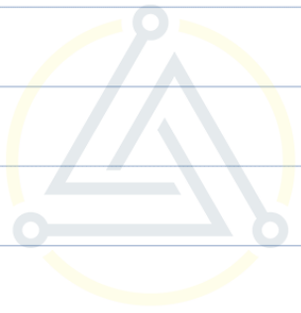
Today's agenda

- Design Pattern Intro
- Singleton design Pattern



* what is design Pattern?
  - software design
  - something 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: (→ OOPS)

(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?

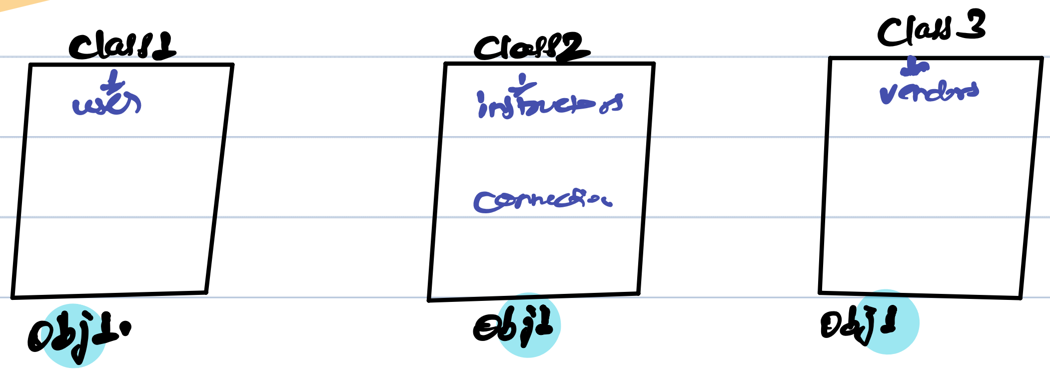
(i) They follow all the Principles of designing.

(ii) Common language for all Software engineers.

* Singleton design Pattern

↳ Allows us to create a single object of class.

Database connection



class dbcl.

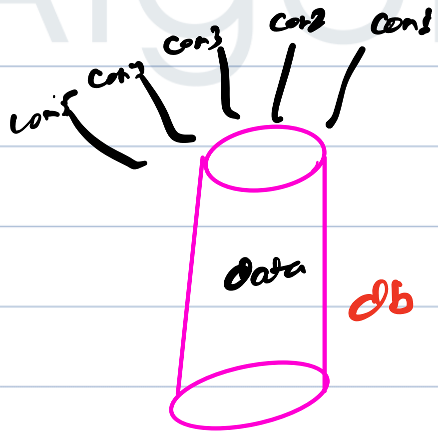
save(msg);

→ dbc obj1 = new dbc();

3 way handshake

creating multiple object is expensive as well.

↳ It should have 1 obj only.



10 requests
allowed 5

req6, req7...10

queue

→ 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 had only methods.

* How to implement Singleton design Pattern ?

```
// follow SOP
class dbc {
    String user;
    String Pswrd;

    void save();
}
```

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

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

```
// follow SOP
class dbc {
    String user;
    String Pswrd;

    private dbc() {
    }
    void save();
}
```

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

→ Static Keywood?.?

// follow SOP

class dbc {

:

:

:

static void save ();

static void ---- ;

dbc.save();

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

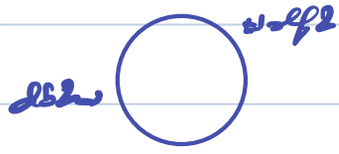
Break till 10:15 Pm

```
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 dbc();
        }
        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 Pswrd;
    |Private dbc() {
    |}
    void save();

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

T1 → T2

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

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

→ Early initialization

```
class dbc {
private static dbC db = new dbc();

    String Pswrd;
    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 Pswrd;
        private dbc() {
        }
        void save();

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

// unlock();

dbc db1 = dbc.getInstance(); // ref

dbc db2 = dbc.getInstance(); // ref

→ Performance is going to be super slow.

Soln 2

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

Is same as soln in terms of performance.

Soln 3

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

T1 → db1 = #ref1

T2 → db2 = #ref2

Is incorrect soln

Soln 4

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

Soln 2

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

Soln 4 → Final Soln

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

lock();

unlock();

Final Code:

```
3  public class database {
4      private static database db = null;
5
6      private database() {
7
8      }
9
10     public static database getInstance() {
11         if(db == null) {
12             synchronized(database.class) {
13                 if(db == null) {
14                     db = new database();
15                 }
16             }
17         }
18
19         return db;
20     }
```