Agenda.

1) Intro to Design Patterns.

2) Singleton Design Pattern.

What are Design Patterns.

$\downarrow$

something that repeats frequently.

Standard ways to solve commonly occurring problems in Software Industry

GOF : Gang of Four. (23 Patterns).

10 design patterns.

Object Orient Prog.

Types of Design Pattern.

1) Creational

↳ Different ways to create an object.

↳ Object creation.

→ Singleton

→ Builder

→ Prototype

→ Factory.

2) Structural

- ↳ How a class|interface should be structures.
- ↳ How classes will interact with each other.

→ Adapter
→ Facade
→ Decorator
→ Flyweight.

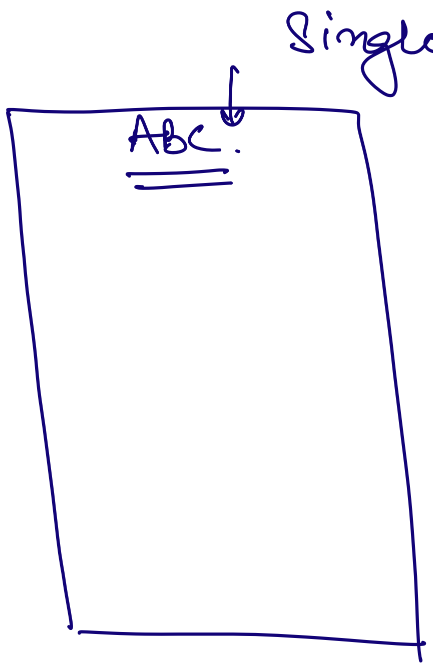
3) Behavioural.

↳ Methods | Behaviours | Actions.

→ Strategy
→ Observer.

SINGLETON.

⇒ Allows us to create only one object of a class across the codebase.



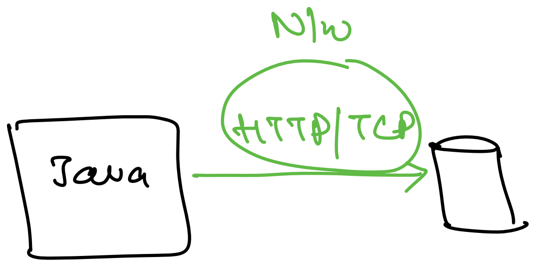
@200

ABC abc = $\boxed{\text{new ABC()}}$

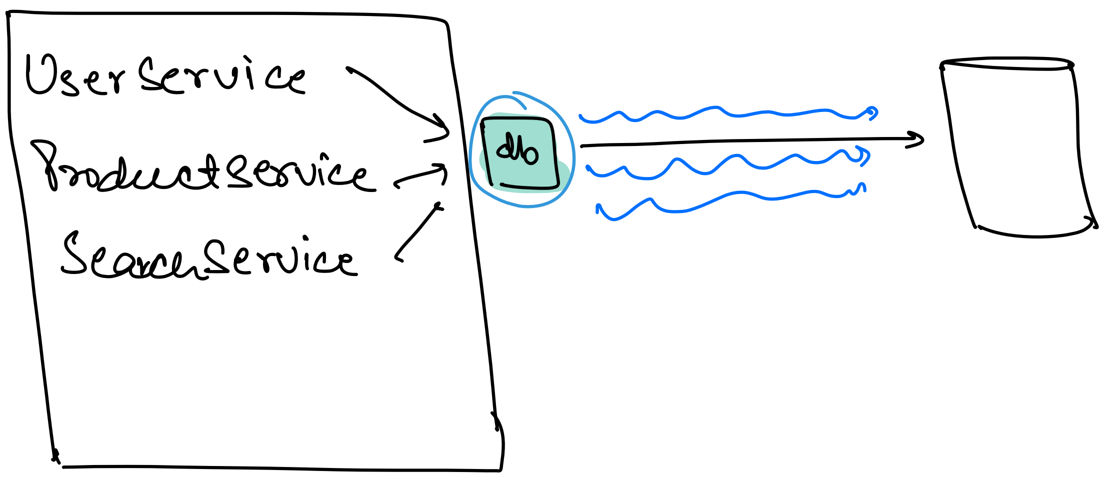
ABC abc1 = abc;

Why?

```
UserService {
        Database db;

        db.save()
        db.update;
}

ProductService {
        Database db;

        db.save()
        db.update;
}

StaudService {
        Database db;

        db.save()
        db.update;
}
Database {
     url;
     username;
     Password
     HTTP/TCP Conn ⇐ List of conn.
}
```

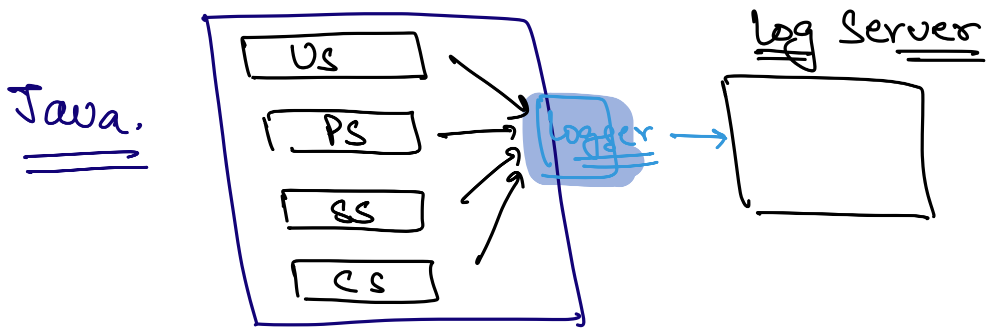


Database object creation is expensive as it involves lot of attrs and also it will have to maintain a conn.



⇒ Shared resource.

Logger.



⇒

When object creation is expensive and we have a shared resource.

How to implement Singleton.

(1)

```
Class DatabaseConnection {
      username
      password
      url
      List<TCP> —
}
```

Not Singleton

```
DBC dbc1 = new DBC();
DBC dbc2 = new DBC();
```

Till the time the constructor is available, anyone can create any no. of objects of DBC.

(2)

```
Class DatabaseConnection {
      username
      password
      url
      List<TCP> —
      private DatabaseConnection() { }
}
```

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

③
```
Class DatabaseConnection {
        username
        password
        url
        List<TCP> —

        private DatabaseConnection(){ }

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

④
```
Class DatabaseConnection {
        username
        password
        url
        List<TCP> —

        private DatabaseConnection(){ }

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

```
DBC dbc = DBC.getInstance();
```

3

```
Class DatabaseConnection {
        private static DBC dbc = null;

         username
         password
         url
         List<TCP> —

         private DatabaseConnection() { }

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

⇒ Static methods can only access Static attrs.

```
if (dbc == null) {
     dbc = new DBC();   ← T1
                        ← T2
}
```

} 2 instances will be created.

Singleton in Multithread environment.

6. Eager | Early Initialization.

```
Class DatabaseConnection {
        private static DBC dbc = new DBC();

        username
        password
        url
        List<TCP> —
                                          UN  PW
        private DatabaseConnection( — — ){ = }

        public static DBC getInstance(){
                return dbc; ↙T1
                            ←T2
        }
}
```

Cons:-

1) App Startup time will increase.
2) Won't be able to pass attributes inside the Constructor.

Lazy Initialization.

7.
```
Class DatabaseConnection {
        private static DBC dbc = null;

        username
        password
        url
        List<TCP> —

        private DatabaseConnection() { }

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

Con. : Low performance.

How to optimize the performance.

⇒ Double Check Locking.

```
Class DatabaseConnection {
        private static DBC dbc = null;

        username
        password
        url
        List<TCP> —

        private DatabaseConnection() { }

        public static DBC getInstance() {
                if (dbc == null) {
                        lock()
                        if (dbc == null) {
                                dbc = new DBC();
                        }
                        unlock()

                }
                return dbc;
        }
}
```

⇒ Best way to implement Singleton in prodⁿ env.

1) Check without lock.
2) lock
3) Check inside lock.

Pros :

Resource efficiency

Cons.

Testing.

HW:- Implement Singleton.

——— * ———