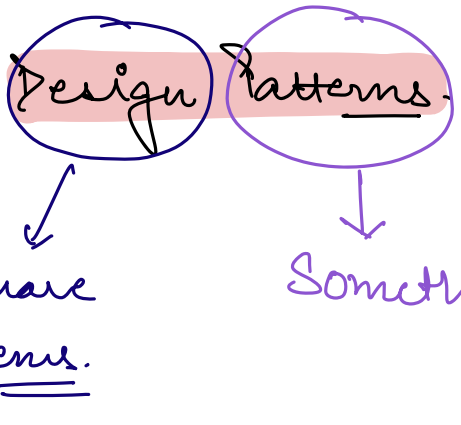
Agenda -

→ What are Design Patterns.

→ Type of Design Patterns.

→ Singleton

What are Design Patterns.



⇒ Well established solutions to commonly occuring usecases in Software System design.

⇒ Type of Design Patterns.

① Creational.

⇒ How to create an object ?

⇒ How many objects to be created ?

② Structural.

⇒ How a class should be structured?

⇒ What all the attrs a class should have?

⇒ How a class can interact with other class.

(3) Behavioural.

⇒ Methods.

⇒ How to code a method?

CREATIONAL. DESIGN PATTERNS.

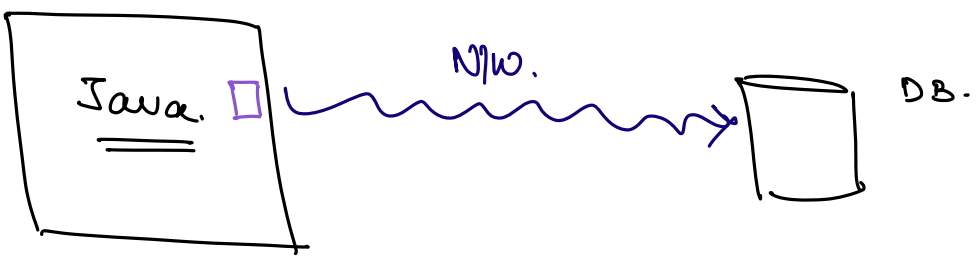
- Singleton
- Builder
- factory
- Prototype.

SINGLETON.

⇒ Allows us to create only a single object for a particular class

Why Singleton?

→ When object creation is expensive.



⇒ Database Connection

↳ Creating DB connection is an expensive operation, so we can use Singleton Pattern.

⇒ Logging : One logger object would be sufficient to write logs across the application.

⇒ Note: Singleton pattern is preffered in case Object creation is expensive.

⇒ Shared resource.

⇒ Class DatabaseConnection {

```
String url;
int portNo;
String userName;
String password;
```

}

⇒ DBC dbc1 = new DBC()

DBC dbc2 = new DBC()

⇒ Till the time constructor is available then we can create any # of objects of a class.

⇒ Make the constructor private.

Class DatabaseConnection {

```
String url;
int portNo;
String userName;
String password;
private DatabaseConnection() {
```

}

}

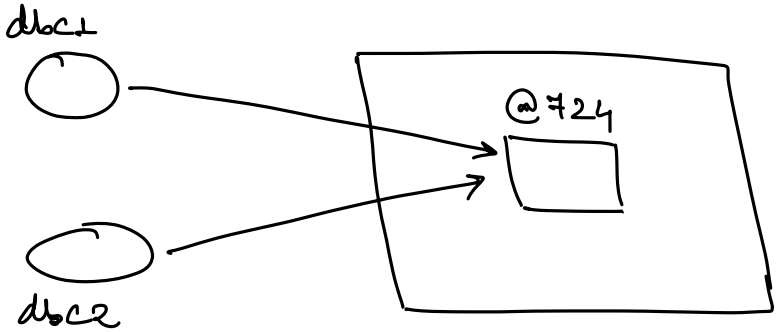
```
DBC dbc1 = new DBC()
```

⇒ We can't even create a single object of the class.

```
Class DatabaseConnection {
    String url;
    int portNo;
    String userName;
    String password;
    Private DatabaseConnection() {
    }
    Public static DBC getInstance() {
        return new DBC();
    }
}
```

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

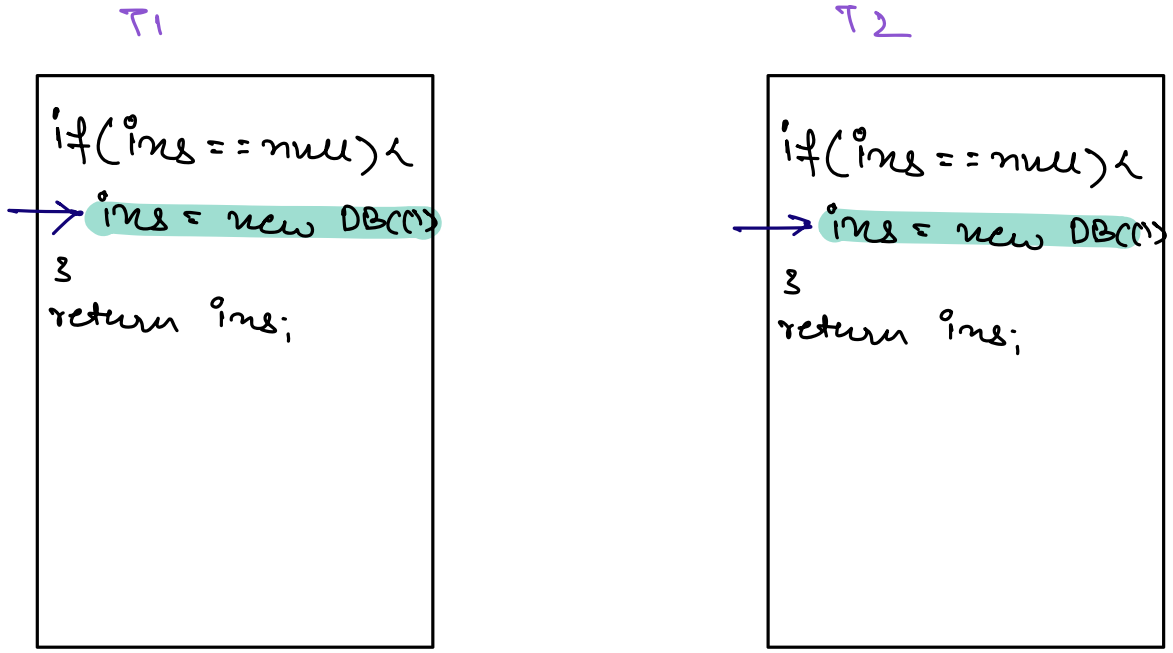
```
Class DatabaseConnection {
    Private Static DBC dbc = null;

    String url;
    int portNo;
    String userName;
    String password;
    Private DatabaseConnection() {
    }
    Public Static DBC getInstance() {
        if (dbc == null) {
            dbc = new DBC();
        }
        return dbc;
    }
}

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



Steps.

1. Make the constructor private.
2. Create a public static getInstance method.
3. Create a static instance of class.

⇒ MULTI-THREADING.



⇒ Above code won't work in case Multi threading environment.

(1) EAGER / EARLY INITIALIZATION.

```
Class DatabaseConnection {
    Private Static DBC dbc = new DBC();

    String url;
    int portNo;
    String userName;
    String password;
    Private DatabaseConnection() {
    }
    Public Static DBC getInstance() {
        return dbc;
    }
}
```

⇒ Lot of static attrs will increase the App load time.

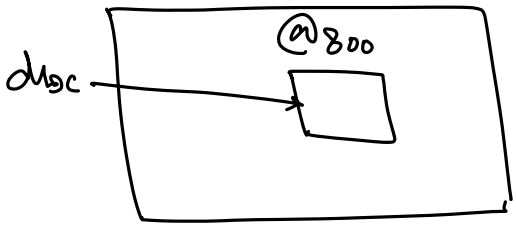
② LAZY.

→ Synchronized getInstance() method.

```
Class DatabaseConnection {
    Private Static DBC dbc = null;

    String url;
    int portNo;
    String userName;
    String password;
    Private DatabaseConnection() {
    }
    Static Synchronised
    Public DBC getInstance() {
        if (dbc == null) {
            dbc = new DBC();
        }
        return dbc;
    }
}
```

⇒ Slow performance.

Double Check Locking.




```
if (dbc == null){
    lock()
      if (dbc == null){
          dbc = new DBC();
      }
    unlock()
}
return dbc;
```

⇒ Best way to implement Singleton in Prod environment.

Steps.

1. Check without lock.
2. Take a lock.
3. Check again after lock.

⇒ Usecase of Singleton.

↳ Only 1 object.

⇒ Concurrency. ⇒ T1, T2, T3 → Commit.