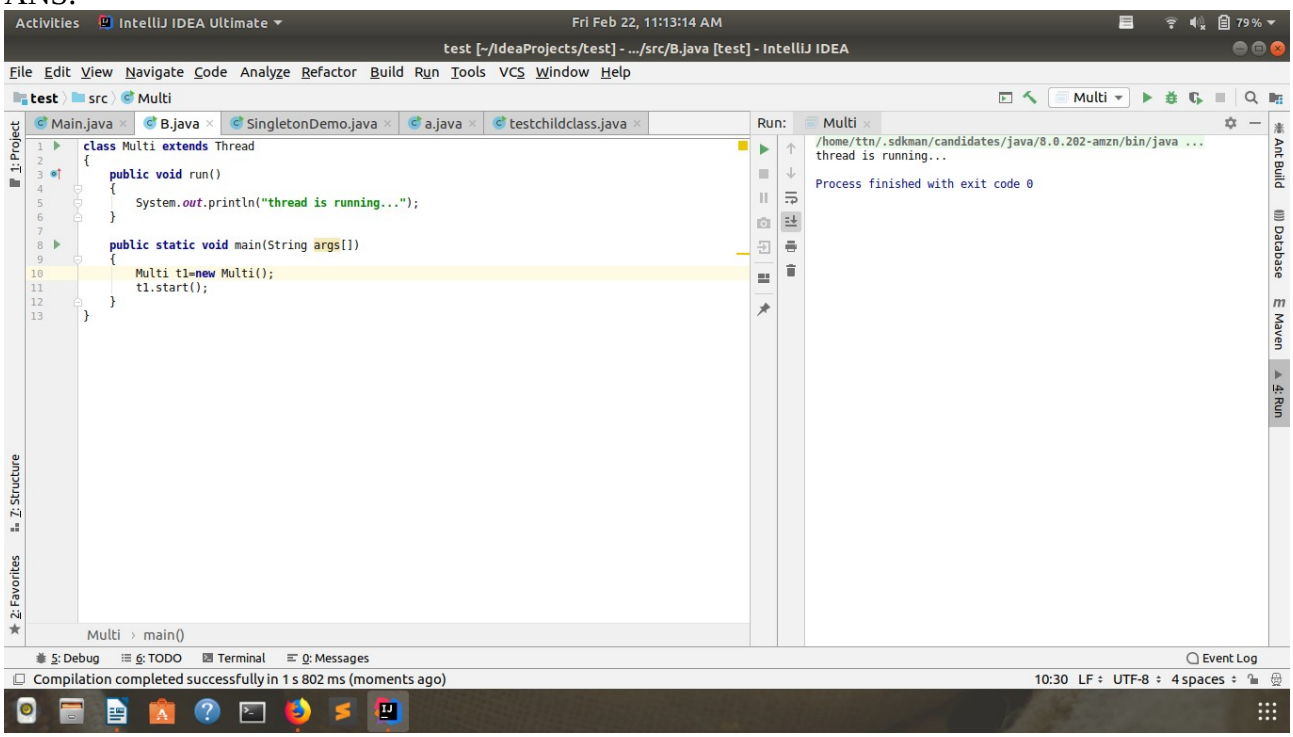
MULTITHREADING

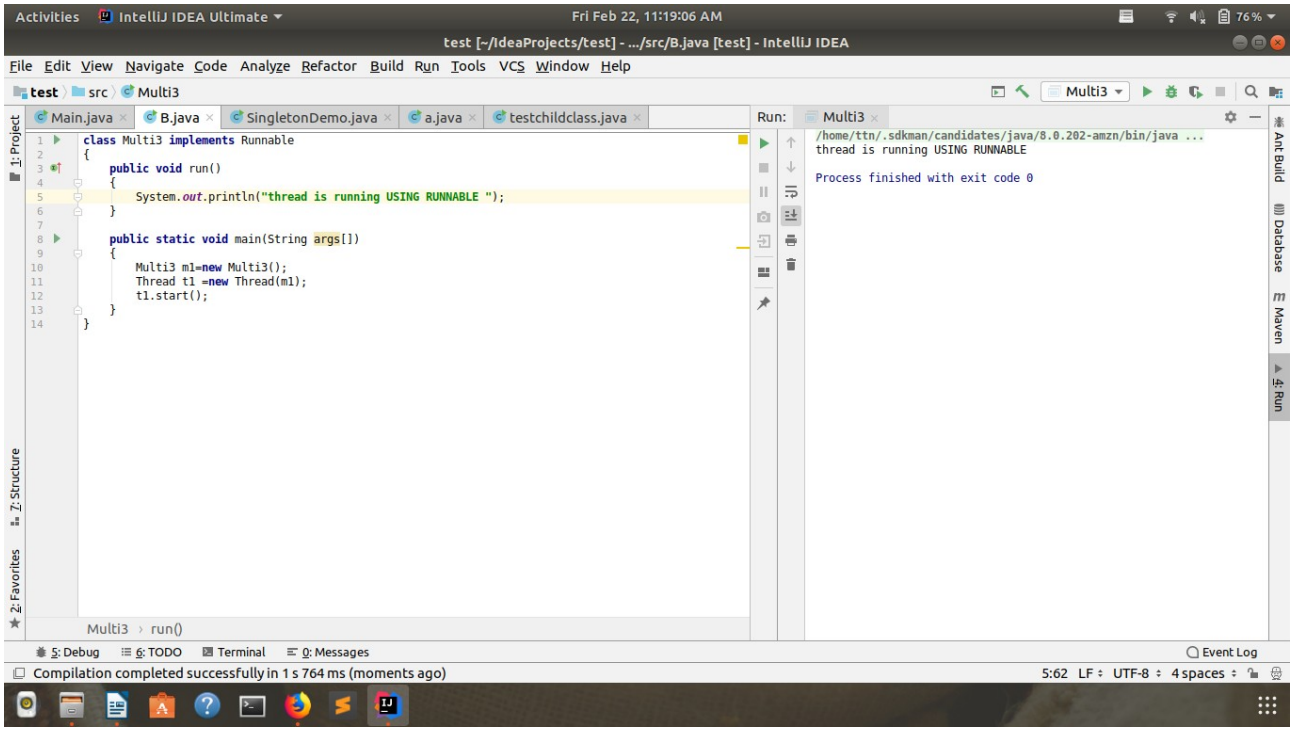
1. Create and Run a Thread using Runnable Interface and Thread class.
2. Use sleep and join methods with thread.
3. Use a singleThreadExecutor to submit multiple threads.
4. Try shutdown() and shutdownNow() and observe the difference.
5. Use isShutDown() and isTerminate() with ExecutorService.
6. Return a Future from ExecutorService by using callable and use get(), isDone(), isCancelled() with the Future object to know the status of task submitted.
7. Submit List of tasks to ExecutorService and wait for the completion of all the tasks.
8. Schedule task using schedule(), scheduleAtFixedRate() and scheduleAtFixedDelay()
9. Increase concurrency with Thread pools using newCachedThreadPool() and newFixedThreadPool().
10. Use Synchronize method to enable synchronization between multiple threads trying to access method at same time.
11. Use Synchronize block to enable synchronization between multiple threads trying to access method at same time.
12. Use Atomic Classes instead of Synchronize method and blocks.
13. Coordinate 2 threads using wait() and notify().
14. Coordinate mulitple threads using wait() and notifyAll()
15. Use Reentract lock for coordinating 2 threads with signal(), signalAll() and wait().
16. Create a deadlock and Resolve it using tryLock().

ANSWERS

Q1]Create and Run a Thread using Runnable Interface and Thread class.
ANS.

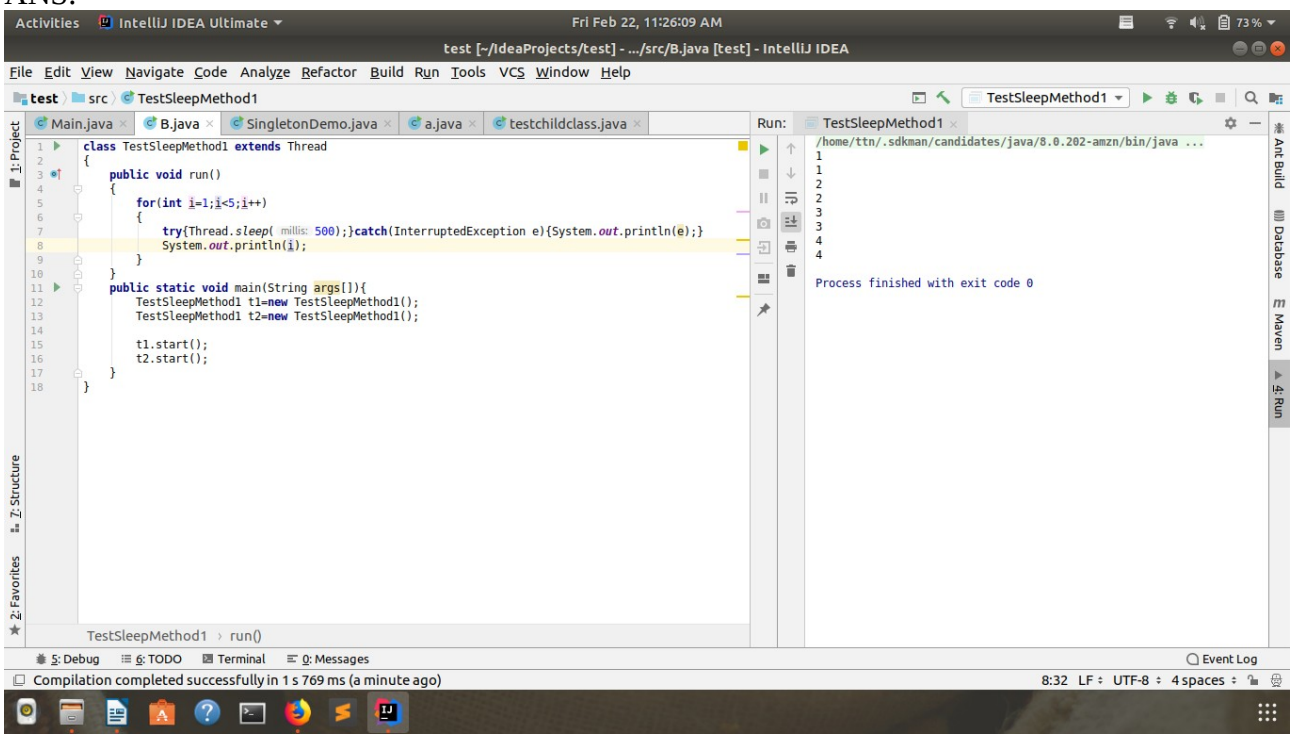


USING RUNNABLE INTERFACE

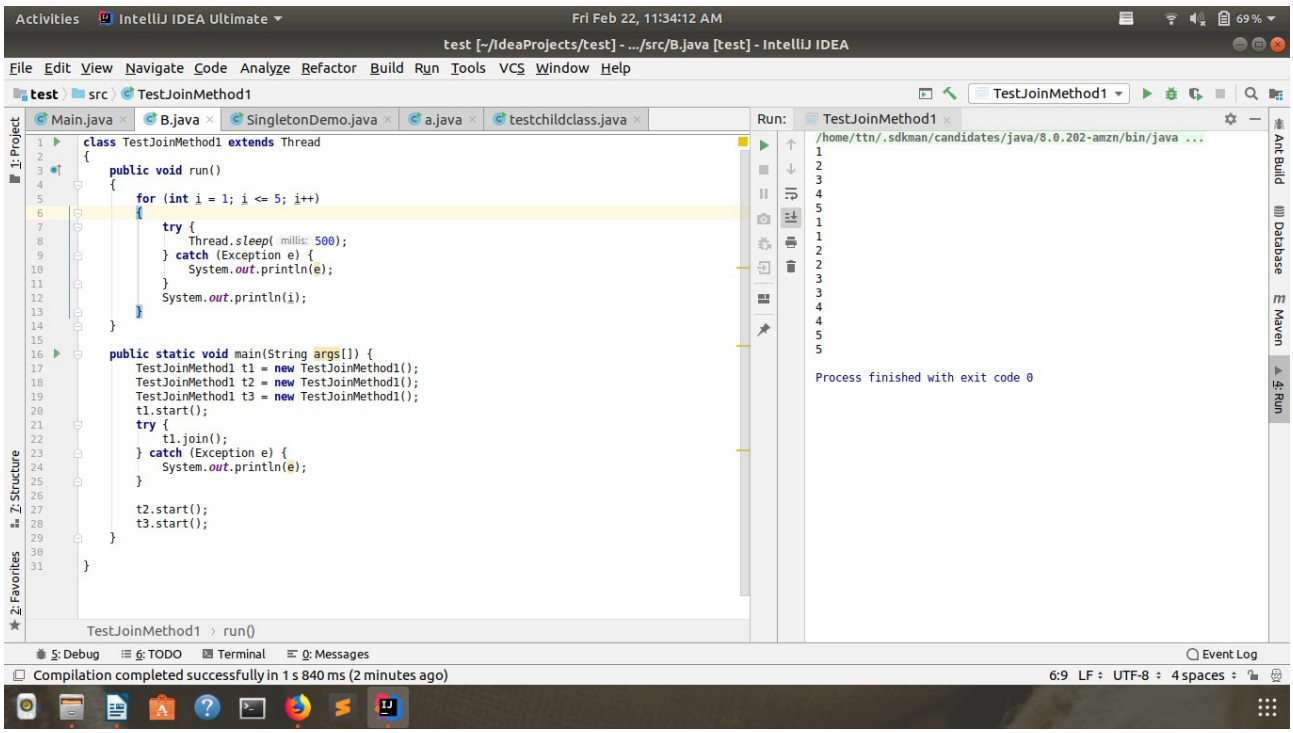


Q2.Use sleep and join methods with thread.

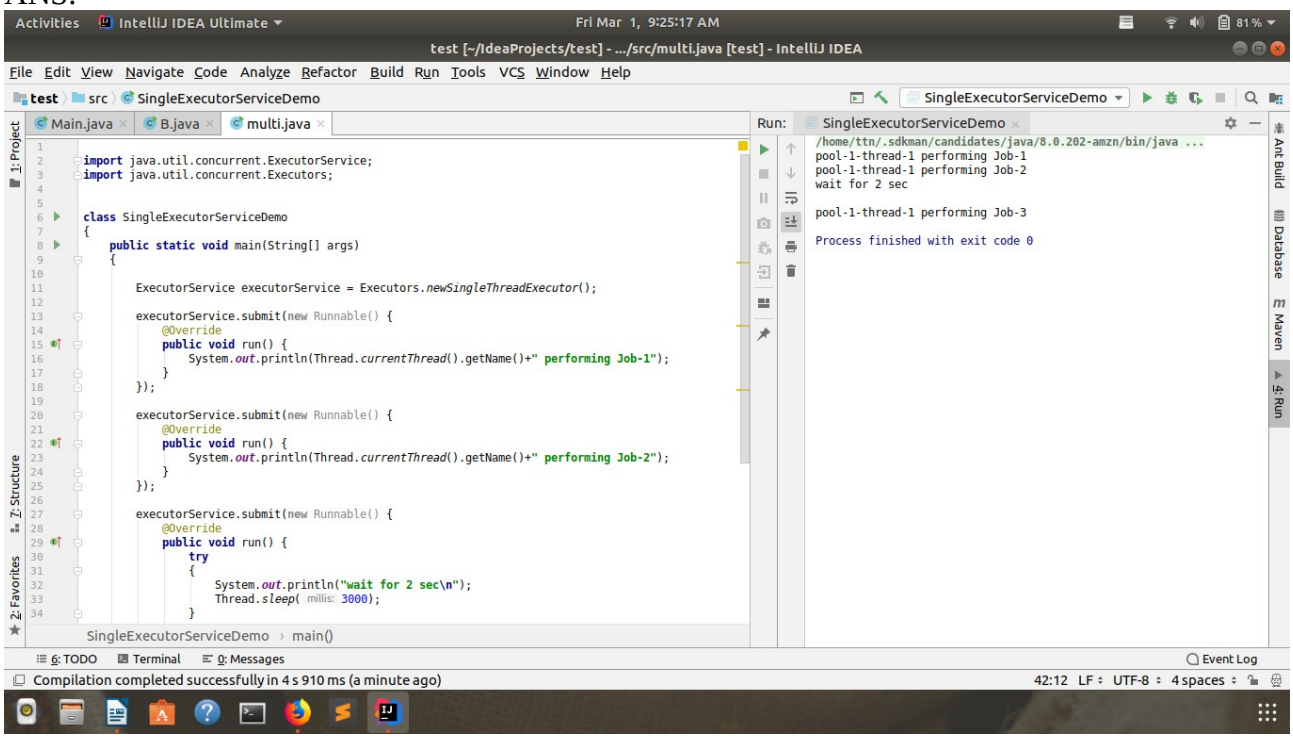
ANS.



Using join



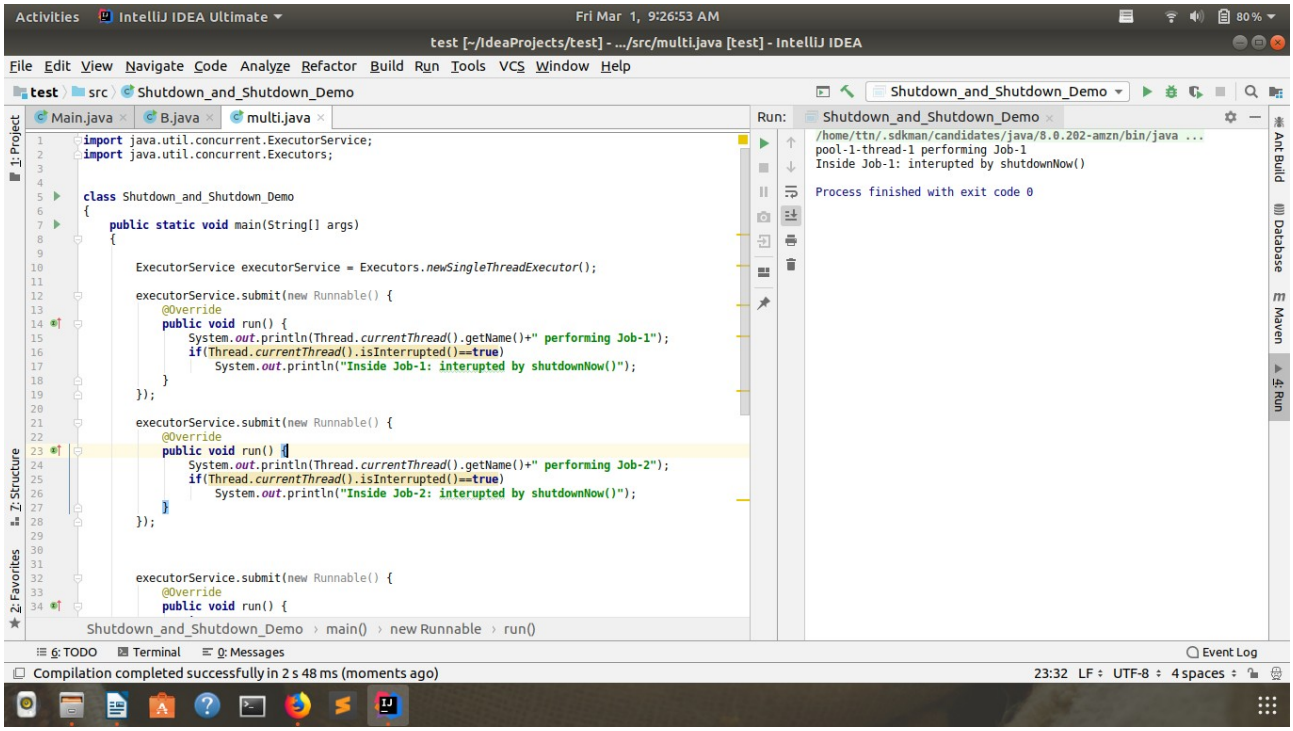
Q3.Use a singleThreadExecutor to submit multiple threads.
ANS.




```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class SingleExecutorServiceDemo
{
    public static void main(String[] args)
    {
        ExecutorService executorService = Executors.newSingleThreadExecutor();
        executorService.submit(new Runnable() {
            @Override
            public void run() {
```

```
                System.out.println(Thread.currentThread().getName()+" performing Job-
1");
            }
        });
        executorService.submit(new Runnable() {
            @Override
            public void run() {
                System.out.println(Thread.currentThread().getName()+" performing Job-
2");
            }
        });
        executorService.submit(new Runnable() {
            @Override
            public void run() {
                try
                {
                    System.out.println("wait for 2 sec\n");
                    Thread.sleep(3000);
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
                System.out.println(Thread.currentThread().getName()+" performing Job-
3");
            }
        });
        executorService.shutdown();
    }
}
```

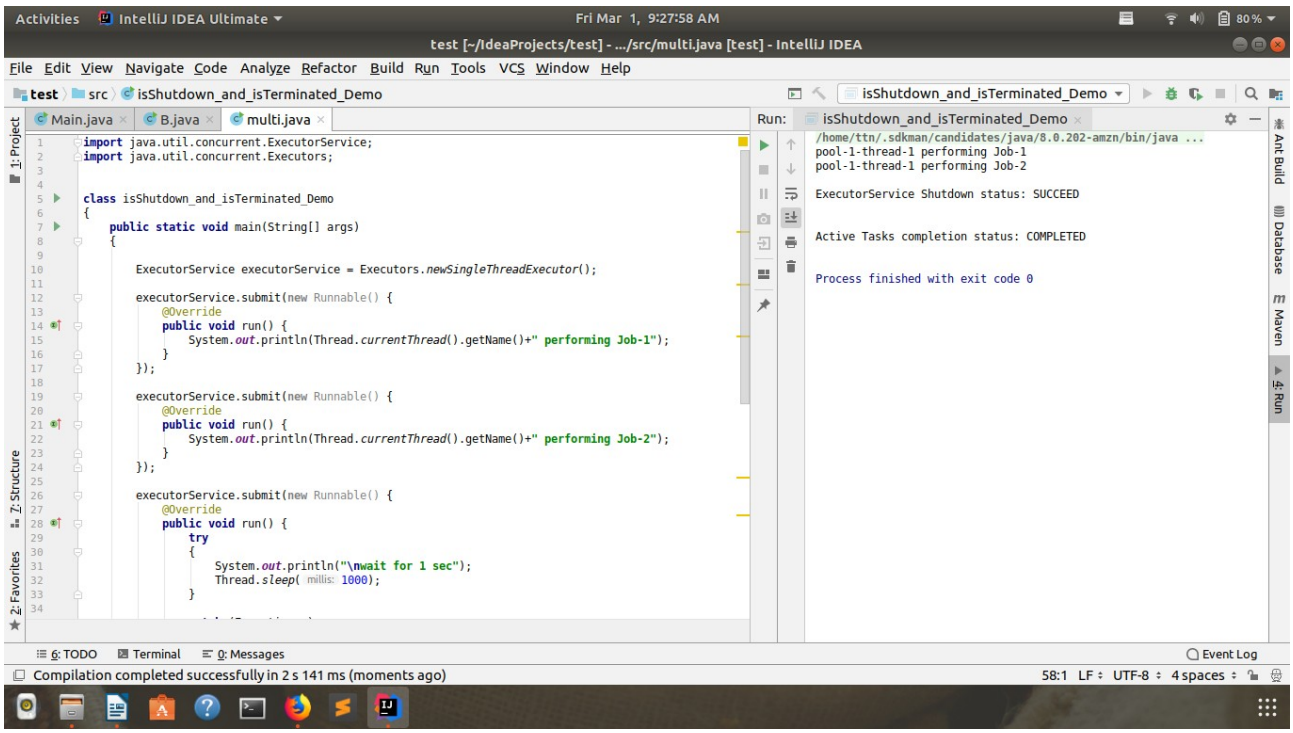
Q4.Try shutdown() and shutdownNow() and observe the difference.
ANS.




```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class Shutdown_and_Shutdown_Demo
{
```

```
    public static void main(String[] args)
    {
        ExecutorService executorService = Executors.newSingleThreadExecutor();
        executorService.submit(new Runnable() {
            @Override
            public void run() {
                System.out.println(Thread.currentThread().getName()+" performing Job-
1");
                if(Thread.currentThread().isInterrupted()==true)
                    System.out.println("Inside Job-1: interupted by shutdownNow()");
            }
        });
        executorService.submit(new Runnable() {
            @Override
            public void run() {
                System.out.println(Thread.currentThread().getName()+" performing Job-
2");
                if(Thread.currentThread().isInterrupted()==true)
                    System.out.println("Inside Job-2: interupted by shutdownNow()");
            }
        });
        executorService.submit(new Runnable() {
            @Override
            public void run() {
                try
                {
                    System.out.println("wait for 1 sec\n");
                    Thread.sleep(1000);
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
                System.out.println(Thread.currentThread().getName()+" performing Job-
3");
                if(Thread.currentThread().isInterrupted()==true)
                    System.out.println("Inside Job-3: interupted by shutdownNow()");
            }
        });
        executorService.shutdownNow();
        executorService.shutdown();
    }
}
```

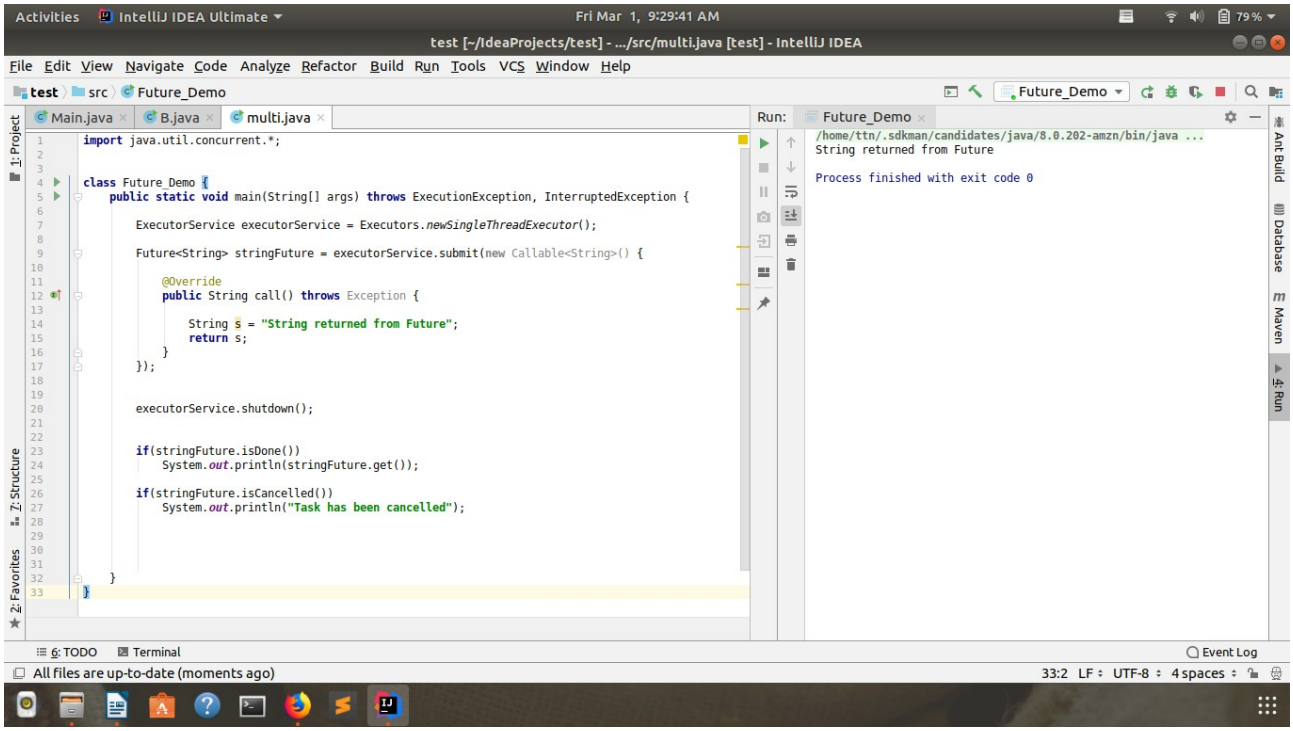
Q5.Use isShutDown() and isTerminate() with ExecutorService.
ANS.



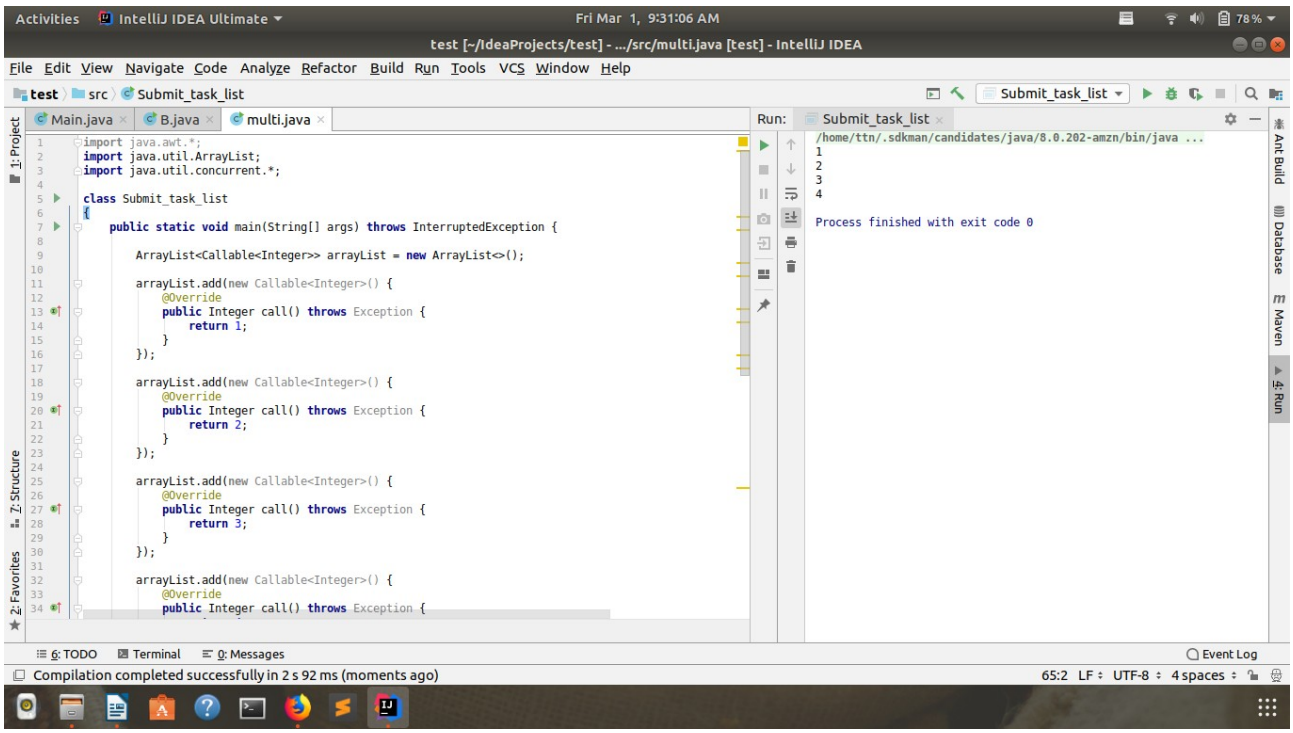

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class isShutdown_and_isTerminated_Demo
{
     public static void main(String[] args)
     {
          ExecutorService executorService = Executors.newSingleThreadExecutor();
          executorService.submit(new Runnable() {
               @Override
               public void run() {
                    System.out.println(Thread.currentThread().getName()+" performing Job-
1");
               }
          });
          executorService.submit(new Runnable() {
               @Override
               public void run() {
                    System.out.println(Thread.currentThread().getName()+" performing Job-
2");
               }
          });
          executorService.submit(new Runnable() {
               @Override
               public void run() {
                    try
                    {
                         System.out.println("\nwait for 1 sec");
                         Thread.sleep(1000);
                    }
                    catch (Exception e)
                    {
                         System.err.println("SLEEP Exception Caughed");
                    }
                    System.out.println(Thread.currentThread().getName()+" performing Job-
3");
               }
          });
          executorService.shutdown();
```

```
            executorService.shutdownNow();
            if(executorService.isShutdown()==true)
                System.out.println("\nExecutorService Shutdown status: SUCCEED\n");
            else
                System.out.println("\nExecutorService Shutdown status: FAILED\n");
            if(executorService.isTerminated()==true)
                System.out.println("\nActive Tasks completion status: COMPLETED\n");
            else
                System.out.println("\nActive Tasks completion status: INTERUPTED\n");
        }
}
```

Q6.Return a Future from ExecutorService by using callable and use get(), isDone(), isCancelled() with the Future object to know the status of task submitted.
ANS.



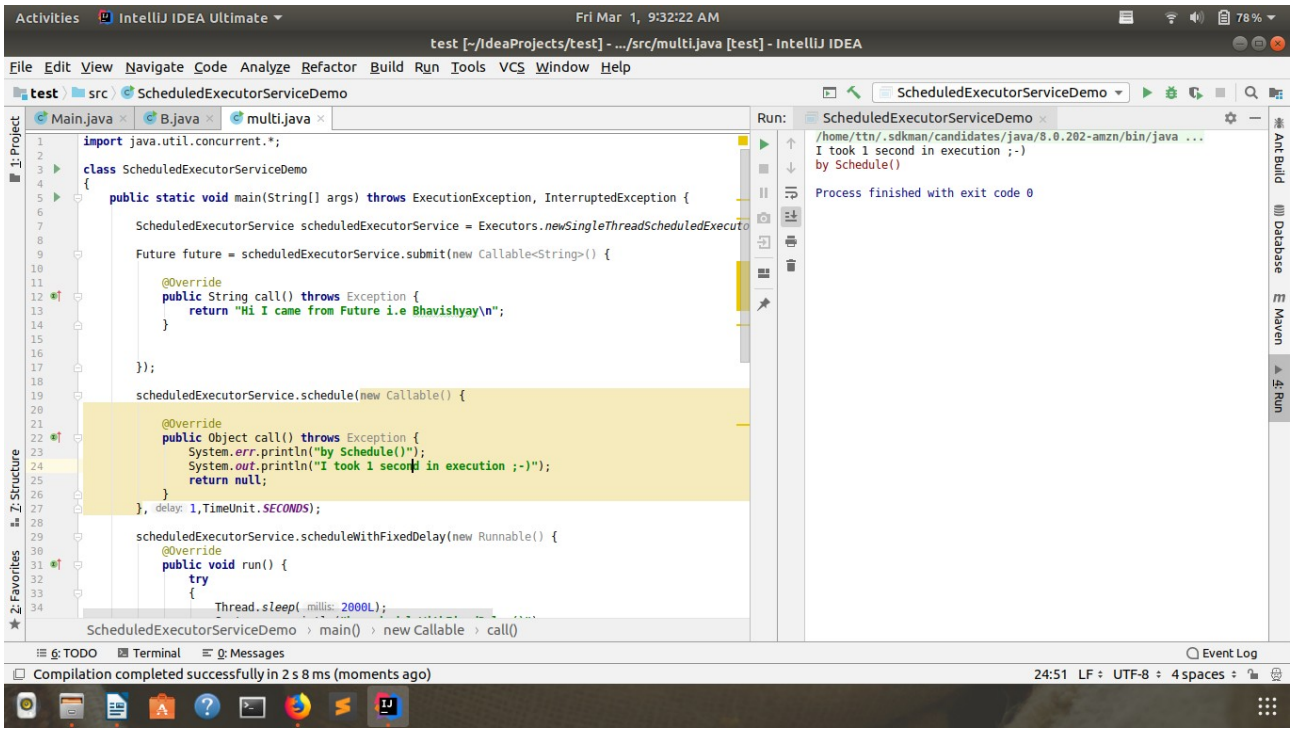
Q7.Submit List of tasks to ExecutorService and wait for the completion of all the tasks.
ANS.




```
import java.awt.*;
import java.util.ArrayList;
import java.util.concurrent.*;
class Submit_task_list
{
     public static void main(String[] args) throws InterruptedException {
          ArrayList<Callable<Integer>> arrayList = new ArrayList<>();
          arrayList.add(new Callable<Integer>() {
               @Override
               public Integer call() throws Exception {
                    return 1;
               }
          });
          arrayList.add(new Callable<Integer>() {
               @Override
               public Integer call() throws Exception {
                    return 2;
               }
          });
          arrayList.add(new Callable<Integer>() {
               @Override
               public Integer call() throws Exception {
                    return 3;
               }
          });
          arrayList.add(new Callable<Integer>() {
               @Override
               public Integer call() throws Exception {
                    return 4;
               }
          });
          ExecutorService executorService = Executors.newSingleThreadExecutor();
          ArrayList<Future<Integer>> futureArrayList = (ArrayList<Future<Integer>>)
executorService.invokeAll(arrayList);
          futureArrayList.forEach((e)->{
               if(e.isDone())
               {
                    try {
```

```
                    System.out.println(e.get());
                }
                catch (ExecutionException el)
                {
                    el.printStackTrace();
                }
                catch (InterruptedException e1)
                {
                    e1.printStackTrace();
                }
            }
        });
        executorService.shutdown();
    }
}
```

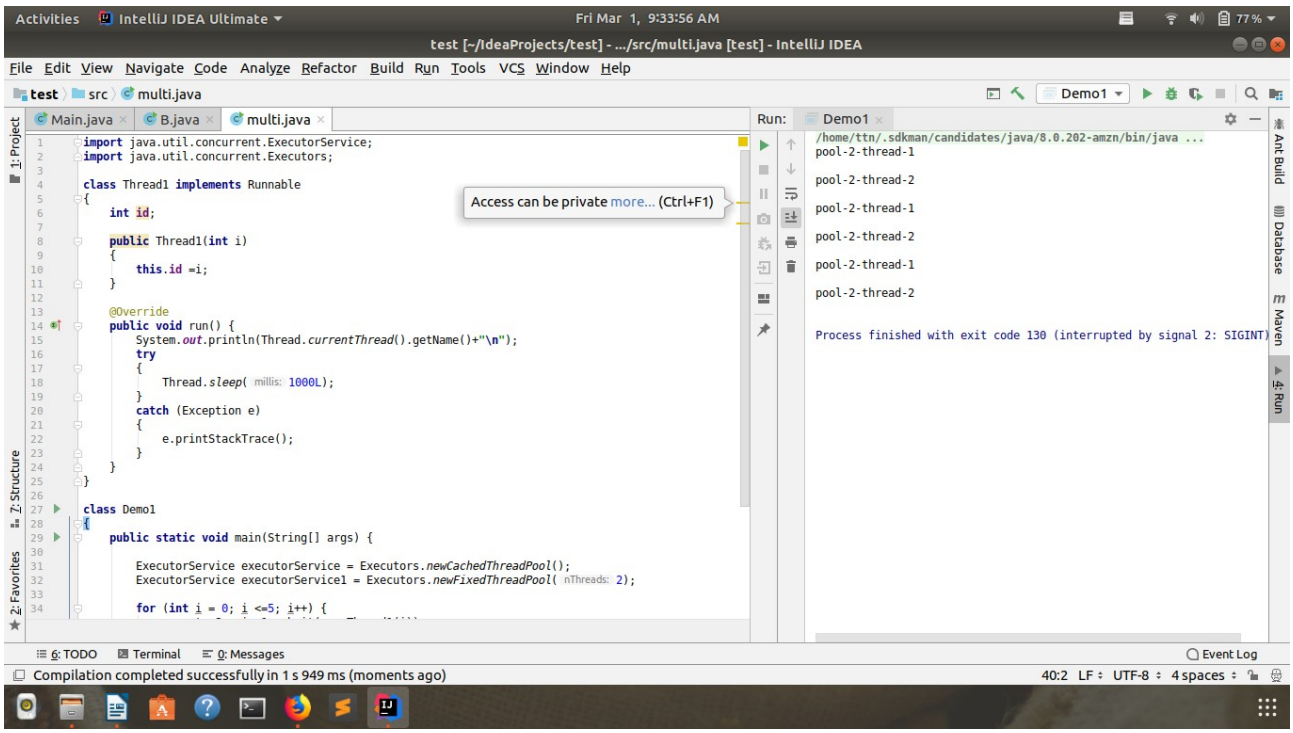
Q8.Schedule task using schedule(), scheduleAtFixedRate() and scheduleAtFixedDelay()
ANS.




```
import java.util.concurrent.*;
class ScheduledExecutorServiceDemo
{
    public static void main(String[] args) throws ExecutionException,
InterruptedException {
        ScheduledExecutorService scheduledExecutorService =
Executors.newSingleThreadScheduledExecutor();
        Future future = scheduledExecutorService.submit(new Callable<String>() {
            @Override
            public String call() throws Exception {
                return "Hi I came from Future i.e Bhavishyay\n";
            }
        });
        scheduledExecutorService.schedule(new Callable() {
            @Override
            public Object call() throws Exception {
                System.err.println("by Schedule()");
                System.out.println("I took 1 second in execution ;-)");
                return null;
```

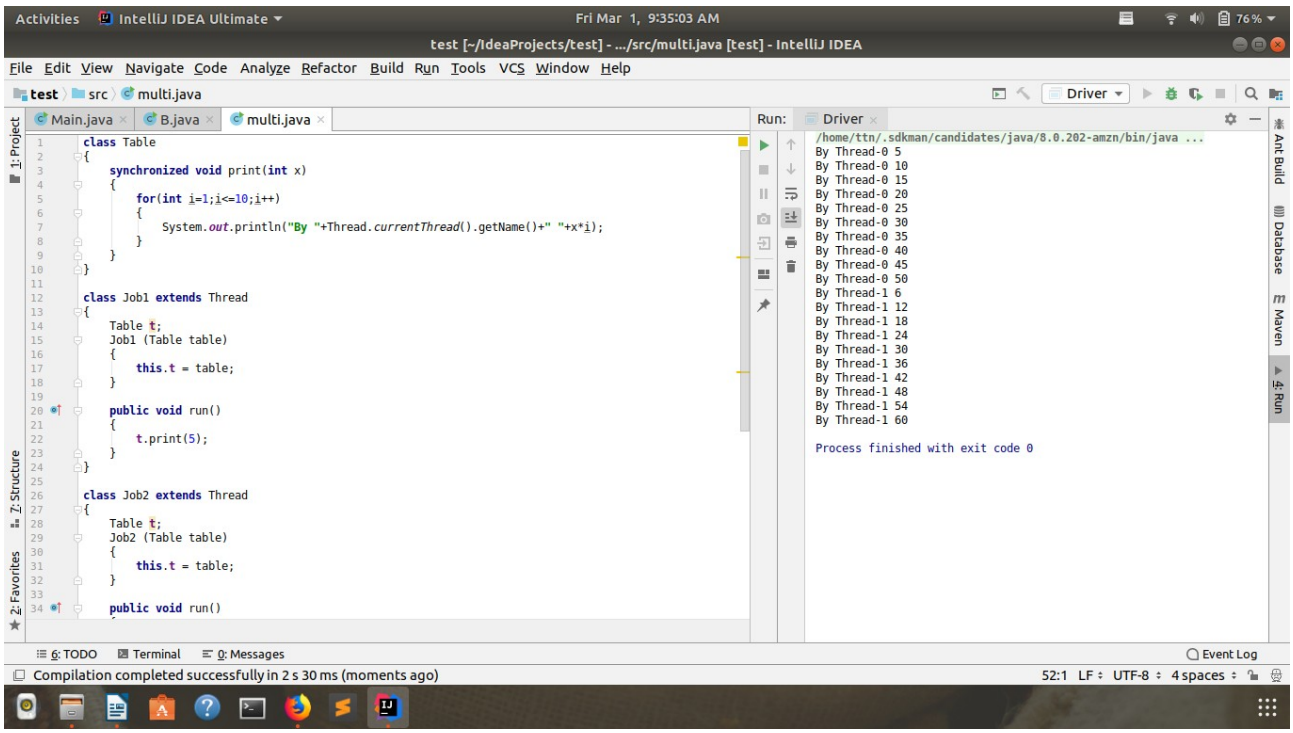
```
            }
        },1,TimeUnit.SECONDS);
        scheduledExecutorService.scheduleWithFixedDelay(new Runnable() {
            @Override
            public void run() {
                try
                {
                    Thread.sleep(2000L);
                    System.err.println("by scheduleWithFixedDelay()");
                    System.out.println("ScheduleWithFixedDelay Scheduled Task to
executed after fixed interval\n");
                }
                catch (InterruptedException e)
                {
                    e.printStackTrace();
                }
            }
        },0,1,TimeUnit.SECONDS);
        scheduledExecutorService.scheduleAtFixedRate(new Runnable() {
                                                                @Override
                                                                public void run() {
                                                                    try {

Thread.sleep(2000L);

System.err.println("by scheduleWithFixedRate()");

System.out.println("ScheduleAtFixedRate Scheduled Task to executed after fixed
interval");
                                                                    } catch
(InterruptedException e) {

e.printStackTrace();
                                                                    }
                                                                }
                                                            }
                ,
                0,
                1,
                TimeUnit.SECONDS);
        if(future.isDone())
            System.out.println(future.get());
        scheduledExecutorService.shutdown();
    }
}
```

Q9.Increase concurrency with Thread pools using newCachedThreadPool() and newFixedThreadPool().
ANS.




```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class Thread1 implements Runnable
{
     int id;
     public Thread1(int i)
     {
          this.id =i;
     }
     @Override
     public void run() {
          System.out.println(Thread.currentThread().getName()+"\n");
          try
          {
               Thread.sleep(1000L);
          }
          catch (Exception e)
          {
               e.printStackTrace();
          }
     }
}
class Demo1
{
     public static void main(String[] args) {
          ExecutorService executorService = Executors.newCachedThreadPool();
          ExecutorService executorService1 = Executors.newFixedThreadPool(2);
          for (int i = 0; i <=5; i++) {
               executorService1.submit(new Thread1(i));
          }
          executorService.shutdown();
     }
}
```

Q10.Use Synchronize method to enable synchronization between multiple threads trying to access method at same time.
ANS.

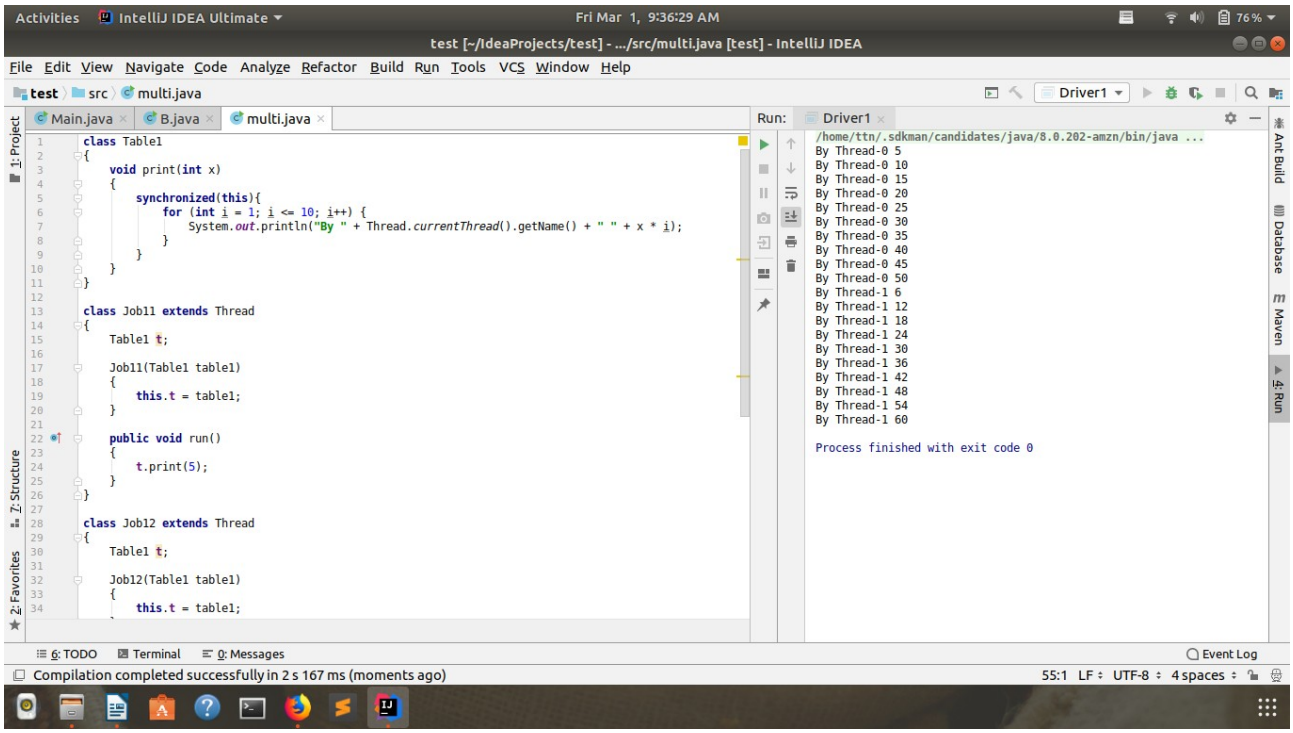



```
class Table
{
     synchronized void print(int x)
     {
          for(int i=1;i<=10;i++)
          {
               System.out.println("By "+Thread.currentThread().getName()+" "+x*i);
          }
     }
}
class Job1 extends Thread
{
     Table t;
     Job1 (Table table)
     {
          this.t = table;
     }
     public void run()
     {
          t.print(5);
     }
}
class Job2 extends Thread
{
     Table t;
     Job2 (Table table)
     {
          this.t = table;
     }
     public void run()
     {
          t.print(6);
     }
}
class Driver
{
     public static void main(String[] args) {
          Table t = new Table();
          Thread j1 = new Job1(t);
```

```
        Thread j2 = new Job2(t);
        //j1.start();
        j2.start();
        j1.start();
    }
}
```

Q11.Use Synchronize block to enable synchronization between multiple threads trying to access method at same time.

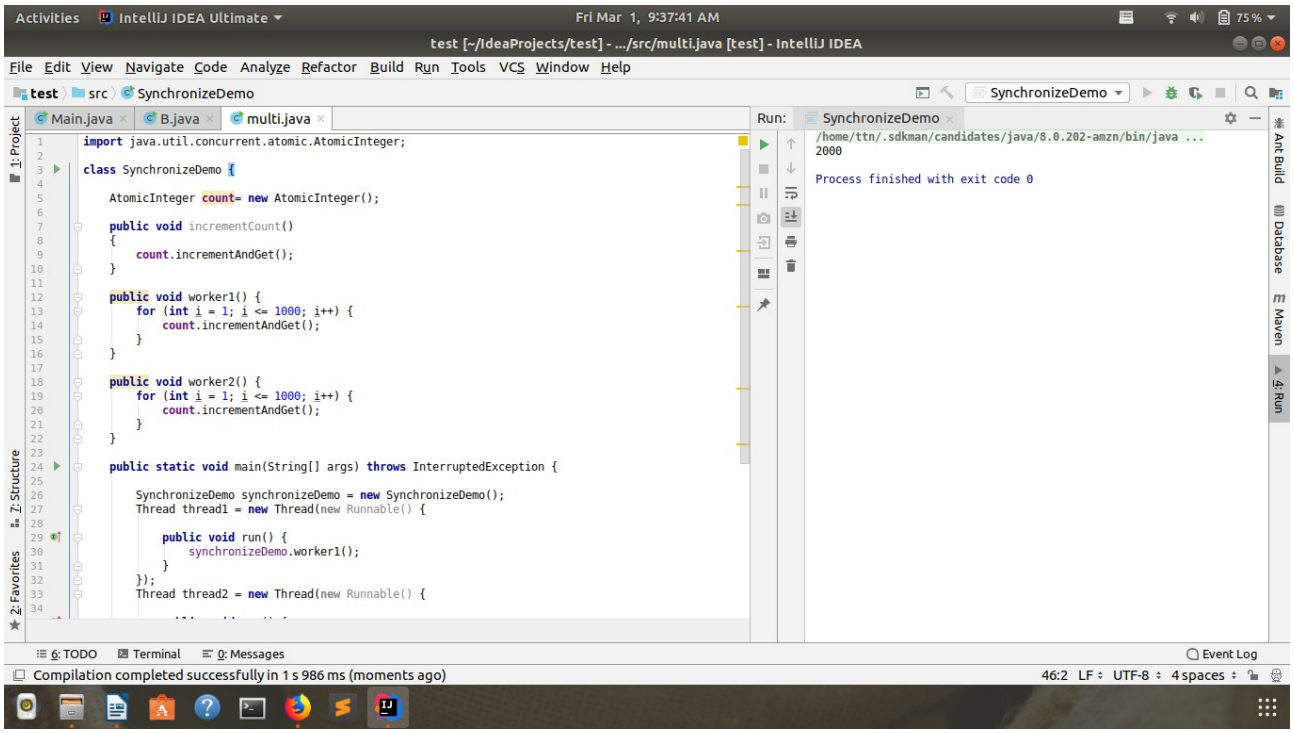
ANS.




```
class Table1
{
    void print(int x)
    {
        synchronized(this){
            for (int i = 1; i <= 10; i++) {
                System.out.println("By " + Thread.currentThread().getName() + " " + x *
i);
            }
        }
    }
}
class Job11 extends Thread
{
    Table1 t;
    Job11(Table1 table1)
    {
        this.t = table1;
    }
    public void run()
    {
        t.print(5);
    }
}
class Job12 extends Thread
{
    Table1 t;
    Job12(Table1 table1)
```

```
    {
        this.t = table1;
    }
    public void run()
    {
        t.print(6);
    }
}
class Driver1
{
    public static void main(String[] args) {
        Table1 t = new Table1();
        Thread j1 = new Job11(t);
        Thread j2 = new Job12(t);
        //j1.start();
        j2.start();
        j1.start();
    }
}
```

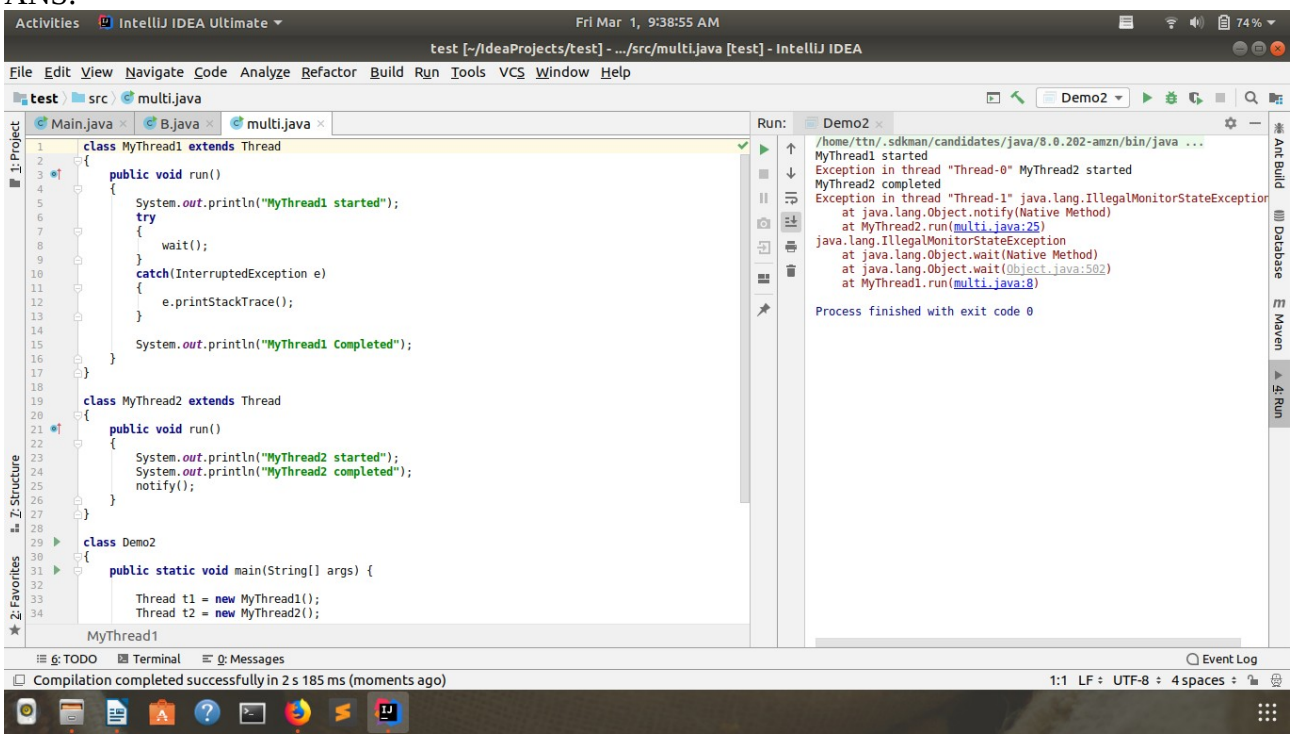
Q12.Use Atomic Classes instead of Synchronize method and blocks.
ANS.




```
import java.util.concurrent.atomic.AtomicInteger;
class SynchronizeDemo {
    AtomicInteger count= new AtomicInteger();
    public void incrementCount()
    {
        count.incrementAndGet();
    }
    public void worker1() {
        for (int i = 1; i <= 1000; i++) {
            count.incrementAndGet();
        }
    }
    public void worker2() {
        for (int i = 1; i <= 1000; i++) {
            count.incrementAndGet();
        }
```

```
    }
    public static void main(String[] args) throws InterruptedException {
        SynchronizeDemo synchronizeDemo = new SynchronizeDemo();
        Thread thread1 = new Thread(new Runnable() {
            public void run() {
                synchronizeDemo.worker1();
            }
        });
        Thread thread2 = new Thread(new Runnable() {
            public void run() {
                synchronizeDemo.worker2();
            }
        });
        thread1.start();
        thread2.start();
        thread1.join();
        thread2.join();
        System.out.println(synchronizeDemo.count);
    }
}
```

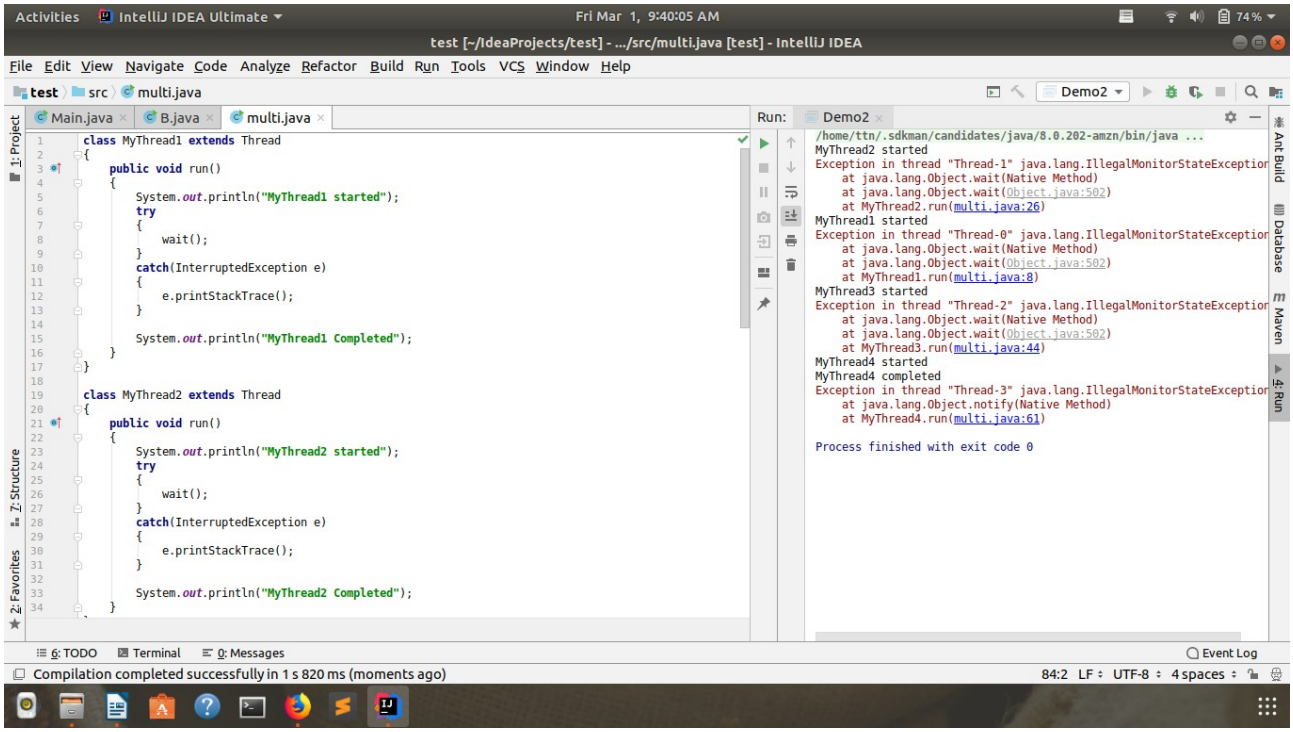
Q13.Coordinate 2 threads using wait() and notify().
ANS.




```
class MyThread1 extends Thread
{
    public void run()
    {
        System.out.println("MyThread1 started");
        try
        {
            wait();
        }
        catch(InterruptedException e)
        {
            e.printStackTrace();
        }
        System.out.println("MyThread1 Completed");
    }
```

```
}
class MyThread2 extends Thread
{
     public void run()
     {
          System.out.println("MyThread2 started");
          System.out.println("MyThread2 completed");
          notify();
     }
}
class Demo2
{
     public static void main(String[] args) {
          Thread t1 = new MyThread1();
          Thread t2 = new MyThread2();
          t1.start();
          t2.start();
     }
}
```

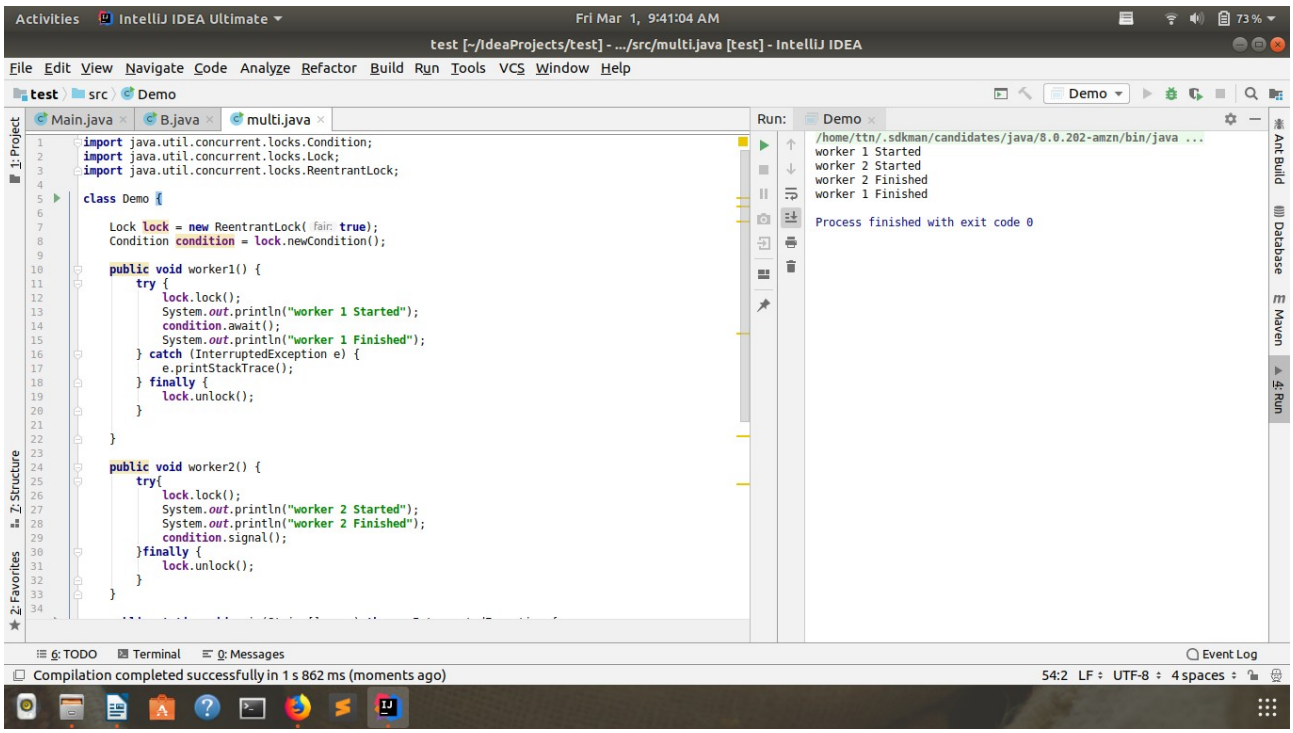
Q14.Coordinate mulitple threads using wait() and notifyAll()
ANS.




```
class MyThread1 extends Thread
{
     public void run()
     {
          System.out.println("MyThread1 started");
          try
          {
               wait();
          }
          catch(InterruptedException e)
          {
               e.printStackTrace();
          }
          System.out.println("MyThread1 Completed");
     }
}
class MyThread2 extends Thread
```

```
{
    public void run()
    {
        System.out.println("MyThread2 started");
        try
        {
            wait();
        }
        catch(InterruptedException e)
        {
            e.printStackTrace();
        }
        System.out.println("MyThread2 Completed");
    }
}
class MyThread3 extends Thread
{
    public void run()
    {
        System.out.println("MyThread3 started");
        try
        {
            wait();
        }
        catch(InterruptedException e)
        {
            e.printStackTrace();
        }
        System.out.println("MyThread3 Completed");
    }
}
class MyThread4 extends Thread
{
    public void run()
    {
        System.out.println("MyThread4 started");
        System.out.println("MyThread4 completed");
        notify();
    }
}
class Demo2
{
    public static void main(String[] args) {
        Thread t1 = new MyThread1();
        Thread t2 = new MyThread2();
        Thread t3 = new MyThread3();
        Thread t4 = new MyThread4();
        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }
}
```

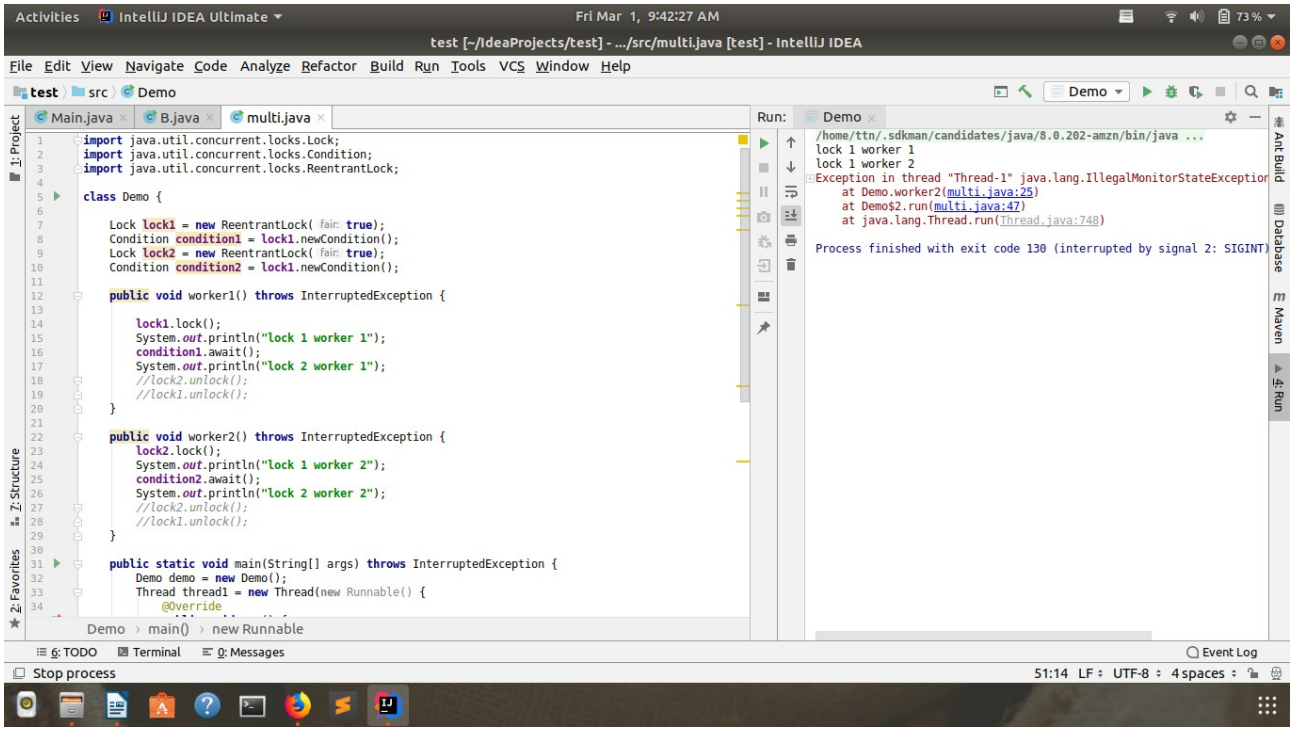
Q15.Use Reentract lock for coordinating 2 threads with signal(), signalAll() and wait().
ANS.




```java
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
class Demo {
     Lock lock = new ReentrantLock(true);
     Condition condition = lock.newCondition();
     public void worker1() {
          try {
               lock.lock();
               System.out.println("worker 1 Started");
               condition.await();
               System.out.println("worker 1 Finished");
          } catch (InterruptedException e) {
               e.printStackTrace();
          } finally {
               lock.unlock();
          }
     }
     public void worker2() {
          try{
               lock.lock();
               System.out.println("worker 2 Started");
               System.out.println("worker 2 Finished");
               condition.signal();
          }finally {
               lock.unlock();
          }
     }
     public static void main(String[] args) throws InterruptedException {
          Demo demo = new Demo();
          Thread thread1 = new Thread(new Runnable() {
               @Override
               public void run() {
                    demo.worker1();
               }
          });
          Thread thread2 = new Thread(new Runnable() {
               @Override
               public void run() {
```

```
                demo.worker2();
            }
        });
        thread1.start();
        thread2.start();
        thread1.join();
        thread2.join();
    }
}
```

Q16.Create a deadlock and Resolve it using tryLock().
ANS.




```
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;
class Demo {
    Lock lock1 = new ReentrantLock(true);
    Condition condition1 = lock1.newCondition();
    Lock lock2 = new ReentrantLock(true);
    Condition condition2 = lock1.newCondition();
    public void worker1() throws InterruptedException {
        lock1.lock();
        System.out.println("lock 1 worker 1");
        condition1.await();
        System.out.println("lock 2 worker 1");
        //lock2.unlock();
        //lock1.unlock();
    }
    public void worker2() throws InterruptedException {
        lock2.lock();
        System.out.println("lock 1 worker 2");
        condition2.await();
        System.out.println("lock 2 worker 2");
        //lock2.unlock();
        //lock1.unlock();
    }
    public static void main(String[] args) throws InterruptedException {
```

```
        Demo demo = new Demo();
        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    demo.worker1();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        Thread thread2 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    demo.worker2();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        thread1.start();
        thread2.start();
        thread1.join();
        thread2.join();
    }
}
```