Agenda

1) Intro to Design Pattern
2) Types of Design Pattern
3) Singleton

Design Patterns

↓ Design Software Systems

↓ Something that occurs frequently

Design patterns are well defined solutions to commonly occuring Software problems

Gang of four : GOF

↳ 23 Design Patterns

10 Design Patterns

98% Problems

Chain of Responsibility

Why Design Pattern

1. Common Vocabulary
2. Saves a lot of time
3. Commonly asked Interview Questions

YES

Types of Design Patterns

1. Creational
   1) How objects should be created
   2) How many objects to create
2. Structural
   1) How will a class to be structured
   2) What attributes will be there
   3) How a class will interact
3. Behavioral
   1) How to code the method/Action

Creational

↳ creation of objects

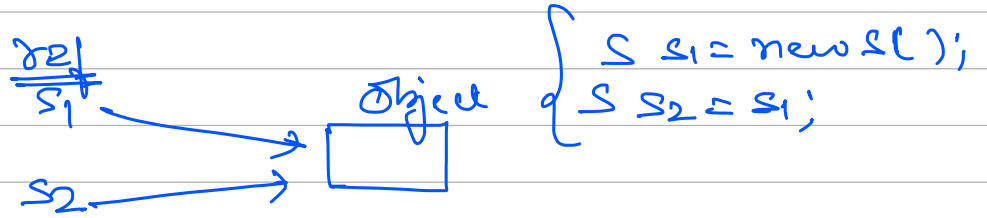
1) Singleton
2) Builder
3) Factory → Simple factory
   → Factory Method
   → Abstract factory
4) Prototypes

Singleton

Definition
Problem statement
How to implement
Pros & Cons
IDE Impl

Singleton

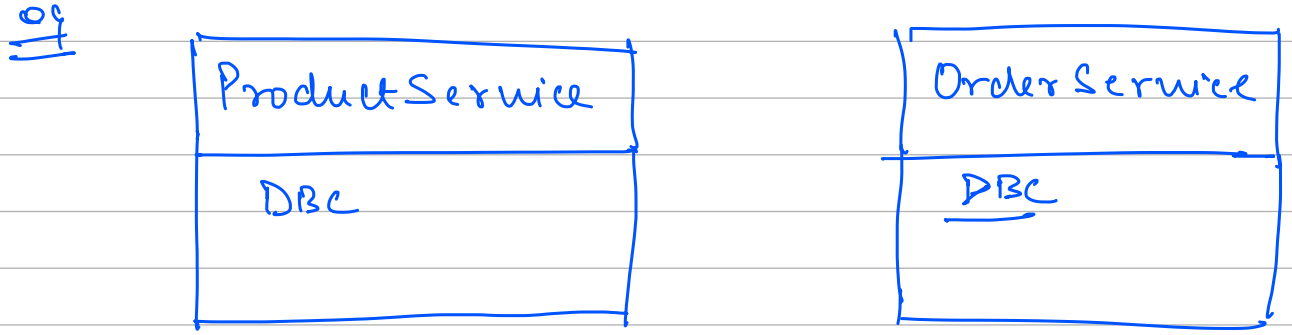
allows to create a class for which only one Object can be created



Why Singleton

Why do you need a Single Object of a particular class?

Monolith vs Microservice



40-50k per min

Logger

log4j

Retail pricing

↳ 1 Request → 234 logs

8350 Requests/secs

8350*234

② Creation of object is expensive

1) Need to pass username/password
2) Send TCP Request
3) Establish TCP Connection

DBC object

→ TCP Pool [ ]

```
DBC {
    String URL;
    String username;
    String password;
    List<Connection> pool;
}
```

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

Version 2

```
DBC {
    String URL
    String username
    ---------
    ---------

    private DBC() {

    }
}
```

Problem statement

1) We need to keep constructor private
2) But we need to create a single object

```
Class DBC {

    ==========

    private DBC () {

    }

    public static getInstance () {
        return new DBC();
    }

}

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

Eager Initialised

```
Class DBC {
    public static DBC instance = new DBC();

    ==========

    private DBC () { }
    public static DBC getInstance () {
        return instance;
    }
}
```

1) Slow performance
2) Unnecessary Initialisation

Lazy Version3

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

    private DBC (){

    }

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

Critical Section

Steps

1) Make Constructor Private
2) Create Static method to create instance
3) Static method checks if objects exist then return same object else create a new object.

Version 3

T_1 DBC db = DBC.getInstance();

T_2 DBC db = DBC.getInstance()

T_1 | T_2 if(Instance==null){

Inst = new DBC();

{

EAGER Loading → Multi Thread Safe

↳ ① Slow performance { Application start time will increase }

② Custom attributes

eg Flag is True → DB

↳ OrderService → MySQL

→ DynamoDB

Retail Pricing

↳ 2 Billion | Events Per week

↳ 8340 Request second

↳ 8.3 Request ms

1 Request | ms

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

    private DBC (){

    }

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

Thread

(critical Section)

T_1 DBC db = DBC.getInstance();

T_2 DBC db = DBC.getInstance();

↳ sequential execution

Performance of the application is impacted

Version 4

```
public static DBC getInstance{
    if(Inst=null)
        lock
        inst= new DBC();
        UNLOCK
    return Instance;
{
```

T_1 | T_2

2 objects

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

Double Checking Lock

SeatBooking
↳ Reserve

Pros of Singleton
↳ Resource efficiency
↳ New Object efficiency

Con
↳ Testing is difficult