Today's agenda

↳ Design Pattern Intro

↳ Singleton design Pattern

* what is design Pattern? → software design
→ same thing occurring again and again

↳ well established solution to frequently occuring software Problems.

→ G.O.F: gang of four
↳ 23 design Patterns

* Types of design Patterns: →2008

(i) Creational design Patterns: Different ways to create an object.
ex: Singleton, factory, builder etc.

(ii) Structural design Patterns: How to decide methods and attributes of a class.
ex: Adapter, bridge etc.

(iii) Behavioural design Patterns: implementation of behaviour in a class.
ex: Strategy, iterator etc.

Q) Why to study design Patterns?

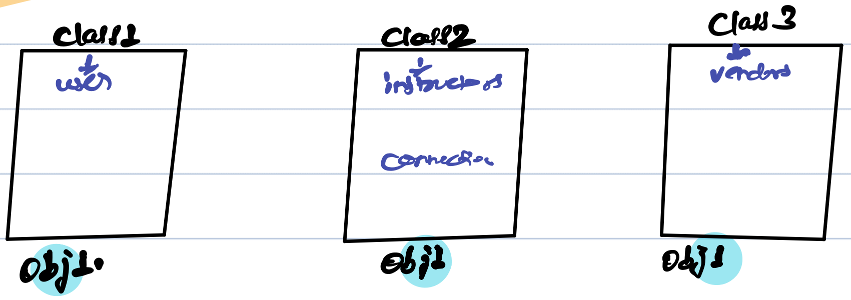
(i) They follow all the Principles of designing.

(ii) Common language for all software engineers.

★ Singleton design Pattern

↳ Allows us to create a single object of class.

Database connection



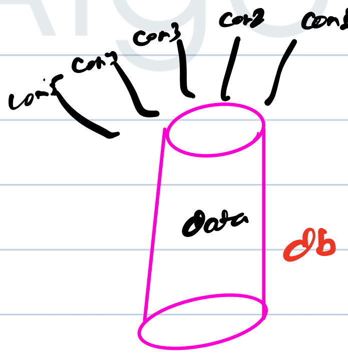
Class dbc{
save (message);
}

→ dbc obj1 = new dbc();

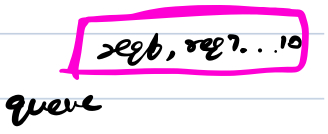
3 way handshake

creating multiple object is expensive as well.

↳ It should have 1 obj only.



10 requests allowed 5



→ logger
→ config files

→ when Singleton design Pattern:

1. when we have a Shared resource behind the Scene, It make sense to have a Single Source of truth for that resource.
   ↓
   one object

2. When creating obj is expensive.

3. When a class has only methods.

* How to implement Singleton design Pattern ?

```
// follow SDP
class dbc {
    String url;
    String Psword;

    void Save ();
}
```

```
→ dbc db1 = new dbc();
→ dbc db2 = new dbc();
```

↳ if we have access of constructor, class can't follow Singleton design Pattern.

```
// follow SDP
class dbc {
    String url;
    String Psword;

    Private dbc () {
    }
    void Save ();
}
```

dbc obj1 = ~~new~~ dbc();

→ Static Keywood??

// follow SOP

```
class dbc {
        :
        ,
        ,
    static void save ();
     static void - - -  ;
                 :
}
```

dbc.save();

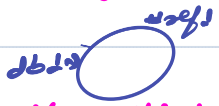
↳ more static method you have used in codebase, the more loadtime it will have.

Break till 10:15 Pm

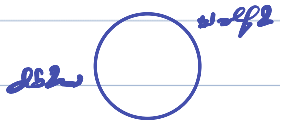
```
class dbc {
    String url;
    String Pswrd;
    Private dbc() {
    }
    void Save();

    Public static dbc getInstance() {
        dbc db = new dbc();
        return db;
    }
}
```

dbc db1 = dbc.getInstance();



dbc db2 = dbc.getInstance();



```
class dbc {
Private Static dbc db = null;
    String Pswrd;
    Private dbc() {
    }
    void Save();

    Public static dbc getInstance() {
        if (db == null) {
            db = new db();
        }
        return db;
    }
}
```

dbc db1 = dbc.getInstance();
ref1

dbc db2 = dbc.getInstance();
ref1

Steps:

1. make Constructor Private
2. Create a Static getInstance method.
3. Create a Private Static reference of the class to hold the Object.

→ Above Solⁿ won't work in multithreaded env.

```
class dbc {
Private static dbC db = null;

    String Pswords;
    Private dbc() {
    }
    void Save();

            static
    Public  ^  dbC getInstance() {
                 if (db = = null) {
       T1          db = new db();
                 }
       T2 →      return db;
    }
}
```

```
dbc db1 = dbC.getInstance();
     ↓
    ref1

dbc db2 = dbC.getInstance();
     ↓
    ref2
```

→ Early initialization

```
class dbc {
private static dbc db =  new dbc();
      String Pswd;
     |Private dbc() {
     |}
      void Save();

       |Public ^static dbc getInstance() {
       |
       |        return db;
       |}
}
```

→ solⁿ is as good/bad as using static methods

→ Lazy initialization

Sol:

```
class dbc {                         'ref'
Private Static dbC db = null;

    String Psword;
    Private dbc() {
    }
    void Save();

     Public  synchronized  Static dbC getInstance() {   lock()
                 if (db = = null) {
                     db = new db();
                 }
     }           return db;
                                                    unlock();
}
```

↳ performance is going to be super slow.

dbc db1 = dbC.getInstance();
'ref'

dbc db2 = dbC.getInstance();
'ref'

Soln 2

```
Public static dbC getInstance () {
    lock ();
    if (db = = null) {
        db = new db ();
    }
    unlock ();
    return db;
}
```

↳ Same as Sol^n in terms of Performance.

Soln 3

```
Public static dbC getInstance () {
    if (db = = null) {
        lock ();
        db = new db ();
        unlock ();
    }
    return db;   T1   T2
}
```

T1 → db1 = #ref1

T2 → db2 = #ref2

↳ incorrect Sol^n

Soln 4

```
Public static dbC getInstance () {
    if (db = = null) {
        lock ();
        if (db = = null) {
            db = new db ();
        }
        unlock ();
    }
    return db;
}
```

Soln 3

db = #ef6

5

```
          synchronized
Public ^   static dbc getInstance() {   lock()
               if (db == null) {
                    db = new db();
               }
           return db;
}
                                         unlock();
```

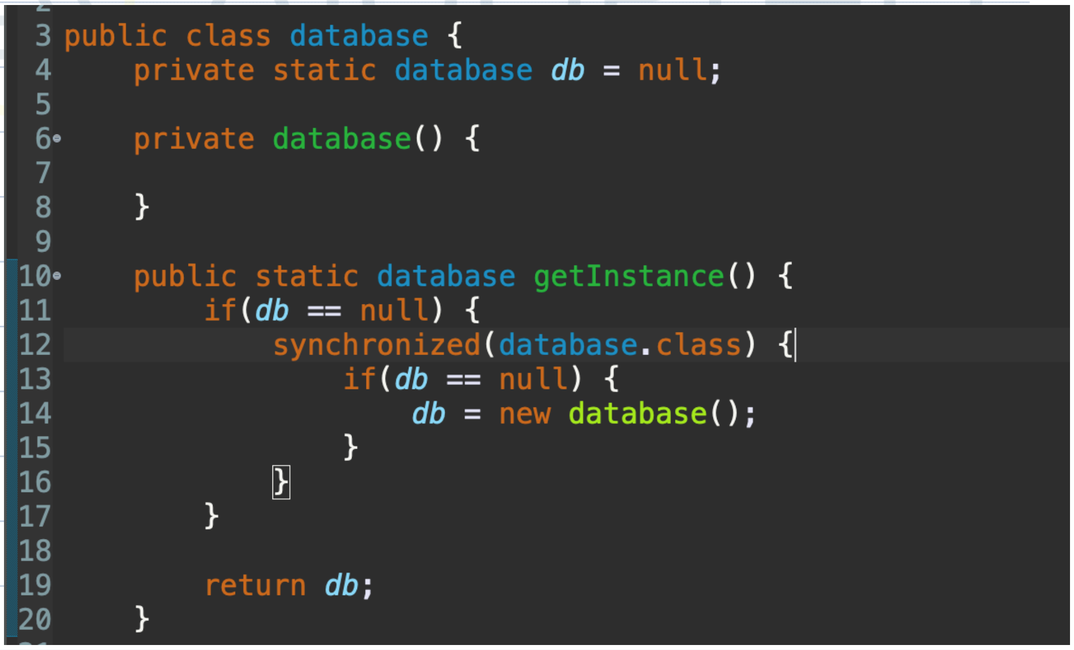
<<<

Soln 4 → Final Soln

db1 = #9ef4

```
Public  static dbc getInstance() {
        if (db == null) {
              lock();
        T1    if (db == null) {
                  db = new db();
              }
              unlock();
        }
        return db;
      S
}
```

Final Code:



```
public class database {
    private static database db = null;

    private database() {

    }

    public static database getInstance() {
        if(db == null) {
            synchronized(database.class) {
                if(db == null) {
                    db = new database();
                }
            }
        }

        return db;
    }
```