Agenda

Intro

Types of design patterns

Singleton

What are design pattern ?

- Software.
- frequently occurring

well established solutions of some common software design problems.

Why learn dp ?

- shared vocabulary.
- saves a lot of development time.

~23 ✗

↓ Gang of four -

~10 design

1. dev
2. interview

Types of dp

1. creational → creation of objects
   - Singleton
   - Builder
   - factory
   - prototype, Registry
2. structural
   - Adapter
   - Facade
   - Decorator
   - Flyweight.
3. behavioural
   - Observer
   - strategy
   - (Command) → LLD 3

Singleton Design Pattern

1) only allows to create a single object of the class
2) delivering the single instance whenever required.

classes which uses shared resources.

- → db connection
- → Logger
- → Thread Pool.
- → Configuration manager
- → Cache manager.
- ⋮

```
class dbconnection {
    url
    path
    .
    .
    .
}
```

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

constructor

private ?

we will not be able to access the constructor outside the class.

```
class dbconn {

private dbconn() {
    ==========
}

static dbconn getInstance() {
    return new dbconn();
}

}
```

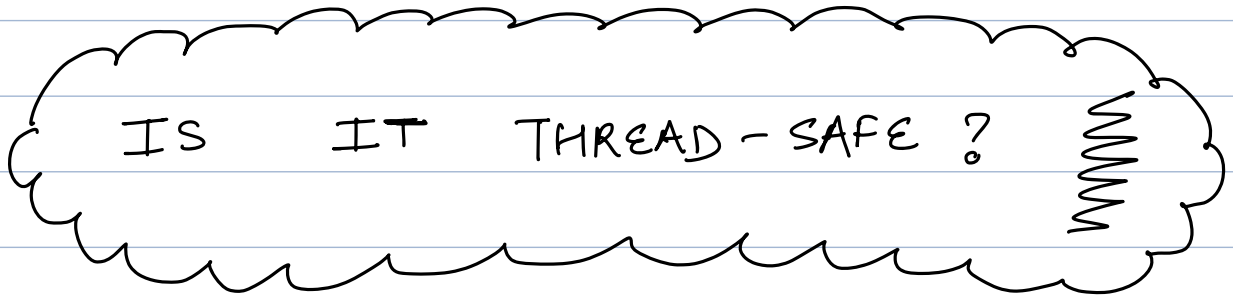
```
dbconn db1 = dbconn.getInstance()
dbconn db2 = dbconn.getInstance()
```

```
class dbconn {
    private static dbconn dbconnInstance = null;
    private dbconn() {
    }
    static dbconn getInstance() {
        if (dbconnInstance == null) {
            dbconnInstance = new dbconn();
        }
        return dbconnInstance;
    }
}
```

private → you can't access the instance var inside static.

Lazy initialization

Normal → private constructor → getInstance() → made it static → ref variable as member → initialising it once & return the same every time → ref → private static, final.



NO

T1

getInstance()

if (db == null)

new obj

T2

getInstance()

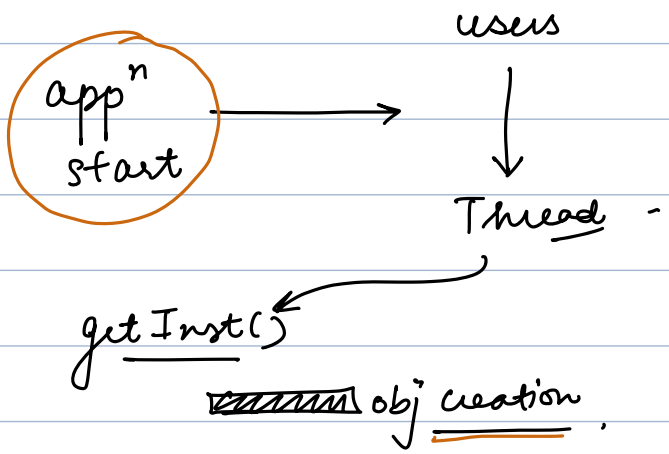
if (db == null)

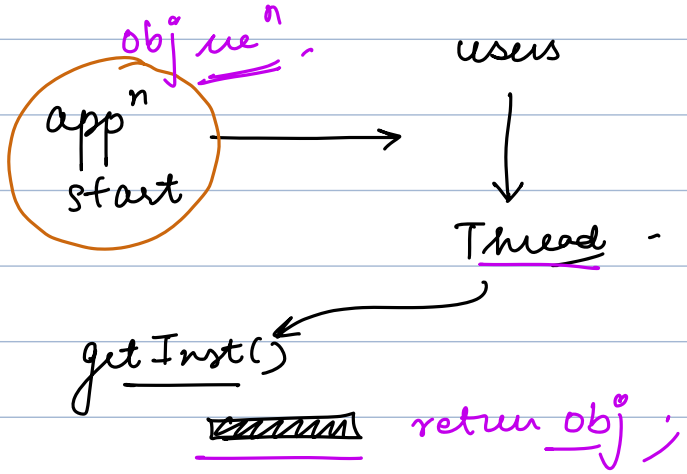
new obj

Eager Initialisation

```
class dbconn {
private static dbconn dbconnInstance = new dbconn();
private dbconn() {
}
static dbconn getInstance() {
    if (dbconnInstance == null) {
        dbconnInstance = new dbconn();
    }
    return dbconnInstance;
}
}
```

we don't need this.





① App^n start time ↑

② we may not use it ∴ wastage.

③ can't give variable config at loading

```
class dbconn {
private static dbconn dbconnInstance = null;
private dbconn() {
```

```
synchronized
static dbconn getInstance() {
    if( dbconnInstance == null) {
        dbconnInstance = new dbconn();
    }
    return dbconnInstance;
}
}
```

performance : slow

- use locks

```
getI() {
    if (dbconn == null)
    {
        lock();
        db = new dbconn();
        unlock();
    }
    return db;
}
```

This doesn't solve the sync issue.

```
getI() {
    lock();
    if (dbconn == null)
    {
        db = new dbconn();
    }
    unlock();
    return db;
}
```

Is it any way diff then Synchronized method?

```
getI(){

    if(dbconn==null)
    {
        lock();
        if(dbconn==null)
            db = new dbconn();
        unlock();
    }

    return db;
}
```

double-check locking

V0 : Single Threaded
V1 : Multi + Eager
V2 : Multi + Synchronized
V3 : Multi + double-check.

3 ques^n

serialization
obj → db/file. → deserialization
you will be able to create a new Instance of that

→ readResolve(){
    return db;
}

Reflections : you can manipulate objects at runtime.