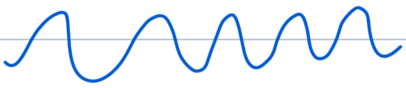
Agenda:

- Intro to Design pattern
- Types of design pattern
- Singleton design pattern

What are design pattern ?

SD

Something the occurs frequently



well established solutions of common software design problems.

~23 dp

~10 design pattern

1. devs
2. interview

Why learn dp ?

1. shared vocabulary
2. Save a lot of time

Types of DP :-

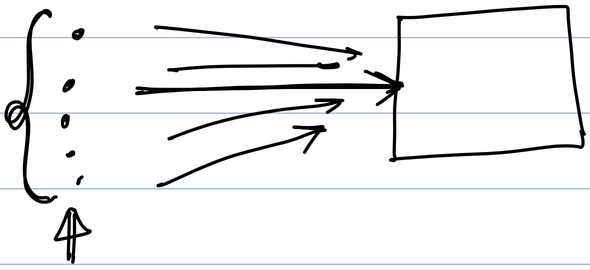
- creational :- creation of objects
    - Singleton
    - Builder
    - Prototype & Registry
    - Factory
- Structural
    → class , structured
- behavioural
    - code on action

Singleton Design pattern

defⁿ :- Allows you to create a class for which only one object is to created

A class which works on shared resources :-

db connection



```
Server {

    Dbconnection db;

     db.save()
     db. execute()

}
```

Logger

```
class    Dbconnection {
             url;
             username;
             password;
             :
  }
```

```
Dbconn   db1 =  new  Dbconn();  } X
Dbconn   d2  =  new  Dbconn();  }
```

Till the my constructor is public, can it be singleton?

$\Downarrow$

private?

$\downarrow$

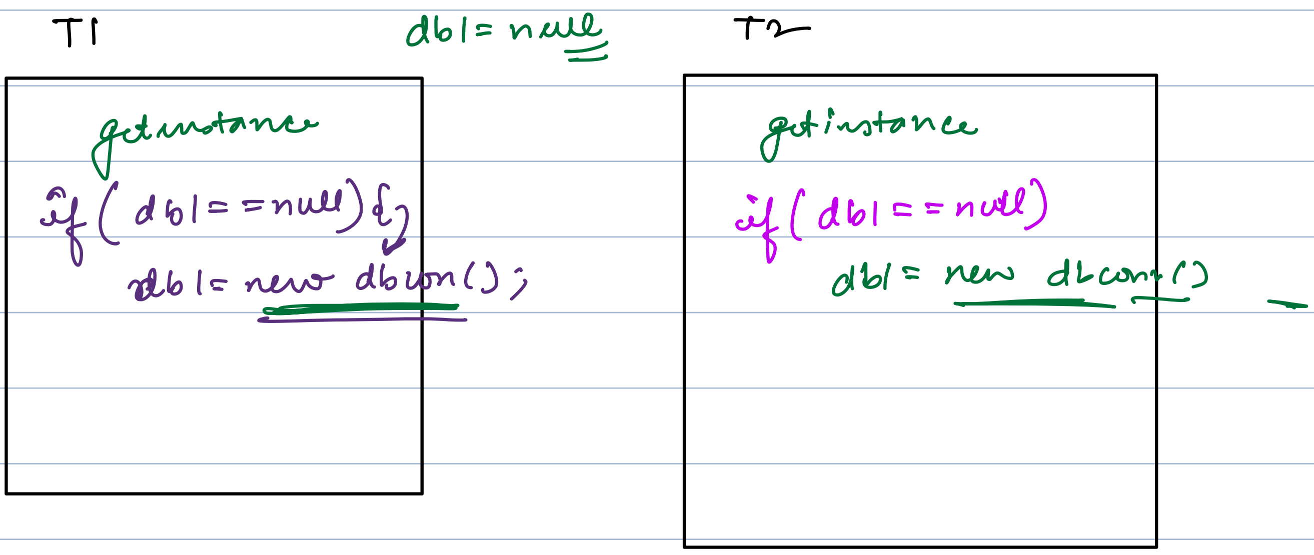
we will not be able to access the constructor outside the class.

```
class  db'conn {

    private  dbconn(){ }
                     static
    public  ^  dbconn   getInstance(){

            return  new  dbconn();

    }

}

      dbconn   db1 =   db.conn.getInstance();
      dbconn   db2  =  db.conn.getInstance();
                                  ⋮
```

```
class   db'conn {
                   static
      private  ^  dbconn   db1 = null;
      private   dbconn(){ }

                static
      public  ^  dbconn   getInstance(){
               if( db1 == null){
                      db1 = new dbconn();

               }
               return  db1;

      }

}
```

Steps:

1. Make the constructor private
2. create a static f^n which will actually help in creating the obj
3. static member → this will hold our object
   ↓
   private



```
getinstance
if( db1==null){
    db1= new dbcon();
```

```
getinstance
if( db1==null)
    db1= new dbcon()
```

error prone before the object gets created first time.

Eager Loading

```
class   db'conn {
                          static
    private   ^   dbconn   db1 = new dbconn();     ↑ f     class got loaded
                                                             ↓
                                                          appⁿ starts
    private   dbconn(){ }

                static
    public  ^   dbconn   getInstance(){

            ~~if ( db1 == null){~~
               ~~db1 = new dbconn();~~
            }

            return   db1;

    }

}
```

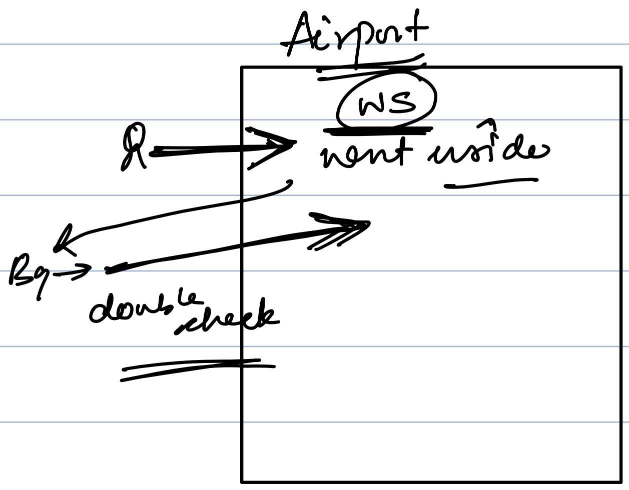
cons : ① Appⁿ startup time will increase.
② can't give variable config while loading the classes.

```
Logger {

        Logger( String env)
    {       if ( even == dev)
               ______
            else if( env == prod)
               ______
        }
    }
```

```
class    db'conn {
                  static
      private  ^  dbconn   dbl = null;
      private   dbconn(){ }

                synch..
   public static ^ dbconn    getInstance(){
           if ( dbl == null){                     CS
                dbl = new dbconn();
           }
           return  dbl;

      }

}
```

performance $\longrightarrow$ slow!

```
    getI()                         |        getI()
{     lock();                      |   {
        if (dbl==null)             |          if (dbl==null)
        {                          |          {   lock();
             dbl= new dbc          |                dbl= new dbc
                                   |                unlock();
        }                          |          }
      unlock();                    |
        return dbl;                |          return dbl;
 }                                 |    }
```



```
get I()
{
    if (db1 == null)
    {
        lock();
        if (db1 == null)
            db1 = new db();
        unlock();
    }
    return db1;
}
```

T1 T2

T2 - wait

T1

{double-check locking}

singleton

Java specific

↓

Enums → thr

↑

reflection
searlizable
desecialc

$$\left\{\begin{array}{l} \text{V0} \rightarrow \overset{\text{a}}{\text{single}}\ \text{Threaded} \\ \text{V1} \rightarrow \text{mult — eager} \\ \text{V2} \rightarrow \text{multi' — double } \underline{\underline{\text{check}}} \end{array}\right.$$