Synchronization in Java

- Synchronization in java is the capability *to control the access of multiple threads to any shared resource*.
- Java Synchronization is better option where we want to allow only one thread to access the shared resource.
- Synchronized keyworld allows one thread at a time to access a method, variable or an object such that making other threads to wait until one finishes its work or get terminated.

Why use Synchronization

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
    1. Synchronized method.
    2. Synchronized block.
2. Co-operation (Inter-thread communication in java)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

1. by synchronized method
2. by synchronized block

Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has a lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

```
Class Table{

void printTable(int n){//method not synchronized
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }
   }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}
  }
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);  }  }

class TestSynchronization{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Java synchronized method

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

```
//example of java synchronized method
class Table{
 synchronized void printTable(int n){//synchronized method
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }

 }
}




class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

public class TestSynchronization{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Example of synchronized method by using annonymous class

```
//Program of synchronized method by using annonymous class
class Table{
 synchronized void printTable(int n){//synchronized method
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }

 }
}

public class TestSynchronization{
public static void main(String args[]){
final Table obj = new Table();//only one object

Thread t1=new Thread(){
public void run(){
obj.printTable(5);
}
};
Thread t2=new Thread(){
public void run(){
obj.printTable(100);
}
};

t1.start();
t2.start();
}
}
```

Synchronized block in java

- Synchronized block can be used to perform synchronization on any specific resource of the method.
- Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.
- If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

Points to remember for Synchronized block

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.

Syntax to use synchronized block

```
synchronized (object reference expression) {
  //code block
}
```

Example of synchronized block

```
class Table{

 void printTable(int n){
   synchronized(this){//synchronized block
     for(int i=1;i<=5;i++){
      System.out.println(n*i);
      try{
       Thread.sleep(400);
      }catch(Exception e){System.out.println(e);}
     }
   }
 }//end of the method
}




class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
```

```
this.t=t;
}
public void run(){
t.printTable(100);
}
}

public class TestSynchronizedBlock{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Example of synchronized block by using annonymous class:

```
class Table{

void printTable(int n){
   synchronized(this){//synchronized block
     for(int i=1;i<=5;i++){
      System.out.println(n*i);
      try{
       Thread.sleep(400);
      }catch(Exception e){System.out.println(e);}
     }
   }
}//end of the method
}

public class TestSynchronizedBlock2{
public static void main(String args[]){
final Table obj = new Table();//only one object

Thread t1=new Thread(){
public void run(){
obj.printTable(5);
}
};
Thread t2=new Thread(){
public void run(){
obj.printTable(100);
}
};

t1.start();
t2.start();
}
}
```

Inter-thread communication in Java

1. **Inter-thread communication** or **Co-operation** is all about allowing synchronized threads to communicate with each other.
2. Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of **Object class**:

   - wait()
   - notify()
   - notifyAll()

1) wait() method

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

2) notify() method

Wakes up a single thread that is waiting on this object's monitor.

3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor.

Understanding the process of inter-thread communication

1. Threads enter to acquire lock.
2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.

Why wait(), notify() and notifyAll() methods are defined in Object class not Thread class?

It is because they are related to lock and object has a lock.

```
class Customer{
int amount=10000;
synchronized void withdraw(int amount){
System.out.println("going to withdraw...");

if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{wait();}catch(Exception e){}
}
this.amount-=amount;
System.out.println("withdraw completed...");
}

synchronized void deposit(int amount){
System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();
}
}

class Test{
public static void main(String args[]){
final Customer c=new Customer();
new Thread(){
public void run(){c.withdraw(15000);}
}.start();
new Thread(){
public void run(){c.deposit(10000);}
}.start();
  }}
```

Implementing the Runnable Interface

The Runnable interface declares the run() method that is required for implementing threads in our programs. TO do this, we must perform the steps listed below:

1. Declare the class as implementing the Runnable interface.
2. Implement the run() method.
3. Create a thread by defining an object that is instantiated from this "runnable" class as the target of the thread.
4. Call the thread's start() method to run the thread.

Eg:

```
class X implements Runnable
{

public void run()
{
int i;
for(i=1;i<=5;i++)
        {
        System.out.println("\t ThreadX:"+i);
        System.out.println("End of the ThreadX");
        }
}

}


class DemoRunnable
{

public static void main(String args[])
{
X runnable=new X();
Thread t=new Thread(runnable);
t.start();

}

}
```

Thread	**Runnable**
1. Thread is a class. It is used to create a thread.	**1. Runnable is a functional interface which is used to create a thread.**
2. It has multiple methods including start() and run().	**2. It has only abstract method i.e run().**
3. Each thread creates object and gets associated with it.	**3. Multiple threads share the same object.**
4. More memory required	**4. Less memory required.**
5. Multiple inheritance is not allowed in Java hence after a class extends Thread class, it cannot extends other class	**5. If a class implementing the runnable interface then your class can extend another class.**

Race Condition in Java Multi-Threading

Race condition in Java occurs in a multi-threaded environment when more than one thread try to access a shared resource (modify, write) at the same time. Since multiple threads try to race each other to finish executing a method thus the name race condition.

- Using synchronization to avoid race condition in Java

.