EXERCISE

1. **Write a program to demonstrate the use of volatile keyword.**

   CODE:

```
public class Q1three
{
    private static volatile boolean running=false;

    public static void main(String[] args) throws Exception
    {
        new Thread(new Runnable() {                                  //new thread
            public void run() {
                while (!running) {                                   //wait
                }
                System.out.println("starting");

                while (running) {                                    //wait
                }
                System.out.println("started");
            }
        }).start();

        Thread.sleep(1000);
        System.out.println("starting");

        running=true;

        Thread.sleep(1000);
        System.out.println("stopping");

        running=false;
    }
}
```

   OUTPUT:

```
^Cpreeti@preeti:~/IdeaProjects/Assessment3/src$ javac Q1three.java
preeti@preeti:~/IdeaProjects/Assessment3/src$ java Q1three
starting
starting
stopping
started
```

2. **Write a program to create a thread using Thread class and Runnable interface each.**

```
import java.lang.*;
class Hello extends Thread
{
    public void run()
    {
        try
        {
            System.out.println("thread"+Thread.currentThread().getId()+" is running");
        }
        catch(Exception e)
        {
            System.out.println("exception is caught");
        }
    }
}
public class Q2three
{
    public static void main(String[] args)
    {
//        int n=5;
        for(int i=0;i<5;i++)
        {
            Hello ob=new Hello();
            ob.start();
        }

    }
}
```

OUTPUT:

```
preeti@preeti:~/IdeaProjects/Assessment3/src$ javac Q2three.java
^[[Apreeti@preeti:~/IdeaProjects/Assessment3/src$ java Q2three
thread10 is running
thread11 is running
thread12 is running
thread13 is running
thread14 is running
```

3. **Write a program using synchronization block and synchronization method**

CODE:

```
class Syncblock extends Thread{
    int n=3;
    public void change() {
        synchronized (this) { // synchronized block
            n++;
        }
        System.out.println(n);
    }
    public void run(){
        change();
    }
}
class Sync{
    synchronized public void Printvalue() // Synchronized method
    {
        for (int i = 0; i < 3; i++)
        {
            System.out.println(i);
            try
            {
                Thread.sleep(100);
            }
            catch (Exception e)
            {
                System.out.println(e);
            }
        }
    }
}
class Sync2 extends Thread{
    Sync sy;

    Sync2(Sync sy)
    {
        this.sy = sy;
    }

    public void run()
    {
        sy.Printvalue();
    }
}
```

```
public class Q4three {
    public static void main(String[] args)
    {
        Sync obj = new Sync();

        Sync2 t1 = new Sync2(obj);
        Sync2 t2 = new Sync2(obj);

        t1.start();
        t2.start();
        try {
            Thread.sleep(100);
        }
        catch(InterruptedException e){}

        Syncblock s1 = new Syncblock();
        Syncblock s2 = new Syncblock();
        s1.start();
        s2.start();
    }
}
```

OUTPUT:

```
preeti@preeti:~/IdeaProjects/Assessment3/src$ javac Q4three.java
preeti@preeti:~/IdeaProjects/Assessment3/src$ java Q4three
0
1
2
0
1
2
```

4. **Write a program to create a Thread pool of 2 threads where one Thread will print even numbers and other will print odd numbers.**

CODE:

```
class OddThread extends Thread
{
    int limit;
    sharedPrinter printer;
    public OddThread(int limit, sharedPrinter printer)
    {
        this.limit = limit;
        this.printer = printer;
    }
    @Override
    public void run()
    {
        int oddNumber = 1;
        while (oddNumber <= limit)
        {
            printer.printOdd(oddNumber);
            oddNumber = oddNumber + 2;
        }
    }
}

class EvenThread extends Thread
{
    int limit;
    sharedPrinter printer;
    public EvenThread(int limit, sharedPrinter printer)
    {
        this.limit = limit;
        this.printer = printer;
    }
    @Override
    public void run()
    {
        int evenNumber = 2;
        while (evenNumber <= limit)
        {
            printer.printEven(evenNumber);
            evenNumber = evenNumber + 2;
        }
    }
}
```

```
class sharedPrinter
{

    boolean isOddPrinted = false;

    synchronized void printOdd(int number)
    {
        while (isOddPrinted)
        {
            try
            {
                wait();
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
        System.out.println(Thread.currentThread().getName()+" : "+number);
        isOddPrinted = true;
        try
        {
            Thread.sleep(1000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        notify();
    }

    synchronized void printEven(int number)
    {
        while (! isOddPrinted)
        {
            try
            {
                wait();
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
        System.out.println(Thread.currentThread().getName()+" : "+number);
```

```
        isOddPrinted = false;
        try
        {
            Thread.sleep(1000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        notify();
    }
}
//Main Class
public class Q3three
{
    public static void main(String[] args)
    {
        sharedPrinter printer = new sharedPrinter();
        OddThread oddThread = new OddThread(10, printer);
        oddThread.setName("Odd-Thread");
        EvenThread evenThread = new EvenThread(10, printer);
        evenThread.setName("Even-Thread");
        oddThread.start();
        evenThread.start();
    }
}
```

OUTPUT:

```
preeti@preeti:~/IdeaProjects/Assessment3/src$ javac Q3three.java
preeti@preeti:~/IdeaProjects/Assessment3/src$ java Q3three
Odd-Thread : 1
Even-Thread : 2
Odd-Thread : 3
Even-Thread : 4
Odd-Thread : 5
Even-Thread : 6
Odd-Thread : 7
Even-Thread : 8
Odd-Thread : 9
Even-Thread : 10
```

5. **Write a program to demonstrate wait and notify methods.**

CODE:

```
class First
{
   public void myProducer() throws InterruptedException {
      synchronized (this)
      {
         System.out.println("By Producer");
         wait();
         System.out.println("The item is consumed and resumed");
      }
   }
   public  void myConsumer() throws InterruptedException {
      synchronized (this)
      {
         Thread.sleep(300);
         System.out.println("By Consumer");
         notify();
      }
   }
}
public class Q5three {
   public static void main(String[] args) {

      First f1=new First();

      Thread t1=new Thread(new Runnable() {
         @Override
         public void run() {

            try {
               f1.myProducer();
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
         }
      });
      Thread t2=new Thread(new Runnable() {
         @Override
         public void run() {

            try {
               f1.myConsumer();
            } catch (InterruptedException e) {
```

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

OUTPUT:

```
^[[A^[[A^Cpreeti@preeti:~/IdeaProjects/Assessment3/src$ javac Q5three.java
preeti@preeti:~/IdeaProjects/Assessment3/src$ java Q5three
By Producer
By Consumer
The item is consumed and resumed
```

6. **Write a program to demonstrate sleep and join methods.**

CODE:

```
class MyThread extends Thread
{
    @Override
    public void run() {
        System.out.println("hey i am executing..."+Thread.currentThread().getName());
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Finished");
    }
}
public class Q6three {
    public static void main(String[] args) {

        MyThread t1= new MyThread();
        MyThread t2=new MyThread();

        t1.start();

        //starts second thread when thread t1 has died
        try{
            t1.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        t2.start();
    }
}
```

OUTPUT:

```
preeti@preeti:~/IdeaProjects/Assessment3/src$ javac Q6three.java
preeti@preeti:~/IdeaProjects/Assessment3/src$ java Q6three
hey i am executing...Thread-0
Finished
hey i am executing...Thread-1
Finished
```

7. **Run a task with the help of callable and store it's result in the Future.**

CODE:

```
import java.util.concurrent.*;
class CallableImpl implements Callable<Integer> {

    private int myName;

    CallableImpl(int i) {
        myName = i;
    }

    @Override
    public Integer call() {
        for (int i = 0; i < 10; i++) {
            System.out.println("Thread : " + getMyName() + " value is : " + i);
        }
        return getMyName();
    }

    public int getMyName() {
        return myName;
    }

    public void setMyName(int myName) {
        this.myName = myName;
    }
}

public class Q7three {

    public static void main(String[] args) throws InterruptedException {

        Callable<Integer> callable = new CallableImpl(2);
        ExecutorService executor = new ScheduledThreadPoolExecutor(1);
        Future<Integer> future = executor.submit(callable);

        try {
            System.out.println("Future value: " + future.get());
        } catch (Exception ignored) {}
        executor.shutdown();
        executor.awaitTermination(1, TimeUnit.HOURS);
    }
}
```

OUTPUT :

```
preeti@preeti:~/IdeaProjects/Assessment3/src$ javac Q7three.java
preeti@preeti:~/IdeaProjects/Assessment3/src$ java Q7three
Thread : 2 value is : 0
Thread : 2 value is : 1
Thread : 2 value is : 2
Thread : 2 value is : 3
Thread : 2 value is : 4
Thread : 2 value is : 5
Thread : 2 value is : 6
Thread : 2 value is : 7
Thread : 2 value is : 8
Thread : 2 value is : 9
Future value: 2
```

8. **Write a program to demonstrate the use of semaphore**

CODE:

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.Semaphore;

class Connection {

    private static Connection instance = new Connection();

    private Semaphore sem = new Semaphore(10, true);
    private int connections = 0;

    private Connection() {
    }

    public static Connection getInstance() {
        return instance;
    }

    public void connect() {
        try {
            sem.acquire();
            doConnect();

        } catch (InterruptedException ignored) {
        } finally {
            sem.release();
        }
    }

    public void doConnect() {
        synchronized (this) {
            connections++;
            System.out.println("Current connections (max 10 allowed): " + connections);
        }
        try {
            System.out.println("Working on connections " + Thread.currentThread().getName());
            Thread.sleep(2000);
        } catch (InterruptedException ignored) {}

        synchronized (this) {
```

```
            connections--;
            System.out.println("I'm done " + Thread.currentThread().getName() + " Connection is
released , connection count: " + connections);
        }
    }
}

class Connect {

    private static Connect instance = new Connect();

    private Semaphore sem = new Semaphore(10, true);

    private Connect() {
    }

    public static Connect getInstance() {
        return instance;
    }

    public void connect() {
        try {
            sem.acquire();

            System.out.printf("%s:: Current connections (max 10 allowed): %d\n",
                    Thread.currentThread().getName(),
                    sem.availablePermits());


            System.out.printf("%s:: WORKING...\n",
                    Thread.currentThread().getName());
            Thread.sleep(2000);

            System.out.printf("%s:: Connection released. Permits Left = %d\n",
                    Thread.currentThread().getName(),
                    sem.availablePermits());

        } catch (InterruptedException ignored) {
        } finally {
            sem.release();
        }
    }
}

public class Q8three {
```

```
    public static void main(String[] args) throws Exception {
        ExecutorService executor = Executors.newCachedThreadPool();

        for (int i = 0; i < 20; i++) { //200 hundred times will be called
            executor.submit(new Runnable() {
                public void run() {
                    Connect.getInstance().connect();
                }
            });
        }

        executor.shutdown();
        executor.awaitTermination(1, TimeUnit.DAYS);
    }
}
```

OUTPUT:

```
preeti@preeti:~/IdeaProjects/Assessment3/src$ javac Q8three.java
preeti@preeti:~/IdeaProjects/Assessment3/src$ java Q8three
pool-1-thread-1:: Current connections (max 10 allowed): 6
pool-1-thread-1:: WORKING...
pool-1-thread-6:: Current connections (max 10 allowed): 4
pool-1-thread-6:: WORKING...
pool-1-thread-9:: Current connections (max 10 allowed): 6
pool-1-thread-9:: WORKING...
pool-1-thread-12:: Current connections (max 10 allowed): 0
pool-1-thread-12:: WORKING...
pool-1-thread-5:: Current connections (max 10 allowed): 0
pool-1-thread-5:: WORKING...
pool-1-thread-2:: Current connections (max 10 allowed): 3
pool-1-thread-2:: WORKING...
pool-1-thread-11:: Current connections (max 10 allowed): 2
pool-1-thread-11:: WORKING...
pool-1-thread-10:: Current connections (max 10 allowed): 6
pool-1-thread-10:: WORKING...
pool-1-thread-8:: Current connections (max 10 allowed): 5
pool-1-thread-8:: WORKING...
pool-1-thread-7:: Current connections (max 10 allowed): 5
pool-1-thread-7:: WORKING...
pool-1-thread-1:: Connection released. Permits Left = 0
pool-1-thread-5:: Connection released. Permits Left = 0
pool-1-thread-12:: Connection released. Permits Left = 0
pool-1-thread-9:: Connection released. Permits Left = 0
pool-1-thread-6:: Connection released. Permits Left = 0
pool-1-thread-14:: Current connections (max 10 allowed): 0
pool-1-thread-14:: WORKING...
```

```
pool-1-thread-13:: Current connections (max 10 allowed): 0
pool-1-thread-13:: WORKING...
pool-1-thread-7:: Connection released. Permits Left = 0
pool-1-thread-3:: Current connections (max 10 allowed): 0
pool-1-thread-3:: WORKING...
pool-1-thread-8:: Connection released. Permits Left = 1
pool-1-thread-10:: Connection released. Permits Left = 0
pool-1-thread-11:: Connection released. Permits Left = 0
pool-1-thread-2:: Connection released. Permits Left = 0
pool-1-thread-4:: Current connections (max 10 allowed): 0
pool-1-thread-4:: WORKING...
pool-1-thread-20:: Current connections (max 10 allowed): 0
pool-1-thread-20:: WORKING...
pool-1-thread-19:: Current connections (max 10 allowed): 0
pool-1-thread-19:: WORKING...
pool-1-thread-18:: Current connections (max 10 allowed): 0
pool-1-thread-18:: WORKING...
pool-1-thread-17:: Current connections (max 10 allowed): 0
pool-1-thread-17:: WORKING...
pool-1-thread-16:: Current connections (max 10 allowed): 0
pool-1-thread-16:: WORKING...
pool-1-thread-15:: Current connections (max 10 allowed): 0
pool-1-thread-15:: WORKING...
pool-1-thread-14:: Connection released. Permits Left = 0
pool-1-thread-13:: Connection released. Permits Left = 1
pool-1-thread-3:: Connection released. Permits Left = 2
pool-1-thread-4:: Connection released. Permits Left = 3
pool-1-thread-20:: Connection released. Permits Left = 4
```

```
pool-1-thread-20:: Connection released. Permits Left = 4
pool-1-thread-19:: Connection released. Permits Left = 5
pool-1-thread-18:: Connection released. Permits Left = 6
pool-1-thread-17:: Connection released. Permits Left = 7
pool-1-thread-16:: Connection released. Permits Left = 8
pool-1-thread-15:: Connection released. Permits Left = 9
```

9. Write a program to demonstrate the use of CountDownLatch

CODE:

```
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

class Processor implements Runnable {

    private CountDownLatch latch;

    public Processor(CountDownLatch latch) {
        this.latch = latch;
    }

    public void run() {
        System.out.println("Started.");

        try {
            Thread.sleep(3000);
        } catch (InterruptedException ignored) {}
        latch.countDown();
    }
}

public class Q9three{

    public static void main(String[] args) {
        CountDownLatch latch = new CountDownLatch(3);
        ExecutorService executor = Executors.newFixedThreadPool(3);
        for (int i = 0; i < 3; i++) {
            executor.submit(new Processor(latch));
        }
        executor.shutdown();

        try {

            latch.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Completed.");
    }}
```

OUTPUT:

```
preeti@preeti:~/IdeaProjects/Assessment3/src$ javac Q9three.java
preeti@preeti:~/IdeaProjects/Assessment3/src$ java Q9three
Started.
Started.
Started.
Completed.
```

10. Write a program which creates deadlock between 2 threads

CODE:

```
public class Q10three {

    public static final Object lock1 = new Object();
    public static final Object lock2 = new Object();
    private int index;

    public static void main(String[] args) {
        Thread t1 = new Thread1();
        Thread t2 = new Thread2();
        t1.start();
        t2.start();
    }

    private static class Thread1 extends Thread {

        public void run() {
            synchronized (lock1) {
                System.out.println("Thread 1: Holding lock 1...");
                try {
                    Thread.sleep(10);
                } catch (InterruptedException ignored) {}
                System.out.println("Thread 1: Waiting for lock 2...");
                synchronized (lock2) {
                    System.out.println("Thread 2: Holding lock 1 & 2...");
                }
            }
        }
    }

    private static class Thread2 extends Thread {

        public void run() {
            synchronized (lock2) {
                System.out.println("Thread 2: Holding lock 2...");
                try {
                    Thread.sleep(10);
                } catch (InterruptedException ignored) {}
                System.out.println("Thread 2: Waiting for lock 1...");
                synchronized (lock1) {
```

```
                System.out.println("Thread 2: Holding lock 2 & 1...");
            }
        }
    }
  }
}
```

OUTPUT:

```
preeti@preeti:~/IdeaProjects/Assessment3/src$ javac Q10three.java
preeti@preeti:~/IdeaProjects/Assessment3/src$ java Q10three
Thread 1: Holding lock 1...
Thread 2: Holding lock 2...
Thread 1: Waiting for lock 2...
Thread 2: Waiting for lock 1...
```