Agenda

- Intro to Design Patterns.
- Type of Design Patterns.
- Singleton

=> What are design Patterns ?

Software

something that repeats frequently.

=> Solutions to well established problems in Software designs

GOF (Gang of four). => 23 Design Patterns.

Type of Design Patterns.

Object oriented Design.

1) Creational.

↳ How we can create an object?

→ How many objects should be created?

- Singleton.
- Builder.
- factory
- Prototype

2) Structural.

↳ How to structure the class.

→ How different classes will interact with each other.

3) Behavioural

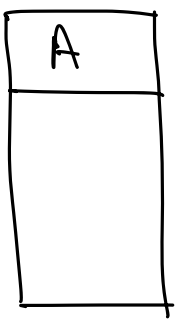
↳ Related to behaviours.

↙ Action / method.

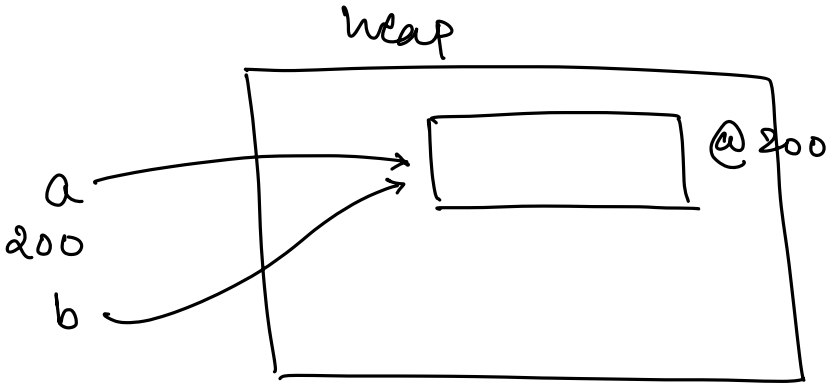
(Strategy).

Singleton.

→ A class that allows us to create a single instance only.



A a = new A()



If (A) is singleton then we should only have one object in the Heap memory.

⇒ Singleton is used when a class has shared resources.

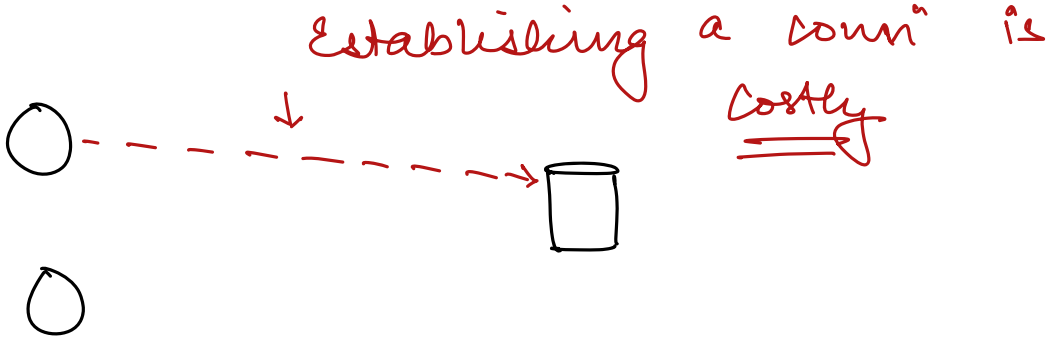
```
DatabaseConnection {
        url
        Password
        username
        port
        ==
}
```

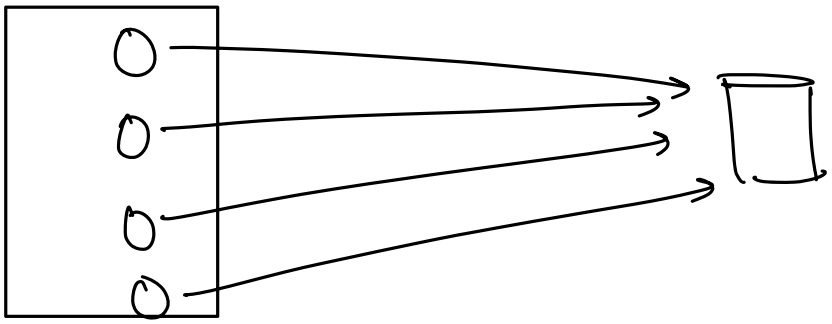
```
# UserService {
        DBC dbc = ——— ;

        save() {
            dbc.execute(≡);
        }
}
```

```
# SearchService {
        DBC dbc = ——— ;

        dbc.execute(——);
}
```

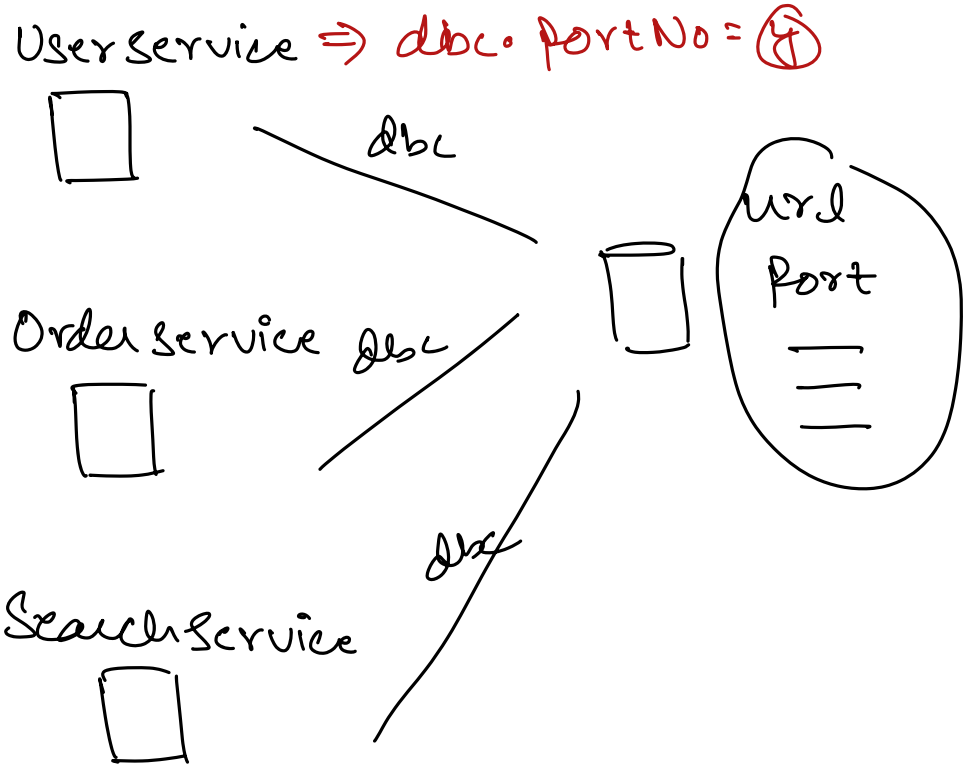




Note: Creating multiple Database connections will be a waste of lot of resources.

⇒ Singleton is generally used when the object creation is costly.

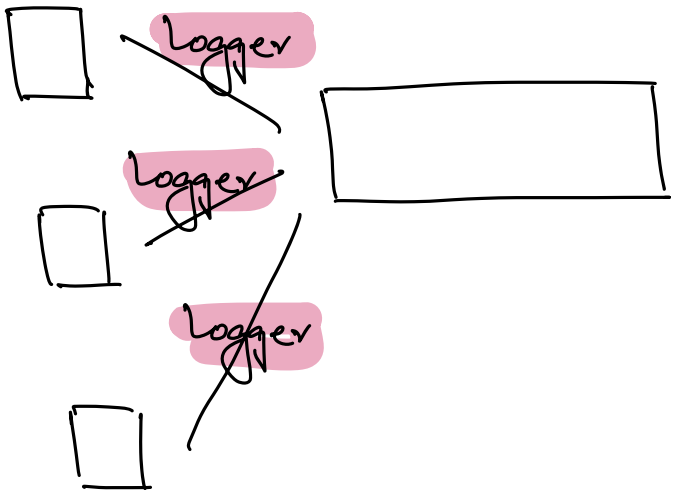
⇒ DBC dbc = new DBC();



⇒ Singleton object should be immutable.

Logger ⇒ To log/print on the command line.

log4j.



⇒ When we can use Singleton?

1) When object creation is heavy.
2) Shared resources.
3) When the object is immutable.

How to implement Singleton?

```
DatabaseConnection {
        url
        Password
        username
        port
}
```

DBC dbc1 = new DBC();
DBC dbc2 = new DBC();

⇒ To create an object of a class we need to call its constructor. Till the time constructor is publicly available anyone can create any no. of objects of a class.

```
DBC {
        private DBC(){
        }
}
```

DBC dbc1 = new DBC(); ✗

→ If the constructor is private, we can't even create a single object.

→ Create a public method getInstance().

```
DBC {
    private DBC() {
    }
    public static DBC getInstance() {
        DBC dbc = new DBC();
        return dbc;
    }
}
```

⇒ DBC dbc1 = DBC.getInstance();
DBC dbc2 = DBC.getInstance();

Q. Is this Singleton? NO.

```
DBC {
    private Static DBC instance = null;

    ≡

    Private DBC() {
        =

    }

    Public Static DBC getInstance() {
        if (instance == null) {
            instance = new DBC();
        }
        return instance;
    }
}
```

```
DBC dbc1 = DBC.getInstance();
```

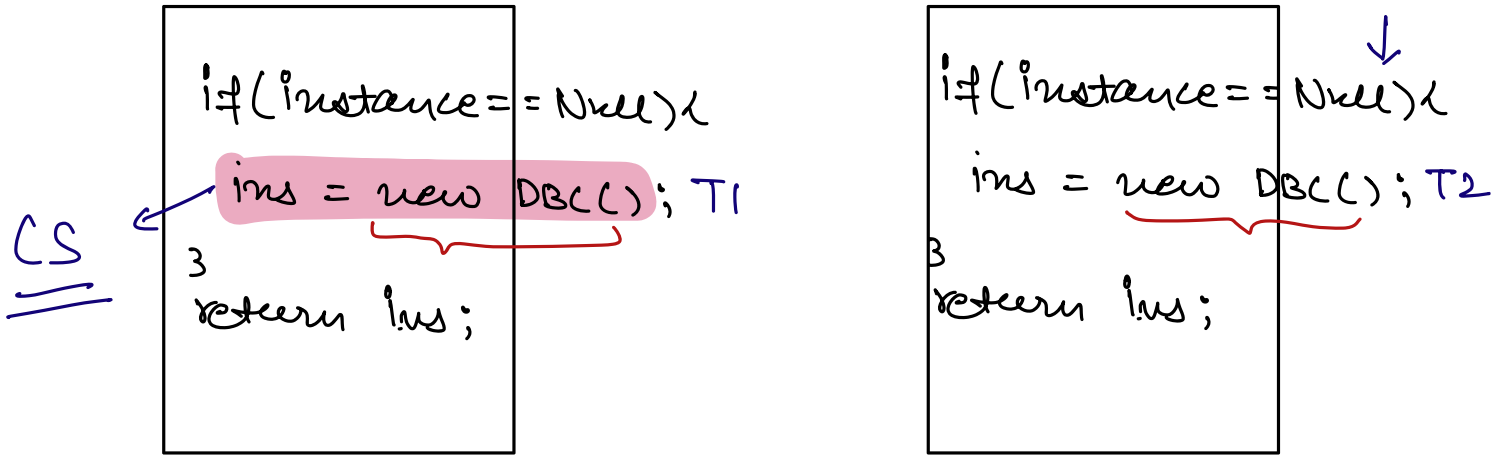
Steps.

1. Constructor private.
2. public static getInstance()
3. Static instance.
4. Before creating an object, first check if object already exists or not.

#

T1

T2



T1: DBC dbc1 = DBC.getInstance();

T2: DBC dbc2 = DBC.getInstance();

⇒ If multiple threads tries to create object at same time then it can lead to creation of more than one Object.

Singleton in Multithreaded environment.

1) Early initialization.

```
DBC {
  private static dbc = new DBC ();
  ≡
  ≡
  ≡
  private DBC ();

T1 → public static getInstance() {
T2 →      return dbc;
  }
}
```

Cons.

1. It increases the app startup time.
2. We can't any Param inside the constructor.

```
DBC (String Param) {
    if (Param is PROD) {
        ≡
    }
    else {
        ≡
    }
}
```

Requirements

→ Singleton should be thread safe.

→ Create an object at run time.

Synchronised (lock).

```
DBC {
    private static DBC instance = null;

    ≡

    private DBC(){
        =

    }                    Synchronised
    Public static ^DBC getInstance(){
        if(instance == null){
            instance = new DBC();
        }
        return instance;
    }

}
```

⇒ Performance will be slow.

#

```
getInstance(){
    if (instance == null) {
lock/Sync(){
        if( instance == null){
            instance = new DB();
        }
    }
    return instance;
}
```

T2 T1 .... T10
T11 T12 ... T20

⇒ Double Check locking.

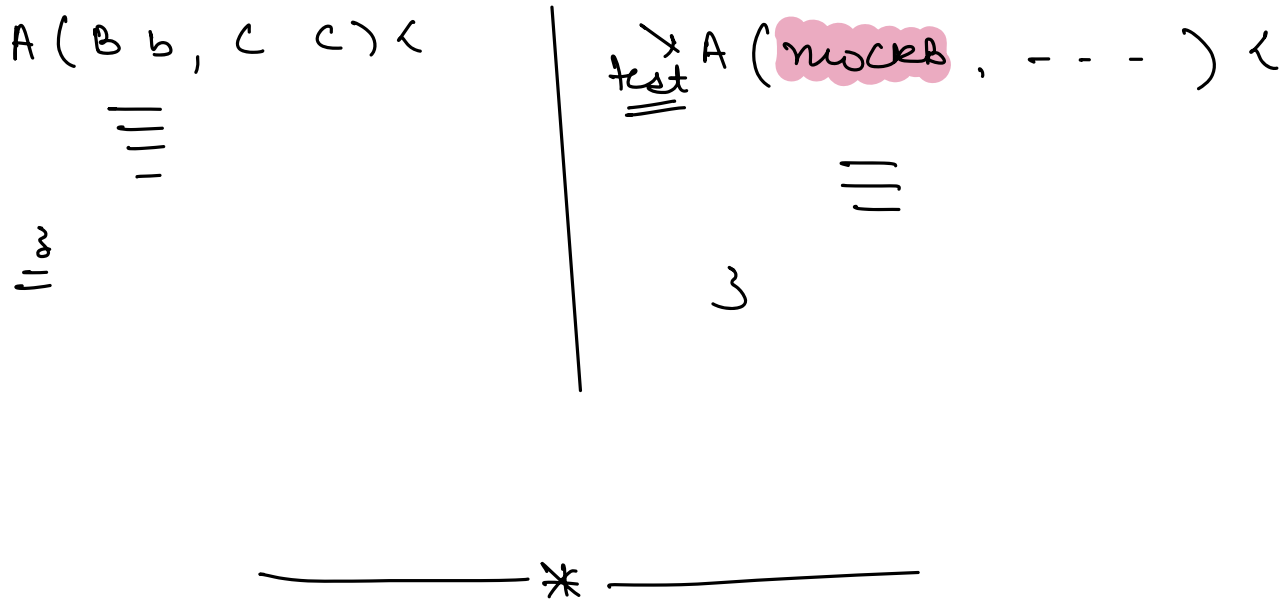
Best & Practical way to implement Singleton.

⇒ Advantages

⇒ Efficient use of resources

Disadvantages

Difficult to test.


A (B b , C c) {
}
test
A (mockB , - - -) {
}