

Agenda :

1. Introduction of Design Patterns
2. Singleton

What are design patterns?

↓
Softwares

↘ frequently occurring

Software design patterns which well
established solutions to frequently occurring
problems

→ Gang of four

Design Pattern :
↓
~23 patterns
↓
~10 patterns
↓
3-4 are used commonly
rest are interviews

Structure of a class
& how class interacts with each other

Patterns:

creation of object

1. Creational

Singleton

Builder

2. Structural

factory

3. Behavioral

prototype & registry

Adapter

Observer

How
code will

Flyweight

Strategy

work in a class.

Decorator

Facade.

command design pattern

↳ LLD 3 (Case Studies)

Singleton :

- ① only allows single object / instance for a particular class

- ② delivers a single instance everytime.

Examples:

```

class dbConnections {
    url
    password
}

new dbConnection();

```

② logger (loggings) console.log
↳ log in a file sysout
cloudwatch
datadog

Where we use: Whenever a class is a Shared Resource & you don't need multiple objects.

Object creation is expensive so avoid creating multiple objects if not required.

(v)

```
class dbConn {  
    url ;  
    userName ;  
    passWord ;  
}
```

}

```
new dbConn(); // new object  
new dbConn();
```

(vi)

```
class dbConn {
```

```
private dbConn() { }
```

}

```
new dbConn(); - X
```

(vii)

```
class dbConn {
```

```
private dbConn() { }
```

```
static dbConn getInstance() {
```

```
return new dbConn();
```

```
} }
```

everytime
creates
new obj { dbConn.getDBInstance();
dbConn.getDBInstance();

V4

```
class dbConn {  
    private static dbConn dbInstance = null;  
    private dbConn() { }  
  
    static dbConn getDBInstance() {  
        if (dbInstance == null) {  
            dbInstance = new dbConn();  
        }  
        return dbInstance;  
    }  
}
```

private Constructor → static getInstance → static ref variable to store instance

↓
if instance ref is null create instance
else return it as-is

IS IT THREAD SAFE !

T₁ ↓

```
class dbConn {  
    private static dbConn dbInstance = null;  
    private dbConn() {}  
  
    static dbConn getDBInstance() {  
        ↓ T1  
        → if (dbInstance == null) {  
            ↓  
            dbInstance = new dbConn();  
        }  
        return dbInstance;  
    }  
}
```

T₂ ↓

```
class dbConn {  
    private static dbConn dbInstance = null;  
    private dbConn() {}  
  
    static dbConn getDBInstance() {  
        ↓ T2  
        → if (dbInstance == null) {  
            ↓  
            dbInstance = new dbConn();  
        }  
        return dbInstance;  
    }  
}
```

Threads coming during creation

↓↓↓↓↓

() this is where problem lies

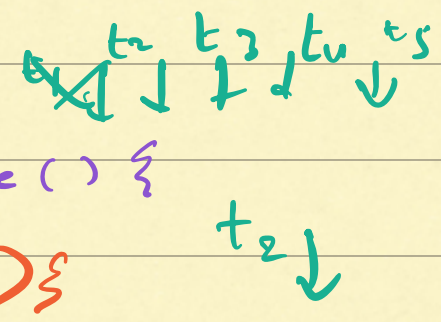
After creation threads

↓↓↓↓↓

no problem.

Synchronised Method :

```
class dbConn {  
    private static dbConn dbInstance = null;  
    private dbConn() { }  
    static synchronized dbConn getInstance() {  
        if (dbInstance == null) {  
            dbInstance = new dbConn();  
        }  
        return dbInstance;  
    }  
}
```



Threads coming during
creation

100 ↓ ↓ ↓ ↓ ↓ ↓

() no problem

Threads come after
will all wait
on each other

100000 ↓ ↓ ↓ ↓ ↓ ↓ → eternity

() waiting threads
increases latency.

Break till 8:25

Option 1 ~~X~~

Option 2

dbConn getInstance {

lock();

if (dbInstance == NULL) {

dbInstance = new dbConn();

} unlock();

return dbInstance;

dbConn getInstance {

if (dbInstance == NULL) {

lock();

dbInstance = new dbConn();

unlock();

return dbInstance;

Option 3 - Double check lock



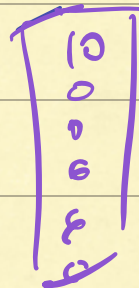
Sec1

Population
control.



Sec2

→ ticket
passport
what you carrying



dbConn getInstance { ↓↓↓ t₁ t₂ t₃

if (dbInstance == NULL) { ← sec1
lock();

if (dbInstance == NULL) { ← sec2

dbInstance = new dbConn();

} unlock();

return dbInstance.

}

Threads coming during
creation
100 ↓↓↓↓↓
() no problem

After creation threads
100000000 ↓↓↓↓↓
no problem.