Experiment No._____ **Date___/___/2020**

TITLE OF EXPERIMENT: - A program in Java to multiple threads

DIVISION:_____ **BRANCH: _____**

BATCH:_____ **ROLL NO.: _____**

PERFORMED ON DATE: _____

SIGNATURE OF TEACHING STAFF:

EXPERIMENT NO. 9

Aim: Write a program to create multiple threads and demonstrate how two threads communicate with each other.

Objective: To learn multiple threads in Java

Software:

1. Eclipse
2. JDK 16

Theory:

Multi-threaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results.

So it needs to be made sure by some synchronization method that only one thread can access the resource at a given point of time.

Java provides a way of creating threads and synchronizing their task by using synchronized blocks.

Java Synchronization

Synchronization is a process of handling resource accessibility by multiple thread requests. The main purpose of synchronization is to avoid thread interference. At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called synchronization. The synchronization keyword in java creates a block of code referred to as critical section.

General Syntax:

```
synchronized (object)

{
```

```
  //statement to be synchronized

}
```

Every Java object with a critical section of code gets a lock associated with the object. To enter critical section a thread need to obtain the corresponding object's lock.

Why we need Syncronization?

If we do not use syncronization, and let two or more threads access a shared resource at the same time, it will lead to distorted results.

Consider an example, Suppose we have two different threads **T1** and **T2**, T1 starts execution and save certain values in a file *temporary.txt* which will be used to calculate some result when T1 returns. Meanwhile, T2 starts and before T1 returns, T2 change the values saved by T1 in the file temporary.txt (temporary.txt is the shared resource). Now obviously T1 will return wrong result.

To prevent such problems, synchronization was introduced. With synchronization in above case, once T1 starts using *temporary.txt* file, this file will be **locked**(LOCK mode), and no other thread will be able to access or modify it until T1 returns.

The synchronization is mainly used to

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

Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Thread Synchronization

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

1. Mutual Exclusive
    1. Synchronized method.
    2. Synchronized block.

3. Static synchronization.

2. Cooperation (Inter-thread communication in java)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. It can be achieved by using the following three ways:

1. By Using Synchronized Method
2. By Using Synchronized Block
3. By Using Static Synchronization

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.

Using Synchronized Methods

Using Synchronized methods is a way to accomplish synchronization.

If no Synchronization

Suppose, we are not using synchronization and creating multiple threads that are accessing display method and produce the random output.

Synchronizes Keyword

To synchronize program, we must *synchronize* access to the shared method, making it available to only one thread at a time. This is done by using keyword **synchronized** with shared method.

```
synchronized void display (String msg)
```

1. 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.

2. Synchronized Block in Java

Synchronized block can be used to perform synchronization on any specific resource of the method.

Suppose we have 50 lines of code in our method, but we want to synchronize only 5 lines, in such cases, we can use synchronized block.

If we put all the codes of the method in the synchronized block, it will work same as the synchronized method.

Points to Remember

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.
- A Java synchronized block doesn't allow more than one JVM, to provide access control to a shared resource.
- The system performance may degrade because of the slower working of synchronized keyword.
- Java synchronized block is more efficient than Java synchronized method.

Syntax

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

Difference between synchronized keyword and synchronized block

When we use synchronized keyword with a method, it acquires a lock in the object for the whole method. It means that no other thread can use any synchronized method until the current thread, which has invoked it's synchronized method, has finished its execution.

synchronized block acquires a lock in the object only between parentheses after the synchronized keyword. This means that no other thread can acquire a lock on the locked object until the synchronized block exits. But other threads can access the rest of the code of the method.

Which is more preferred - Synchronized method or Synchronized block?

In Java, synchronized keyword causes a performance cost. A synchronized method in Java is very slow and can degrade performance. So we must use synchronization keyword in java when it is necessary else, we should use Java synchronized block that is used for synchronizing critical section only.

Inter-thread Communication in Java

Inter-thread communication or **Co-operation** is all about allowing synchronized threads to communicate with each other.

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

The 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.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Method	Description
public final void wait()throws InterruptedException	It waits until object is notified.
public final void wait(long timeout)throws InterruptedException	It waits for the specified amount of time.

2) notify() method

The notify() method wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

Syntax:

1. **public final void** notify()

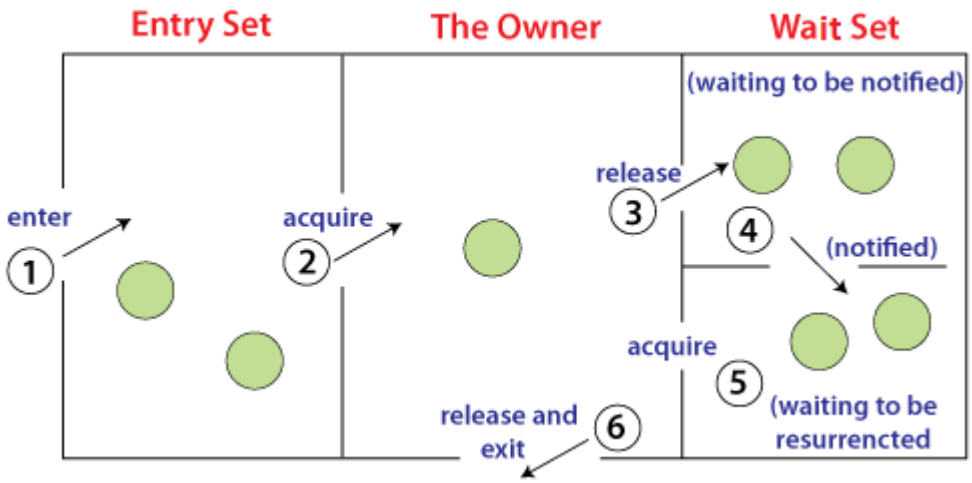
3) notifyAll() method

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

Syntax:

```
public final void notifyAll()
```

Understanding the process of inter-thread communication:



The point to point explanation of the above diagram is as follows:

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.

Difference between wait and sleep:

Let's see the important differences between wait and sleep methods.

wait()	sleep()
The wait() method releases the lock.	The sleep() method doesn't release the lock.
It is a method of Object class	It is a method of Thread class
It is the non-static method	It is the static method
It should be notified by notify() or notifyAll() methods	After the specified amount of time, sleep is completed.

Program:

a. Program for Main Thread

```
public class Lab10 {


    public static void main(String[] args) {


        Thread obj = Thread.currentThread();
       System.out.println("Current thread is " +obj);
       System.out.println("Name of current thread is "
+obj.getName());
      obj.setName("Multi Thread"); // Changing name of main thread.
      System.out.println("Name of current thread after changing name
is " +obj);
       Account accObj= new Account(100);
       Thread t1=  new Thread( new DepositeThread(accObj,30));
       t1.setName("Deposite Thread");
        System.out.println("Name of current thread is "
+t1.getName());
       t1.start();
        Thread t2= new Thread( new DepositeThread(accObj,10));
                t2.start();
         Thread t3=new Thread( new WithdrawThread(accObj,150));
         t3.start();
         Thread t4=new Thread( new DepositeThread(accObj,20));
         t4.start();
         new Thread( new DepositeThread(accObj,30)).start();
```

```
        new Thread( new WithdrawThread(accObj,30)).start();



    }

}
```

b. Program for Account Thread

```
public class Account {
    private double balance=0;

    public Account(double balance)
    {
        this.balance=balance;
    }
    public synchronized void deposit(double amount)
    {
        if (amount<0)
        {
            System.out.println("can't be deposite");
        }
        this.balance+=amount;
        System.out.println("Deposite :"+amount+"in
thread"+Thread.currentThread().getId()+" Balance is :"+balance);
    }

    public synchronized void withdraw(double amount)
    {
        if (amount<0|| amount>this.balance)
        {
            System.out.println("can't be withdraw");
        }
        this.balance-=amount;
        System.out.println("Withdraw:"+amount+"in
thread"+Thread.currentThread().getId()+" Balance is :"+balance);



    }
}
```

c. Program for Deposit Thread

```
public class DepositeThread extends Thread{
    private Account account;
    private double amount;

    public DepositeThread(Account account, double amount) {
        this.account = account;
        this.amount = amount;
    }
    public void run()
    {
        account.deposit(amount);
    }

}
```

d. Program for Withdraw Thread

```
public class WithdrawThread implements Runnable{
    private Account account;
    private double amount;

    public WithdrawThread(Account account, double amount) {
        this.account = account;
        this.amount = amount;
    }
    public void run()
    {
        account.withdraw(amount);
    }


}
```

Output:

```
Current thread is Thread[main,5,main]
Name of current thread is main
Name of current thread after changing name is Thread[Multi
Thread,5,main]
Name of current thread is Deposite Thread
```

```
Deposite :30.0in thread15 Balance is :130.0
Withdraw:30.0in thread23 Balance is :100.0
Deposite :30.0in thread22 Balance is :130.0
Deposite :20.0in thread20 Balance is :150.0
Withdraw:150.0in thread18 Balance is :0.0
Deposite :10.0in thread17 Balance is :10.0
```

Conclusion:

Screenshot's of Program and Result: