Session: Basics of Multithreading

Assignment

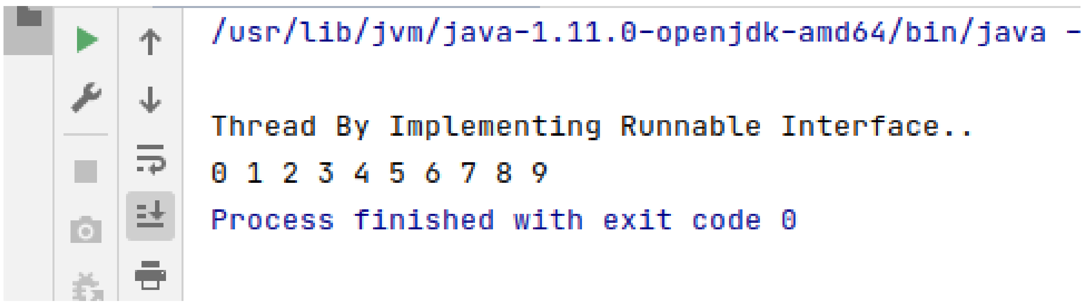
1. Create and Run a Thread using Runnable Interface and Thread class.

Thread Using Runnable Interface

CODE

```
class Threademo implements Runnable {
    public void run() {

        for (int i = 0; i < 10; i++) {
            System.out.print(i + " ");
        }
    }
}

public class Ques1_Runnable {
    public static void main(String[] args) {

        System.out.println("\nThread By Implementing Runnable Interface..");
        Threademo t1 = new Threademo();
        Thread ob1 = new Thread(t1);
        ob1.start();
    }
}
```

OUTPUT



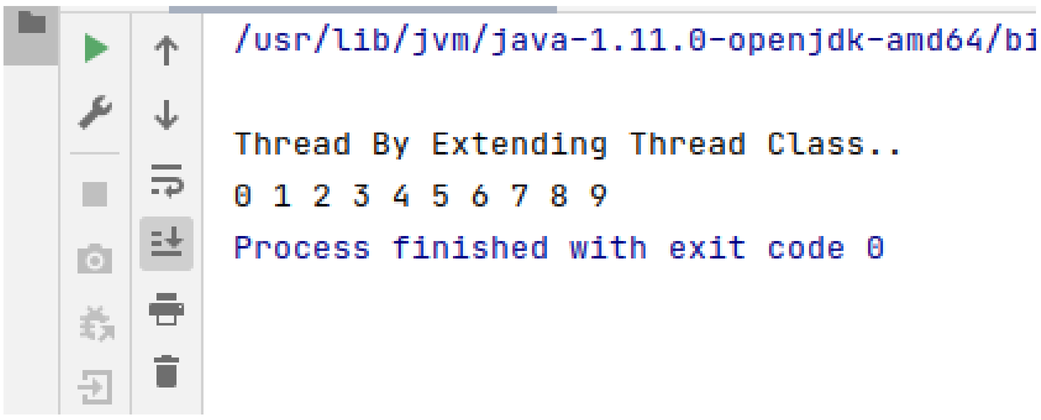
```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java -

Thread By Implementing Runnable Interface..
0 1 2 3 4 5 6 7 8 9
Process finished with exit code 0
```

Thread by Extending Thread class

CODE

```
class Threademo2 extends Thread {
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.print(i + " ");
        }
    }
}

public class Ques1_Thread {
    public static void main(String[] args) {

        System.out.println("\nThread By Extending Thread Class..");
        Threademo2 t2 = new Threademo2();
        t2.start();
    }
}
```

OUTPUT



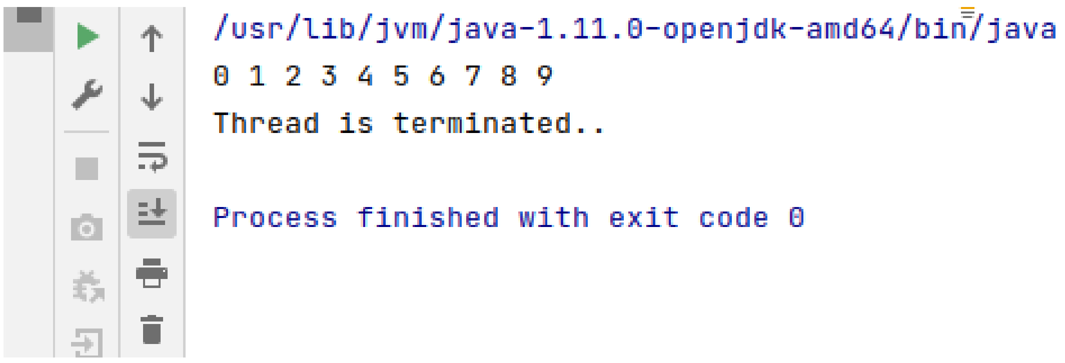
```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bi

Thread By Extending Thread Class..
0 1 2 3 4 5 6 7 8 9
Process finished with exit code 0
```

2. Use sleep and join methods with thread.

CODE

```
class Sleep_join extends Thread {
    public void run() {

        for (int i = 0; i < 10; i++) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.print(i + " ");
        }
    }
}

public class Ques2 {
    public static void main(String[] args) {
        Sleep_join sl = new Sleep_join();
        sl.start();
        try {
            sl.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("\nThread is terminated..");
    }
}
```

OUTPUT



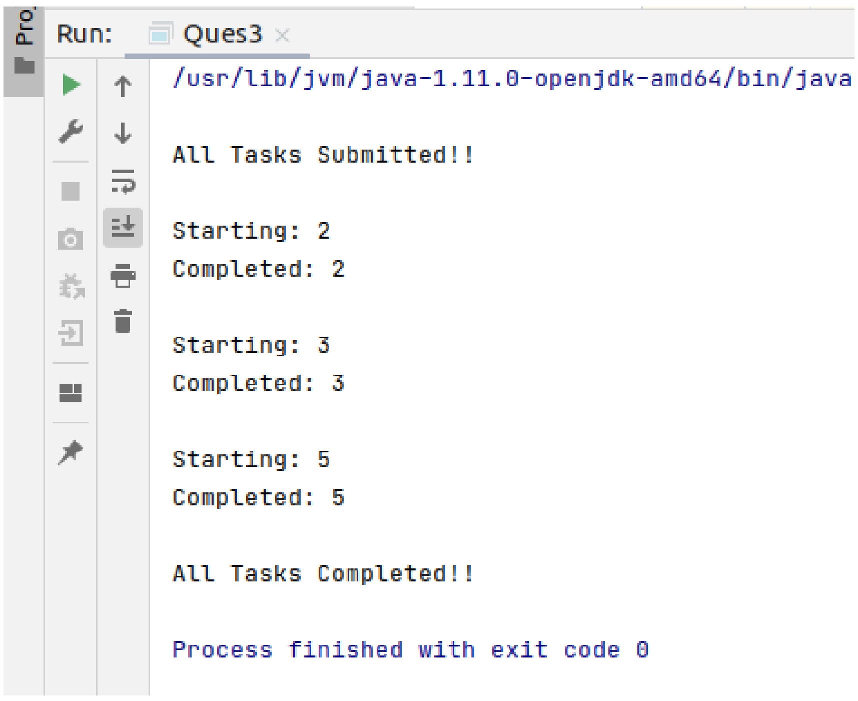
```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java
0 1 2 3 4 5 6 7 8 9
Thread is terminated..

Process finished with exit code 0
```

3. Use a singleThreadExecutor to submit multiple threads.

CODE

```
class Processor implements Runnable
{
    private int id;
    Processor(int id)
    {
        this.id = id;
    }

    @Override
    public void run() {
        System.out.println("\nStarting: "+id);
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Completed: "+id);
    }
}

public class Ques3 {
    public static void main(String[] args) {
        ExecutorService executorService = Executors.newSingleThreadExecutor();
        executorService.submit(new Processor(2));
        executorService.submit(new Processor(3));
        executorService.submit(new Processor(5));

        executorService.shutdown();
        System.out.println("\nAll Tasks Submitted!!");
        try {
            executorService.awaitTermination(1, TimeUnit.HOURS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

       System.out.println("\nAll Tasks Completed!!");


    }
}
```

OUTPUT



```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java

All Tasks Submitted!!

Starting: 2
Completed: 2

Starting: 3
Completed: 3

Starting: 5
Completed: 5

All Tasks Completed!!

Process finished with exit code 0
```

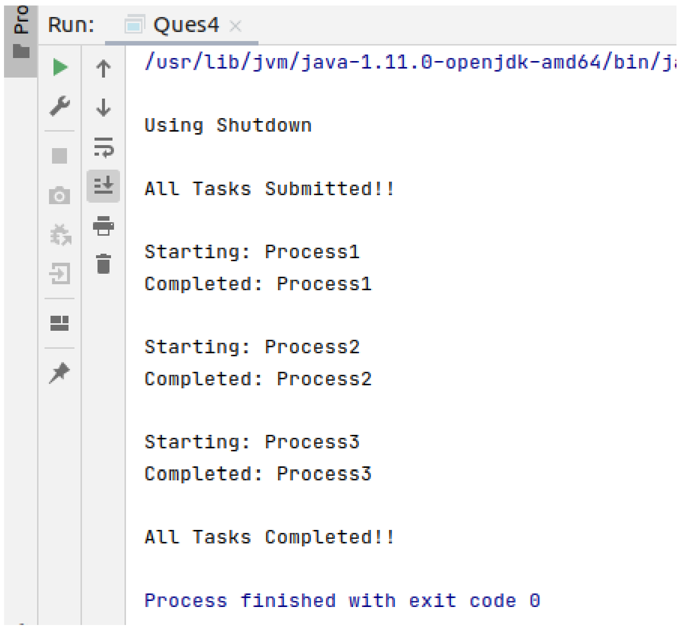
4. Try shutdown() and shutdownNow() and observe the difference.

CODE

Using shutdown()

```
class Processor1 implements Runnable
{
    private String name;
    Processor1(String name)
    {
        this.name = name;
    }

    @Override
    public void run() {
        System.out.println("\nStarting: "+name);
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Completed: "+name);
    }
}

public class Ques4 {
    public static void main(String[] args) {
        ExecutorService executorService = Executors.newSingleThreadExecutor();
        executorService.submit(new Processor1("Process1"));
        executorService.submit(new Processor1("Process2"));
        executorService.submit(new Processor1("Process3"));
        System.out.println("\nUsing Shutdown");
         executorService.shutdown();
        System.out.println("\nAll Tasks Submitted!!");
    try {
            executorService.awaitTermination(1, TimeUnit.HOURS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("\nAll Tasks Completed!!");
    }
}
```

OUTPUT

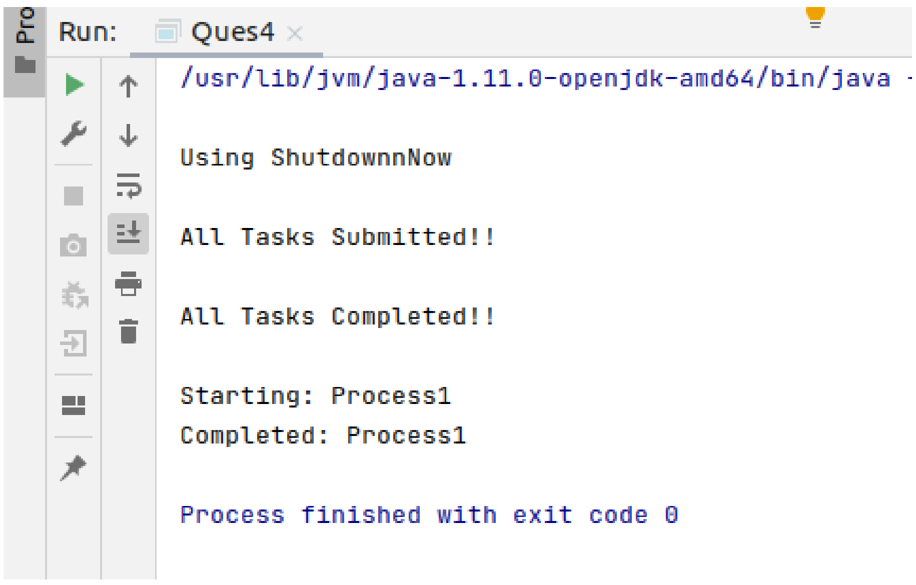


```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/j

Using Shutdown

All Tasks Submitted!!

Starting: Process1
Completed: Process1

Starting: Process2
Completed: Process2

Starting: Process3
Completed: Process3

All Tasks Completed!!

Process finished with exit code 0
```

Using shutdownNow()

```
class Processor1 implements Runnable
{
    private String name;
    Processor1(String name)
    {
        this.name = name;
    }

    @Override
    public void run() {
        System.out.println("\nStarting: "+name);
        System.out.println("Completed: "+name);
    }
}

public class Ques4 {
    public static void main(String[] args) {
        ExecutorService executorService = Executors.newSingleThreadExecutor();
        executorService.submit(new Processor1("Process1"));
        executorService.submit(new Processor1("Process2"));
        executorService.submit(new Processor1("Process3"));
        System.out.println("\nUsing ShutdownnNow");
        executorService.shutdownNow();
        System.out.println("\nAll Tasks Submitted!!");
        System.out.println("\nAll Tasks Completed!!");
    }
}
```

OUTPUT



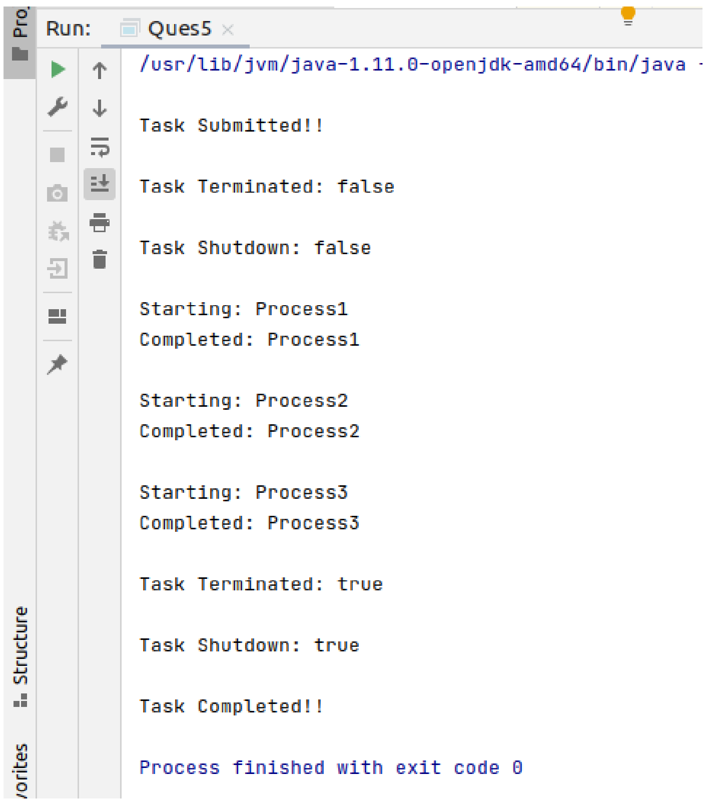
```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java -

Using ShutdownnNow

All Tasks Submitted!!

All Tasks Completed!!

Starting: Process1
Completed: Process1

Process finished with exit code 0
```

5. Use isShutDown() and isTerminated() with ExecutorService.

CODE

```
class Processor2 implements Runnable {
    private String name;

    Processor2(String name) {
        this.name = name;
    }

    @Override
    public void run() {
        System.out.println("\nStarting: " + name);
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Completed: " + name);
    }
}

public class Ques5 {
    public static void main(String[] args) {

        ExecutorService executorService = Executors.newSingleThreadExecutor();
        executorService.submit(new Processor2("Process1"));
        executorService.submit(new Processor2("Process2"));
        executorService.submit(new Processor2("Process3"));
        System.out.println("\nTask Submitted!!");
        System.out.println("\nTask Terminated: " + executorService.isTerminated());
        System.out.println("\nTask Shutdown: " + executorService.isShutdown());


        executorService.shutdown();
        try {
            executorService.awaitTermination(1, TimeUnit.HOURS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("\nTask Terminated: " + executorService.isTerminated());
        System.out.println("\nTask Shutdown: " + executorService.isShutdown());
        System.out.println("\nTask Completed!!");
    }
}
```

OUTPUT



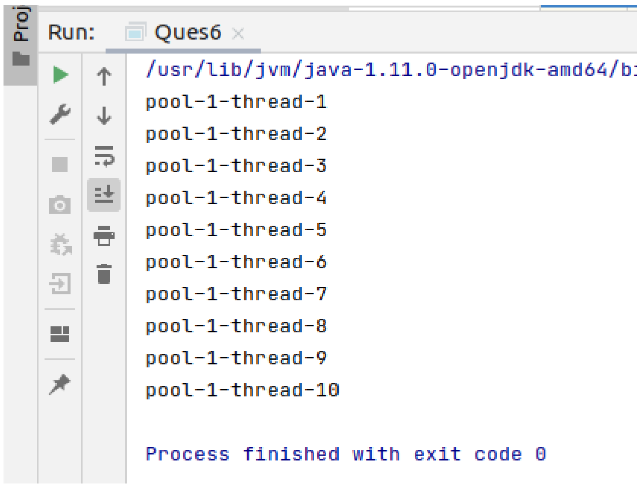
```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java

Task Submitted!!

Task Terminated: false

Task Shutdown: false

Starting: Process1
Completed: Process1

Starting: Process2
Completed: Process2

Starting: Process3
Completed: Process3

Task Terminated: true

Task Shutdown: true

Task Completed!!

Process finished with exit code 0
```

6. Return a Future from ExecutorService by using callable and use get(), isDone(), isCancelled() with the Future object to know the status of task submitted.

CODE

```
public class Ques6 implements Callable<String> {

    @Override
    public String call() throws Exception {
        Thread.sleep(1000);
        return Thread.currentThread().getName();
    }

    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(10);
        List<Future<String>> list = new ArrayList<Future<String>>();
        Callable<String> callable = new Ques6();
        for (int i = 0; i < 10; i++) {
            Future<String> future = executor.submit(callable);
            list.add(future);
        }
        int count = 1;
        for (Future<String> fut : list) {
            if (count == 10) {
                fut.cancel(true);
            }
            count++;
            try {
                if (fut.isCancelled()) {
                    System.out.println("10th thread is cancelled");
                } else {
                    String str = fut.get();
                    if (fut.isDone()) {
                        System.out.println(str);
                    }
                }
            } catch (InterruptedException | ExecutionException e) {
                e.printStackTrace();
            }
        }
        executor.shutdown();
    }
}
```

OUTPUT



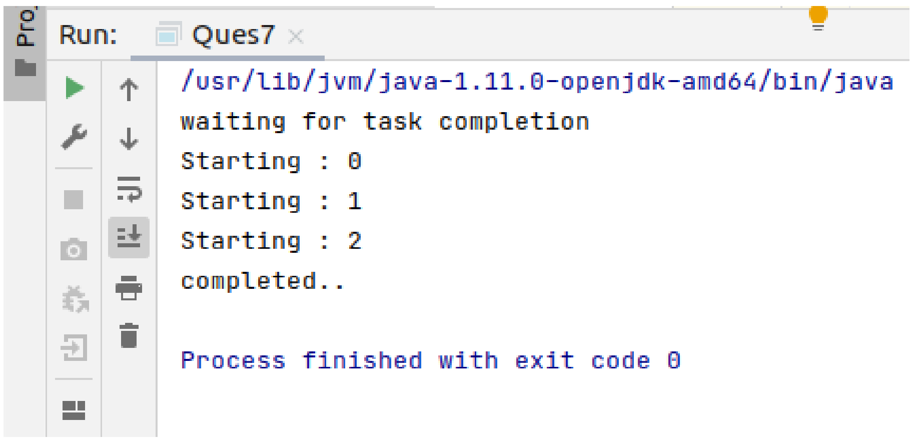
```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/b
pool-1-thread-1
pool-1-thread-2
pool-1-thread-3
pool-1-thread-4
pool-1-thread-5
pool-1-thread-6
pool-1-thread-7
pool-1-thread-8
pool-1-thread-9
pool-1-thread-10

Process finished with exit code 0
```

7. Submit List of tasks to ExecutorService and wait for the completion of all the tasks.

CODE

```
class Processor3 implements Runnable {
    public CountDownLatch latch;
    private int id;

    public Processor3(CountDownLatch latch, int id) {
        this.latch = latch;
        this.id = id;
    }

    @Override
    public void run() {
        System.out.println("Starting : " + id);
        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        latch.countDown();
    }
}

public class Ques7 {
    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 Processor3(latch, i));
        }
        try {
            System.out.println("waiting for task completion");
            latch.await();

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        executor.shutdown();
        System.out.println("completed..");
    }
}
```

OUTPUT



```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java
waiting for task completion
Starting : 0
Starting : 1
Starting : 2
completed..

Process finished with exit code 0
```

8. Schedule task using schedule(), scheduleAtFixedRate() and scheduleAtFixedDelay()

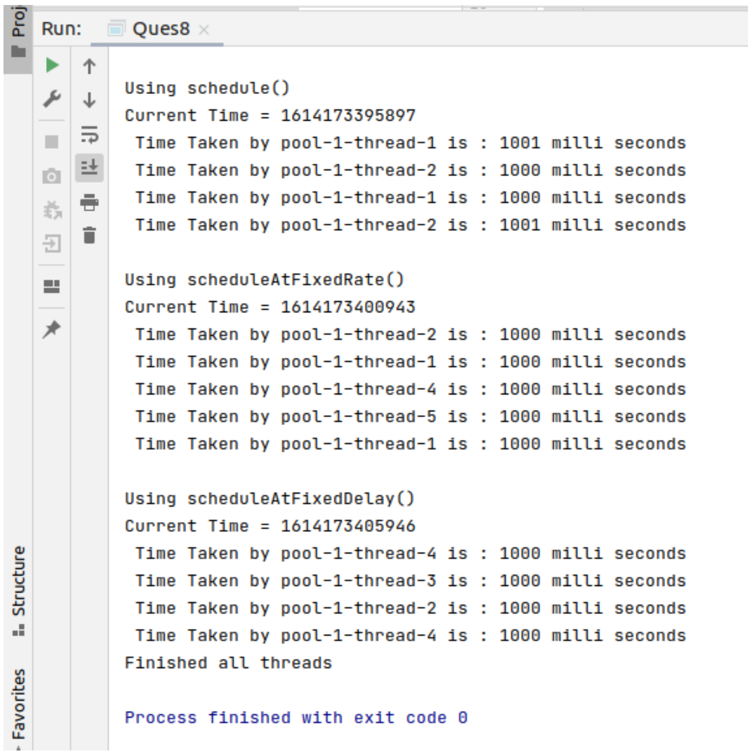
CODE

```
class WorkerThread implements Runnable {
    @Override
    public void run() {
        long start, end;
        start = System.currentTimeMillis();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        end = System.currentTimeMillis();
        System.out.println(" Time Taken by " + Thread.currentThread().getName() + " is : "
+ (end - start) + " milli seconds");
    }
}
```

```
public class Ques8 {
   public static void main(String[] args) {

      ScheduledExecutorService scheduledThreadPool =
Executors.newScheduledThreadPool(5);
      System.out.println("\nUsing schedule()\nCurrent Time = " +
System.currentTimeMillis());
      for (int i = 0; i < 5; i++) {
         try {
            Thread.sleep(1000);
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
         WorkerThread worker = new WorkerThread();
         scheduledThreadPool.schedule(worker, 0, TimeUnit.SECONDS);
      }
      System.out.println("\nUsing scheduleAtFixedRate()\nCurrent Time = " +
System.currentTimeMillis());
      for (int i = 0; i < 5; i++) {
         try {
            Thread.sleep(1000);
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
         WorkerThread worker = new WorkerThread();
         scheduledThreadPool.scheduleAtFixedRate(worker, 0, 10, TimeUnit.SECONDS);
      }
      System.out.println("\nUsing scheduleAtFixedDelay()\nCurrent Time = " +
System.currentTimeMillis());
      for (int i = 0; i < 3; i++) {
         try {
            Thread.sleep(1000);
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
         WorkerThread worker = new WorkerThread();
         scheduledThreadPool.scheduleWithFixedDelay(worker, 0, 10,
TimeUnit.SECONDS);
      }

      try {
         Thread.sleep(500);
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
```

```
        scheduledThreadPool.shutdown();
        while (!scheduledThreadPool.isTerminated()) {
            //wait for all tasks to finish
        }
        System.out.println("Finished all threads");
    }
}
```

OUTPUT



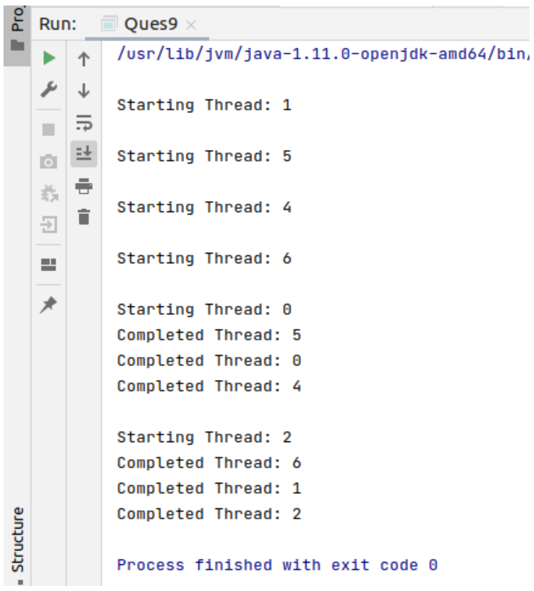
```
Using schedule()
Current Time = 1614173395897
 Time Taken by pool-1-thread-1 is : 1001 milli seconds
 Time Taken by pool-1-thread-2 is : 1000 milli seconds
 Time Taken by pool-1-thread-1 is : 1000 milli seconds
 Time Taken by pool-1-thread-2 is : 1001 milli seconds

Using scheduleAtFixedRate()
Current Time = 1614173400943
 Time Taken by pool-1-thread-2 is : 1000 milli seconds
 Time Taken by pool-1-thread-1 is : 1000 milli seconds
 Time Taken by pool-1-thread-4 is : 1000 milli seconds
 Time Taken by pool-1-thread-5 is : 1000 milli seconds
 Time Taken by pool-1-thread-1 is : 1000 milli seconds

Using scheduleAtFixedDelay()
Current Time = 1614173405946
 Time Taken by pool-1-thread-4 is : 1000 milli seconds
 Time Taken by pool-1-thread-3 is : 1000 milli seconds
 Time Taken by pool-1-thread-2 is : 1000 milli seconds
 Time Taken by pool-1-thread-4 is : 1000 milli seconds
Finished all threads

Process finished with exit code 0
```

9. Increase concurrency with Thread pools using newCachedThreadPool() and newFixedThreadPool().

CODE

```
class Demo implements Runnable {
    private int id;

    public Demo(int id) {
        this.id = id;
    }

    public void run() {
        System.out.println("\nStarting Thread: " + id);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Completed Thread: " + id);
    }
}

public class Ques9 {
    public static void main(String[] args) {

        ExecutorService executor1 = Executors.newFixedThreadPool(2);
        ExecutorService executor2 = Executors.newCachedThreadPool();

        for (int i = 0; i < 3; i++) {
            executor1.submit(new Demo(i));
        }
        for (int i = 4; i < 7; i++) {
            executor2.submit(new Demo(i));
        }
        executor1.shutdown();
        executor2.shutdown();

        try {
            executor1.awaitTermination(1, TimeUnit.HOURS);
            executor2.awaitTermination(1, TimeUnit.HOURS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    }
}
```

OUTPUT



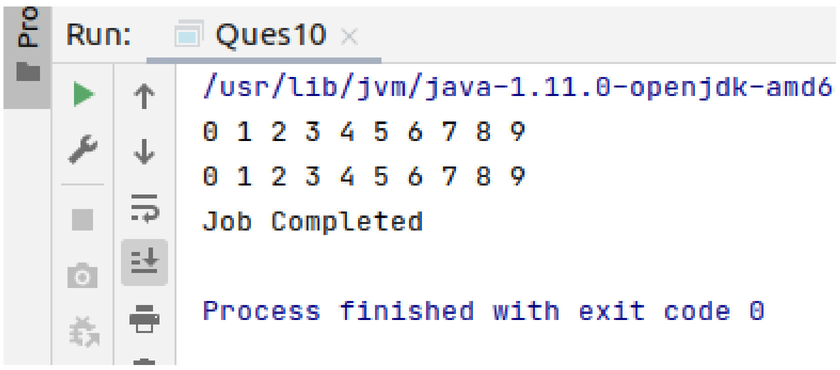
```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin,

Starting Thread: 1

Starting Thread: 5

Starting Thread: 4

Starting Thread: 6

Starting Thread: 0
Completed Thread: 5
Completed Thread: 0
Completed Thread: 4

Starting Thread: 2
Completed Thread: 6
Completed Thread: 1
Completed Thread: 2

Process finished with exit code 0
```

10. Use Synchronize method to enable synchronization between multiple threads trying to access method at same time.

CODE

```
public class Ques10 implements Runnable {

   @Override
   public synchronized void run() {
      for (int i = 0; i < 10; i++)
         System.out.print(i + " ");
      System.out.println();
   }

   public static void main(String[] args) {

      Ques10 obj1 = new Ques10();
      Ques10 obj2 = new Ques10();

      Thread t1 = new Thread(obj1);
      Thread t2 = new Thread(obj2);
      t1.start();
      t2.start();
      try {
         t1.join();
         t2.join();
      } catch (InterruptedException e) {
         e.printStackTrace();
      }

      System.out.println("Job Completed");

   }
}
```

OUTPUT



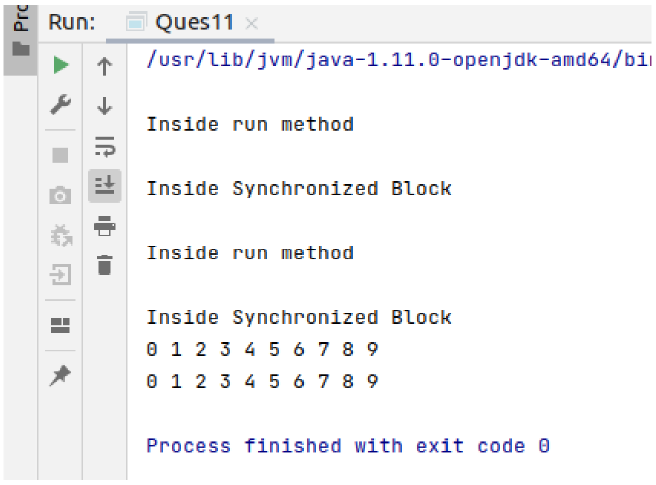
```
/usr/lib/jvm/java-1.11.0-openjdk-amd6
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
Job Completed

Process finished with exit code 0
```

11. Use Synchronize block to enable synchronization between multiple threads trying to access method at same time.

CODE

```
public class Ques11 implements Runnable {

    void increment() {
        synchronized (this) {
            System.out.println("\nInside Synchronized Block");
            for (int i = 0; i < 10; i++)
                System.out.print(i + " ");
        }
    }

    @Override
    public void run() {
        System.out.println("\nInside run method");
        increment();
        System.out.println();
    }

    public static void main(String[] args) {
        Ques11 obj1 = new Ques11();
        Ques11 obj2 = new Ques11();
        Thread t1 = new Thread(obj1);
        Thread t2 = new Thread(obj2);
        t1.start();
        t2.start();

    }
}
```

OUTPUT



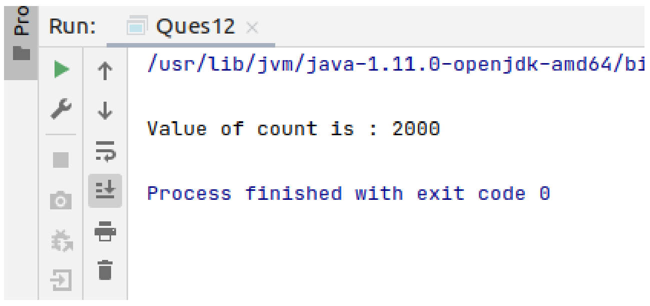
```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bi

Inside run method

Inside Synchronized Block

Inside run method

Inside Synchronized Block
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9

Process finished with exit code 0
```

12. Use Atomic Classes instead of Synchronize method and blocks.

CODE

```
class Adder extends Thread {
    AtomicInteger count;

    Adder() {
        count = new AtomicInteger();
    }

    public void run() {
        for (int i = 0; i < 1000; i++) {
            count.addAndGet(1);
        }
    }
}

public class Ques12 {
    public static void main(String[] args) {

        Adder obj = new Adder();

        Thread first = new Thread(obj, "Thread1");
        Thread second = new Thread(obj, "Thread2");

        first.start();
        second.start();
        try {
            first.join();
            second.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("\nValue of count is : " + obj.count);
    }
}
```

OUTPUT



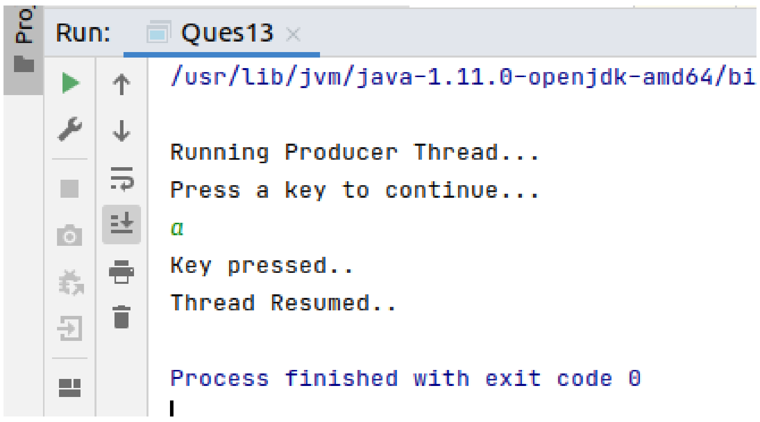
```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bi

Value of count is : 2000

Process finished with exit code 0
```

13. Coordinate 2 threads using wait() and notify().

CODE

```
class Processor4 {
   public void produce() throws InterruptedException {
      synchronized (this) {
         System.out.println("\nRunning Producer Thread...");
         wait();
         System.out.println("Thread Resumed..");
      }
   }

   public void consume() throws InterruptedException {

      Thread.sleep(2000);
      synchronized (this) {
         System.out.println("Press a key to continue...");
         Scanner scanner = new Scanner(System.in);
         String str = scanner.nextLine();
         if (str.equals(" ")) {
            System.out.println("\nPlease enter a valid String");
            exit(1);
         } else
            System.out.println("Key pressed..");
         notify();
      }
   }
}

public class Ques13 {
   public static void main(String[] args) {

      Processor4 obj1 = new Processor4();

      Thread t1 = new Thread(new Runnable() {
         @Override
         public void run() {
            try {
               obj1.produce();
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
         }
      });

      Thread t2 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    obj1.consume();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        t1.start();
        t2.start();
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT



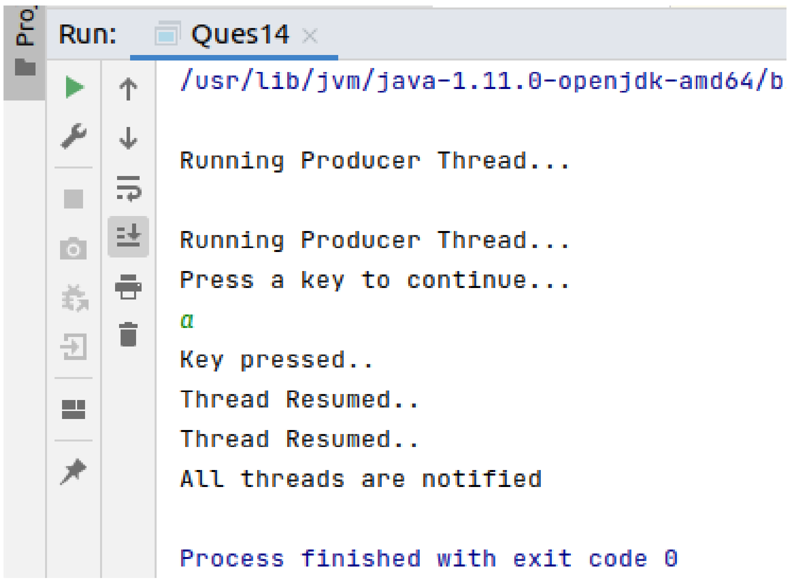
```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bi

Running Producer Thread...
Press a key to continue...
a
Key pressed..
Thread Resumed..

Process finished with exit code 0
```

14. Coordinate multiple threads using wait() and notifyAll()

CODE

```
class Processor5 {
    public void produce() throws InterruptedException {
        synchronized (this) {
            System.out.println("\nRunning Producer Thread...");
            wait();
            System.out.println("Thread Resumed..");
        }
    }

    public void consume() throws InterruptedException {
        Thread.sleep(2000);
        synchronized (this) {
            System.out.println("Press a key to continue...");
            Scanner scanner = new Scanner(System.in);
            String str = scanner.nextLine();
            if (str.equals(" ")) {
                System.out.println("\nPlease enter a valid String");
                exit(1);
            } else
                System.out.println("Key pressed..");
            notifyAll();
        }
    }
}

public class Ques14 {
    public static void main(String[] args) {

        Processor5 obj1 = new Processor5();
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    obj1.produce();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }, "First");
        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    obj1.produce();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }, "Second");

        Thread t3 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    obj1.consume();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        t1.start();
        t2.start();
        t3.start();

        try {
            t1.join();
            t2.join();
            t3.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("All threads are notified");
    }
}
```

OUTPUT



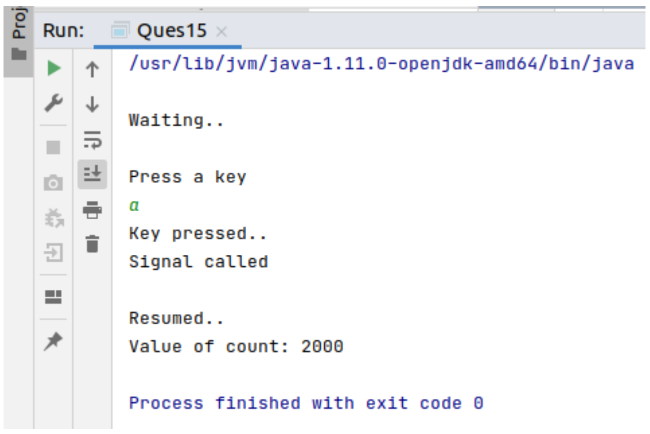
```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/b

Running Producer Thread...

Running Producer Thread...
Press a key to continue...
a
Key pressed..
Thread Resumed..
Thread Resumed..
All threads are notified

Process finished with exit code 0
```

15. Use Reentract lock for coordinating 2 threads with signal(), signalAll() and wait().

CODE

```
class Runner {
    int count = 0;
    private Lock lock = new ReentrantLock();
    private Condition cond = lock.newCondition();

    private void increment() {
        for (int i = 0; i < 1000; i++) {
            count++;
        }
    }

    public void firstThread() throws InterruptedException {
        lock.lock();
        System.out.println("\nWaiting..");
        cond.await();
        System.out.println("\nResumed..");
        try {
            increment();
        } finally {
            lock.unlock();
        }

    }

    public void SecondThread() throws InterruptedException {
        Thread.sleep(1000);
        lock.lock();
        System.out.println("\nPress a key");
        Scanner scanner = new Scanner(System.in);
        String str = scanner.nextLine();
        if (str.equals(" ")) {
            System.out.println("\nPlease enter a valid String");
            exit(1);
        } else
            System.out.println("Key pressed..");
        cond.signal();
        /*Here using signal() and signalAll() will not make any difference in the program
and its output
         **Because we are using only 2 threads as mentioned in the question
         **So the output will remain same for both of them in our case
         **Hence no separate execution is performed for signalAll()*/

        //cond.signalAll();
        System.out.println("Signal called");
        try {
            increment();
        } finally {
            lock.unlock();
        }
    }
}

public class Ques15 {
    public static void main(String[] args) {
        Runner r1 = new Runner();
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    r1.firstThread();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    r1.SecondThread();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        t1.start();
        t2.start();

        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Value of count: " + r1.count);
    }
}
```

OUTPUT



```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java

Waiting..

Press a key
a
Key pressed..
Signal called

Resumed..
Value of count: 2000

Process finished with exit code 0
```

16. Create a deadlock and Resolve it using tryLock().

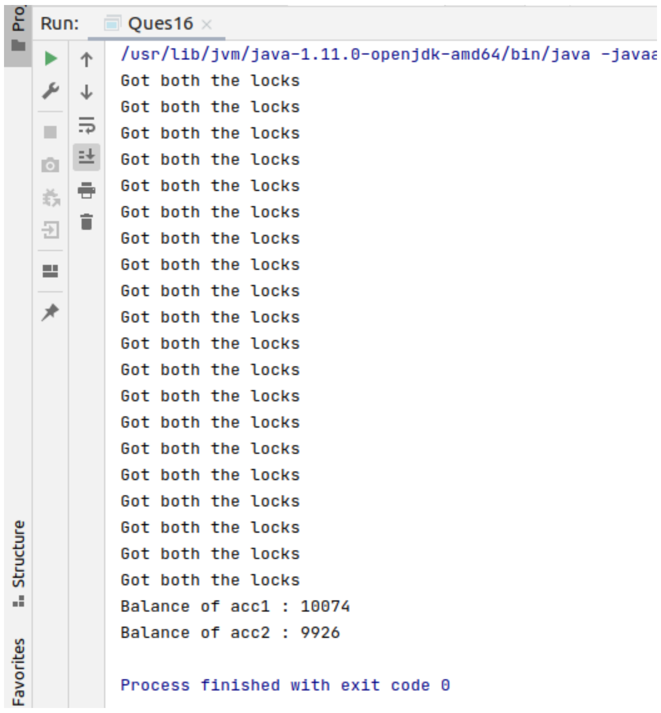
CODE

```
class Account {
   private int balance = 10000;

   private void deposit(int amount) {
      balance += amount;
   }

   private void withdraw(int amount) {
      if (balance < 500)
         System.out.println("\nCan not withdraw..");
      else
         balance -= amount;
   }

   public int getBalance() {
      return balance;
   }

   public static void transfer(Account acc1, Account acc2, int amount) {
      acc1.withdraw(amount);
      acc2.deposit(amount);
   }
}

class DemoLock {
   private Account acc1 = new Account();
   private Account acc2 = new Account();

   private Lock lock1 = new ReentrantLock();
   private Lock lock2 = new ReentrantLock();

   DemoLock(Account acc1, Account acc2) {
      this.acc1 = acc1;
      this.acc2 = acc2;
   }

   private void acquirelocks(Lock firstLock, Lock secondLock) throws
InterruptedException {
      while (true) {
         boolean gotFirstLock = false;
         boolean gotSecondLock = false;
         try {
            gotFirstLock = firstLock.tryLock();
```

```
                gotSecondLock = secondLock.tryLock();
                System.out.println("Got both the locks");

            } finally {
                if (gotFirstLock && gotSecondLock)
                    return;
                if (gotFirstLock) {
                    System.out.println("Release lock1");
                    firstLock.unlock();
                }
                if (gotSecondLock) {
                    System.out.println("Release lock2");
                    secondLock.unlock();
                }
            }
            Thread.sleep(100);
        }
    }

    public void firstThread() throws InterruptedException {
        Random random = new Random();
        for (int i = 0; i < 10; i++) {
            acquirelocks(lock1, lock2);

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

    public void secondThread() throws InterruptedException {
        Random random = new Random();
        for (int i = 0; i < 10; i++) {
            acquirelocks(lock1, lock2);
            try {
                Account.transfer(acc2, acc1, random.nextInt(100));
                Thread.sleep(1000);
            } finally {
                lock1.unlock();
                lock2.unlock();
            }
        }
    }

}
```

```
public class Ques16 {
   public static void main(String[] args) {

      Account acc1 = new Account();
      Account acc2 = new Account();
      DemoLock ty = new DemoLock(acc1, acc2);
      Thread t1 = new Thread(new Runnable() {
         @Override
         public void run() {
            try {
               ty.firstThread();
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
         }
      });
      Thread t2 = new Thread(new Runnable() {
         @Override
         public void run() {
            try {
               ty.secondThread();
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
         }
      });
      t1.start();
      t2.start();
      try {
         t1.join();
         t2.join();
      } catch (InterruptedException e) {
         e.printStackTrace();
      }

      System.out.println("Balance of acc1 : " + acc1.getBalance() + "\nBalance of acc2 : "
+ acc2.getBalance());

   }
}
```

OUTPUT



```
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java -javaa
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Got both the locks
Balance of acc1 : 10074
Balance of acc2 : 9926

Process finished with exit code 0
```