
```

Whenever we create an object with "new A()" a constructor of this class gets called.

Constructor is always called 1st, once object is created.

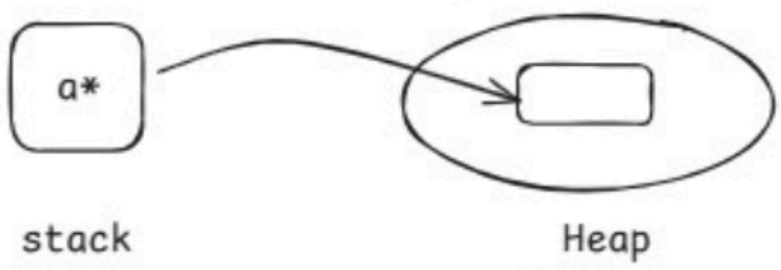
If we don't write constructor then C++/java language will give default constructor with same class name.

```
A* a = new A();
```

1.In heap memory due to "new" keyword memory gets reserved based on class size.

2.Class A() constructor gets called, if user not defined default constructor gets called.

3.*a (stored in stack) assigned to heap memory object



* How to stop a class with more than 1 object creation?

->We make public constructor to private such that it cannot accessed from outer of class.
  By this we can't create a single object.

->To create a single object from class we create in public with Singleton* getInstance().
  To call getInstance() we need a object, we make our getInstance() as static.
  static methods belong to class not objects. To call static methods by "classname and methodname" not objectname
                                                                        "Singleton::getInstance()"
->To control getInstance() to create one object,we will create a private variable that holds singleton class's instance, to access we do static.
 Variable checks whether object created or not, if not getinstance method creates object and returns, otherwise it returns already created object.

* How to Make Class ThreadSafe?

-> In threading we lock to not enter critical section(where multiplethreads runs parallely)

-> We create a static variable type of mutex (which helps to lock or unlock in c++).

-> In getInstance()[where object is created] we lock that "lock_guard<mutex>lock(mtx).
   With this, now only one thread can enters into this method.

->Locking or Unlocking Mechanism in Threading environment is very expensive in any language.
  So, we try not to use as much as possible as consumes many resources and need to hold many threads.

->Double Locking to check when multi threads enetered 1st if condition same time. In mutex one gets locked other one proceeds.Once that 1thread creates object & returns, 2nd thread gets unlocked will also proceed(here no check again whether object created/not). In this process 2 objects gets created. To solve this issue, we double check.

```
static Singleton* getInstance() {
    if(instance == nullptr){//1stCheck(no locking)
       lock_guard<mutex>lock(mtx);//Lock only needed
            if (instance == nullptr) {// Second check (after acquiring lock)
                instance = new Singleton();
            }

    }

    return instance;
}
```

* Eager Initialization

```
class Singleton {
private:
    static Singleton* instance;

    Singleton() {
        cout << "Singleton Constructor Called!" << endl;
    }

public:
    static Singleton* getInstance() {
        return instance;
    }
};

// Initialize static members
Singleton* Singleton::instance = new Singleton();
```

- It is prefered to use only when object consumes less memory otherwise this is very expensive.

While initialization in static memory at that time only we created a Singleton object in heap and assigned to instance variable.

Singleton Design Overview

1. Create a private Constructor.
2. Create a static instance(getInstance that returns same instance every time)

Practical UseCase

1. Logging System
2. Database Connection
3. Configuration Manager

- Loggers we want to take only once, as taking multiple consumes lots of memory
- Whenever we connect database with an application i.e a expensive If we won't use singleton then multiple objects gets created when a new user comes. We need one instance i.e sharable to entire application.
- API key, we stored in a cofiguration file ,the services there all in appliaction, should be hold in a single configuration manager object.

Where Singleton should not be used?

->In application for a class there is a requiement of multiple instances (already we know), we shouldn't use Singleton.
Eg: MultiPlayer Game

->In singleton, ThreadSafety is very important. Multithreading code gets complicated.