Q1 Write a programe do to demonstrate the use of volatile keyword.

Code -

```
import java.util.Scanner;
class Pro extends Thread {

        private volatile boolean running = true;

        @Override
        public void run() {
            while (running) {
                System.out.println("Running...");
                try {
                    Thread.sleep( millis: 50);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
        public void shutDown() {
            running = false;
        }
    }
public class multi1{
    public static void main(String[] args) {
        Pro pro=new Pro();
        pro.start();
        new Scanner(System.in).nextLine();
        pro.shutDown();
    }
}
```

Output-

```
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
```

Q2 Write a program to create a thread using Thread class and Runnable interface each.

Code -Thread

```
public class Multithreading extends Thread{
    @Override
    public void run() {
        for(int i=0;i<10;i++){
            System.out.println("Iter:"+i);
        }
    }
    public static void main(String[] args) {
        Multithreading multithreading=new Multithreading();
        Multithreading multithreading1=new Multithreading();
        multithreading.start();
        multithreading1.start();
    }
}
```

Output - Thread

```
/home/tushar/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Iter:0
Iter:0
Iter:1
Iter:2
Iter:3
Iter:4
Iter:5
Iter:6
Iter:7
Iter:8
Iter:9
Iter:1
Iter:2
Iter:3
Iter:4
Iter:5
Iter:6
Iter:7
Iter:8
Iter:9

Process finished with exit code 0
```

Code -

```java
public class MultiInterface implements Runnable {
    @Override
    public void run() {
        for(int i=0;i<10;i++){
            System.out.println("Iter:"+i);
            try {
                Thread.sleep( millis: 5000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
    public static void main(String[] args) {
        Thread t1=new Thread(new MultiInterface());
        Thread t2=new Thread(new MultiInterface());
        t1.start();
        t2.start();
    }
}
```

Q3 -Write a program using synchronization block and synchronization method

Code-

```java
public class Synchronized_block {
    private int count = 0;
    public synchronized void increment() {
        count++;
    }
    public static void main(String[] args) {
        Synchronized_block synchronized_block = new Synchronized_block();
        synchronized_block.run();
    }
    public void run() {
        Thread th = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i < 1000; i++) {
                    increment();
                }
            }
        });

        Thread th1 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i < 1000; i++) {
                    increment();
                }
            }
        });
        th.start();
        th1.start();
        try {
            th.join();
```

```java
        try {
            th.join();
            th1.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(count);
    }
}
```

Output-

```
/home/tushar/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
2000

Process finished with exit code 0
```

Q4 - 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-

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class TestThreadPool {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool( nThreads: 2);
        for (int i = 0; i < 10; i++) {
            Runnable worker = new WorkerThread( s: "" + i);
            executor.execute(worker);
        }
        executor.shutdown();
        while (!executor.isTerminated()) {   }
        System.out.println("Finished all threads");
    }
}
class WorkerThread implements Runnable {
    private String message;
    public WorkerThread(String s){
        this.message=s;
    }
    public void run() {
        System.out.println(Thread.currentThread().getName()+" (Start) message = "+message);
        processmessage();
        System.out.println(Thread.currentThread().getName()+" (End)");
    }
    private void processmessage() {
        try {  Thread.sleep( millis: 2000);  } catch (InterruptedException e) { e.printStackTrace(); }
    }
}
```

Output-

Q5 -Write a program to demonstrate wait and notify methods.

Code-

```
import java.util.Scanner;
class Processor {
  public void produce() throws InterruptedException {
     synchronized (this){
        System.out.println("Producer.....");
        wait();
        System.out.println("Producer is Back.....");
     }
  }
  public void consumer() throws InterruptedException {
     Scanner sc=new Scanner(System.in);
     Thread.sleep(2000);
     synchronized (this){
        System.out.println("Waiting for Return key....");
        sc.nextLine();
```

```
            System.out.println("Key pressed");
            notify();
            Thread.sleep(5000);
        }
    }
}
class demo2 {
    public static void main(String[] args) {
        final Processor processor=new Processor();

        Thread t1=new Thread(new Runnable() {
            apublic void run() {
                try {
                    processor.produce();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        Thread t2=new Thread(new Runnable() {
            public void run() {
                try {
                    processor.consumer();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        t1.start();
        t2.start();
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Output-

```
Run:  demo2
/home/tushar/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Producer.....
Waiting for Return key....

Key pressed

Producer is Back.....

Process finished with exit code 0
```

Q6Write a program to demonstrate sleep and join methods.

Code -

```
class TestJoinMethod1 extends Thread{
    public void run(){
        for(int i=1;i<=5;i++){
            try{
                Thread.sleep( millis: 500);
            }catch(Exception e){System.out.println(e);}
            System.out.println(i);
        }
    }
    public static void main(String args[]){
        TestJoinMethod1 t1=new TestJoinMethod1();
        TestJoinMethod1 t2=new TestJoinMethod1();
        TestJoinMethod1 t3=new TestJoinMethod1();
        t1.start();
        try{
            t1.join();
        }catch(Exception e){System.out.println(e);}
        t2.start();
        t3.start();
    }
}
```

Output -

```
/home/tushar/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
1
2
3
4
5
1
1
2
2
3
3
4
4
5
5

Process finished with exit code 0
```

Q7-Run a task with the help of callable and store it's result in the Future.

Code-

```java
import java.util.Random;
import java.util.concurrent.*;
public class RunCallable {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newCachedThreadPool();
Future<Integer> future = executor.submit(new Callable<Integer>() {
public Integer call() throws Exception {
        Random random=new Random();
        int duration = random.nextInt( bound: 4000);
        System.out.println("Starting...");
        Thread.sleep(duration);
        System.out.println("Finished...");
        return duration;
        }
        });
        executor.shutdown();
        try {
        System.out.println("result is:"+ future.get());
        } catch (InterruptedException e) {
        e.printStackTrace();
        } catch (ExecutionException e) {
        e.printStackTrace();
        }
        }
        }
```

Output -

Q8 Write a program to demonstrate the use of semaphore

Code -

```java
import java.util.concurrent.*;
class Connections{
  private static Connections instance = new Connections();
  private Semaphore semaphore = new Semaphore(2);
```

```
    private int connection=0;
    private Connections(){
    }
    public static Connections getInstance(){
        return instance;
    }
    public void connect(){
        try {
            semaphore.acquire();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        try{
            doconnect();
        }
        finally {
            semaphore.release();
        }
    }
    public void doconnect(){
        synchronized (this){
            connection++;
            System.out.println("Current Connection:"+connection);
        }
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        synchronized (this){
            connection--;
        }
    }
}
class demo2 {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newCachedThreadPool();
        for(int i=0;i<5;i++){
            executor.submit(new Runnable() {
                public void run() {
                    Connections.getInstance().connect();
                }
            });
        }
        executor.shutdown();
        try {
            executor.awaitTermination(1, TimeUnit.DAYS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

```
  }
}
```

Output -

```
/home/tushar/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Current Connection:1
Current Connection:2
Current Connection:1
Current Connection:2
Current Connection:1

Process finished with exit code 0
```

Q9 Write a program to demonstrate the use of CountDownLatch

Code-

```java
import java.util.concurrent.*;
class Processor implements Runnable{
    private CountDownLatch latch;
    public Processor(CountDownLatch latch){
        this.latch = latch;
    }
    public void run() {
        System.out.println("Started.....");
        try {
            Thread.sleep( millis: 400);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        latch.countDown();
    }
}
class CountDownDemo {
    public static void main(String[] args) {
        CountDownLatch countDownLatch = new CountDownLatch(3);
        ExecutorService executor = Executors.newFixedThreadPool( nThreads: 3);
        for(int i=0;i<3;i++){
            executor.submit(new Processor(countDownLatch));
        }
        try {
            countDownLatch.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Completed......");
    }
}
```

Output -

```
/home/tushar/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Started.....
Started.....
Started.....
Completed......
```

Q10 Write a program which creates deadlock between 2 threads

Code -

```
class Shared {
  synchronized void test1(Shared s2) throws InterruptedException {
```

```
        System.out.println("test1-begin");
        Thread.sleep(1000);
        s2.test2(this);
        System.out.println("test1-end");
    }
    synchronized void test2(Shared s1) throws InterruptedException {
        System.out.println("test2-begin");
        Thread.sleep(1000);
        s1.test1(this);
        System.out.println("test2-end");
    }
}
class Thread1 extends Thread
{
    private Shared s1;
    private Shared s2;
    public Thread1(Shared s1, Shared s2)
    {
        this.s1 = s1;
        this.s2 = s2;
    }
    public void run()
    {
        try {
            s1.test1(s2);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
class Thread2 extends Thread
{
    private Shared s1;
    private Shared s2;
    public Thread2(Shared s1, Shared s2)
    {
        this.s1 = s1;
        this.s2 = s2;
    }
    public void run()
    {
        try {
            s2.test2(s1);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
class DeadlockDemo {
    public static void main(String[] args) throws InterruptedException {
```

```
        Shared s1 = new Shared();
        Shared s2 = new Shared();
        Thread1 t1 = new Thread1(s1, s2);
        t1.start();
        Thread2 t2 = new Thread2(s1, s2);
        t2.start();
        Thread.sleep(2000);
    }
}
```

Output-

```
/home/tushar/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
test1-begin
test2-begin
```