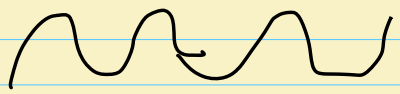
1. Good Evening
2. Lecture begins at 9:10 pm
3. Introduction to Design Patterns.

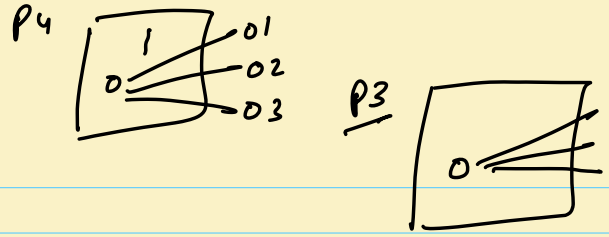
Agenda

1. Introduction to Design Patterns
2. Categories of Design Patterns.
3. Creational Design Patterns
   - Singleton Design Pattern.

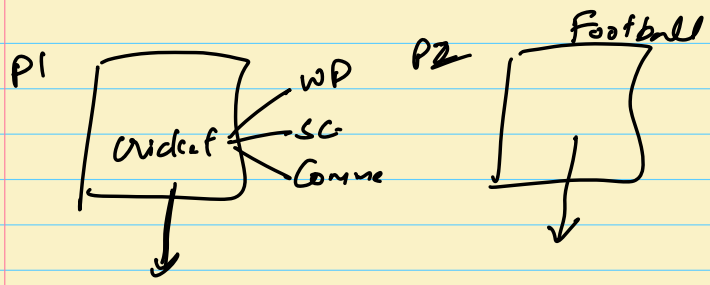
What are Design Patterns?

Pattern → 
- Something repititive

Design → Design of maintainable ... software systems.



Design Pattern



* Well established solutions to common design problem

→ 4 authors : GoF — Gang of Four

→ 23 Design Problems

~ 10 Design Patterns

→ Common patterns in codebase

→ " " asked in interviews.

Solid Principles → 23 problems *

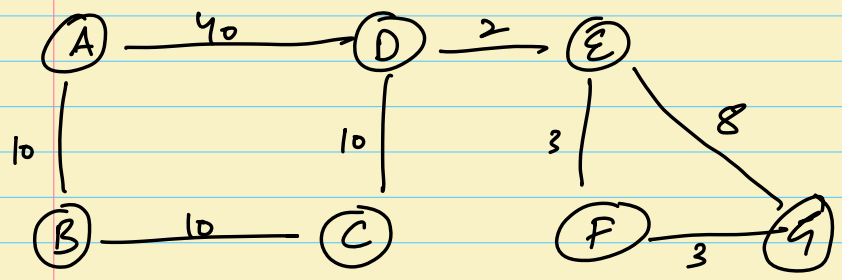
↓ Problem + Solution

Guidelines

Design Pattern.

Why should we learn Design Patterns?

1. Shared Vocabulary? [To save time]



Shortest Path [in kms] A to G

Djikstra → Heap / PQ

Hashmap

30 minutes [60 minutes]

* How to design a class which allows only one object to be created?

* Singleton

2. Interview Questions?

Category of Design Patterns?

- ★ 10 - 15 minutes
- ★ Solution [to me]

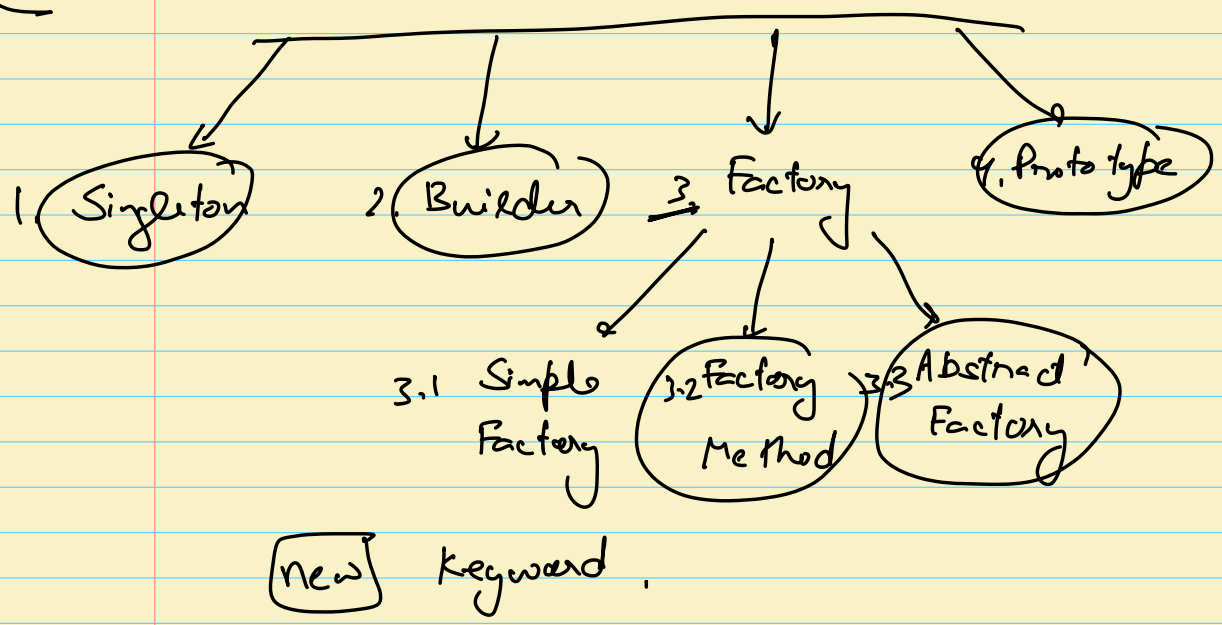
Common problems & Solutions in OOD?

Life Cycle of objects → Created

1. Creational Design Patterns?

   How an object can be created?

   How many objects can be created?

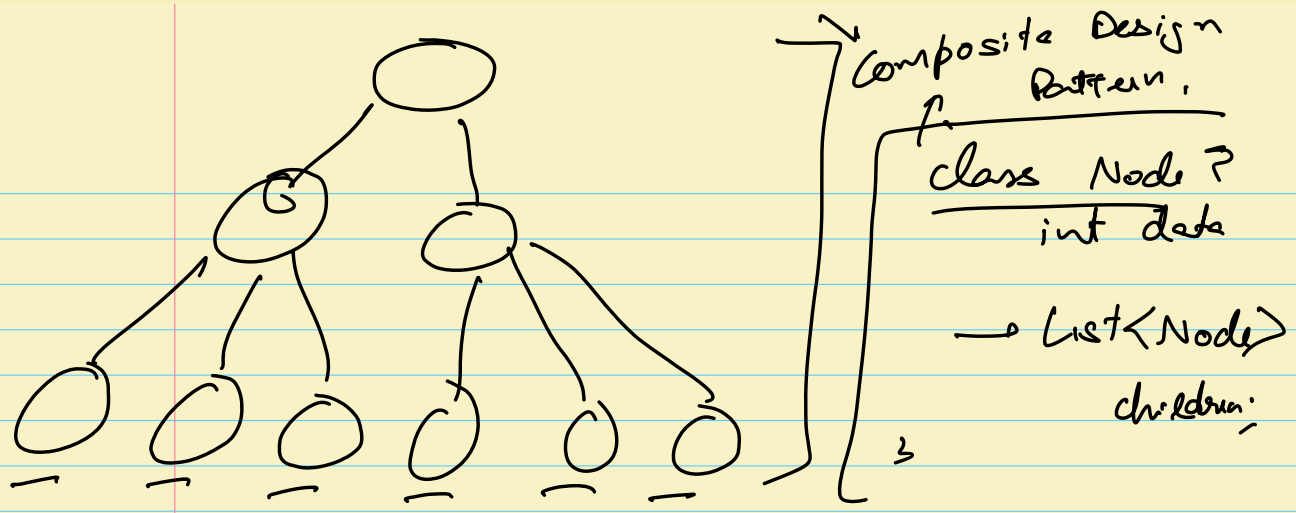


new keyword.

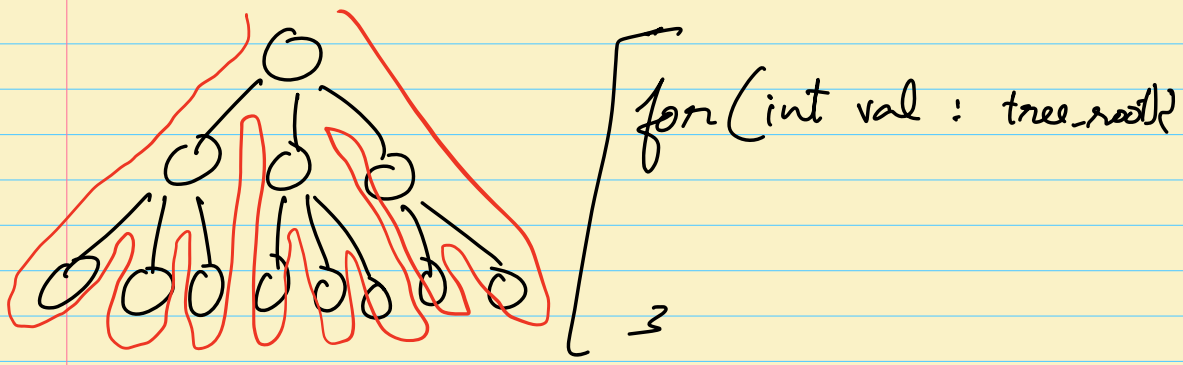
2. Structural Design Patterns.

   → How will one class be composed

   → Attributes



3. Behavioural Design Patterns.
   - To add a behaviour



★ Iterator Design Pattern.

Creational Design Pattern
- Singleton Design Pattern.

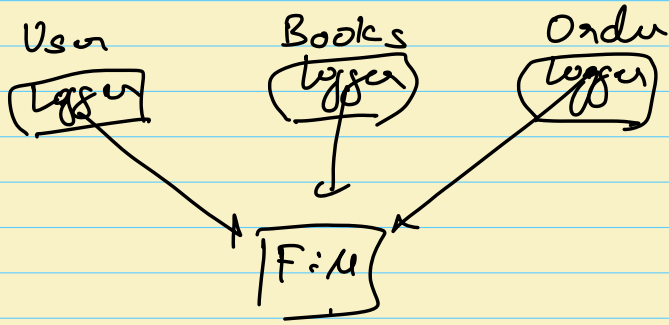
Singleton Design Pattern. [Problem Solution]

- Defn.
- Why? [Problem Statement]
- How? [Implementation]
- Pros
- Cons
- Code

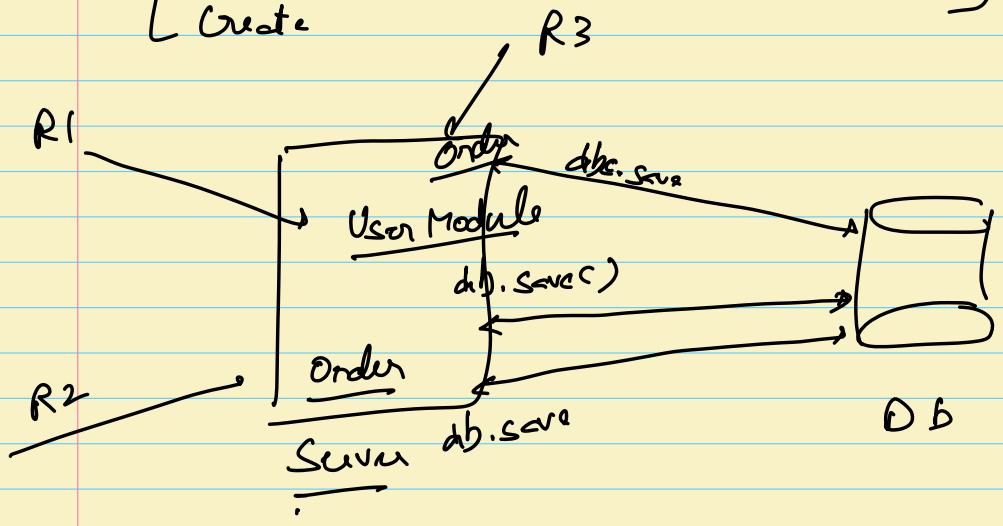
Definition → [Allows you to create a class for which only one object can be created.]

Why? → 1. Shared resources.

e.g. Logger. → only one object.



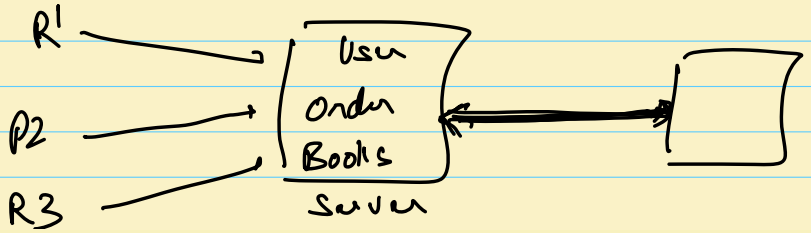
2. [If an object is slow to Create]



* Creating a db connection is Costly
  - db connection → TCP connection.

. class DbConnection ?
  - String url;
  - String userName;
  - String pwd.
  - TCPConnection con] → List<TCPConnection>

3



☆ A costly shared resource.

↓

Single object to be created.

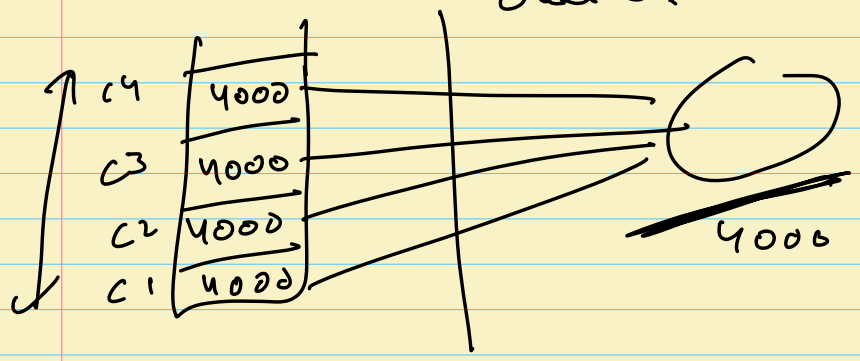
How to implement Singleton?

Version 1

```
class DbConnection {
    pvt. String url;
    pvt. String userName;
    pvt  "     pwd;
    "    TCPConnection conn;

    public DbConnection() {

    }
}
```

```
DbConnection c1 = new DbConnection();
"            c2 = new DbConnection();
```

Problem 1 → multiple instances getting created.



version 1 has a public constructor.

---

version 2

```
class DbConnection {
    // same as version 1

    // removed ctor.
}

DbConnection c1 = new DbConnection()
// c2 = new DbConnection();
```

Default ctor

Problem 2 → ∵ of default constructor, still we are able to create multiple instances

---

Version 3 → Make the ctor private

```
class DbConnection {
    // same as version 1
    private DbConnection() {
    }
}
```

```
DbConnection c1 = new DbConnection();
//            c2 = //          // ();
```

Problem 3 → Not able to create instances at all.

Version 4

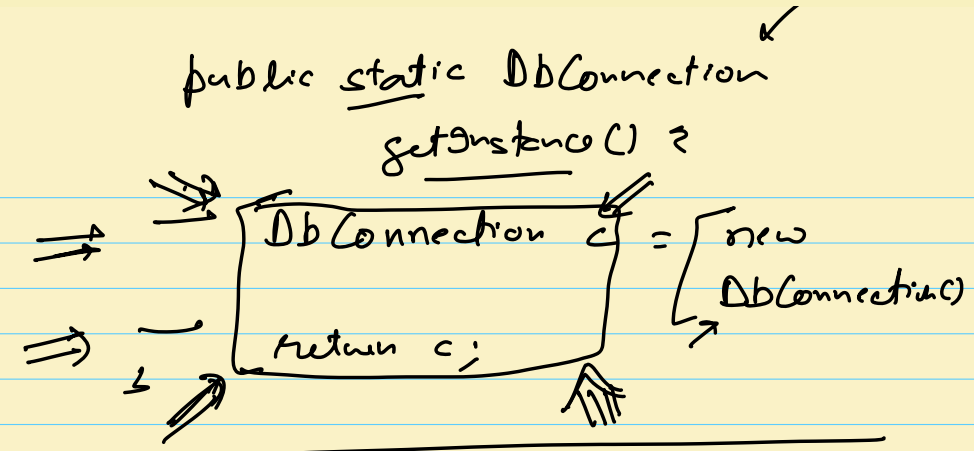
```
class DbConnection {
    // same as version 1
    private DbConnection() {}

    public static DbConnection getInstance() {
        DbConnection c = new DbConnection();
        return c;
    }
}
```

```
DbConnection cl = ?.getInstance();
```

Problem → Not able to call getInstance without having an instance.

Version 5

```
class DbConnection {
    // same as version 1
    private DbConnection() {}
}
```



DbConnection c1 = DbConnection.getInstance()

—— c2 = " . " ();

Problems → multiple instances getting created again

Version 6

✓ → local variable ✓

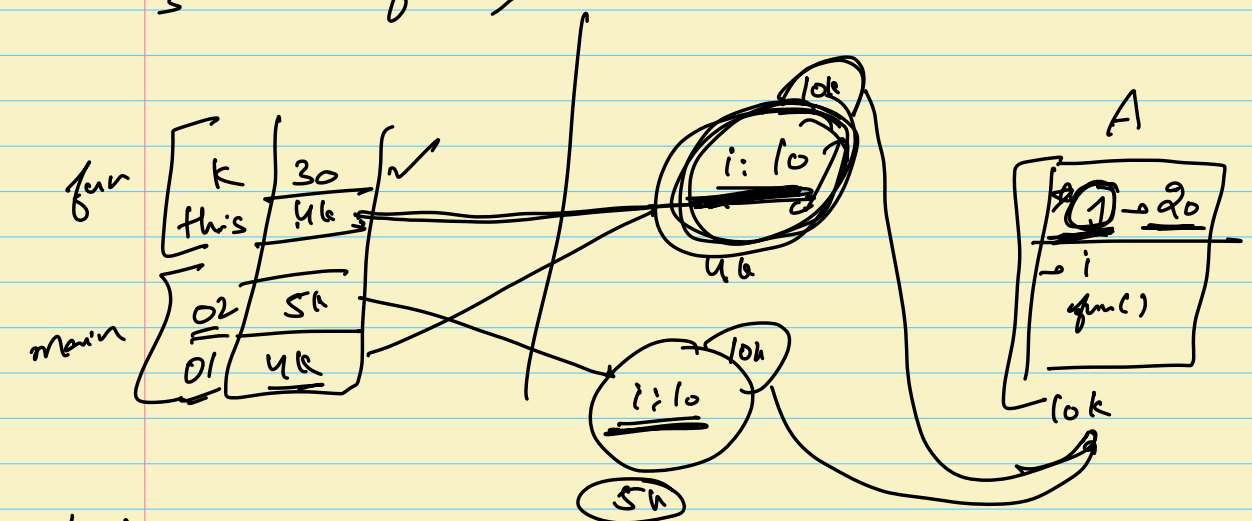
✓ → instance level data member ✓

✓ → class " " " ✓

↓

static

```
class A {
    int i = 10;
    static int j = 20;

    void fun() {
        int i = 30;
        sout(i);
        sout(j);
        sout(F);
    }
}

main() {
    A o1 = new A();
    A o2 = new A();
    o1.fun();
    o2.fun();
}
```



o1.i
o2.i

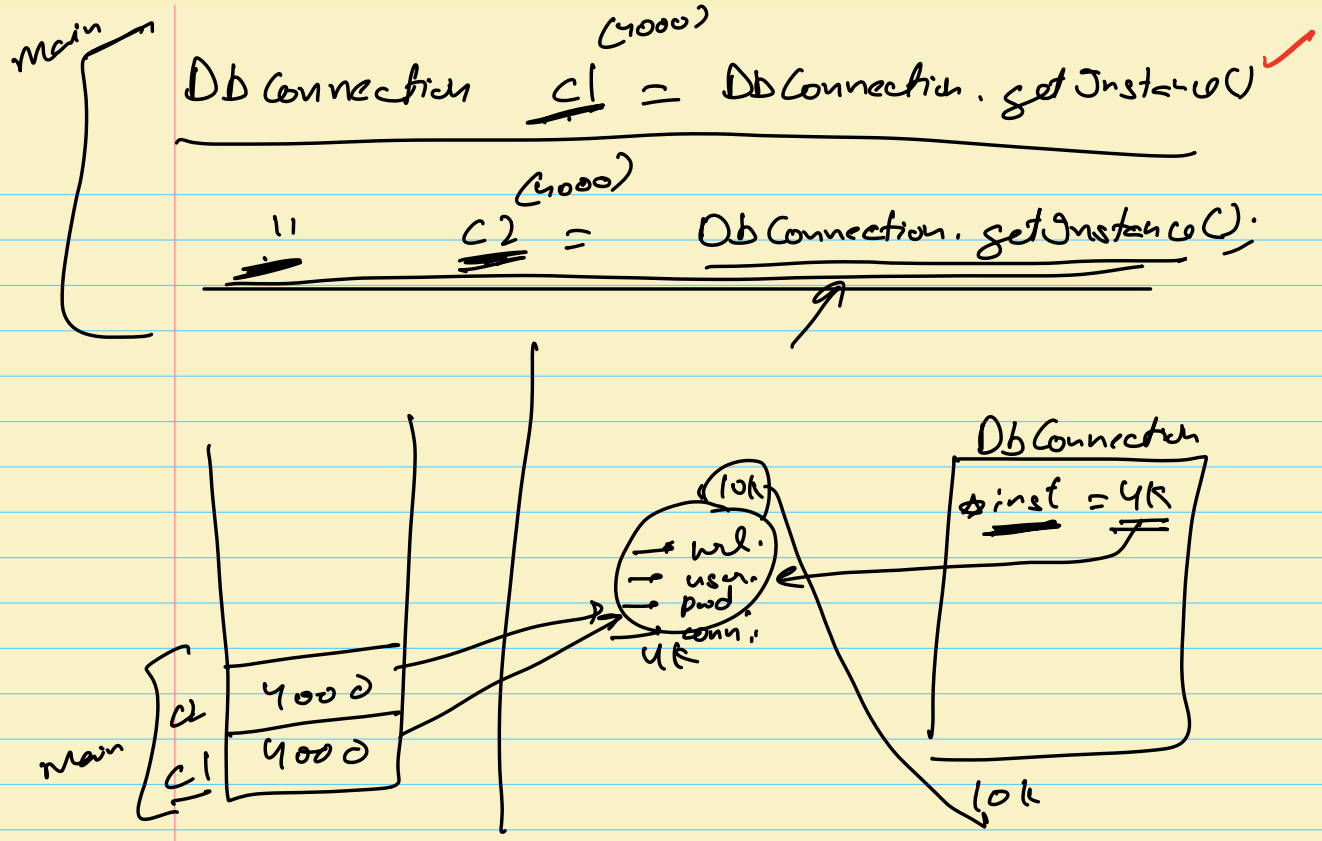
o1.j A.j
o2.j

Version 6

```
class DbConnection {
    // same as version 1
    private DbConnection() {}

    private static DbConnection inst = null
    public static DbConnection getInstance() {
        if (inst == null) {
            inst = new DbConnection();
        }
        return inst;
    }
}
```

1. Can static methods access non-static data members? No

2. Can non-static methods access static data members?



Version 6 has nearly solved the problem

or

completely?

* Is there a situation where version 6 fails?

On multi-threaded application version 6 will fail?

Break → 10:36 to 10:46

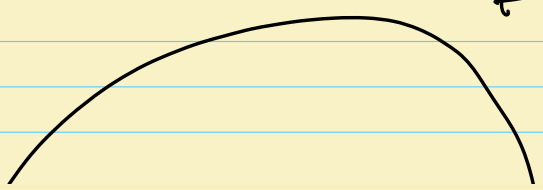
- How multi-threading can break the code?
- How to solve?

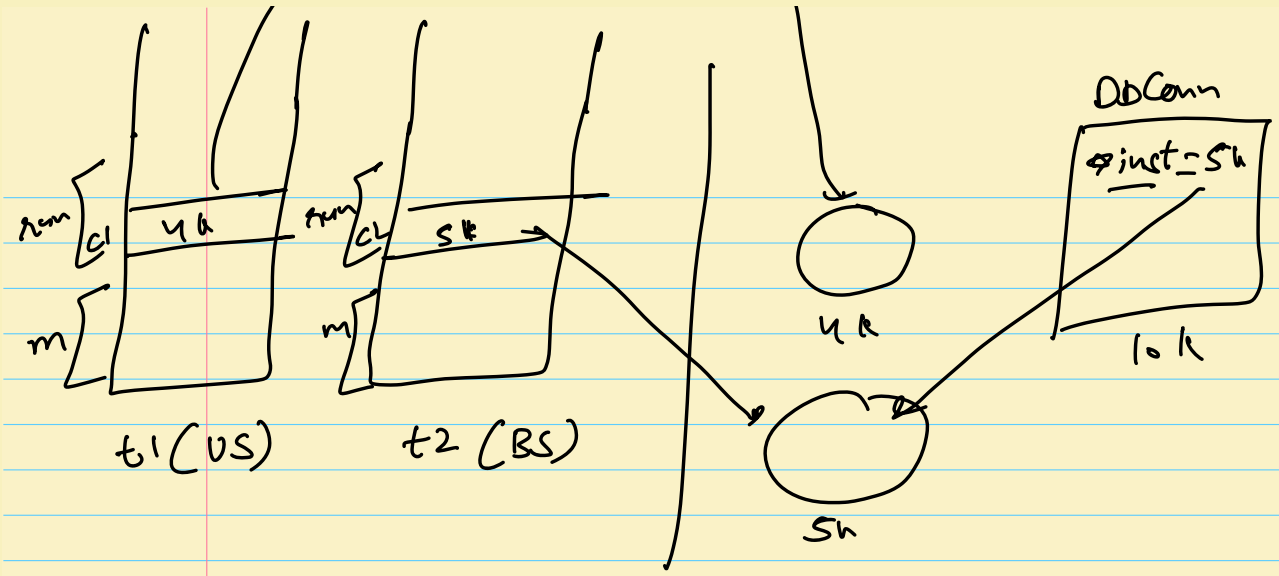
t1 → UserService [Runnable]
run() ?
DBconn c1 = DBC.getC()
}

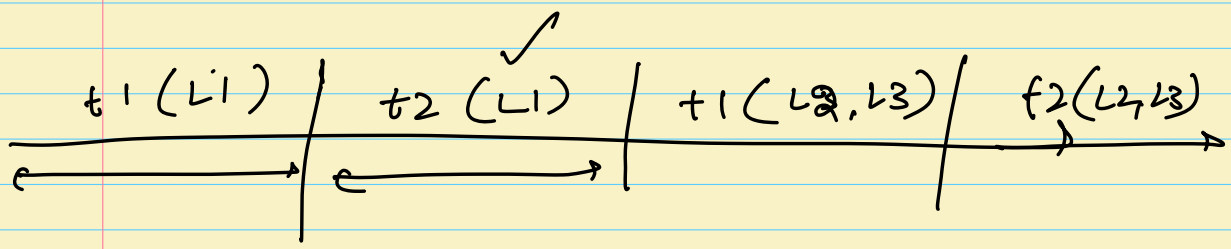
t2 → BookService [Runnable]
run() ?
DBcon c2 = DBC.getC()
}

class DbConnection {
// variables
// private cton
pvt. static DbConnection inst = null
public static ' getInstance()
L1 if (inst == null) {
L2 inst = new DBConn ();
}
L3 return inst;
}

t1(L1) / t2(L1) / t1(L2) / t2(L2)
t / t / ✓ / ✓







☆ Thread Switching is controlled by OS.

→ [Problem of Synchronization]

- A shared data
- A critical section
- Race condition

☆ Solution of Synchronization

- lock
- synchronized

Lock or synchronized

- Allow only 1 thread in the critical section

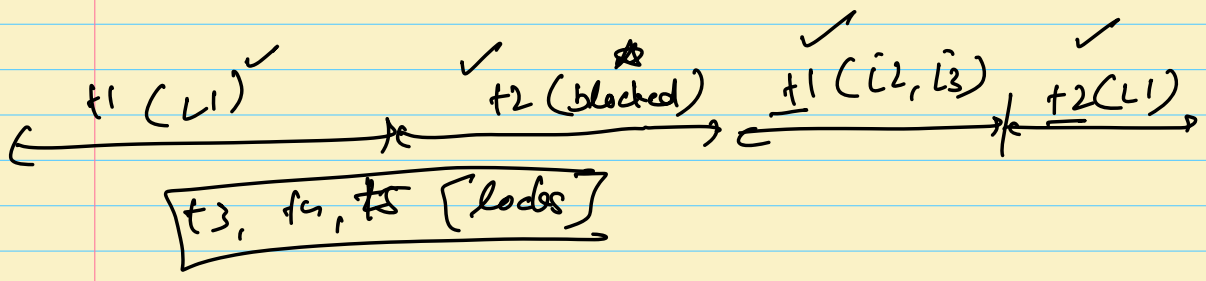
☆ [If 1 thread is in CS, the other thread trying to enter CS blocks]

Version 7

- Everything remain same as Version 6
- → Just add synchronized keyword to getInstance

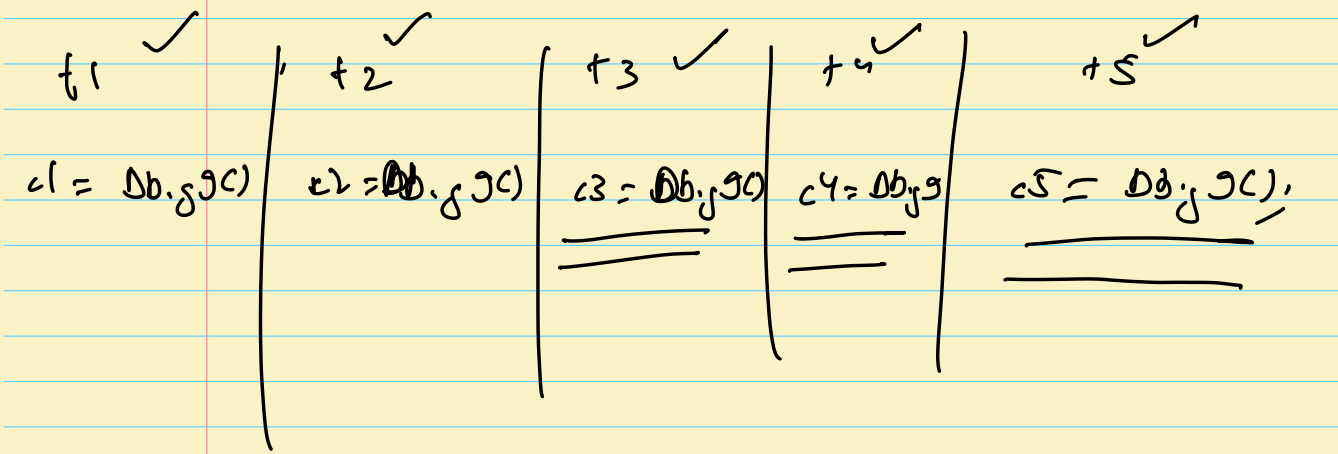
7.1

```
class DbConnection {
// same as version 6
    public synchronized static getIns() {
L1      if (inst == null)
L2          inst = new DbCon();
L3      return inst;
    }
}
```



version 7.1 → A good solution, works in concurrent environment also.

Problem 7.1 → { synchronized keywords adds performance bottleneck



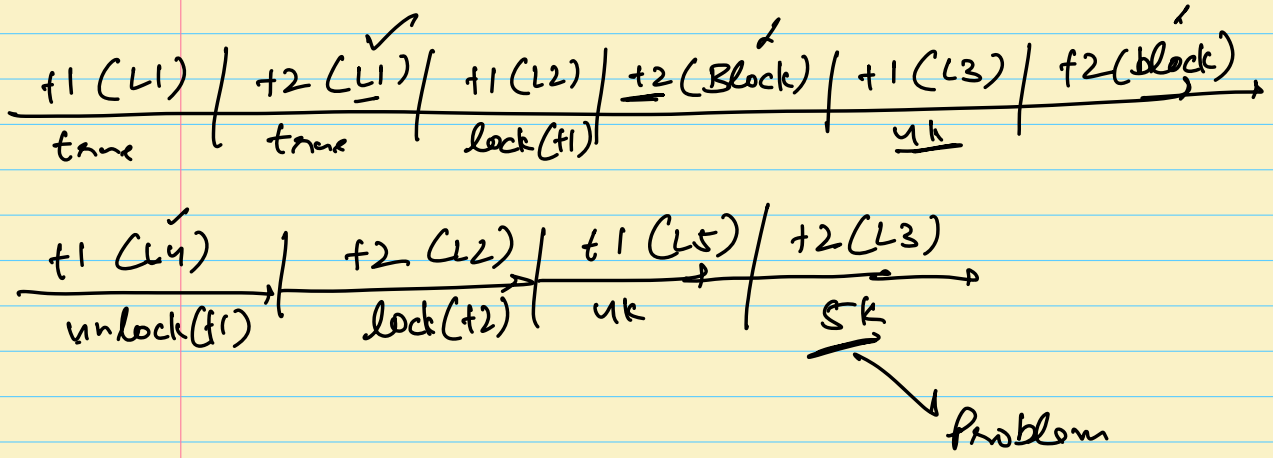
If inst is initialized, after that should we still keep threads blocking while entering the getInstance()

★ → Once inst is initialized, even if multiple threads enter the method there is no harm, infact will be better for performance.

version 7.2

```
class DbConnection {
    // version 6 variables
    // version 6 ctor
    // " " 6 inst

    public static DbConn getInstance() {
L1      if (inst == null) {
L2          lock.acquire()
L3  ✓           inst = new DbConn();
L4          lock.release()
        }
L5  return inst;
    }
```

t1 | t2 | t3



Problem 7.2 → Not singleton, multiple object

Version 7.3

class DbConnection ?

// variable, ctor, inst same as version 6

public static DbConnection getInstance() ?

L1 — if (inst == null) ?

L2 — lock.acquire(); ✓

L3 — if (inst == null)

L4 — inst = new DbConn();

L5 — lock.release();

}

L6 — return inst; ✓

}

t1 t2 t3 t4

t1(L1) | t2(L1) | t1(L2) | t2(block) | t1(L3) | t2(block)
true | true | lock(t1) | L2 | true

t1(L4) | t2(block) | t1(L5) | t2(L2) | t1(L6)
UK | lock(*) | lock(t2) | UK

t2(L3) | Any thread | t2(L4 L5) | Any thread | t2(L6)
false | lock(*) | UK

t3(L1) | t4(L1) | t3(L6) | t4(L6)
false | false | UK | UK

* After the instance is created, no thread even came across a lock.

Version 7.3	vs	Version 7.1
* Threads deal with locks only till instance is not created		Even after instance is created, all threads acquire & release a lock

- Once the instance is created, in version 7.3, the future threads never acquire or releasing a lock.
  - Does everything as version 7.1
  - & has better

- version 7.3 is Dual check locking

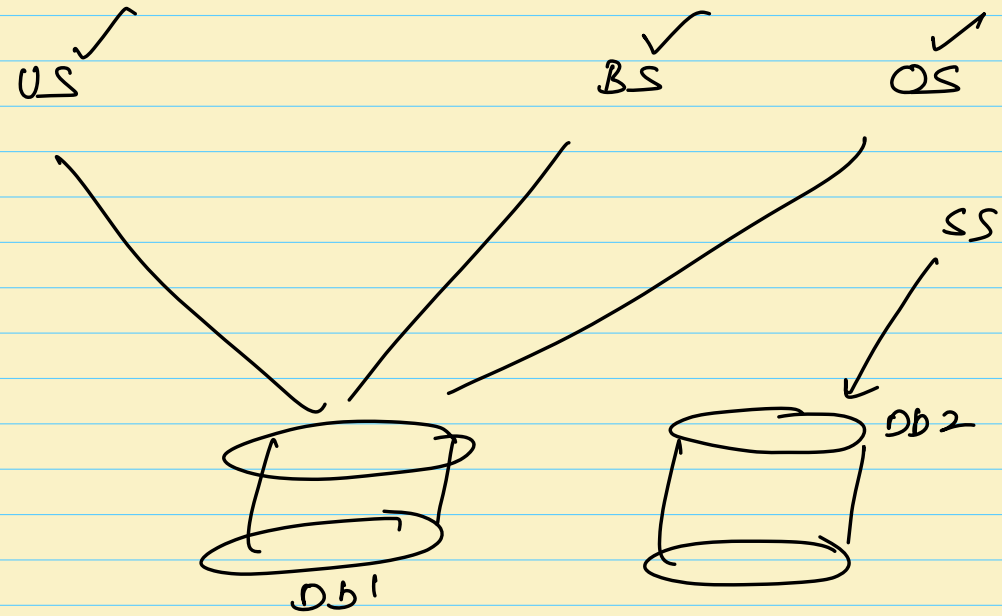
- Pros → Resource saving when there is a costly object creation for a shared resource.

- Cons → If the properties can mutate.

Use Singleton when the objects are immutable.

DbConnection ?

String url ;



- Code
- Interview Questions