**Multithreading – Part-08 – Synchronization part-02**

* **Case Study:**

class SynchronizedDemo{

public static void main(String[] args){

Display d1 = new Display();

Display d2 = new Display();

MyThread t1 = new MyThread(d1, “Dhoni”);

MyThread t2 = new MyThread(d2, “Yuvraj”);

t1.start();

t2.start();

}

}

(d1) 🡪 t1 🡪 lock(d1) (d2) 🡪 t2 🡪 lock(d2)

t1.wish(“Dhoni”); t2.wish(“Yuvraj”);

Note:

In this case even though wish method is synchronized we will get irregular output. Because threads are operating on different Java objects.

Conclusion:

If multiple threads are operating on same Java object, then synchronization is required.

If multiple threads are operating on multiple Java objects then synchronization is not require.

In simple words:

3 dogs one briyani plate, then we need synchronization.

3 dogs 3 briynai plate, then we don’t need synchronization.

* **Class Leve Lock**

Every class in Java has a unique lock which is nothing but class level lock.

If a thread wants to execute a static synchronized method then thread requires class level lock.

Once a thread got class level lock then it is allowed to execute any static synchronized method of that class.

Once method execution completes automatically thread releases the lock.

While a thread executing static synchronized method the remaining threads are not allowed to execute any static synchronized of that class simultaneously. But remaining threads are allowed to execute the following methods simultaneously.

Normal static methods.

Synchronized instance methods.

Normal instance methods.

Example:

class X{

static synchronized m1();

static synchronized m2();

static m3();

synch m4();

void m5();

}

(Object X)

t1 🡪 CL(X) and executing method m1.

Same time t2 wants to execute m1(), but the lock is already acquired by t1, t2 has to wait.

Same time t3 wants to execute m2(), but the lock is already acquired by t1, t3 has to wait.

Same time t4 wants to execute m3(), since m3 is a normal static method it will get a chance.

Same time t5 wants to execute m4 since it’s a normal synchronized method it needs an object level lock so it will get a chance.

Same time t6 wants to execute m5() since it is a normal method it will get a chance.

* **Example:**

class Display{

public synchronized void display(){

for(int i = 0; I < 10; i++){

System.out.print(i)

try{

Thread.sleep(2000);

} catch(InterruptedException ie){

}

}

}

public synchronized void displayc(){

for(int i = 65; i <= 75; i++){

System.out.print((char)i)

try{

Thread.sleep(2000);

} catch(InterruptedException ie){

}

}

}

}

class MyThread1 extends Thread{

Display d;

MyThread1(Display d){

this.d = d;

}

public void run(){

d.displayn();

}

}

class MyThread2 extends Thread{

Display d;

MyThread2(Display d){

this.d = d;

}

public void run(){

d.displayc();

}

}

class SynchronizeDemo{

public static void main(String[] args){

Display d = new Display();

MyThread1 t1= new MyThread1();

MyThread2 t2 = new MyThread2();

t1.start();

t2.start();

}

}

Output:

12345678910ABCDEFGHIJ