

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

### Sol 1.

```
1 package Day3Threads;
2
3 import java.util.Scanner;
4
5 class process extends Thread{
6     private volatile boolean running=true;
7
8     public void run(){
9         while(running) {
10             System.out.println("Hello");
11             try {
12                 Thread.sleep(1000);
13             } catch (InterruptedException e) {
14                 System.out.println("Exception");
15             }
16         }
17         public void shutdown(){
18             running=false;
19         }
20     }
21
22     public class threadVolatile {
23
24         public static void main(String[] args) {
25             process p=new process();
26             p.start();
27             System.out.println("Press return to stop...");
28             Scanner scanner=new Scanner(System.in);
29             scanner.nextLine();
30             p.shutdown();
31         }
32     }
33 }
```

```
threadVolatile
/home/aman/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Hello
Press return to stop...
Hello
Hello
Hello
Hello

Process finished with exit code 0
|
final 0: Messages 4: Run 5: Debug 6: TODO
```

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

### Sol 2.

```
1 package Day3Threads;
2
3 class Runners implements Runnable
4 {
5     @Override
6     public void run()
7     {
8         for(int i=0; i<10; i++)
9         {
10             try{System.out.println("Hello"+i);
11                 Thread.sleep(100);}
12             catch(Exception e)
13             {
14                 System.out.println("Error");
15             }
16         }
17     }
18 }
19
20 public class Threadexample {
21     public static void main(String[] args)
22     {
23         Thread t1=new Thread(new Runners());
24         Thread t2=new Thread(new Runners());
25         t1.start();
26         t2.start();
27     }
28 }
29
30 }
```

```
Threadexample
/home/aman/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Hello0
Hello0
Hello1
Hello1
Hello2
Hello2
Hello3
Hello3
Hello4
Hello4
Hello5
Hello5
Hello6
Hello6
Hello7
Hello7
Hello8
Hello8
Hello9
Hello9

Process finished with exit code 0
final 0: Messages 4: Run 5: Debug 6: TODO
```

```
1 package Day3Threads;
2
3 class Runners extends Thread
4 {
5     @Override
6     public void run()
7     {
8         for(int i=0;i<10;i++)
9         {
10             try{System.out.println("Hello"+i);
11                 Thread.sleep(100);}
12             catch(Exception e)
13             {
14                 System.out.println("Error");
15             }
16         }
17     }
18 }
19
20 public class Threadexample {
21
22     public static void main(String[] args)
23     {
24         Runners t1=new Runners();
25         Runners t2=new Runners();
26         t1.start();
27         t2.start();
28     }
29 }
30
```

Threadexample

```
Threadexample x
/home/aman/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Hello0
Hello0
Hello1
Hello1
Hello2
Hello2
Hello3
Hello3
Hello4
Hello4
Hello5
Hello5
Hello6
Hello6
Hello7
Hello7
Hello8
Hello8
Hello9
Hello9

Process finished with exit code 0
```

Run Debug TODO

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

#### Sol 3.

```
1 package Day3Threads;
2
3
4 public class Worker {
5     private int count = 0;
6
7     public synchronized void increment() {
8         count++;
9     }
10
11 public static void main(String[] args)
12 {
13     Worker worker = new Worker();
14     worker.run();
15 }
16
17 public void run() {
18     Thread thread1 = new Thread(new Runnable() {
19         public void run() {
20             for(int i = 0; i < 10000; i++) {
21                 increment();
22             }
23         }
24     });
25     thread1.start();
26
27     Thread thread2 = new Thread(new Runnable() {
28         public void run() {
29             for(int i = 0; i < 10000; i++) {
30                 increment();
31             }
32         }
33     });
34     thread2.start();
35
36     try {
37         thread1.join();
38         thread2.join();
39     } catch (InterruptedException e) {
40         e.printStackTrace();
41     }
42
43     System.out.println("Count is: " + count);
44 }
45 }
```

```
20         increment();
21     }
22 }
23
24 thread1.start();
25
26 Thread thread2 = new Thread(new Runnable() {
27     public void run() {
28         for(int i = 0; i < 10000; i++) {
29             increment();
30         }
31     }
32 });
33 thread2.start();
34
35 try {
36     thread1.join();
37     thread2.join();
38 } catch (InterruptedException e) {
39     e.printStackTrace();
40 }
41
42 System.out.println("Count is: " + count);
43 }
44 }
```

```
Worker <
/home/aman/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Count is: 20000

Process finished with exit code 0
```

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.

#### Sol 4.

```
1 package Day3Threads;
2
3
4 import java.util.concurrent.ExecutorService;
5 import java.util.concurrent.Executors;
6 import java.util.concurrent.TimeUnit;
7
8 class ThreadPoolDemo implements Runnable{
9
10     public void run() {
11         for (int i=0; i<100;i=i+2){
12             System.out.println(i);
13         }
14
15         try {
16             Thread.sleep( millis: 500);
17         } catch (InterruptedException e) {
18         }
19         System.out.println("\n");
20
21         for (int i=1;i<100;i=i+2){
22             System.out.println(i);
23         }
24     }
25 }
26
27
28
29
30 public class Q4Process {
31     public static void main(String[] args) {
32
33         ExecutorService executor = Executors.newFixedThreadPool( nThreads: 2);
34
35
36         executor.submit(new ThreadPoolDemo());
37
38         executor.shutdown();
39
40         System.out.println("All tasks submitted.");
41
42         try {
43             executor.awaitTermination( timeout: 1, TimeUnit.DAYS);
44         } catch (InterruptedException e) {
45         }
46
47         System.out.println("All tasks completed.");
48     }
49 }
```

Q4Process → main()

```
CDlatch x Q4Process x
/home/aman/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
56
58
nal ▶ 4: Run 5: Debug 6: TODO
```

```
CDlatch x Q4Process x
All tasks submitted.
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49
51
53
55
nal ▶ 4: Run 5: Debug 6: TODO
```

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

### Sol 5.

```
1 package Day3Threads;
2
3 public class Q5WaitNotifyDemo {
4     public static void main(String[] args) throws InterruptedException {
5         final Q5Process process = new Q5Process();
6         Thread t1 = new Thread(new Runnable() {
7             @Override
8             public void run() {
9                 try {
10                     process.produce();
11                 } catch (InterruptedException e) {}
12             }
13         });
14         Thread t2 = new Thread(new Runnable() {
15             @Override
16             public void run() {
17                 try {
18                     process.consumer();
19                 } catch (InterruptedException e) {}
20             }
21         });
22         t1.start();
23         t2.start();
24         t1.join();
25         t2.join();
26     }
27 }
```

```
1 import java.util.Scanner;
2
3 public class Q5Process {
4     public void produce() throws InterruptedException {
5         synchronized (this) {
6             System.out.println("Producer thread running ....");
7             wait();
8             System.out.println("Resumed.");
9         }
10    }
11
12    public void consumer() throws InterruptedException {
13        Scanner scanner = new Scanner(System.in);
14        Thread.sleep(2000);
15
16        synchronized (this) {
17            System.out.println("Waiting for return key.");
18            scanner.nextLine();
19            System.out.println("Return key pressed.");
20            notify();
21            Thread.sleep(5000);
22        }
23    }
24 }
```

```
CDlatch x Q5WaitNotifyDemo x
/home/aman/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Producer thread running ....
Waiting for return key.

Return key pressed.
Resumed.

Process finished with exit code 0
```

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

### Sol 6.

```
1 package Day3Threads;
2 public class Q6SleepJoinDemo implements Runnable{
3     private int count = 0;
4     public static void main(String[] args)
5     {
6         Q6SleepJoinDemo sleepJoinDemo = new Q6SleepJoinDemo();
7         sleepJoinDemo.run();
8     }
9     public void run() {
10        Thread thread1 = new Thread(new Runnable() {
11            public void run() {
12                for(int i = 0; i < 10000; i++) {
13                    count++; }
14            });
15        thread1.start();
16
17        Thread thread2 = new Thread(new Runnable() {
18            public void run() {
19                for(int i = 0; i < 10000; i++) {
20                    count++; }
21            });
22        thread2.start();
23        try {
24            thread1.join();
25            thread2.join();
26        } catch (InterruptedException e) {
27            e.printStackTrace();
28        }
29        System.out.println("Count is: " + count);
30    }
31 }
```

```
Q6SleepJoinDemo
/home/aman/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Count is: 20000

Process finished with exit code 0

Run Debug TODO
```



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

### Sol 7.

```
1 package Day3Threads;
2 import java.io.IOException;
3 import java.util.Random;
4 import java.util.concurrent.*;
5 public class Q7CallFutureDemo {
6     public static void main(String[] args) {
7         ExecutorService executor = Executors.newCachedThreadPool();
8         Future<Integer> future = executor.submit(new Callable<Integer>() {
9             @Override
10             public Integer call() throws Exception {
11                 Random random = new Random();
12                 int duration = random.nextInt( bound: 400);
13
14                 if (duration > 2000) {
15                     throw new IOException("Sleeping for too long.");
16                 }
17                 System.out.println("Starting ...");
18                 try {
19                     Thread.sleep(duration);
20                 } catch (InterruptedException e) {
21                     e.printStackTrace();
22                 }
23                 System.out.println("Finished.");
24                 return duration;
25             }
26         });
27         executor.shutdown();
28         try {
29             System.out.println("Result is: " + future.get());
30         } catch (InterruptedException e) {
31             e.printStackTrace();
32         } catch (ExecutionException e) {
33             IOException ex = (IOException) e.getCause();
34             System.out.println(ex.getMessage());
35         }
36     }
37 }
```

```
Q7CallFutureDemo x
/home/aman/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Starting ...
Finished.
Result is: 348

Process finished with exit code 0

Run Debug TODO
```

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

**Sol 8.**

```
1 package Day3Threads;
2
3
4 import java.util.concurrent.ExecutorService;
5 import java.util.concurrent.Executors;
6 import java.util.concurrent.TimeUnit;
7
8 public class Q8SemaphoreDemo {
9     public static void main(String[] args) {
10         ExecutorService executor = Executors.newCachedThreadPool();
11
12         for (int i = 0; i < 200; i++) {
13             executor.submit(new Runnable() {
14                 public void run() {
15                     Q8Connections.getInstance().connect();
16                 }
17             });
18         }
19
20         executor.shutdown();
21
22         try {
23             executor.awaitTermination(1, TimeUnit.DAYS);
24         } catch (InterruptedException e) {
25
26         }
27     }
28 }
```

```
Thu 14:12
core-java [~/IdeaProjects/core-java] - .../src/Day3Threads/Q8Connections.java
Build Run Tools VCS Window Help
connections
Q7CallFutureDemo
Q8Connections.java
1 package Day3Threads;
2 import java.util.concurrent.Semaphore;
3 public class Q8Connections {
4     private static Q8Connections instance = new Q8Connections();
5     private Semaphore sem = new Semaphore(permits: 10, fair: true);
6     private int connections = 0;
7     private Q8Connections() { }
8     public static Q8Connections getInstance() {
9         return instance;
10    }
11    public void connect() {
12        try {
13            sem.acquire();
14        } catch (InterruptedException e1) {
15            e1.printStackTrace();
16        }
17        try {
18            doConnect();
19        } finally {
20            sem.release();
21        }
22    }
23    public void doConnect() {
24        synchronized (this) {
25            connections++;
26            System.out.println("Current connections: " + connections);
27            try {
28                Thread.sleep(2000);
29            } catch (InterruptedException e) {
30                e.printStackTrace();
31            }
32            synchronized (this) {
33                connections--;
34            }
35        }
36    }
37 }
```

```
Q8SemaphoreDemo
/home/aman/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Current connections: 1
Current connections: 2
Current connections: 3
Current connections: 4
Current connections: 5
Current connections: 6
Current connections: 7
Current connections: 8
Current connections: 9
Current connections: 10
```

9. Write a program to demonstrate the use of CountdownLatch

### Sol 9.

```
1 package Day3Threads;
2
3
4 import java.util.concurrent.CountDownLatch;
5 import java.util.concurrent.ExecutorService;
6 import java.util.concurrent.Executors;
7
8 public class CDlatch {
9
10     public static void main(String[] args) {
11
12         CountDownLatch latch = new CountDownLatch(3);
13         ExecutorService executor = Executors.newFixedThreadPool(3);
14
15         for(int i=0; i < 3; i++) {
16             executor.submit(new CDprocessor(latch));
17         }
18
19         try {
20             latch.await();
21         } catch (InterruptedException e) {
22             e.printStackTrace();
23         }
24         System.out.println("Completed.");
25     }
26 }
27
28 CDlatch > main()
```

```
1 package Day3Threads;
2
3 import java.util.concurrent.CountDownLatch;
4 import java.util.concurrent.ExecutorService;
5 import java.util.concurrent.Executors;
6
7 class CDprocessor implements Runnable {
8     private CountDownLatch latch;
9
10     public CDprocessor(CountDownLatch latch) {
11         this.latch = latch;
12     }
13
14     public void run() {
15         System.out.println("Started.");
16
17         try {
18             Thread.sleep(3000);
19         } catch (InterruptedException e) {
20             e.printStackTrace();
21         }
22         latch.countDown();
23     }
24 }
25
26 CDprocessor > run()
```

Debug | TODO | Event Log | 20:13 LF UTF-8 4 spaces

```
CDlatch x
/home/aman/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Started.
Started.
Started.
Completed.
```

10. Write a program which creates deadlock between 2 threads

**Sol 10.**

```
1 package Day3Threads;
2
3 class DeadAcc {
4     private int balance = 10000;
5
6     public void deposit(int amount) {
7         balance += amount;
8     }
9
10    public void withdraw(int amount) {
11        balance -= amount;
12    }
13
14    public int getBalance() {
15        return balance;
16    }
17
18    public static void transfer(DeadAcc acc1, DeadAcc acc2, int amount) {
19        acc1.withdraw(amount);
20        acc2.deposit(amount);
21    }
22 }
23
```

```
1 package Day3Threads;
2 public class DeadApp {
3     public static void main(String[] args) throws Exception {
4
5         final DeadRunner runner = new DeadRunner();
6
7         Thread t1 = new Thread(new Runnable() {
8             public void run() {
9                 try {
10                     runner.firstThread();
11                 } catch (InterruptedException e) {
12                     // TODO Auto-generated catch block
13                     e.printStackTrace();
14                 }
15             }
16         });
17
18         Thread t2 = new Thread(new Runnable() {
19             public void run() {
20                 try {
21                     runner.secondThread();
22                 } catch (InterruptedException e) {
23                     // TODO Auto-generated catch block
24                     e.printStackTrace();
25                 }
26             }
27         });
28         t1.start();
29         t2.start();
30         t1.join();
31         t2.join();
32         runner.finished();
33     }
34 }
```

```
1 package Day3Threads;
2
3 import java.util.Random;
4 import java.util.concurrent.locks.Lock;
5 import java.util.concurrent.locks.ReentrantLock;
6
7 public class DeadRunner {
8     private DeadAcc acc1 = new DeadAcc();
9     private DeadAcc acc2 = new DeadAcc();
10
11     private Lock lock1 = new ReentrantLock();
12     private Lock lock2 = new ReentrantLock();
13
14     private void acquireLocks(Lock firstLock, Lock secondLock) throws InterruptedException {
15         while(true) {
16             // Acquire locks
17
18             boolean gotFirstLock = false;
19             boolean gotSecondLock = false;
20
21             try {
22                 gotFirstLock = firstLock.tryLock();
23                 gotSecondLock = secondLock.tryLock();
24             }
25             finally {
26                 if(gotFirstLock && gotSecondLock) {
27                     return;
28                 }
29             }
30         }
31     }
32 }
```

```

    }
    finally {
        if (gotFirstLock && gotSecondLock) {
            return;
        }

        if (gotFirstLock) {
            firstLock.unlock();
        }

        if (gotSecondLock) {
            secondLock.unlock();
        }
    }

    // Locks not acquired
    Thread.sleep( millis: 1);
}

}

public void firstThread() throws InterruptedException {
    Random random = new Random();

    for (int i = 0; i < 10000; i++) {
        acquireLocks(lock1, lock2);

        try {
            DeadAcc.transfer(acc1, acc2, random.nextInt( bound: 100));
            Account.transfer(acc1, acc2, );
        }
    }
}

```

```

    }
}

public void secondThread() throws InterruptedException {
    Random random = new Random();

    for (int i = 0; i < 10000; i++) {
        acquireLocks(lock2, lock1);

        try {
            DeadAcc.transfer(acc2, acc1, random.nextInt( bound: 100));
        } finally {
            lock1.unlock();
            lock2.unlock();
        }
    }
}

public void finished() {
    System.out.println("Account 1 balance: " + acc1.getBalance());
    System.out.println("Account 2 balance: " + acc2.getBalance());
    System.out.println("Total balance: "
        + (acc1.getBalance() + acc2.getBalance()));
}
}

```

```

DeadApp --mainD --newRandom --1000
DeadApp --
mainD --mainD --newRandom --1000
ce: 14593
ce: 2407
20000
d with exit code 0

```