INTRODUCTION & SINGLETON

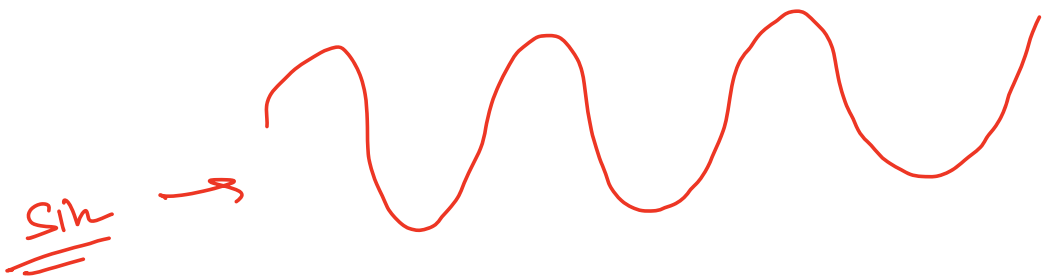
⊛ Agenda :-

- ⊙ Introduction to Design Patterns
- ⊙ Types of Design Patterns
- ⊙ Creational Design Pattern
  - ⊙ Singleton (with variations)

⊛ Introduction to Design Patterns:-

Design
↳ s/w Design

Pattern
↳ Something that occurs frequently.



Design Patterns :-

↳ well established solutions to common software design problems

GoF → Gang of Four (23 patterns)

→ at least 10 different Design Patterns (interviews)

⊙ Why Learn Design Patterns :-

1. It gave a "shared vocabulary" to Developers/Engineers.
2. It saves a lot of time.
3. Commonly asked in interviews :D.

✳ Types of Design patterns :-

1. Creational Design Pattern :-

   ↳ How will an object be created.

   ↳ How many objects will be created.

(2.) Structural Design patterns:-

↳ How will a class will be structured.

↳ what attributes will be there in a class.

↳ How will a particular class interacts with other classes.

3. Behavioural Design Patterns :-

↳ How to code an Action.

⊛ Creational Design Pattern :-

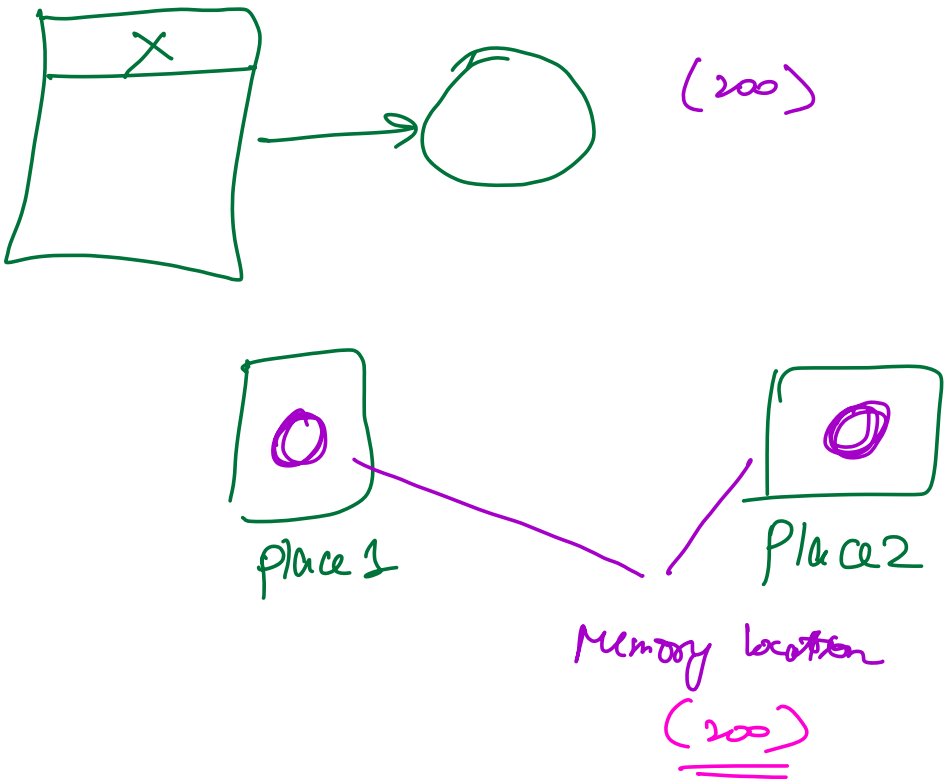
- Singleton
- Builder
- Factory → Simple factory.
  → Factory method.
  → Abstract factory
- Prototypes

⊛ Singleton :-

↳ Definition
↳ Problem statement (why ?)
↳ How to implement.
↳ Pros
↳ Cons

⊙ Definition :-

↳ It allows to create a Class for which only one object can be created (across the system).



Why do we need one object?

1. A class which is shared resource behind the scenes.

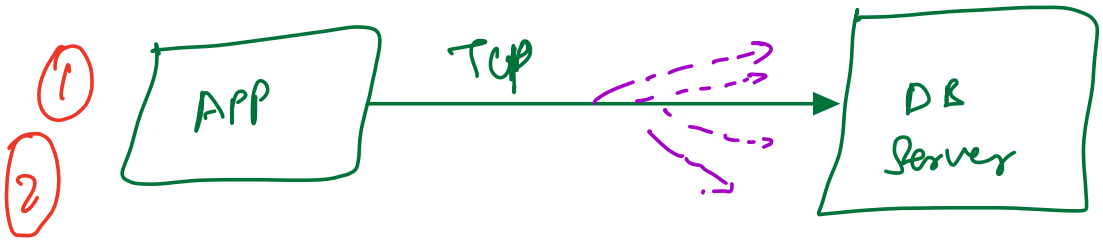
Ex - DB connection

```
Server {
    DBConnection db

    db.save();
    db.execute();
}
```

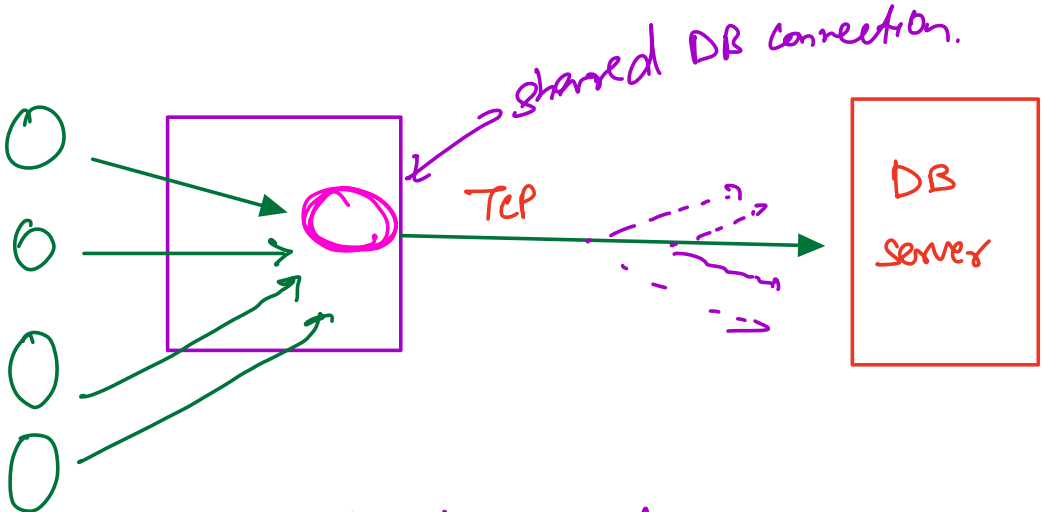
```
DBConnection {
- username
- password
- url
- List<TCP connection> Pool;

- cache
}
```

```
UserService {
    Database db;

    db.save();
}
```

```
OrderService {
    Database db;

    db.save()
}
```



(1) wasting a lot of CPU resources + memory.

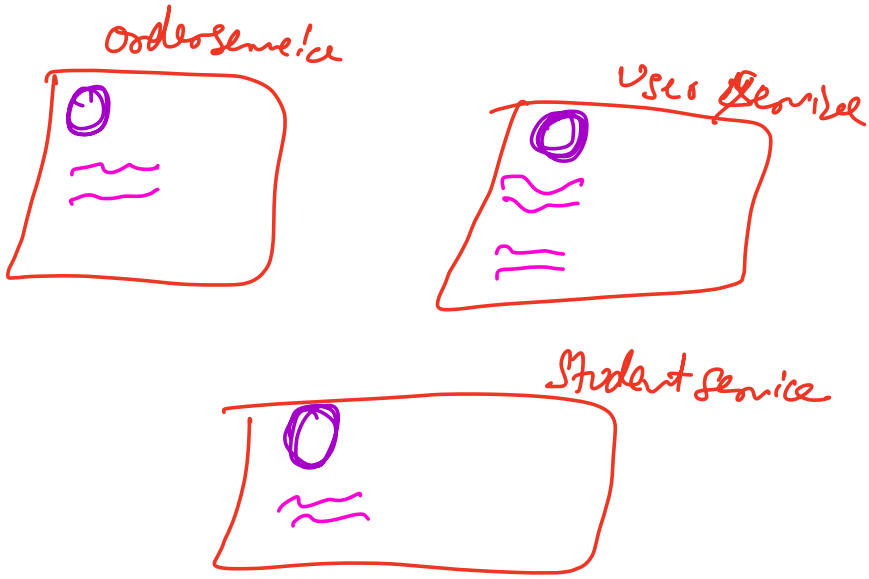


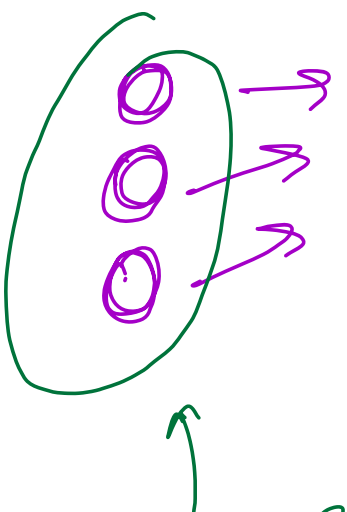
Yes! It's better! (optimal use of resources)

Ex — Logger !

↓

Library → prints the info on command line.





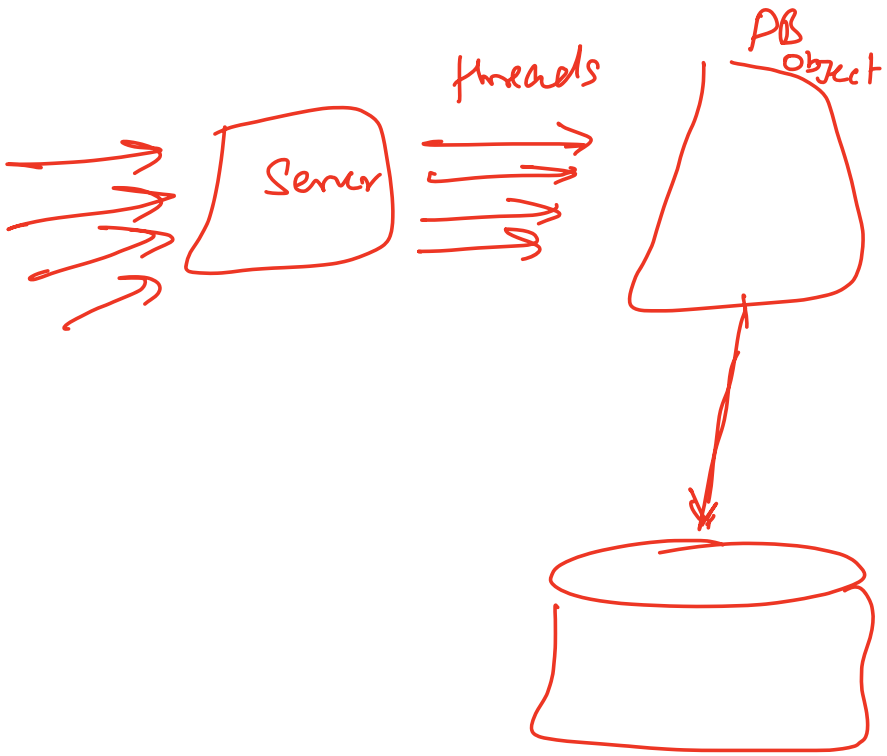
prints to command line.

prints to command line.

prints to command line.

↑

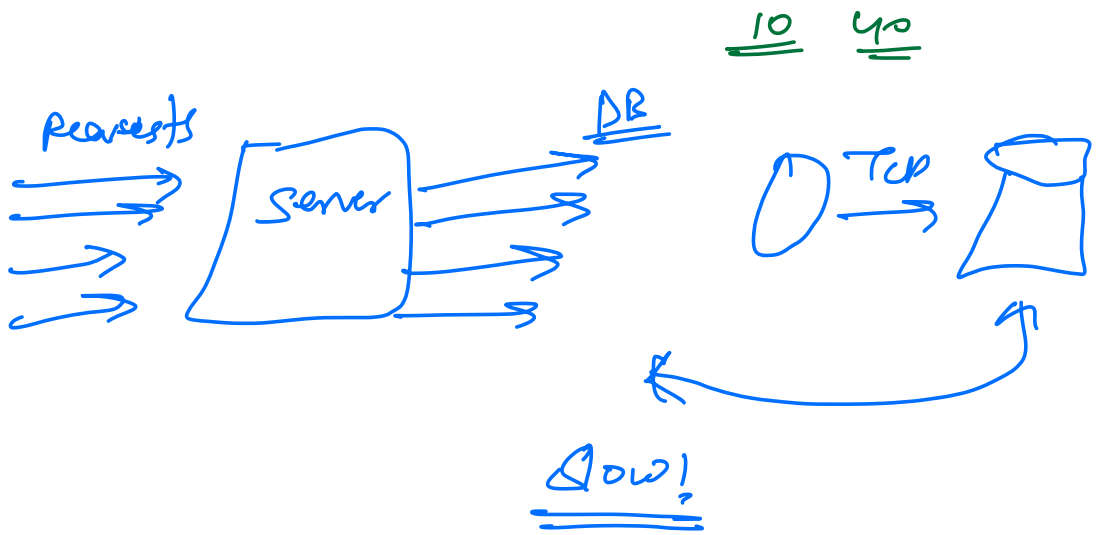
Singleton !


threads
PB
object
Server

2. Creation of object is expensive.!

(singleton)

Ex - DBConnection



If we create a singleton from a mutable class/object.

```
Db db = new Db();
db.setPassword(  );
```

Singletons are always immutable!

Summary! -

- Singleton → Class for which only 1 object can be created
- Why?
  - When shared resource are used.
  - Creation of object → expensive.
  - Wherever it makes common sense !!.

⊙ How do implement Singleton:-

```
class DBConnection {

    String url;
    String password;
    String username;
    List<Connection> pool;
        :
}
```

```
DBC db1 = new DBC();
DBC db2 = new DBC();
```

```
class DBconnection {

    String url;
    String password;
    String username;
    List<Connection> pool;

    Private DBconnection() { }

}
```

1. we need to have constructor → Private
2. But we need to create one object.
3. TO create a object
   ↳ we need to call the constructor.

⊙ Private members are accessible within the class itself.

```
class DBConnection {

    String url;
    String password;
    String username;
    List<Connection> pool;

    Private DBConnection() { }

    Static
    Public DBConnection getInstance() {
        return new DBConnection();
    }
}
```

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

db1 == db2 (false)

```
class DBConnection {

    Private static DBConnection instance = null;

    String url;
    String password;
    String username;
    List<Connection> pool;

    Private DBConnection() { }

    Public static DBConnection getInstance() {

        if (instance == null) {
            instance = new DBConnection();
        }
        return instance;
    }
}
```

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

db1 == db2 (false) → True.

Steps:-

1. Make the constructor private.
2. Create a static method to create the instance.
3. Have checks;
   if object created → return
   otherwise;
   Create object & return;

T1

```
if (instance == null) {
    instance = new DBConnection();
}
return instance;
```

T2

```
if (instance == null) {
    instance = new DBConnection();
}
return instance;
```

instance = ~~null~~; ~~ins1~~ ins2;

100

99

(object will still be there until garbage collected).

(*) Singleton in Concurrent Environment:-

(1) Eager loading / execution:-

↳ we can create an object during class load time or when application starts

```
Class DBC {
    ;
    Private Static DBC instance = new DBC();
    .
    Private DBC() { }

    Public static DBC getInstance() {
        return instance;
    }
}
```

Cons:-

↳ It will slow down the application start time.

↳ we can't pass parameters/ configurations while creating the very first object.

Logger ——

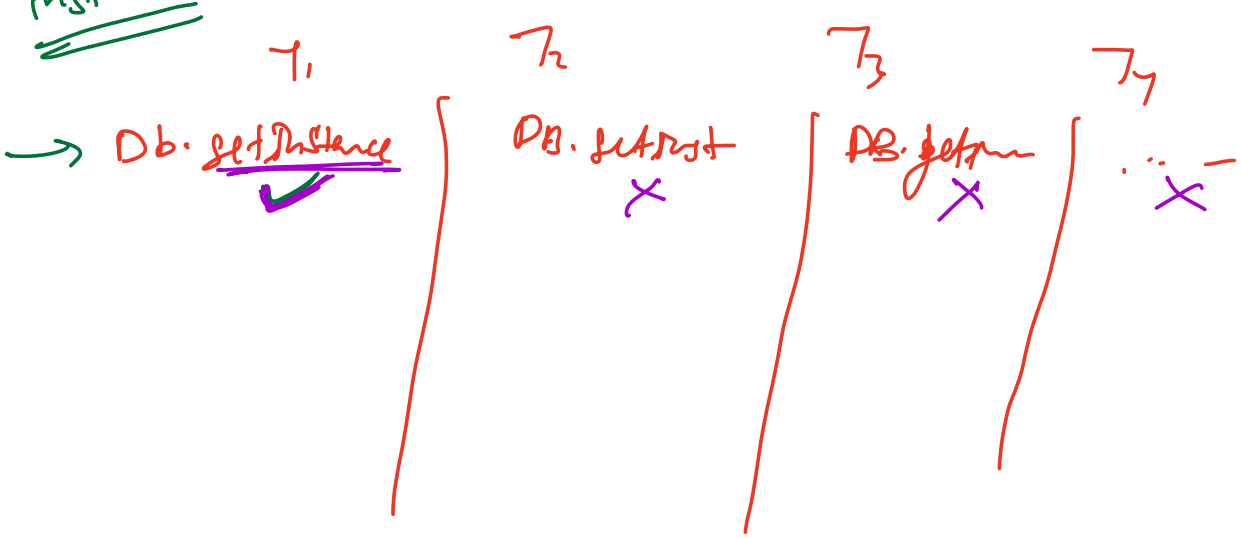
```
Public Logger getLogger (String env) {
    If (env == Prod) {
        info.
    }
    else if (env == "dev") {
        debug
    }
}
```

(1) we want to create singleton at runtime

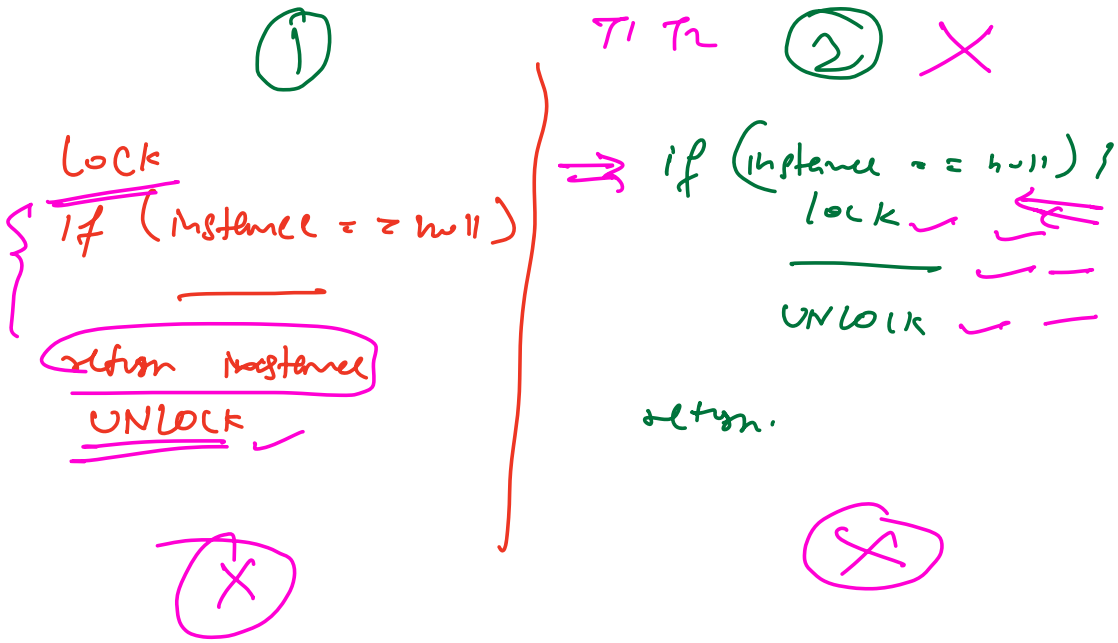
(2) It should work fine for Concurrent env.

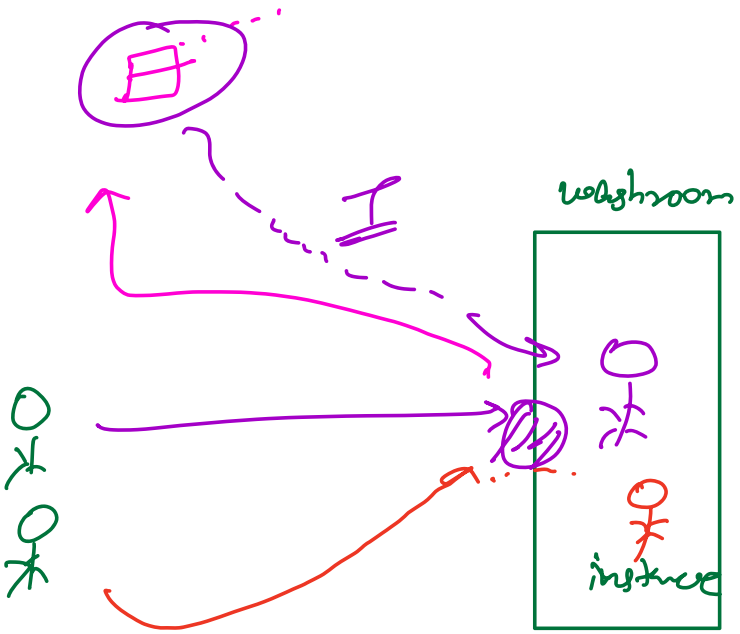
```
Public synchronized static DBConnection getInstance(){
    if (instance == null){
        instance = new DBConnection();
    }
    return instance;
}
```

instance = null



DB.getInstance();

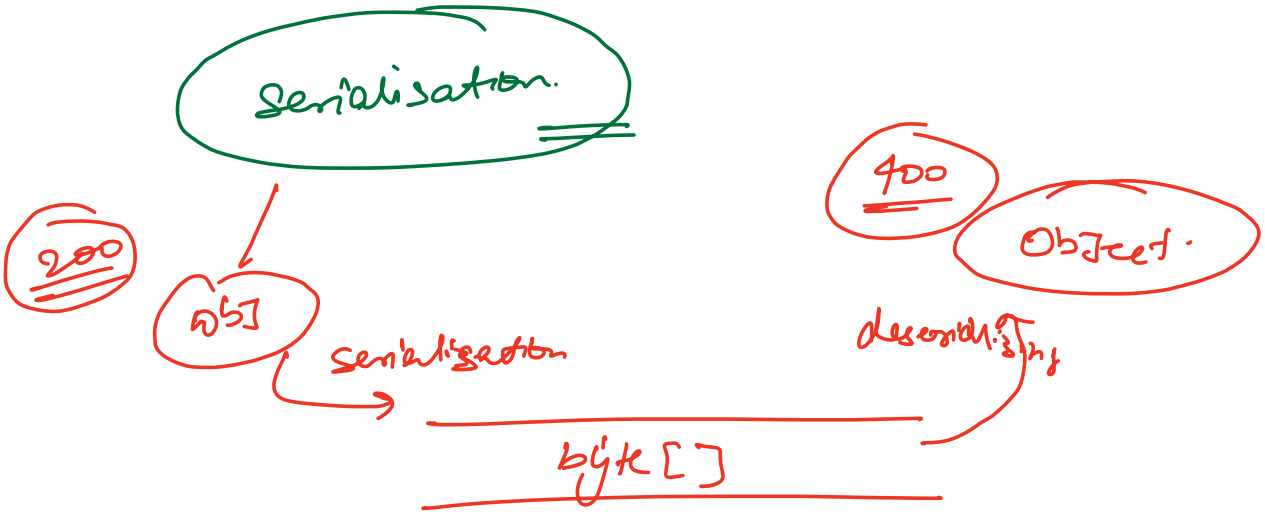




Double-Check locking !

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

    Private DBC() { }

    Public static DBC getInstance() {

        if ( instance == null ) {
            LOCK;
              if (instance == null) {
            instance = new DBC();
            }
            UNLOCK
        } }

        return instance;

}
```

(Best Solution)



Hlw

Singleton with Enums (Java).

⇓ ↑

Best way to implement singleton.

(*) Pros :-

↳ It provides Resource efficiency.

↳ singleton is like caching the objects ⇒ creation of object is inefficient.

(*) Cons:-

1. Difficult to Testing.

↓

Mocking the dependencies becomes difficult.