May
28-05-2025
Friday

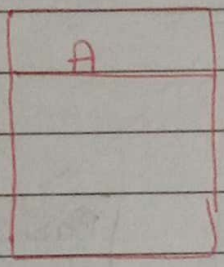
Day - 12
Lecture - 10

Singleton Design Pattern | Thread-Safe

* Singleton Design Pattern:

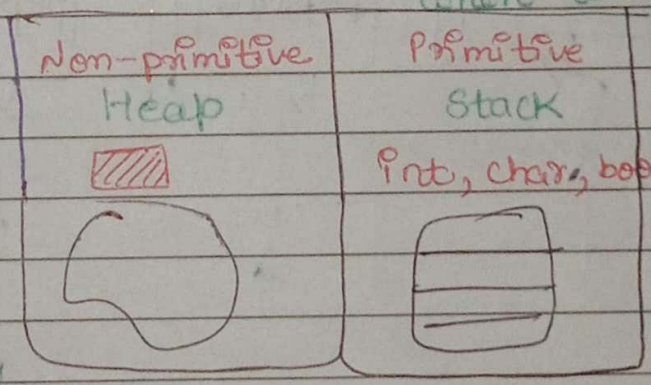
Singleton - We can't create multiple object. A class where we can only create one objects. If we make another object then it return the previous object.

* Understanding object creation logic



A * a = new A();

what happen when we write new



→ We have two type of memory

→ Primitive - where we store int, char bool in stack

→ Non-primitive are store in heap

A* a = new A();

In the same way here we use parenthesis to call constructor

→ Constructor is call when we create object

m1();

When we call method we use parenthesis

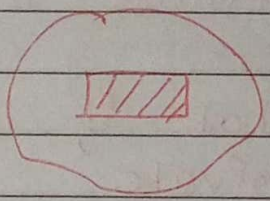
```
Class A {
    int x;
    char y;
}
```

Constructor is used to intiate the object means may provide default ~~per~~ value.

→ If we don't write constructor then language like C++, Java call their defaut constructor

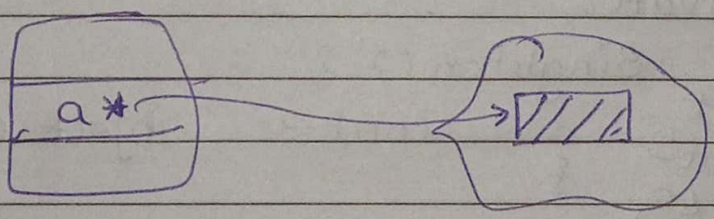
```
A * a = new A();
```

Step - 1 Memory is reserved in heap due to new Keyword



Step - 2 Constructor will call because we use A() parenthesis.

Step - 3



We will take a* in stack and point it to heap memory

* Creating Singleton Class

```
class Singleton {
public:
    Singleton() {
        cout << " object created" << endl;
    }
};
```

```
int main() {
    Singleton * s1 = new Singleton();
    Singleton * s2 = new Singleton();

    cout << (s1 == s2) << endl;        // output 0 (object are not same)
}
```

In the above code we can create multiple objects of the same class because the constructor is public.

What if we make the constructor private

```
Class Singleton {
    private:
        Singleton() {
            cout << "object created" << endl;
        }
    public:
        static Singleton * getInstance() {
            return new Singleton();
        }
};
```

We declare it as private but we can't create object easily so we use getter and setter which we studied in encapsulation

static → belong to class not object

```
int main() {
    Singleton * s1 = Singleton::getInstance();
    Singleton * s2 = Singleton::getInstance();
    cout << (s1 == s2) << endl;
}
```

we can't create the object directly so we use the static keyword and scope resolution operator

But still we can create multiple objects.

So we can create one variable which hold the pointer of singleton, which created in heap memory.

```
Class Singleton {
  private:

      static Singleton * instance;

    Singleton () {
            cout << " Object created" <<endl;
    }

  public:
        static Singleton * getInstance () {
            if (instance == nullptr) {
                instance = new Singleton();
            }
        return instance;
        }
};
Singleton :: instance = nullptr;
 int main () {
        Singleton * S1 = Singleton :: getInstance();
        Singleton * S2 = Singleton :: getInstance();

        cout << (S1 == S2) <<endl;

}
```

logic to create singleton object

* Making a Class Thread safe

→ If application supports multiple threading then in ~~mult~~ multiple threading every thread will create a new object.

→ So to stop this we will use locking system in Cpp.

* Sir give code example so you can watch it

* Introducing double checking

→ If t1 and t2 both enters in the in the code.

→ ~~first check if t1~~

both t1 and t2 enter at the same time both get true

→ `if (instance == nullptr) {`
`lock_guard<mutex> lock(mtx);`
`instance = new Singleton();`
`}`
`return instance;`

then t1 acquire lock and go to this line and return instance and then it do lock to unlock

then t2 enter here and it doesn't check the instance == nullptr and then t2 create new object.

Again two object are created.

So we use double Locking system to check it.

```
if (instance == nullptr) {
    lock_guard <mutex> lock(mtx);
    if (instance == nullptr) {
        instance = new Singleton();
    }
}
return instance;
```

* Eager Initialization:

```
Class Singleton {
private:
    static Singleton * instance;
    Singleton () {
        cout << "Object created"
    }
public:
    Static Singleton * getInstance () {
        return instance;
    }
};
// Initialize Static members
Singleton* Singleton :: instance = new Singleton();

int main(){
    ==
    ==
}
```

→ we initialize it early then it will store in heap memory

Disadvantage

→ May this object never used in application but we create it early and initialize the object and waste the resources if we never used

* Singleton Design : • Create a private constructor
  • Create a static instance (getInstance()) that returns the same instance every time.

* Practical use-Case
  • Logging System
  • Database Connection
  • Configuration Manager

* Where not to use singleton.
  • Game - where we have to create multiple objects.