Intro to Design Pattern

⇓ S/w System

⇓ something that repeats frequently

Well defined solution for commonly occuring problems in S/w system

⇒ Gangs of 4 (GoF) : 23 Design patterns

Why to learn Design Patterns

1) Common Vocabulary
2) Standard Solution.
3) Interviews

Type of Design Patterns → Object Oriented Design

1) Creational Design Pattern
   - → How an object can be created?
   - → Different ways of creating objects)
   - → How many objects can be created)

2) Structural Design Pattern
   - → How a class should be structured)
   - → What attrs & behaviors should be there in the class?

3) Behavioral Design Pattern
   - → Deals with Methods

Creational Design Pattern

- Singleton
- Builder
- Factory
- Prototype

Singleton Design Pattern

⇒ Allows us to create a class for which single object can be created

```
X x = new X();
```

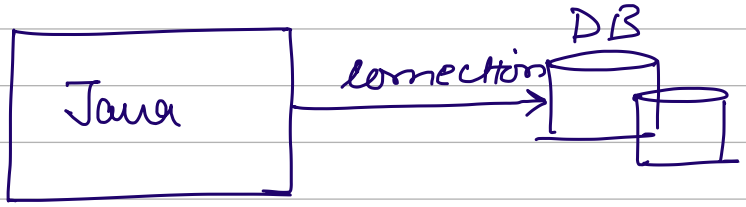
Why?

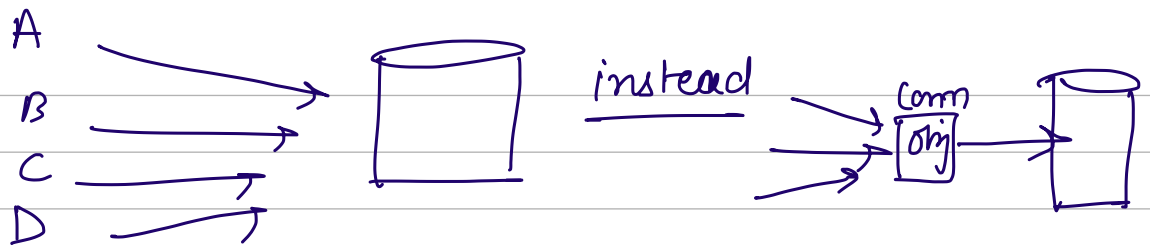
A class which is having a shared resource.
& its expensive

eg

```
Service {
    Database db;
    ____
    ____
}
```

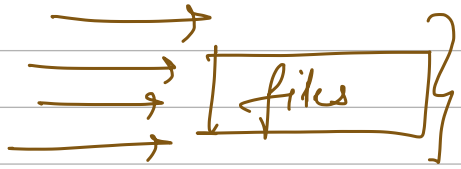
```
Database {
    Url;
    password;
    List<TCP connection>;

    PchageURL();
}
```



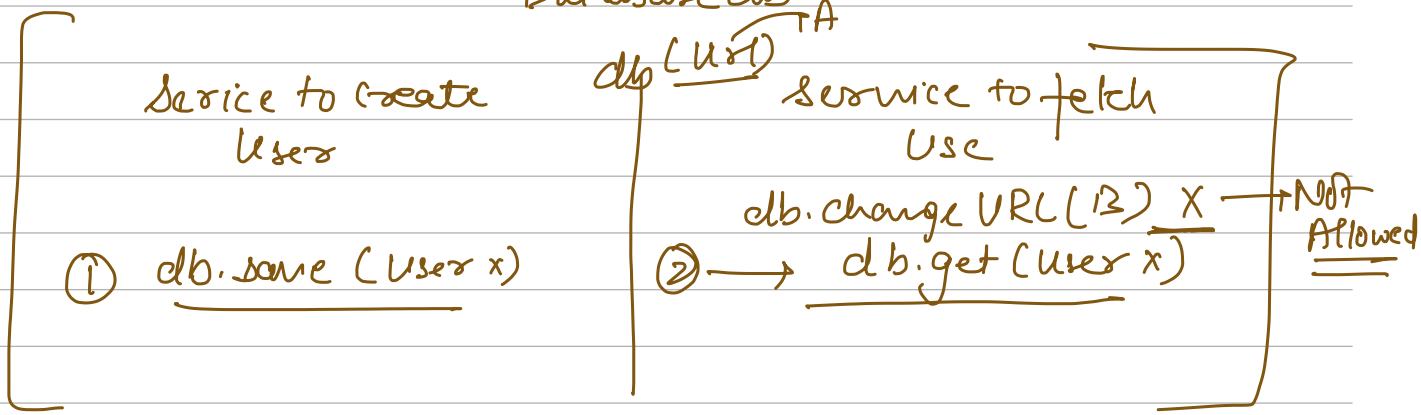


2) Maintaing a DB connection is very costly

2) Logger



2) If object creation is costly then singleton design pattern is used



Singleton are immutable

How to implement Singleton

```
Class DatabaseConn {

    Url;
    password;
    List<TCPConnection> connections;

}
```

Private + Mosetters

```
DBConn db1 = new DbConn();
DBConn db2 = new DBConn();
```

2) Till the time constructor are available then anyone can create objects of class

2) To restrict the object creation, we should make const PRIVATE

```
DBConn {

    private DBConn() {

    }
}
```

2) Because const is private we can't create even a single object.

```
class DBConn {

    private DBConn(){

    }

    public static DBConn getInstance(){
        return new DBConn();
    }
}
```

```
DBConn db1 = DBConn.getInstance();
DBConn db2 = DBConn.getInstance();
```

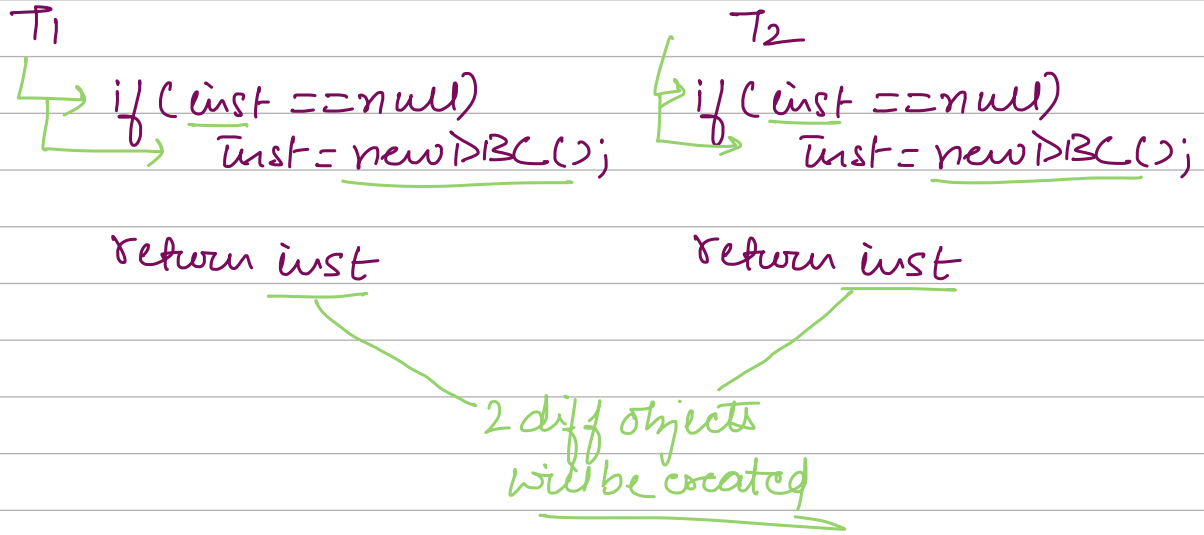
```
class DBC {
    private static DBC instance = null;

    private DBC(){

    }

    public static DBC getInstance(){
        if(instance == null){
            instance = new DBC();
        }
        return instance;
    }
}
```

(ReturnType)

1) Constructor private
2) getInstance() static
3) Static Instance variable

```
DBC db1 = DBC.getInst()
DBC d2 = DBC.getInst()
```

Single vs Multithreaded



Soln ① Eager Initialisation

App startup

```
class DBC {
    private static DBC instance
        = new DBC();

    private DBC() { }

    public static DBC getInstc() {
        return instance;
    }
}
```

```
DBC db = DBC.getInstc();
```

Disadvantage

1) If we have more no. of singleton then App start will be slow.
2) We won't be able to pass an parameter inside const.

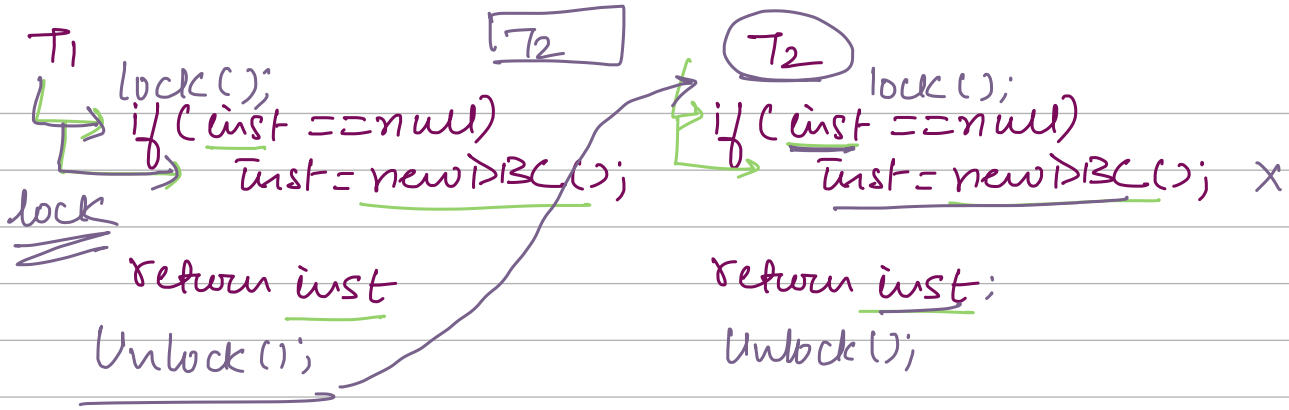
```
Class DBC {
    private static DBC inst = new DBC(5);

    private DBC (String url)
    {   if (url == null)
            throw exception;        // No DBC will be created
    }
}
```

8:12

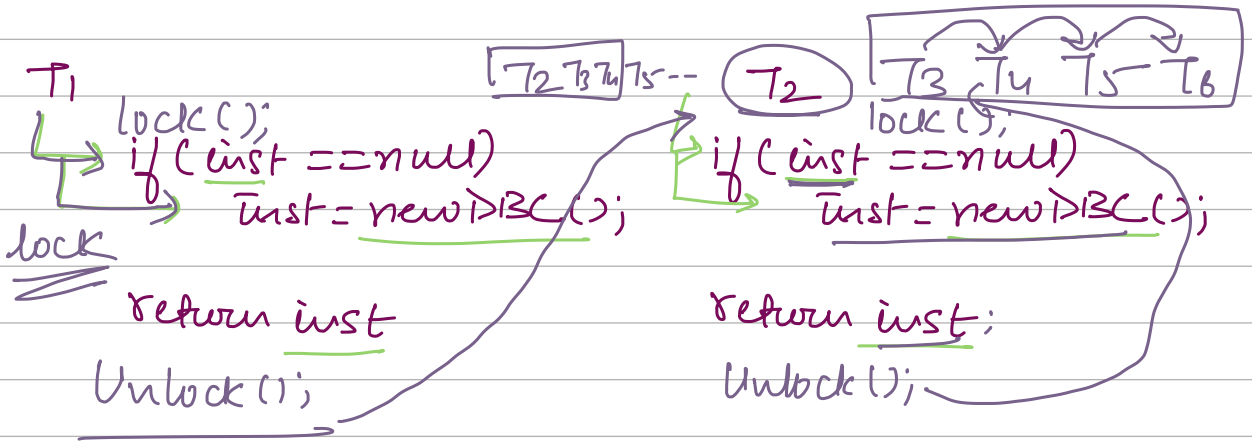
Lazy initialisation

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

    private DBC() {
    }

    public static synchronized DBC getInstance() {
        // Lock / Critical Section
        if (instance == null) {
            instance = new DBC();
        }
        return instance;
    }
}
```

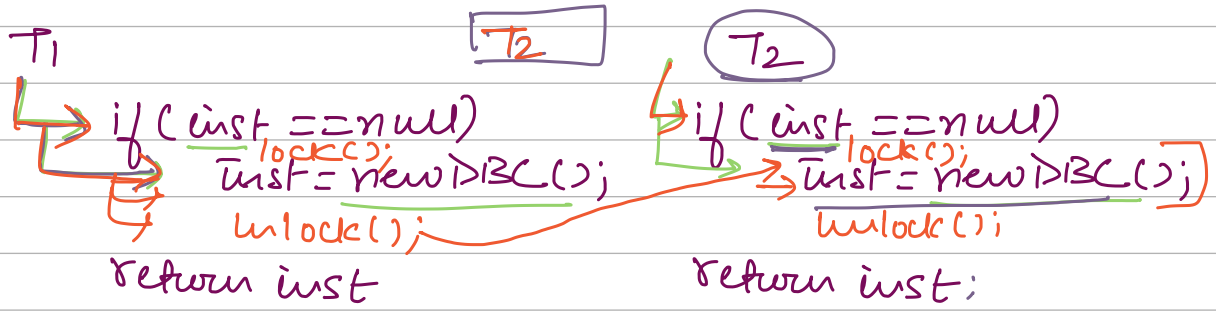
⇒ 2 ways → Lock
↳ Synchronised.



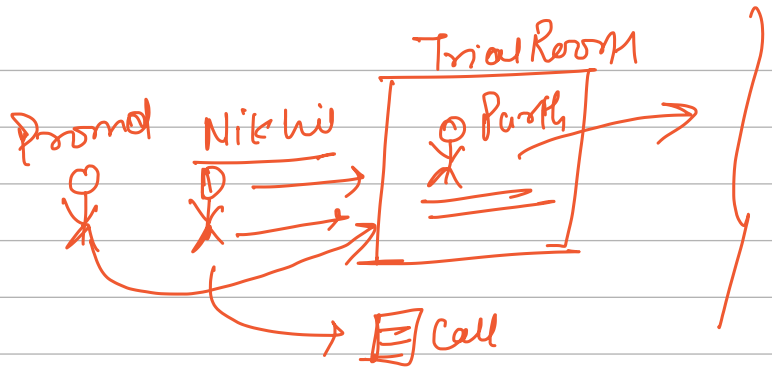
2) This implementation is Thread-safe
2) Slow performance = Disadvantage




```
if (inst == null)
    lock();
    inst = new DBC();   } critical Section
    unlock();
return inst;
```



2 Object will be created



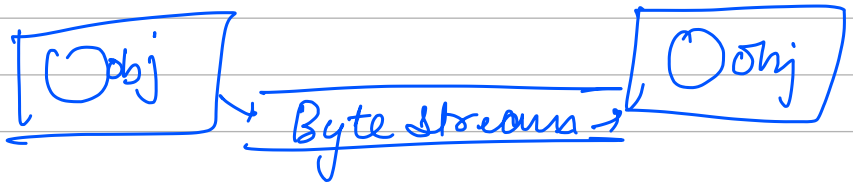

```
public static DBC getInstance(){
    if(instance == null){
        lock();
        if(instance == null){
            instance = new DBC();
        }
        unlock();
    }
    return instance;
}
```

Double Check Locking

1) Check the object without lock
2) Acquire lock
3) Check the object with lock

Note: For best performance, we should only acquire the lock only if instance is NULL

Singleton class for serialization



ENUMs