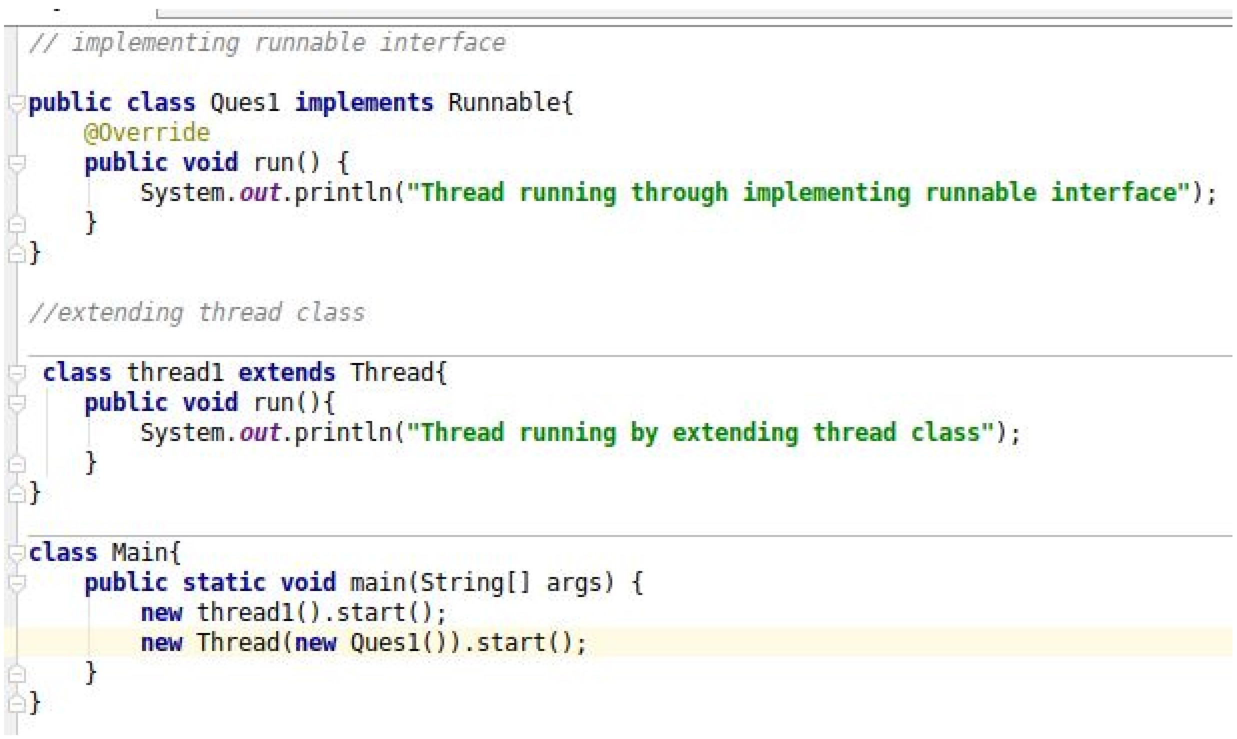
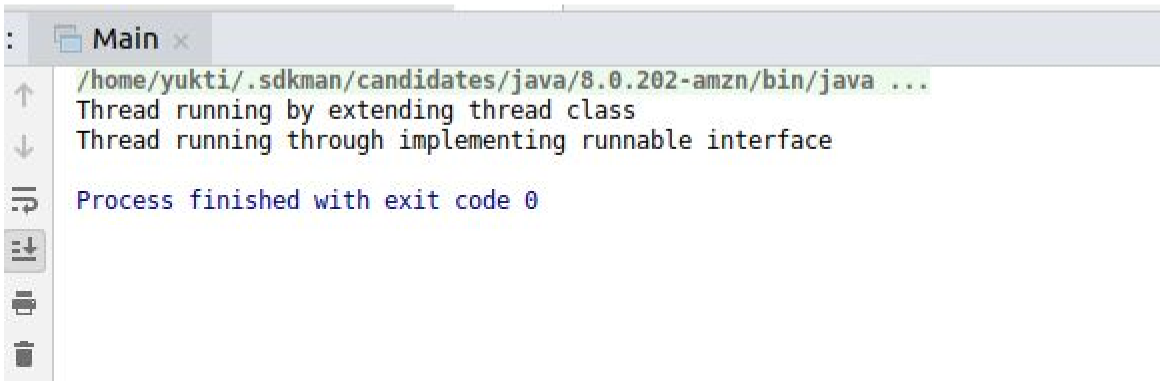
BASICS OF MULTI THREADING

YUKTI SHARMA

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

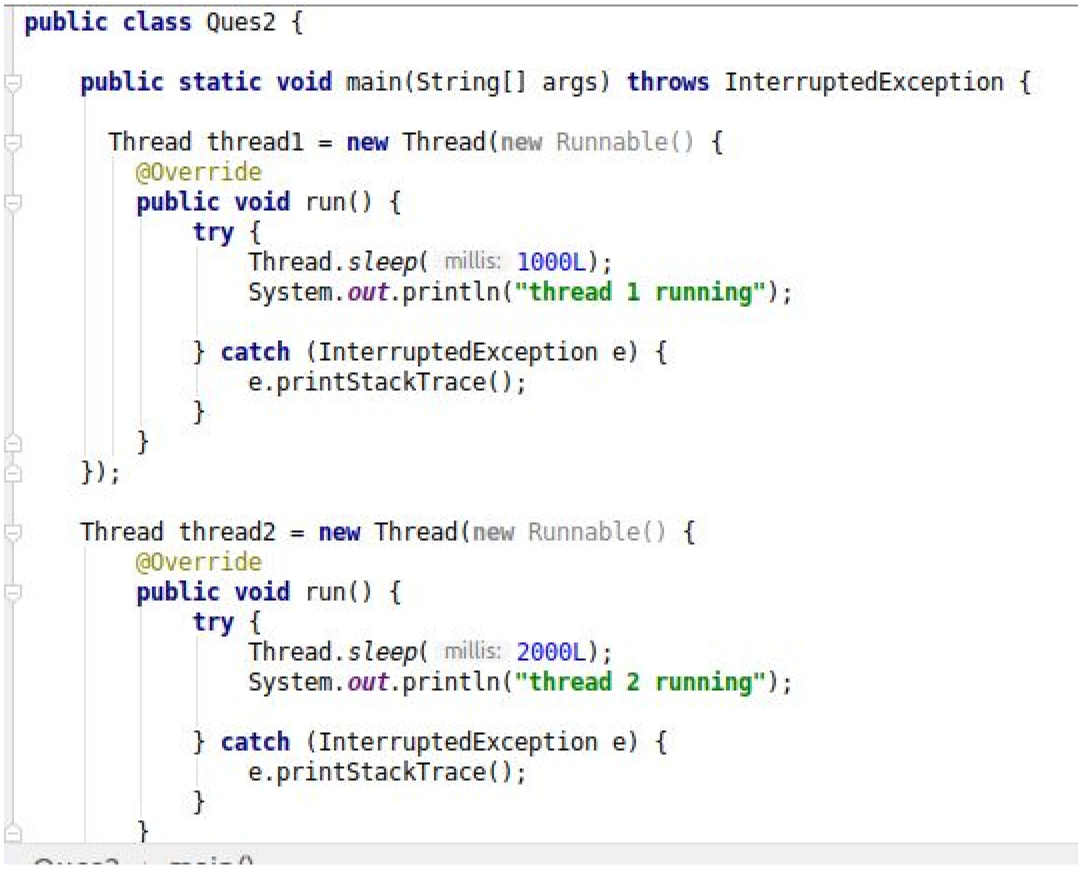


```java
// implementing runnable interface

public class Ques1 implements Runnable{
    @Override
    public void run() {
        System.out.println("Thread running through implementing runnable interface");
    }
}

//extending thread class

 class thread1 extends Thread{
    public void run(){
        System.out.println("Thread running by extending thread class");
    }
}

class Main{
    public static void main(String[] args) {
        new thread1().start();
        new Thread(new Ques1()).start();
    }
}
```

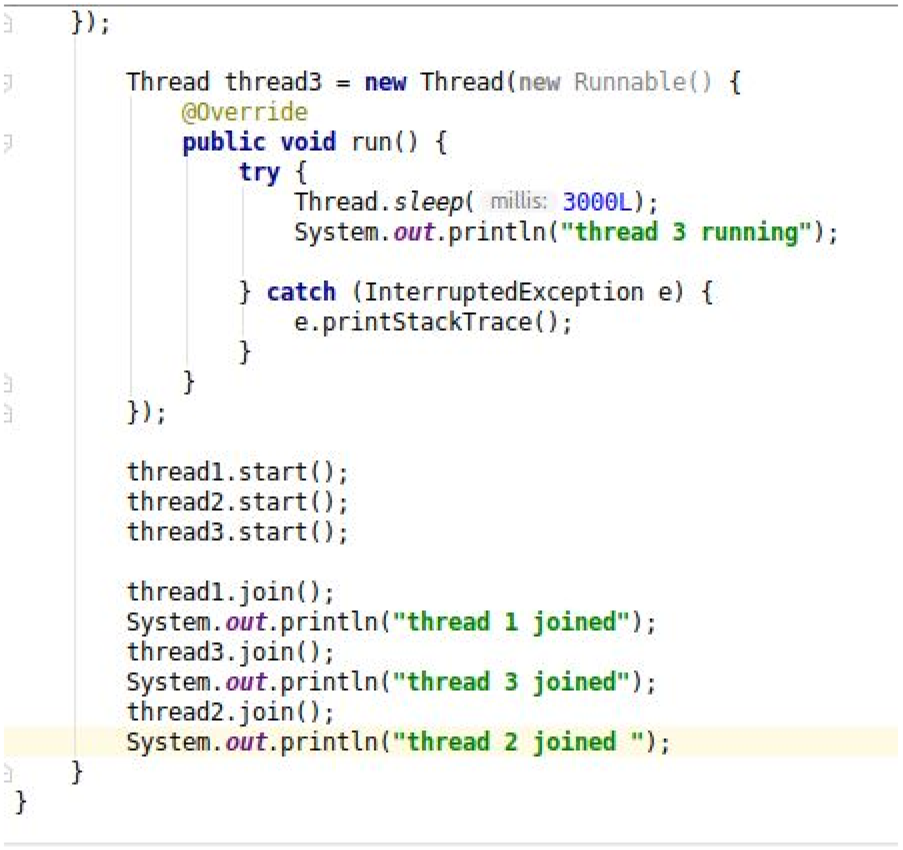


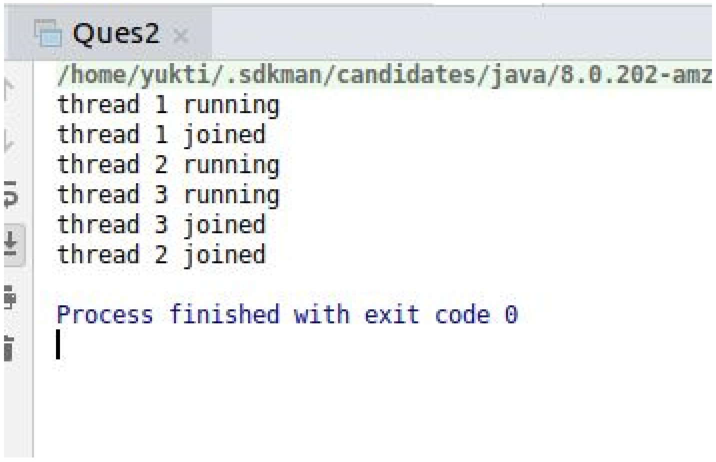
```
/home/yukti/.sdkman/candidates/java/8.0.202-amzn/bin/java ...
Thread running by extending thread class
Thread running through implementing runnable interface

Process finished with exit code 0
```

2. **Use sleep and join methods with thread.**



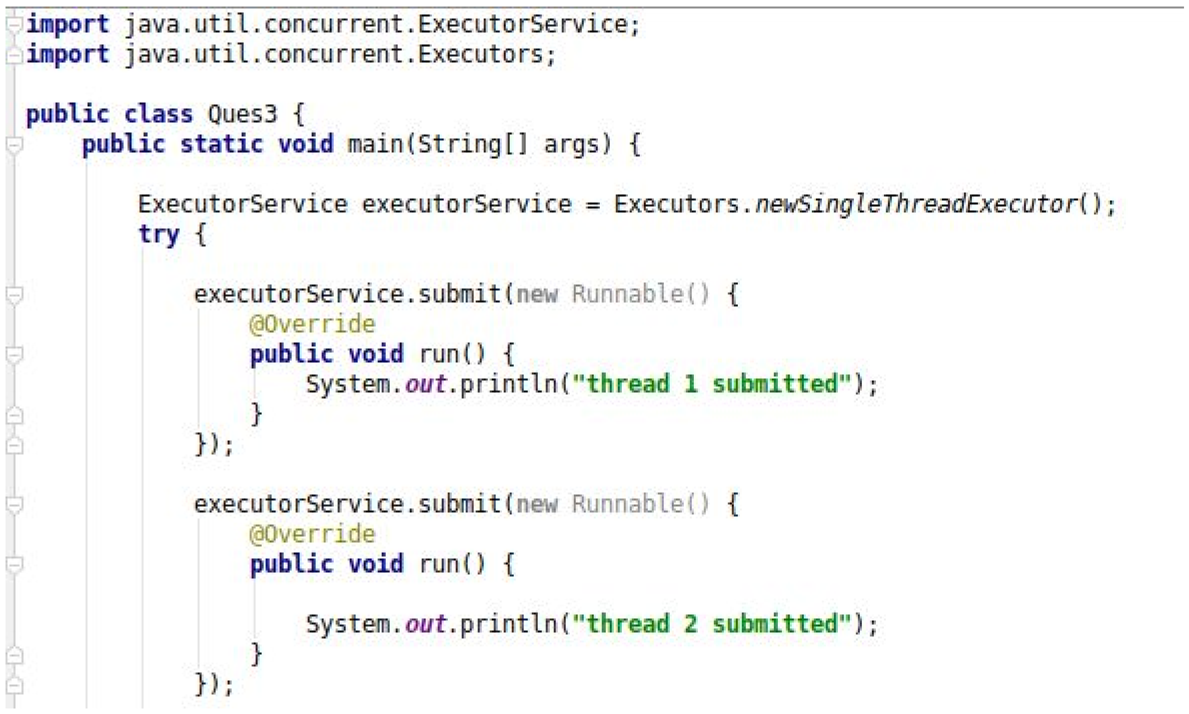
```
public class Ques2 {

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

      Thread thread1 = new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                Thread.sleep( millis: 1000L);
                System.out.println("thread 1 running");

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

    Thread thread2 = new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                Thread.sleep( millis: 2000L);
                System.out.println("thread 2 running");

            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
```


```java
    });

        Thread thread3 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep( millis: 3000L);
                    System.out.println("thread 3 running");

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

        thread1.start();
        thread2.start();
        thread3.start();

        thread1.join();
        System.out.println("thread 1 joined");
        thread3.join();
        System.out.println("thread 3 joined");
        thread2.join();
        System.out.println("thread 2 joined ");
    }
}
```

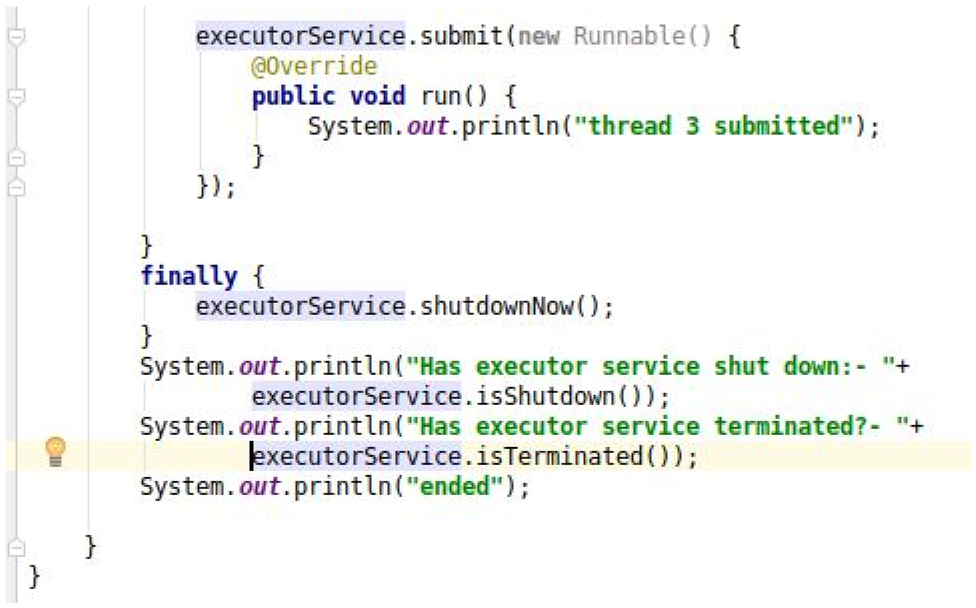

Ques2

```
/home/yukti/.sdkman/candidates/java/8.0.202-amz
thread 1 running
thread 1 joined
thread 2 running
thread 3 running
thread 3 joined
thread 2 joined

Process finished with exit code 0
```

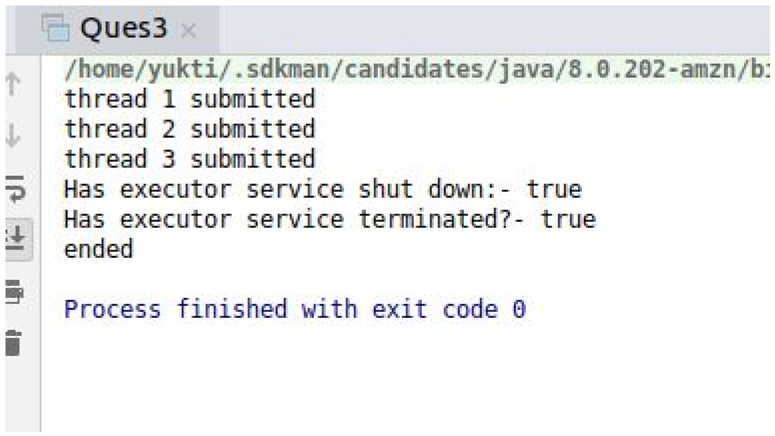
3. Use a singleThreadExecutor to submit multiple threads.



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

public class Ques3 {
    public static void main(String[] args) {

        ExecutorService executorService = Executors.newSingleThreadExecutor();
        try {

            executorService.submit(new Runnable() {
                @Override
                public void run() {
                    System.out.println("thread 1 submitted");
                }
            });

            executorService.submit(new Runnable() {
                @Override
                public void run() {

                    System.out.println("thread 2 submitted");
                }
            });
```



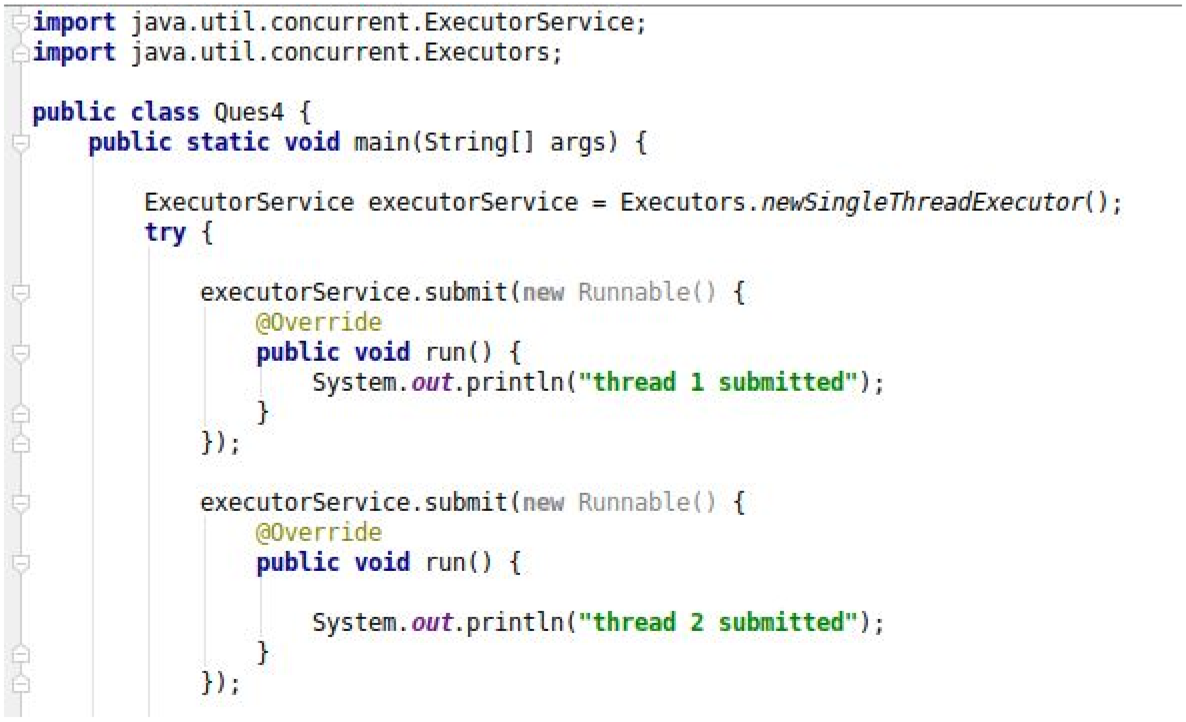
```
            executorService.submit(new Runnable() {
                @Override
                public void run() {
                    System.out.println("thread 3 submitted");
                }
            });

        }
        finally {
            executorService.shutdownNow();
        }
        System.out.println("Has executor service shut down:- "+
                executorService.isShutdown());
        System.out.println("Has executor service terminated?- "+
                executorService.isTerminated());
        System.out.println("ended");

    }
}
```



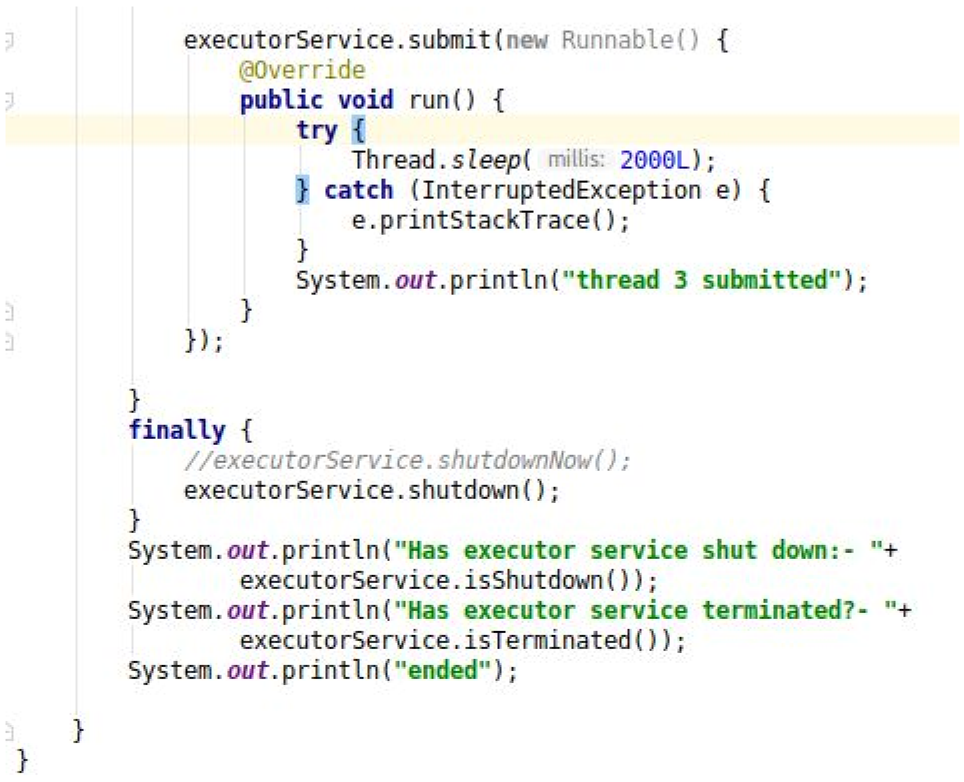
```
/home/yukti/.sdkman/candidates/java/8.0.202-amzn/b
thread 1 submitted
thread 2 submitted
thread 3 submitted
Has executor service shut down:- true
Has executor service terminated?- true
ended

Process finished with exit code 0
```

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

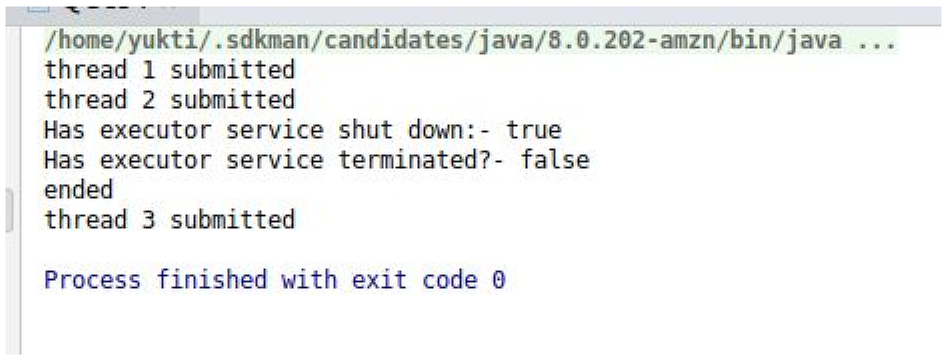
shutdown()-



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

public class Ques4 {
    public static void main(String[] args) {

        ExecutorService executorService = Executors.newSingleThreadExecutor();
        try {

            executorService.submit(new Runnable() {
                @Override
                public void run() {
                    System.out.println("thread 1 submitted");
                }
            });

            executorService.submit(new Runnable() {
                @Override
                public void run() {

                    System.out.println("thread 2 submitted");
                }
            });
```

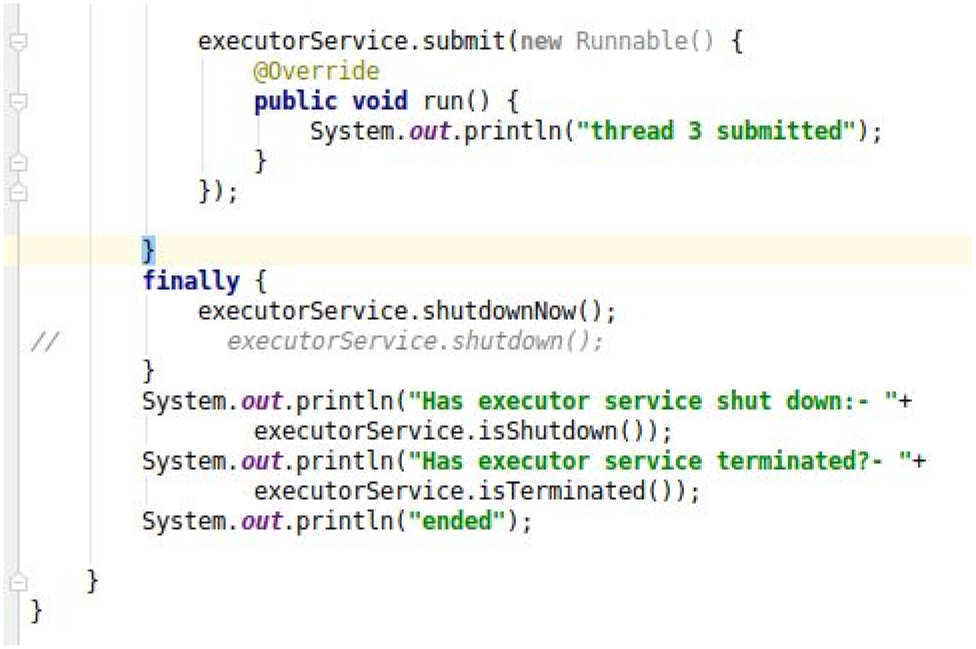


```java
            executorService.submit(new Runnable() {
                @Override
                public void run() {
                    try {
                        Thread.sleep( millis: 2000L);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    System.out.println("thread 3 submitted");
                }
            });

        }
        finally {
            //executorService.shutdownNow();
            executorService.shutdown();
        }
        System.out.println("Has executor service shut down:- "+
                executorService.isShutdown());
        System.out.println("Has executor service terminated?- "+
                executorService.isTerminated());
        System.out.println("ended");

    }
}
```

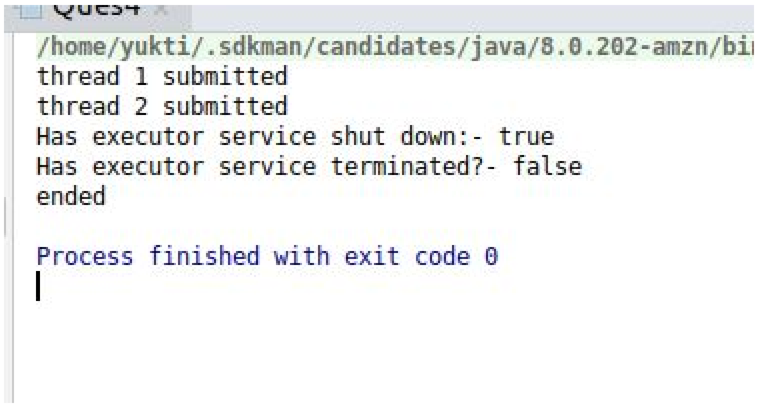


```
/home/yukti/.sdkman/candidates/java/8.0.202-amzn/bin/java ...
thread 1 submitted
thread 2 submitted
Has executor service shut down:- true
Has executor service terminated?- false
ended
thread 3 submitted

Process finished with exit code 0
```

shutdownNow()-



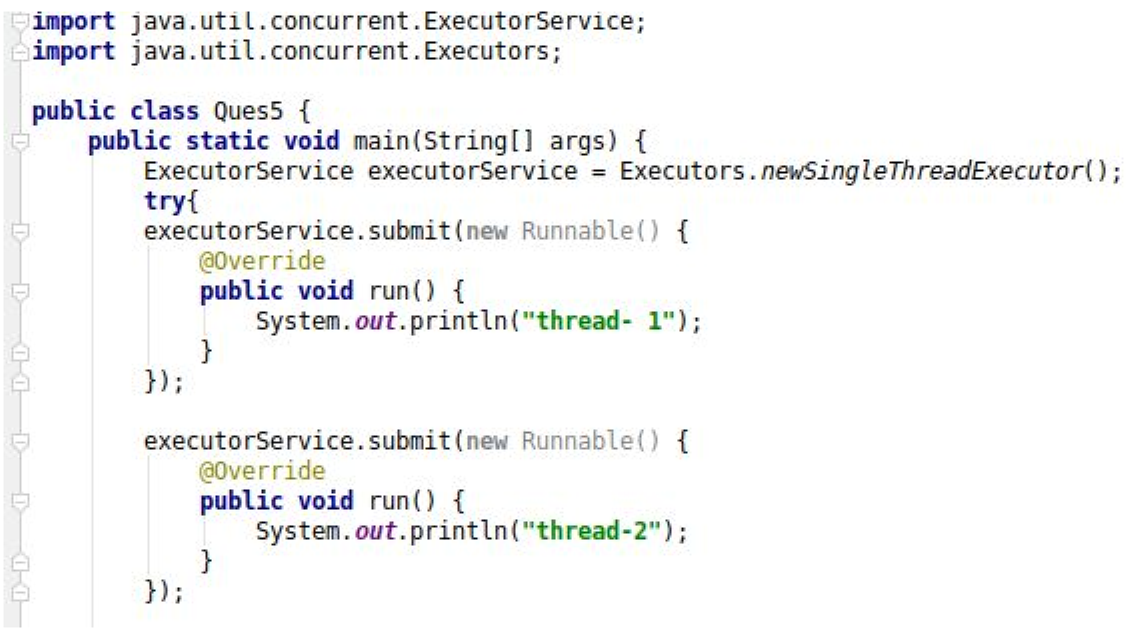
```
            executorService.submit(new Runnable() {
                @Override
                public void run() {
                    System.out.println("thread 3 submitted");
                }
            });

        }
        finally {
            executorService.shutdownNow();
//            executorService.shutdown();
        }
        System.out.println("Has executor service shut down:- "+
                executorService.isShutdown());
        System.out.println("Has executor service terminated?- "+
                executorService.isTerminated());
        System.out.println("ended");

    }
}
```



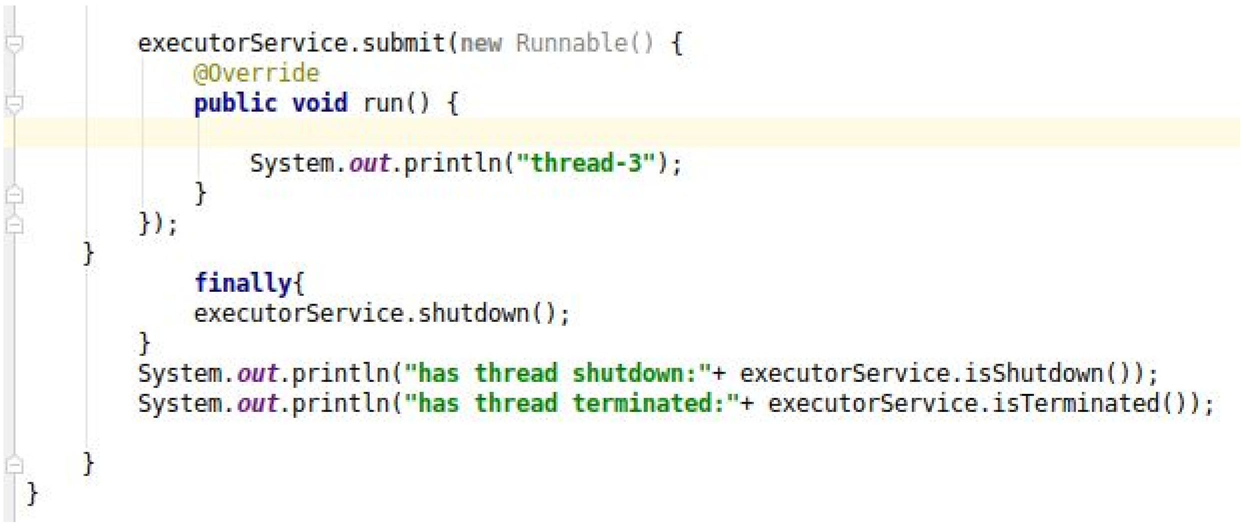
Ques4

```
/home/yukti/.sdkman/candidates/java/8.0.202-amzn/bi
thread 1 submitted
thread 2 submitted
Has executor service shut down:- true
Has executor service terminated?- false
ended

Process finished with exit code 0
```

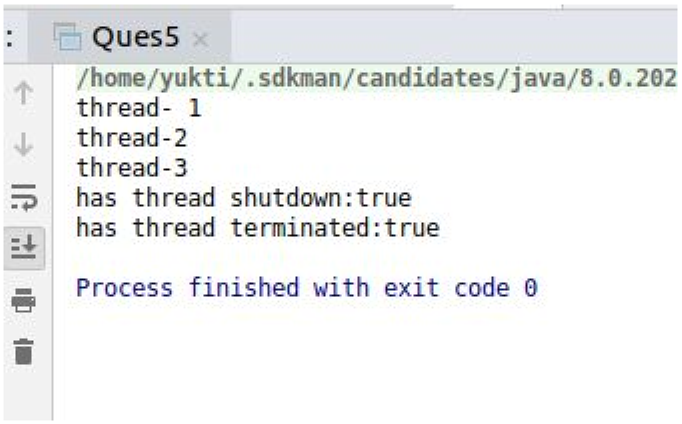
5. **Use isShutDown() and isTerminate() with ExecutorService.**



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

public class Ques5 {
    public static void main(String[] args) {
        ExecutorService executorService = Executors.newSingleThreadExecutor();
        try{
        executorService.submit(new Runnable() {
            @Override
            public void run() {
                System.out.println("thread- 1");
            }
        });

        executorService.submit(new Runnable() {
            @Override
            public void run() {
                System.out.println("thread-2");
            }
        });
```



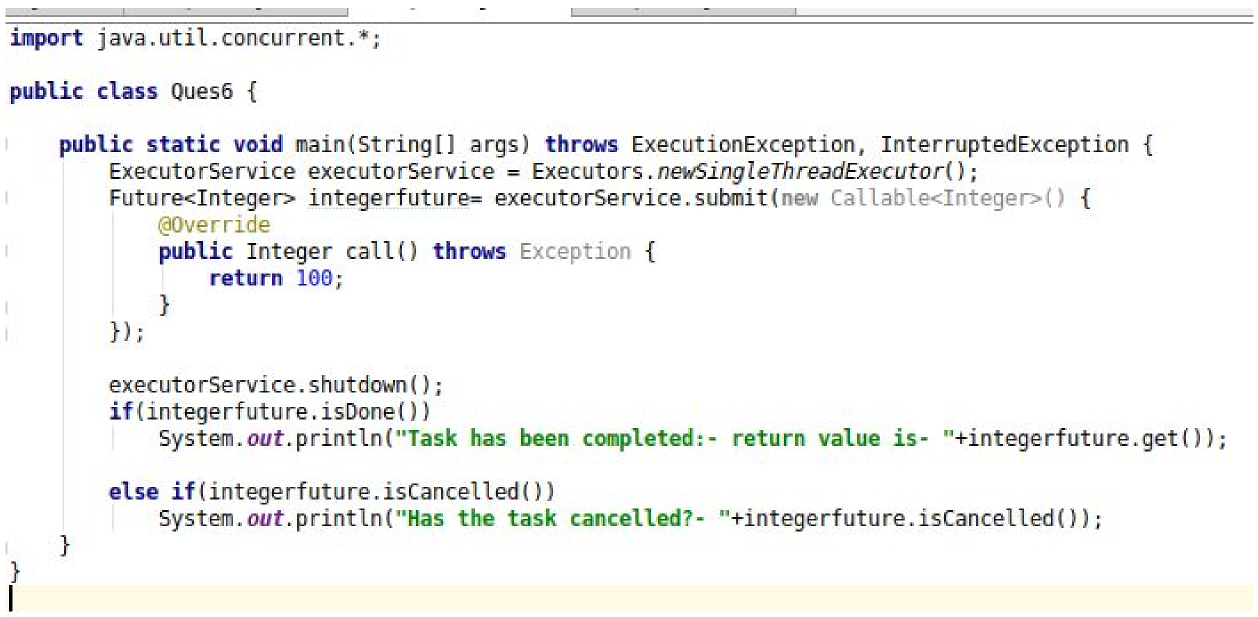
```java
        executorService.submit(new Runnable() {
            @Override
            public void run() {

                System.out.println("thread-3");
            }
        });
    }
            finally{
            executorService.shutdown();
        }
        System.out.println("has thread shutdown:"+ executorService.isShutdown());
        System.out.println("has thread terminated:"+ executorService.isTerminated());

    }
}
```



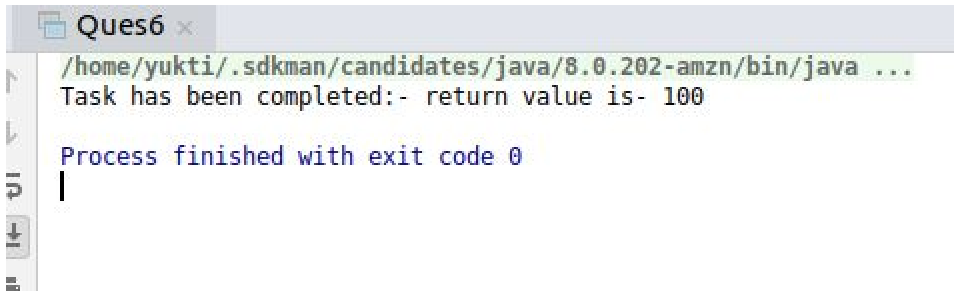
Ques5

```
/home/yukti/.sdkman/candidates/java/8.0.202
thread- 1
thread-2
thread-3
has thread shutdown:true
has thread terminated:true

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.**

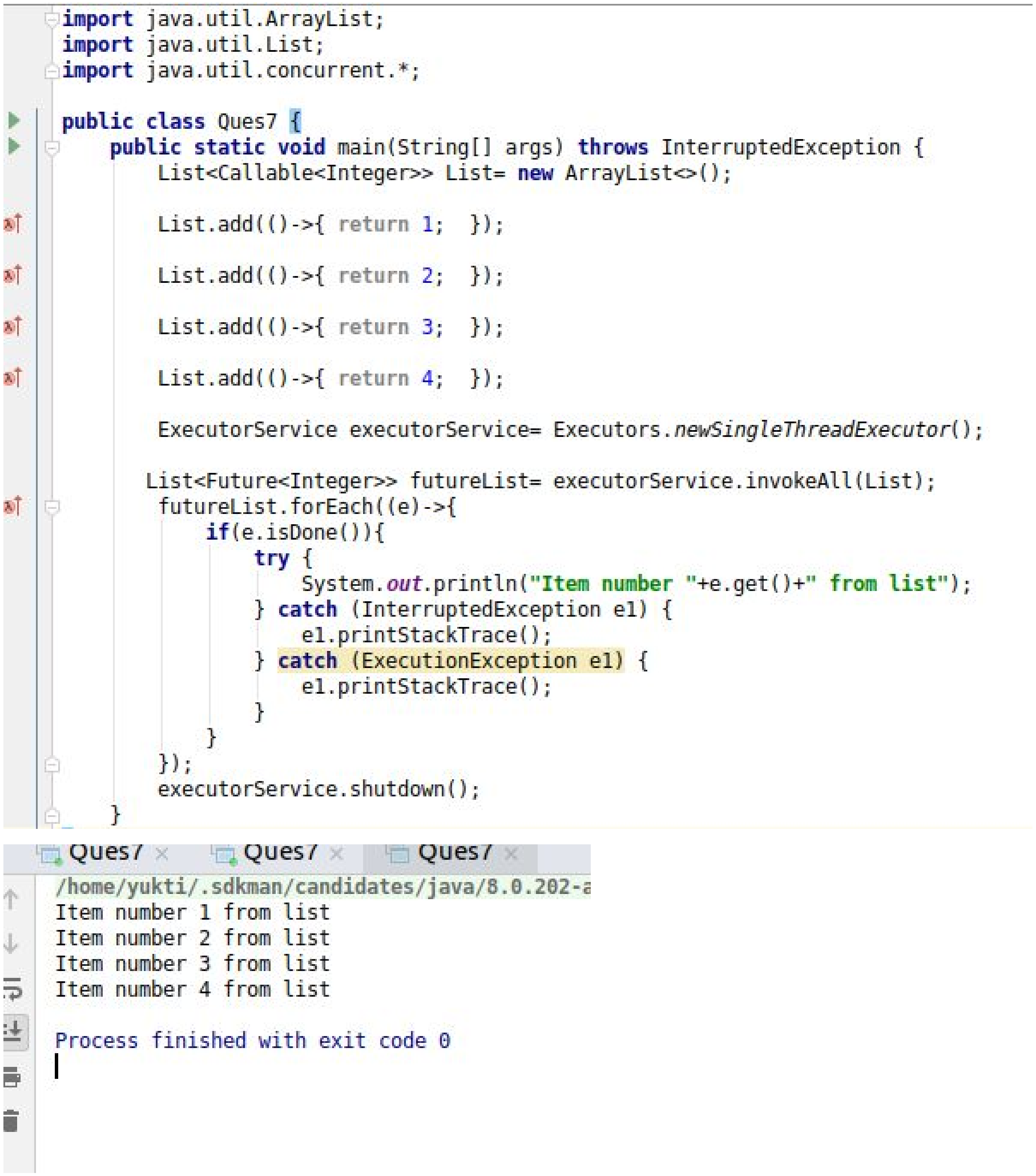


```
import java.util.concurrent.*;

public class Ques6 {

    public static void main(String[] args) throws ExecutionException, InterruptedException {
        ExecutorService executorService = Executors.newSingleThreadExecutor();
        Future<Integer> integerfuture= executorService.submit(new Callable<Integer>() {
            @Override
            public Integer call() throws Exception {
                return 100;
            }
        });

        executorService.shutdown();
        if(integerfuture.isDone())
            System.out.println("Task has been completed:- return value is- "+integerfuture.get());

        else if(integerfuture.isCancelled())
            System.out.println("Has the task cancelled?- "+integerfuture.isCancelled());
    }
}
```



```
/home/yukti/.sdkman/candidates/java/8.0.202-amzn/bin/java ...
Task has been completed:- return value is- 100

Process finished with exit code 0
```

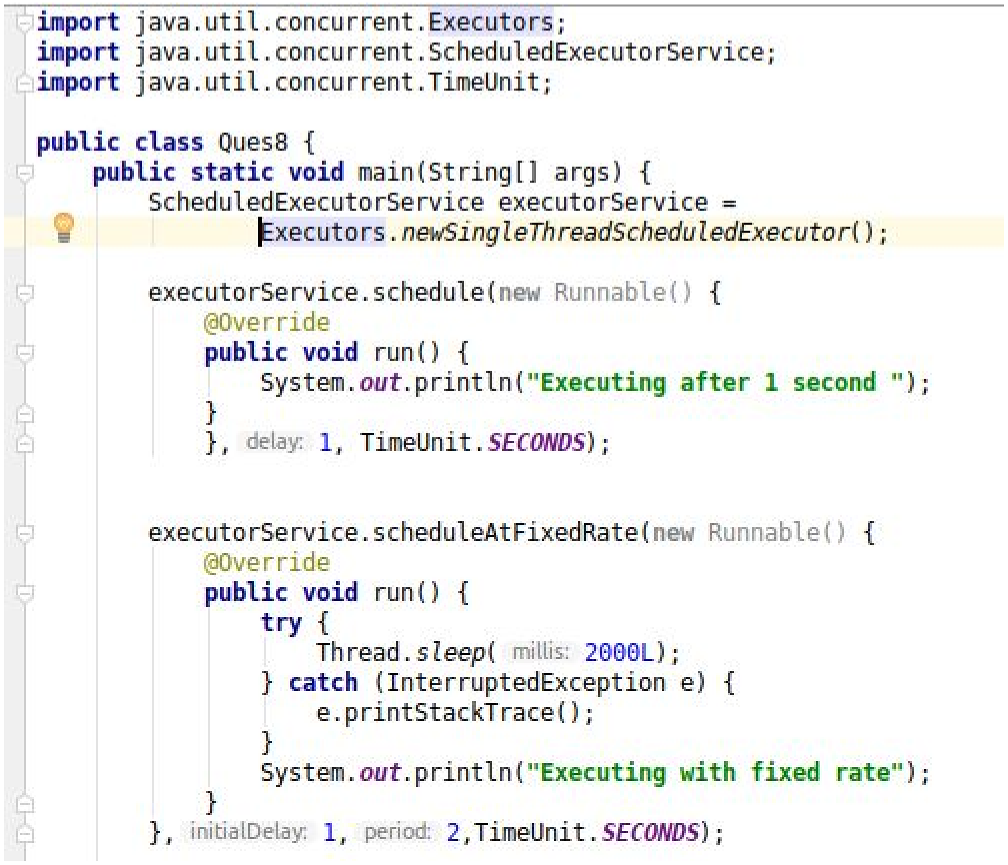
7. **Submit List of tasks to ExecutorService and wait for the completion of all the tasks.**



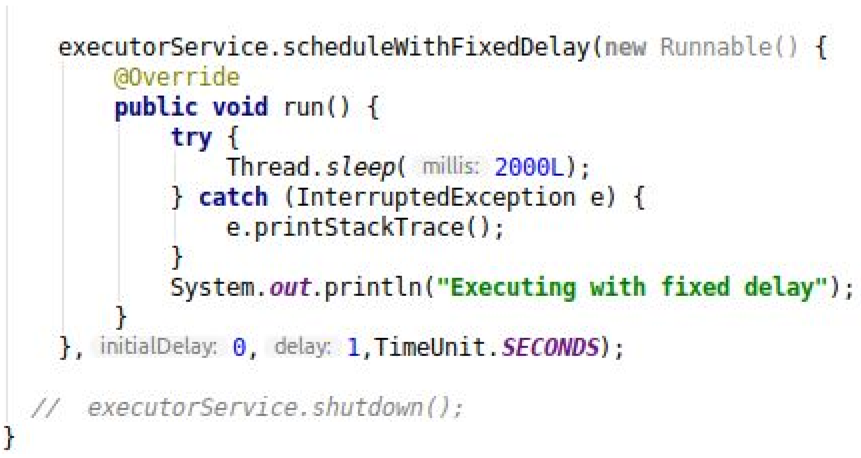
```java
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.*;

public class Ques7 {
    public static void main(String[] args) throws InterruptedException {
        List<Callable<Integer>> List= new ArrayList<>();

        List.add(()->{ return 1;  });

        List.add(()->{ return 2;  });

        List.add(()->{ return 3;  });

        List.add(()->{ return 4;  });

        ExecutorService executorService= Executors.newSingleThreadExecutor();

       List<Future<Integer>> futureList= executorService.invokeAll(List);
        futureList.forEach((e)->{
            if(e.isDone()){
                try {
                    System.out.println("Item number "+e.get()+" from list");
                } catch (InterruptedException e1) {
                    e1.printStackTrace();
                } catch (ExecutionException e1) {
                    e1.printStackTrace();
                }
            }
        });
        executorService.shutdown();
    }
```

```
/home/yukti/.sdkman/candidates/java/8.0.202-a
Item number 1 from list
Item number 2 from list
Item number 3 from list
Item number 4 from list

Process finished with exit code 0
```

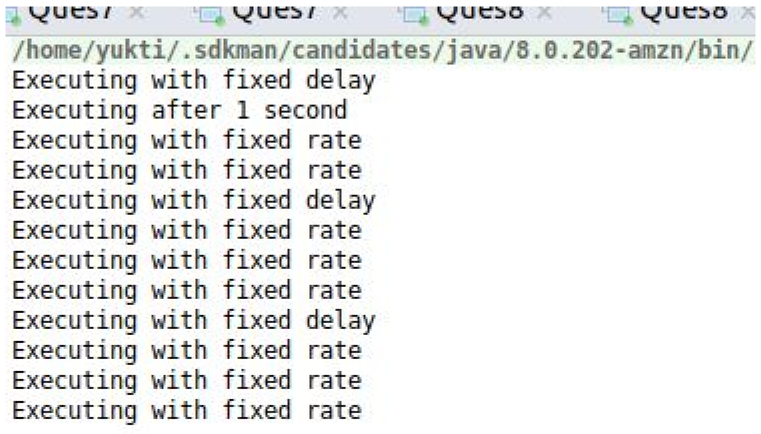
8. **Schedule task using schedule(), scheduleAtFixedRate() and scheduleAtFixedDelay()**



```
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

public class Ques8 {
    public static void main(String[] args) {
        ScheduledExecutorService executorService =
                Executors.newSingleThreadScheduledExecutor();

        executorService.schedule(new Runnable() {
            @Override
            public void run() {
                System.out.println("Executing after 1 second ");
            }
        }, delay: 1, TimeUnit.SECONDS);


        executorService.scheduleAtFixedRate(new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep( millis: 2000L);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("Executing with fixed rate");
            }
        }, initialDelay: 1, period: 2,TimeUnit.SECONDS);
```



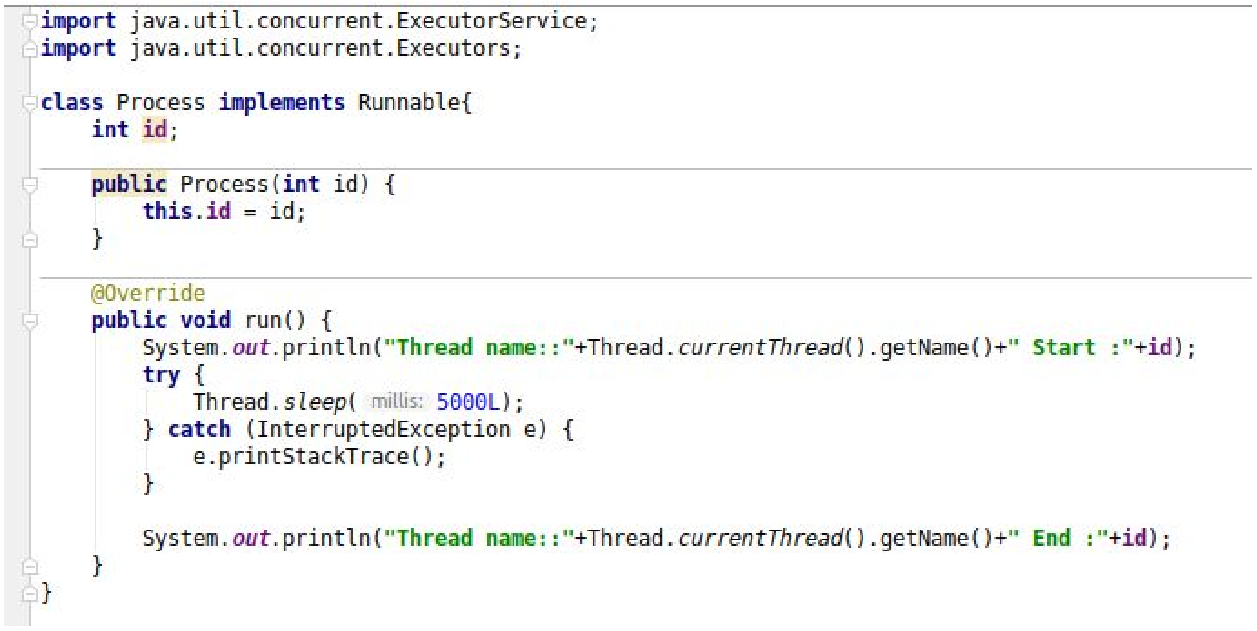
```
    executorService.scheduleWithFixedDelay(new Runnable() {
        @Override
        public void run() {
            try {
                Thread.sleep( millis: 2000L);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Executing with fixed delay");
        }
    }, initialDelay: 0, delay: 1,TimeUnit.SECONDS);

  //  executorService.shutdown();
}
```



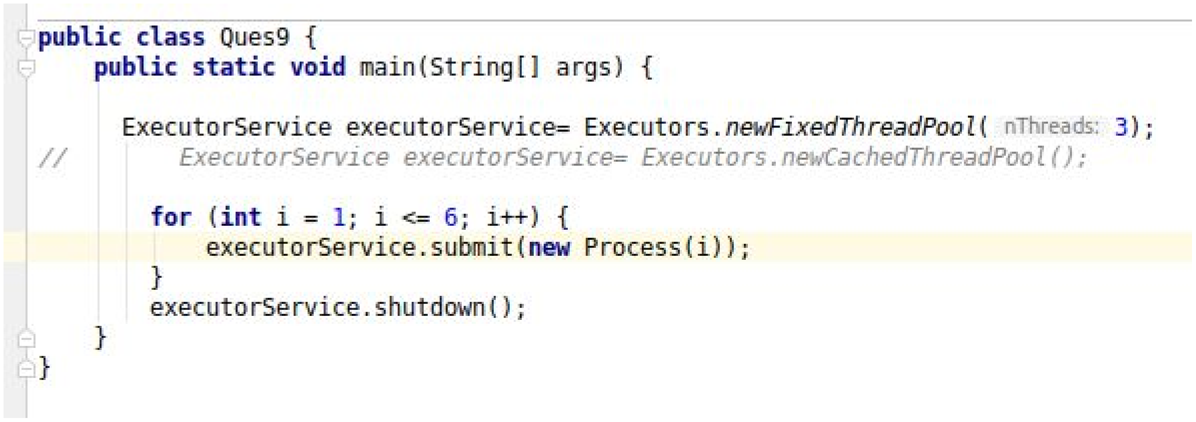
```
Ques7 × Ques7 × Ques8 × Ques8 ×
/home/yukti/.sdkman/candidates/java/8.0.202-amzn/bin/
Executing with fixed delay
Executing after 1 second
Executing with fixed rate
Executing with fixed rate
Executing with fixed delay
Executing with fixed rate
Executing with fixed rate
Executing with fixed rate
Executing with fixed delay
Executing with fixed rate
Executing with fixed rate
Executing with fixed rate
```

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

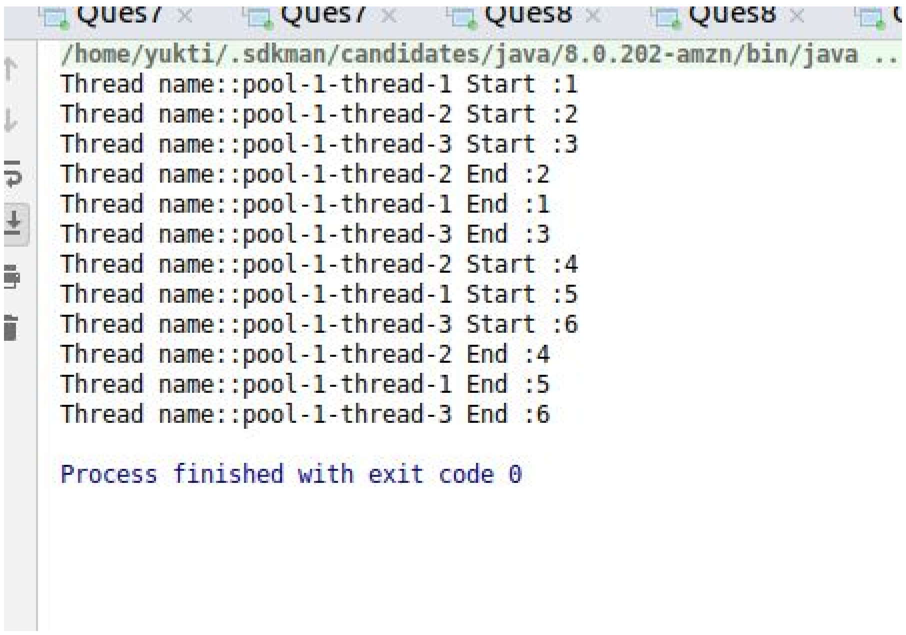
   newFixedThreadPool()-



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

class Process implements Runnable{
    int id;

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

    @Override
    public void run() {
        System.out.println("Thread name::"+Thread.currentThread().getName()+" Start :"+id);
        try {
            Thread.sleep( millis: 5000L);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Thread name::"+Thread.currentThread().getName()+" End :"+id);
    }
}
```

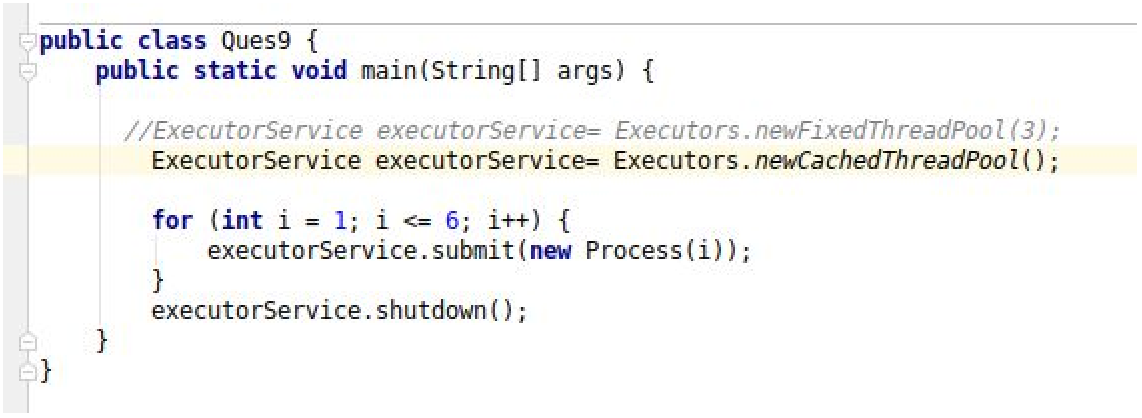


```
public class Ques9 {
    public static void main(String[] args) {

      ExecutorService executorService= Executors.newFixedThreadPool( nThreads: 3);
//        ExecutorService executorService= Executors.newCachedThreadPool();

        for (int i = 1; i <= 6; i++) {
            executorService.submit(new Process(i));
        }
        executorService.shutdown();
    }
}
```

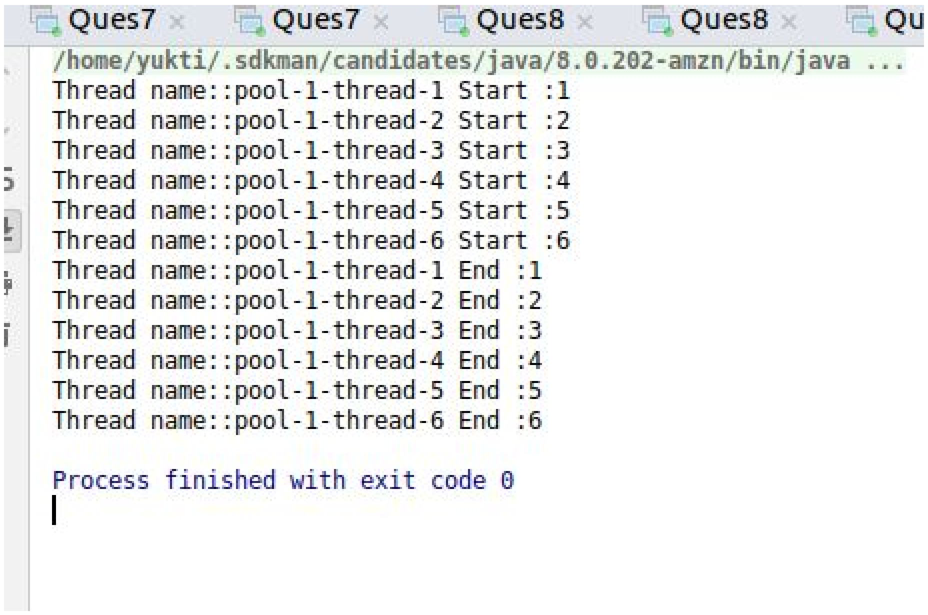


```
/home/yukti/.sdkman/candidates/java/8.0.202-amzn/bin/java ..
Thread name::pool-1-thread-1 Start :1
Thread name::pool-1-thread-2 Start :2
Thread name::pool-1-thread-3 Start :3
Thread name::pool-1-thread-2 End :2
Thread name::pool-1-thread-1 End :1
Thread name::pool-1-thread-3 End :3
Thread name::pool-1-thread-2 Start :4
Thread name::pool-1-thread-1 Start :5
Thread name::pool-1-thread-3 Start :6
Thread name::pool-1-thread-2 End :4
Thread name::pool-1-thread-1 End :5
Thread name::pool-1-thread-3 End :6

Process finished with exit code 0
```

newCachedThreadPool()-

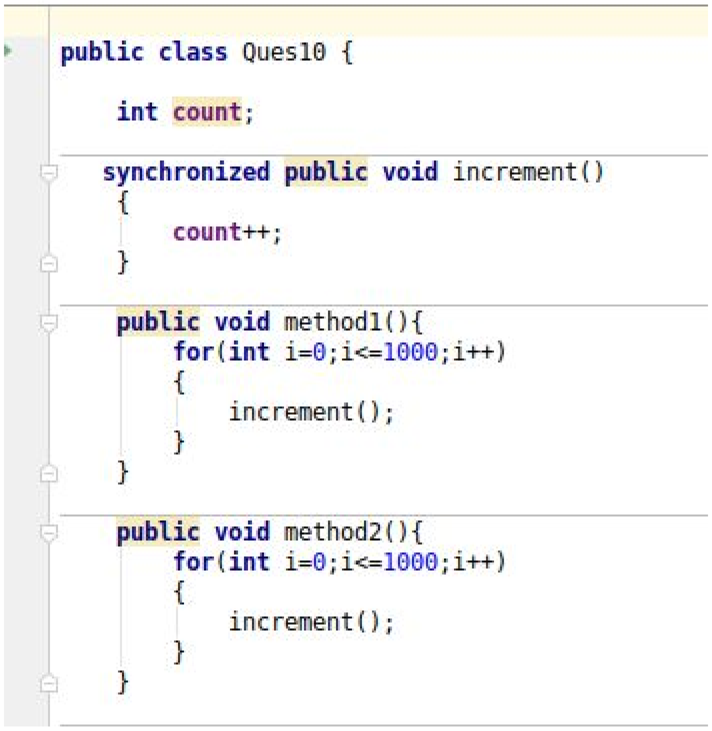


```java
public class Ques9 {
    public static void main(String[] args) {

      //ExecutorService executorService= Executors.newFixedThreadPool(3);
        ExecutorService executorService= Executors.newCachedThreadPool();

        for (int i = 1; i <= 6; i++) {
            executorService.submit(new Process(i));
        }
        executorService.shutdown();
    }
}
```

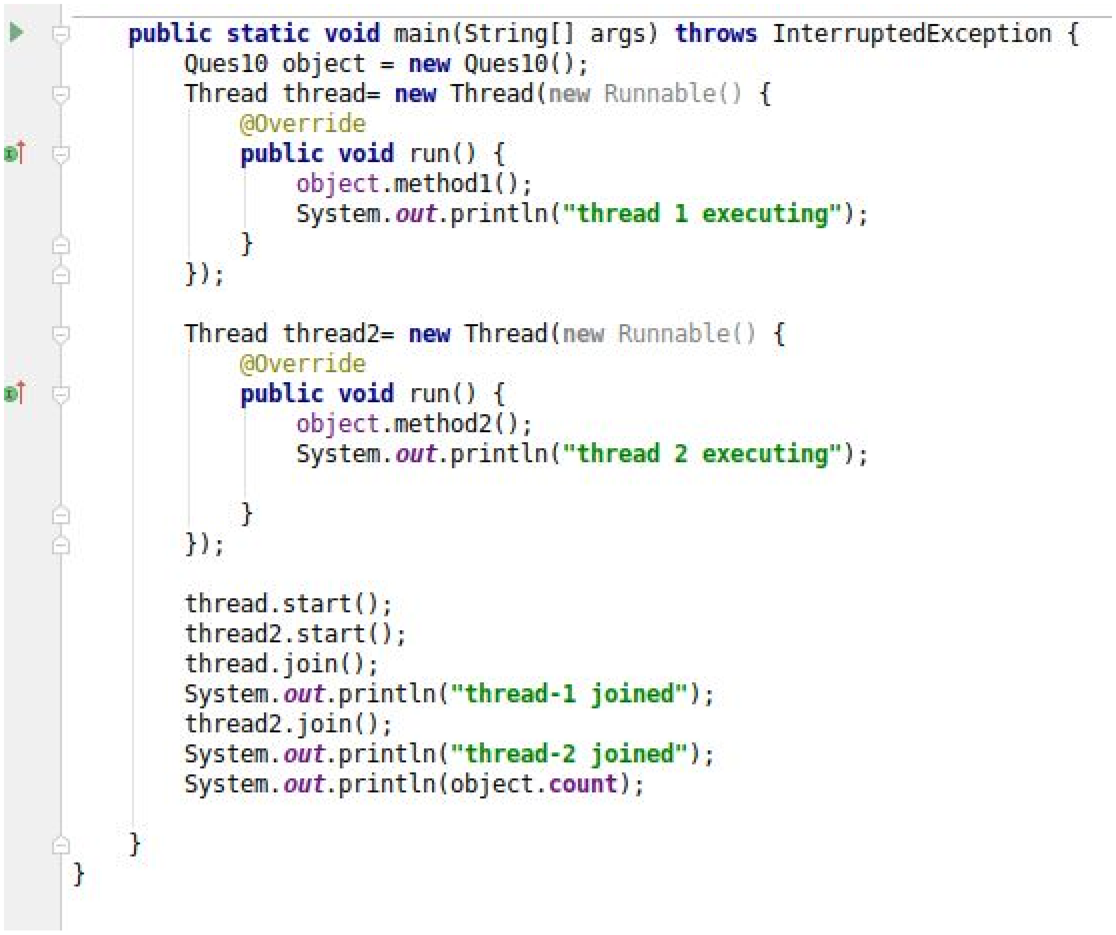


```
/home/yukti/.sdkman/candidates/java/8.0.202-amzn/bin/java ...
Thread name::pool-1-thread-1 Start :1
Thread name::pool-1-thread-2 Start :2
Thread name::pool-1-thread-3 Start :3
Thread name::pool-1-thread-4 Start :4
Thread name::pool-1-thread-5 Start :5
Thread name::pool-1-thread-6 Start :6
Thread name::pool-1-thread-1 End :1
Thread name::pool-1-thread-2 End :2
Thread name::pool-1-thread-3 End :3
Thread name::pool-1-thread-4 End :4
Thread name::pool-1-thread-5 End :5
Thread name::pool-1-thread-6 End :6

Process finished with exit code 0
```

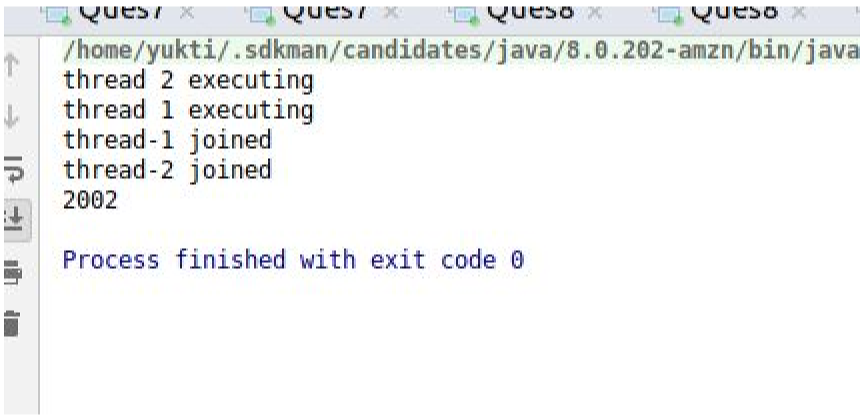
10. **Use Synchronize method to enable synchronization between multiple threads trying to access method at same time.**



```
public class Ques10 {

    int count;

   synchronized public void increment()
    {
        count++;
    }

    public void method1(){
        for(int i=0;i<=1000;i++)
        {
            increment();
        }
    }

    public void method2(){
        for(int i=0;i<=1000;i++)
        {
            increment();
        }
    }
```

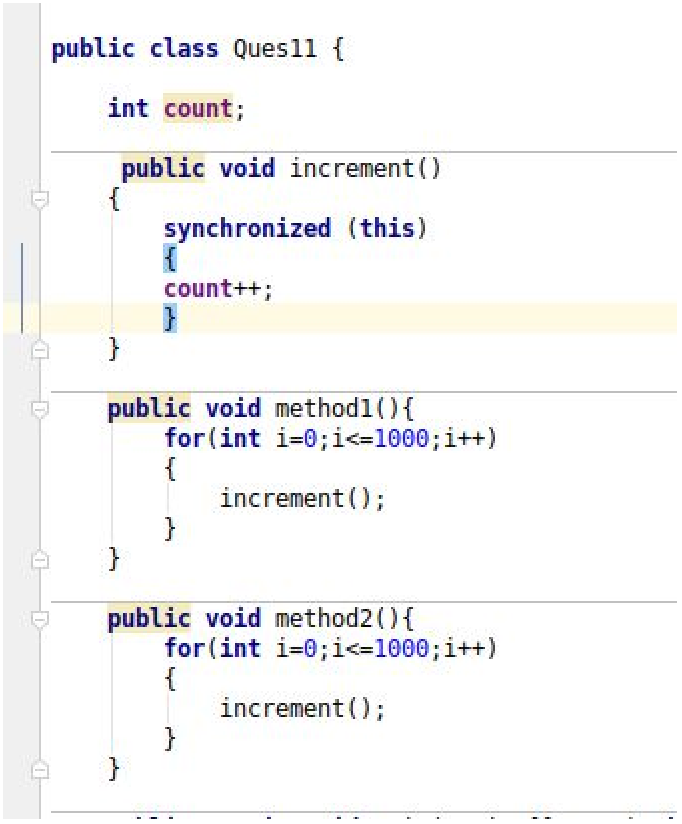


```
    public static void main(String[] args) throws InterruptedException {
        Ques10 object = new Ques10();
        Thread thread= new Thread(new Runnable() {
            @Override
            public void run() {
                object.method1();
                System.out.println("thread 1 executing");
            }
        });

        Thread thread2= new Thread(new Runnable() {
            @Override
            public void run() {
                object.method2();
                System.out.println("thread 2 executing");

            }
        });

        thread.start();
        thread2.start();
        thread.join();
        System.out.println("thread-1 joined");
        thread2.join();
        System.out.println("thread-2 joined");
        System.out.println(object.count);

    }
}
```

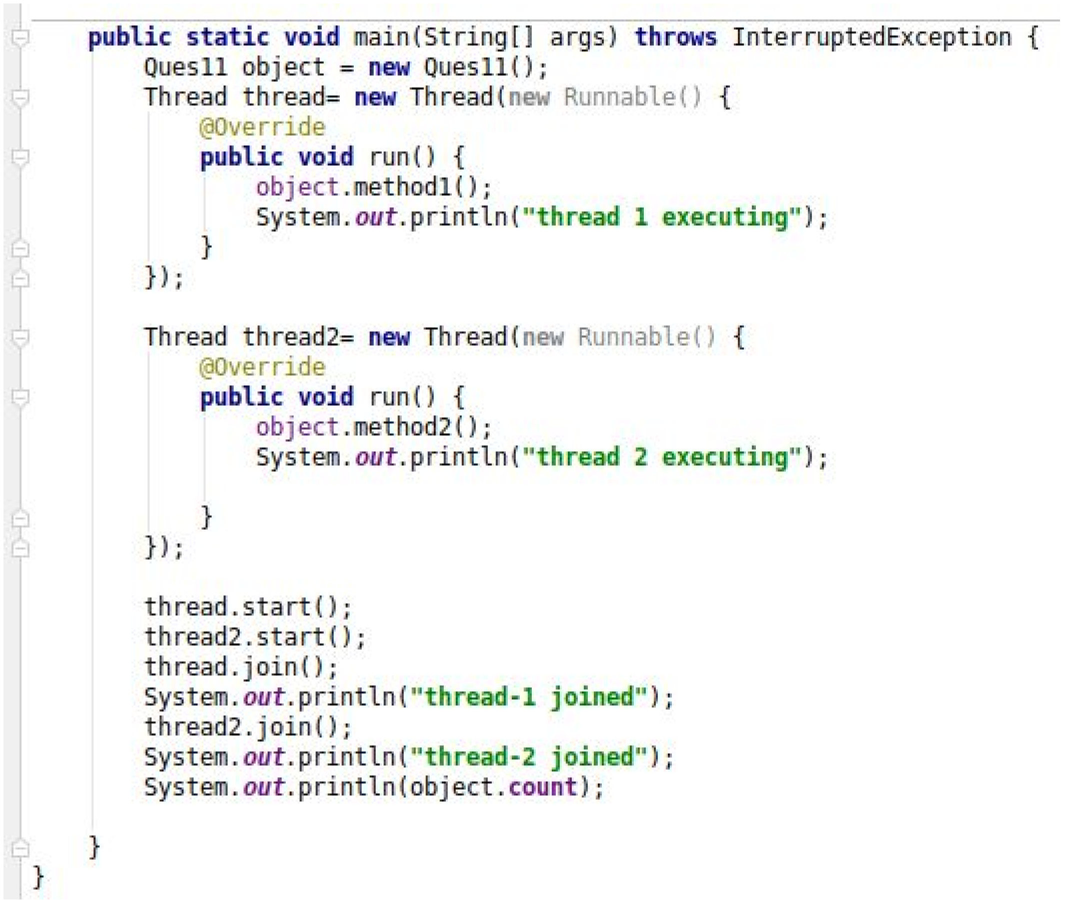


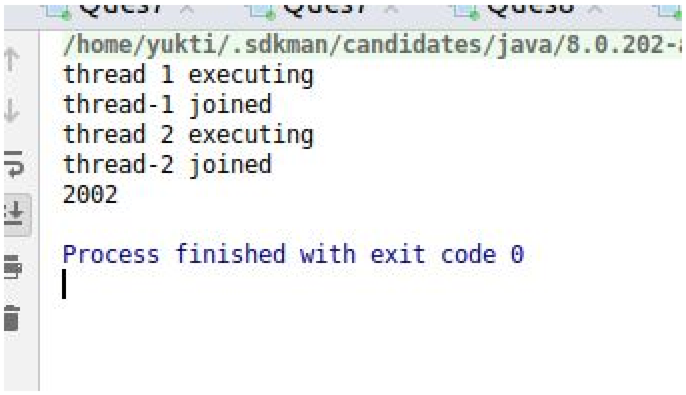
```
/home/yukti/.sdkman/candidates/java/8.0.202-amzn/bin/java
thread 2 executing
thread 1 executing
thread-1 joined
thread-2 joined
2002

Process finished with exit code 0
```

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

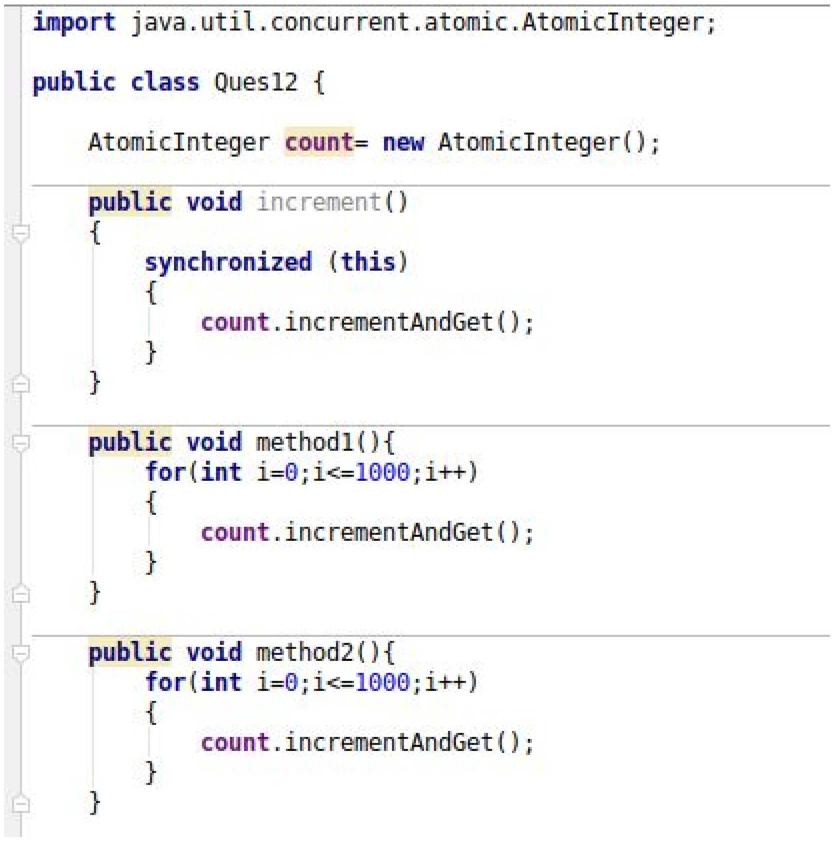


```
public class Ques11 {

    int count;

     public void increment()
    {
        synchronized (this)
        {
        count++;
        }
    }

    public void method1(){
        for(int i=0;i<=1000;i++)
        {
            increment();
        }
    }

    public void method2(){
        for(int i=0;i<=1000;i++)
        {
            increment();
        }
    }
```

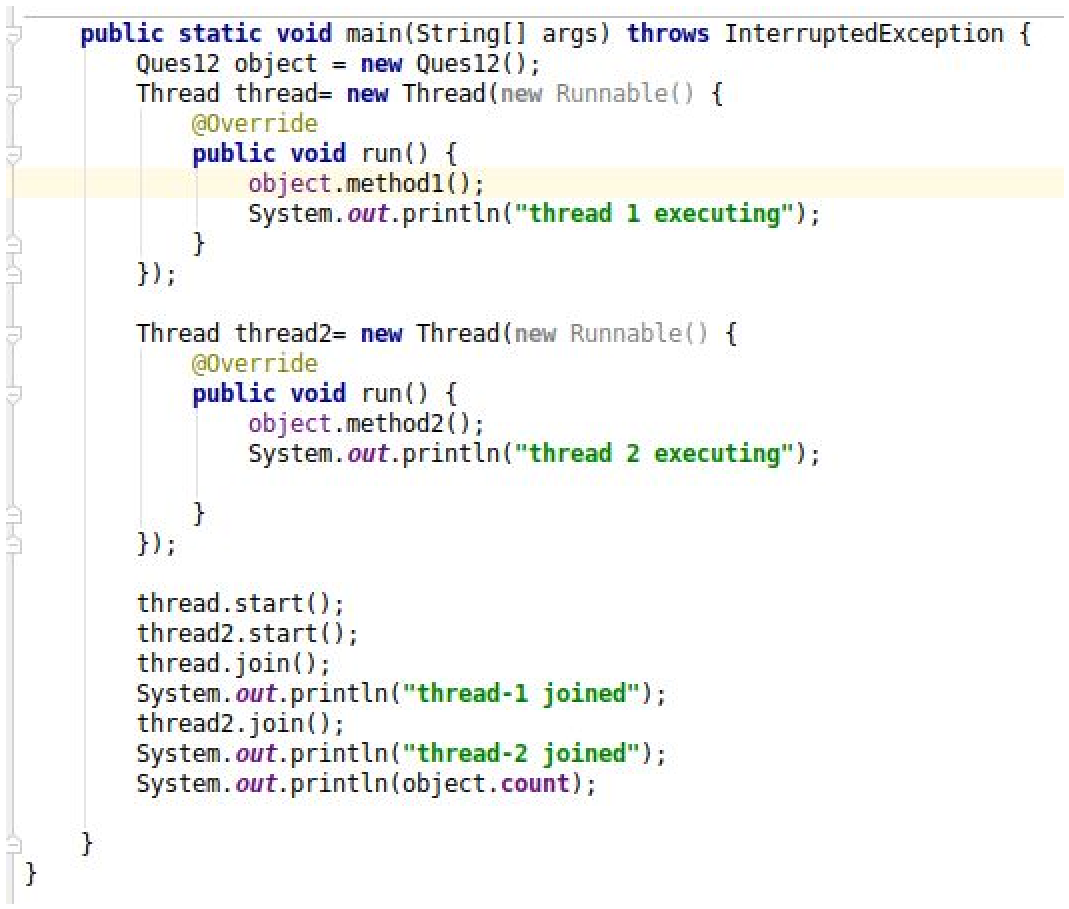

```java
    public static void main(String[] args) throws InterruptedException {
        Ques11 object = new Ques11();
        Thread thread= new Thread(new Runnable() {
            @Override
            public void run() {
                object.method1();
                System.out.println("thread 1 executing");
            }
        });

        Thread thread2= new Thread(new Runnable() {
            @Override
            public void run() {
                object.method2();
                System.out.println("thread 2 executing");

            }
        });

        thread.start();
        thread2.start();
        thread.join();
        System.out.println("thread-1 joined");
        thread2.join();
        System.out.println("thread-2 joined");
        System.out.println(object.count);

    }
}
```


```
/home/yukti/.sdkman/candidates/java/8.0.202-
thread 1 executing
thread-1 joined
thread 2 executing
thread-2 joined
2002

Process finished with exit code 0
```

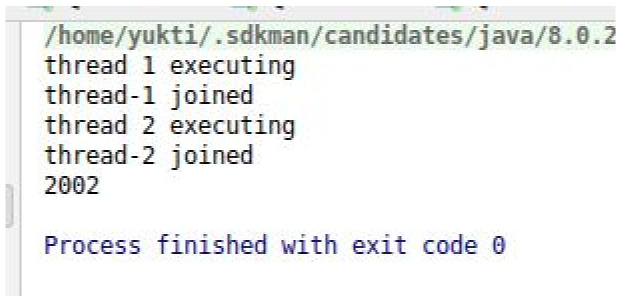
12. Use Atomic Classes instead of Synchronize method and blocks.



```java
import java.util.concurrent.atomic.AtomicInteger;

public class Ques12 {

    AtomicInteger count= new AtomicInteger();

    public void increment()
    {
        synchronized (this)
        {
            count.incrementAndGet();
        }
    }

    public void method1(){
        for(int i=0;i<=1000;i++)
        {
            count.incrementAndGet();
        }
    }

    public void method2(){
        for(int i=0;i<=1000;i++)
        {
            count.incrementAndGet();
        }
    }
```

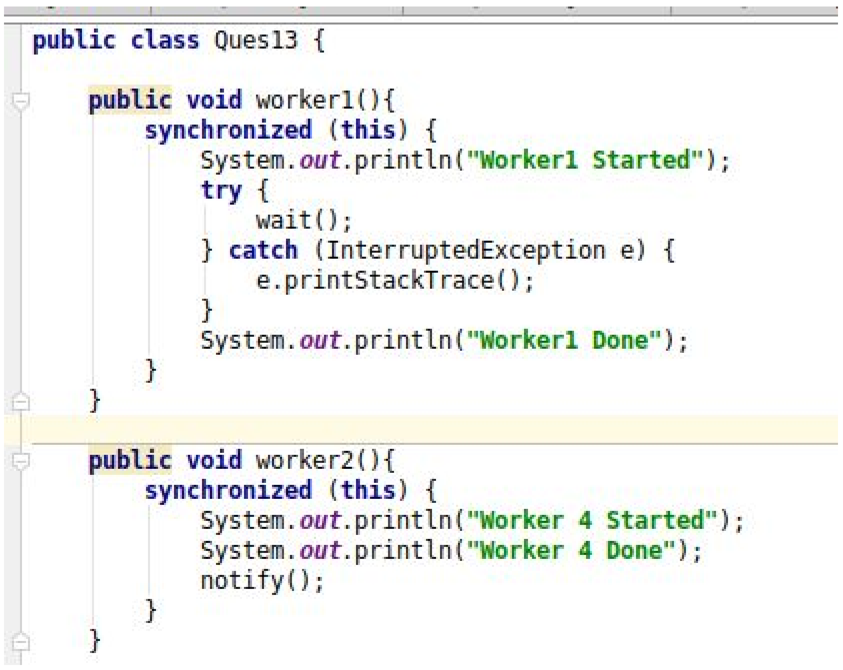


```
    public static void main(String[] args) throws InterruptedException {
        Ques12 object = new Ques12();
        Thread thread= new Thread(new Runnable() {
            @Override
            public void run() {
                object.method1();
                System.out.println("thread 1 executing");
            }
        });

        Thread thread2= new Thread(new Runnable() {
            @Override
            public void run() {
                object.method2();
                System.out.println("thread 2 executing");

            }
        });

        thread.start();
        thread2.start();
        thread.join();
        System.out.println("thread-1 joined");
        thread2.join();
        System.out.println("thread-2 joined");
        System.out.println(object.count);

    }
}
```

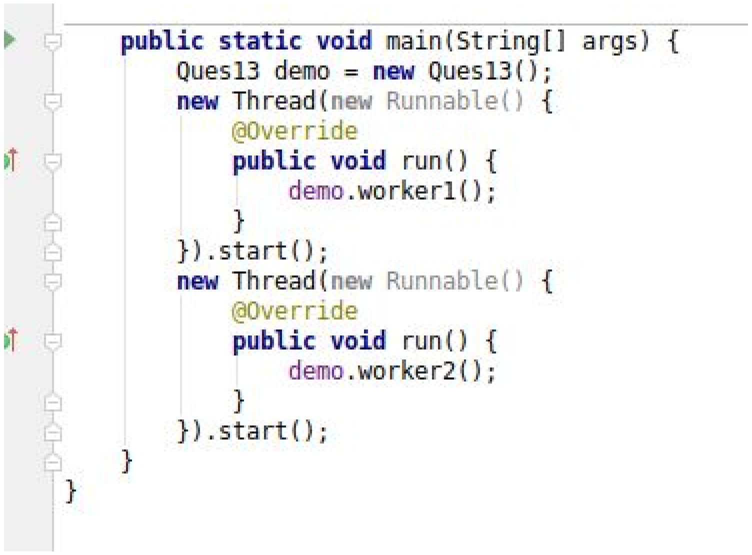


```
/home/yukti/.sdkman/candidates/java/8.0.2
thread 1 executing
thread-1 joined
thread 2 executing
thread-2 joined
2002

Process finished with exit code 0
```

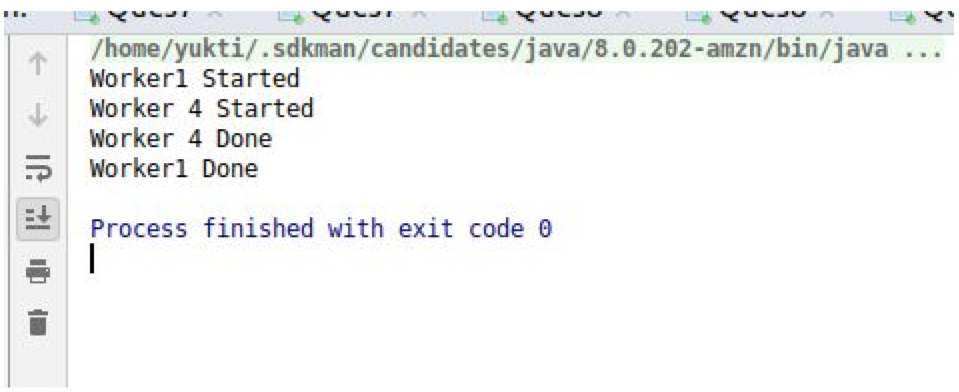
13. Coordinate 2 threads using wait() and notify().



```java
public class Ques13 {

    public void worker1(){
        synchronized (this) {
            System.out.println("Worker1 Started");
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Worker1 Done");
        }
    }

    public void worker2(){
        synchronized (this) {
            System.out.println("Worker 4 Started");
            System.out.println("Worker 4 Done");
            notify();
        }
    }
```

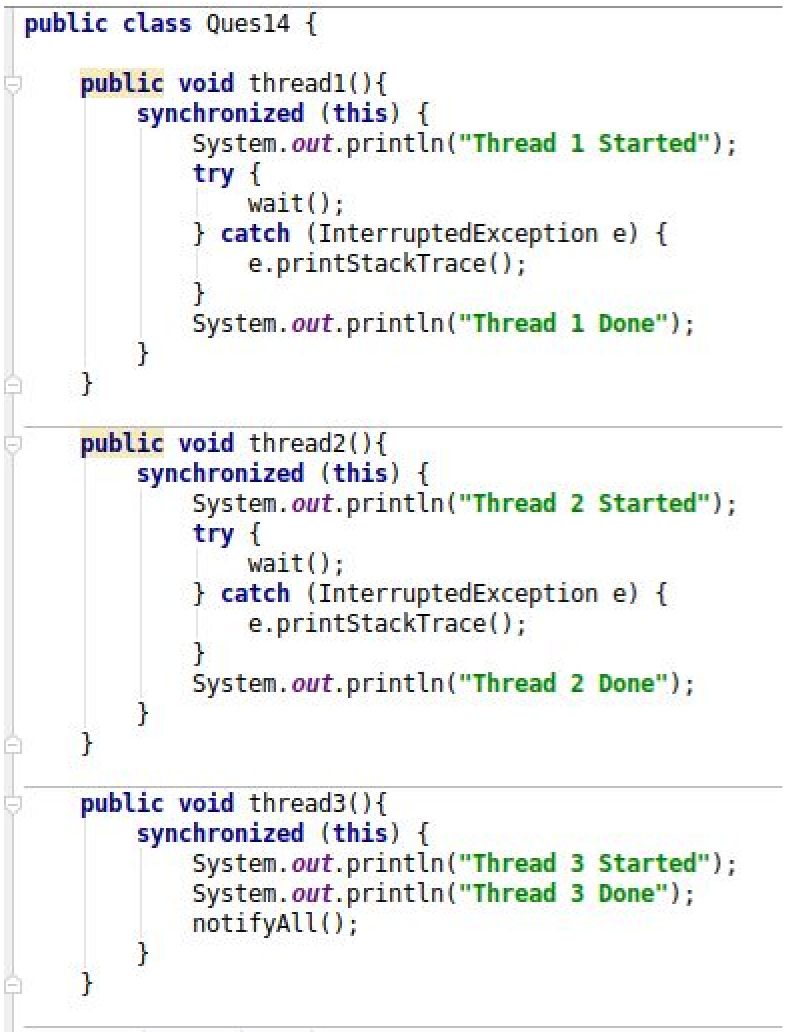


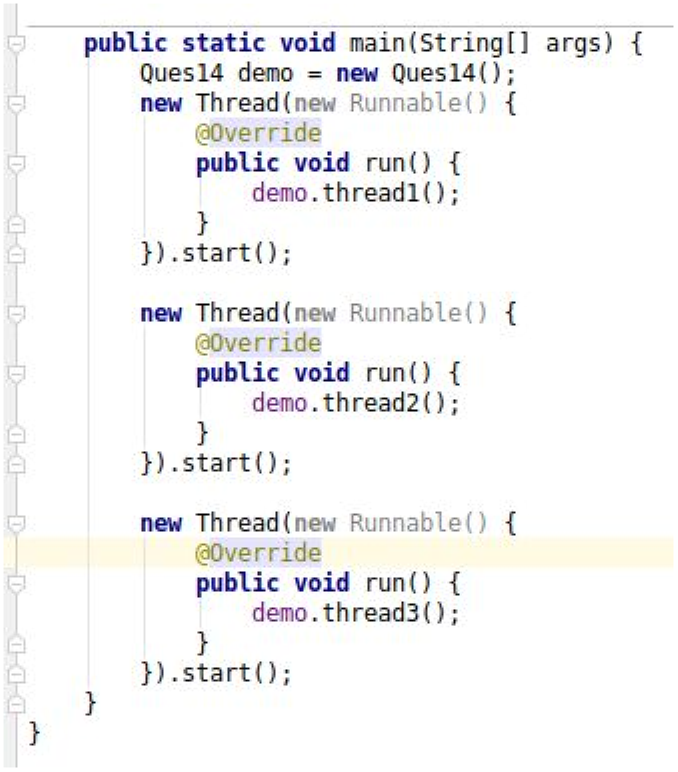
```java
    public static void main(String[] args) {
        Ques13 demo = new Ques13();
        new Thread(new Runnable() {
            @Override
            public void run() {
                demo.worker1();
            }
        }).start();
        new Thread(new Runnable() {
            @Override
            public void run() {
                demo.worker2();
            }
        }).start();
    }
}
```



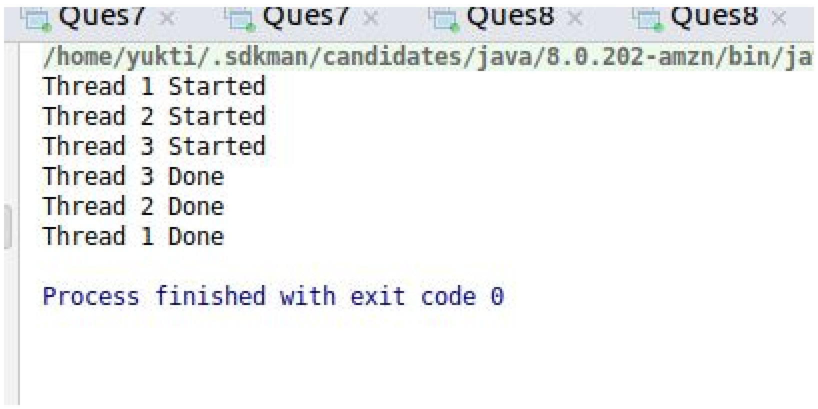
```
/home/yukti/.sdkman/candidates/java/8.0.202-amzn/bin/java ...
Worker1 Started
Worker 4 Started
Worker 4 Done
Worker1 Done

Process finished with exit code 0
```

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


```java
public class Ques14 {

    public void thread1(){
        synchronized (this) {
            System.out.println("Thread 1 Started");
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Thread 1 Done");
        }
    }

    public void thread2(){
        synchronized (this) {
            System.out.println("Thread 2 Started");
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Thread 2 Done");
        }
    }

    public void thread3(){
        synchronized (this) {
            System.out.println("Thread 3 Started");
            System.out.println("Thread 3 Done");
            notifyAll();
        }
    }
```

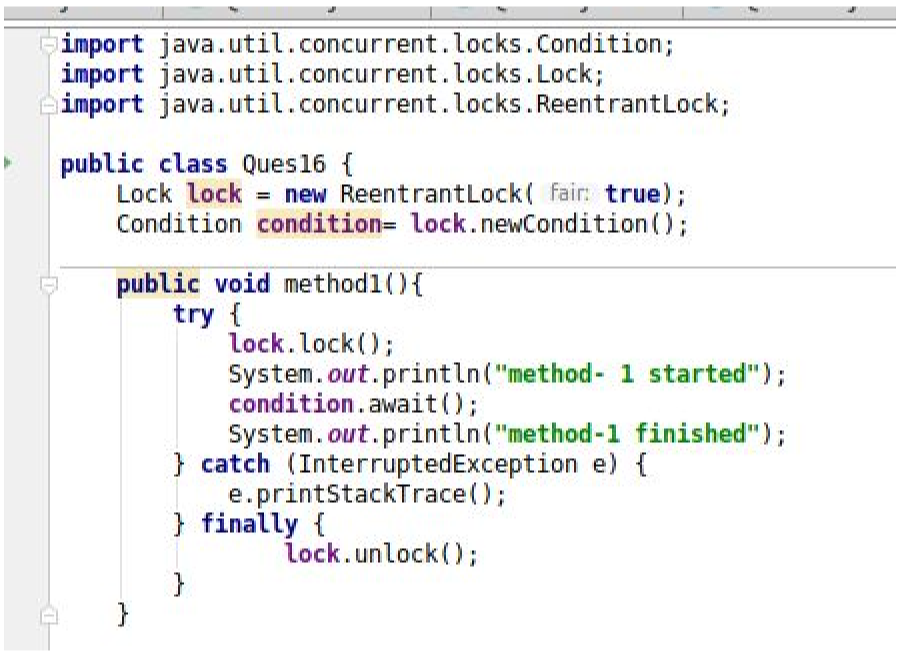


```java
    public static void main(String[] args) {
        Ques14 demo = new Ques14();
        new Thread(new Runnable() {
            @Override
            public void run() {
                demo.thread1();
            }
        }).start();

        new Thread(new Runnable() {
            @Override
            public void run() {
                demo.thread2();
            }
        }).start();

        new Thread(new Runnable() {
            @Override
            public void run() {
                demo.thread3();
            }
        }).start();
    }
}
```

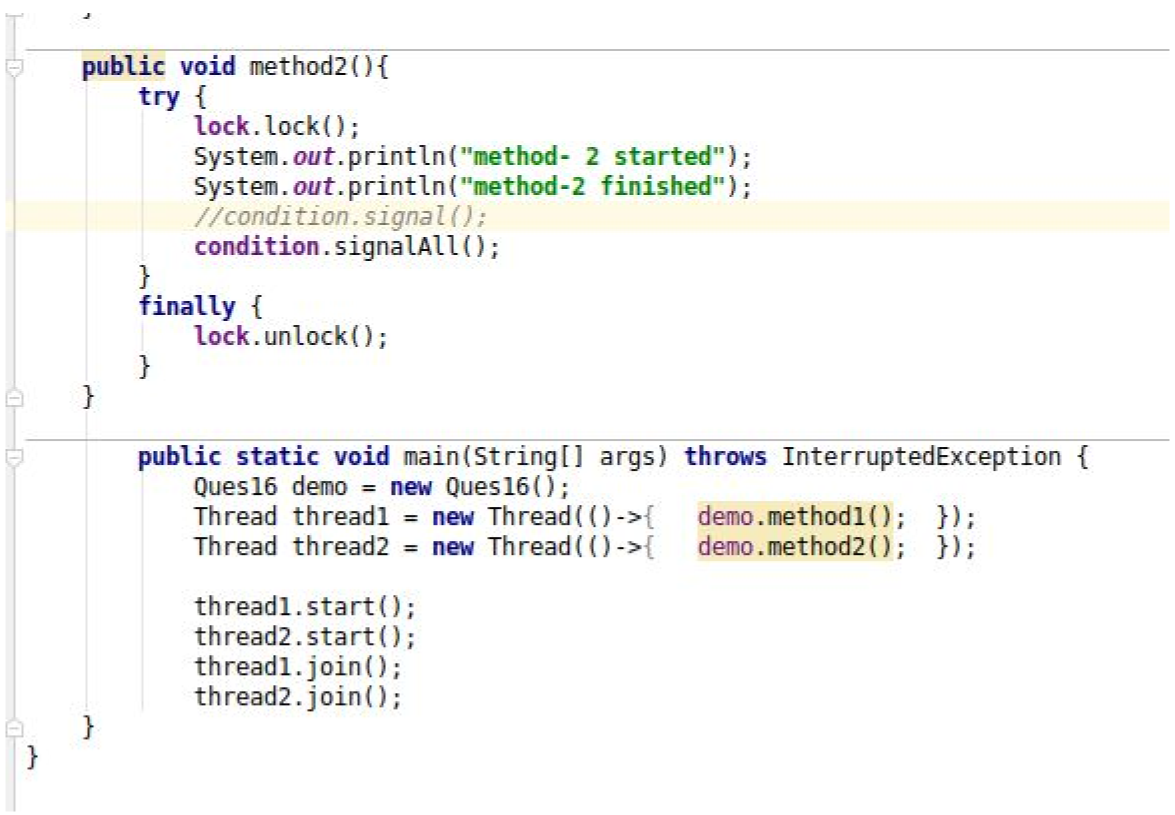


```
/home/yukti/.sdkman/candidates/java/8.0.202-amzn/bin/ja
Thread 1 Started
Thread 2 Started
Thread 3 Started
Thread 3 Done
Thread 2 Done
Thread 1 Done

Process finished with exit code 0
```

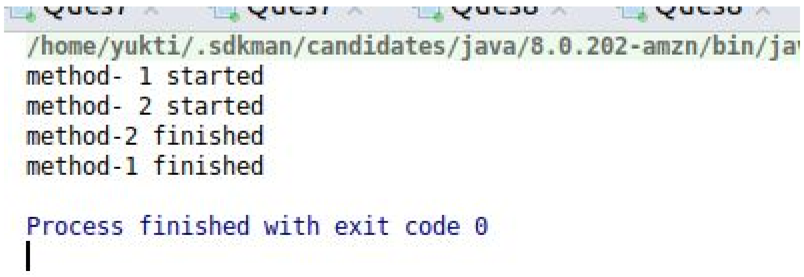
15. Use Reentract lock for coordinating 2 threads with signal(), signalAll() and wait().



```
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class Ques16 {
    Lock lock = new ReentrantLock( fair: true);
    Condition condition= lock.newCondition();

    public void method1(){
        try {
            lock.lock();
            System.out.println("method- 1 started");
            condition.await();
            System.out.println("method-1 finished");
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
                lock.unlock();
        }
    }
```



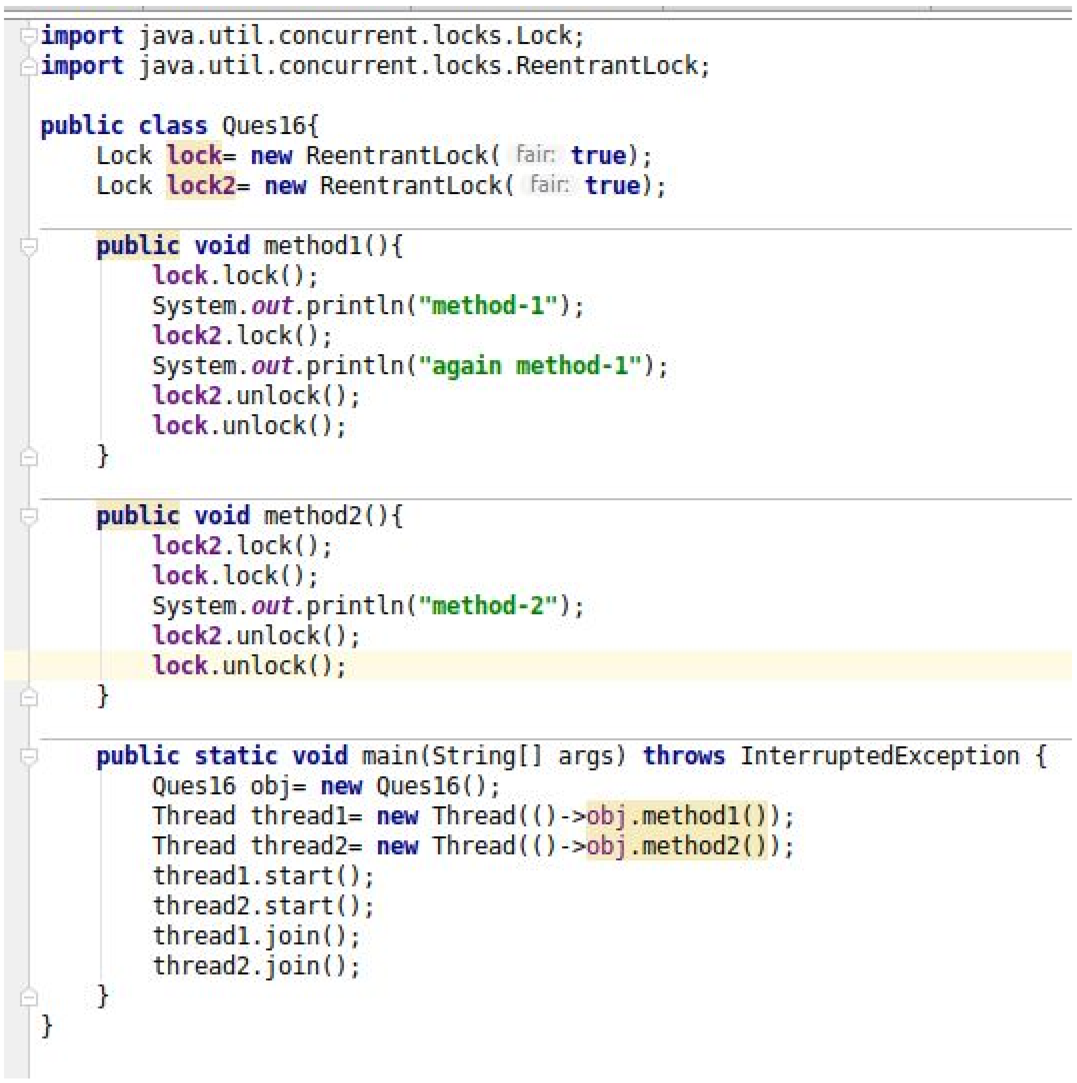
```
    public void method2(){
        try {
            lock.lock();
            System.out.println("method- 2 started");
            System.out.println("method-2 finished");
            //condition.signal();
            condition.signalAll();
        }
        finally {
            lock.unlock();
        }
    }

        public static void main(String[] args) throws InterruptedException {
            Ques16 demo = new Ques16();
            Thread thread1 = new Thread(()->{   demo.method1();  });
            Thread thread2 = new Thread(()->{   demo.method2();  });

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



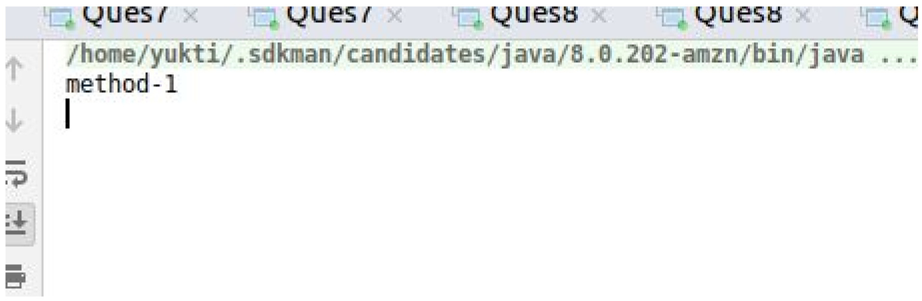
```
/home/yukti/.sdkman/candidates/java/8.0.202-amzn/bin/ja
method- 1 started
method- 2 started
method-2 finished
method-1 finished

Process finished with exit code 0
```

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

Creating a deadlock-

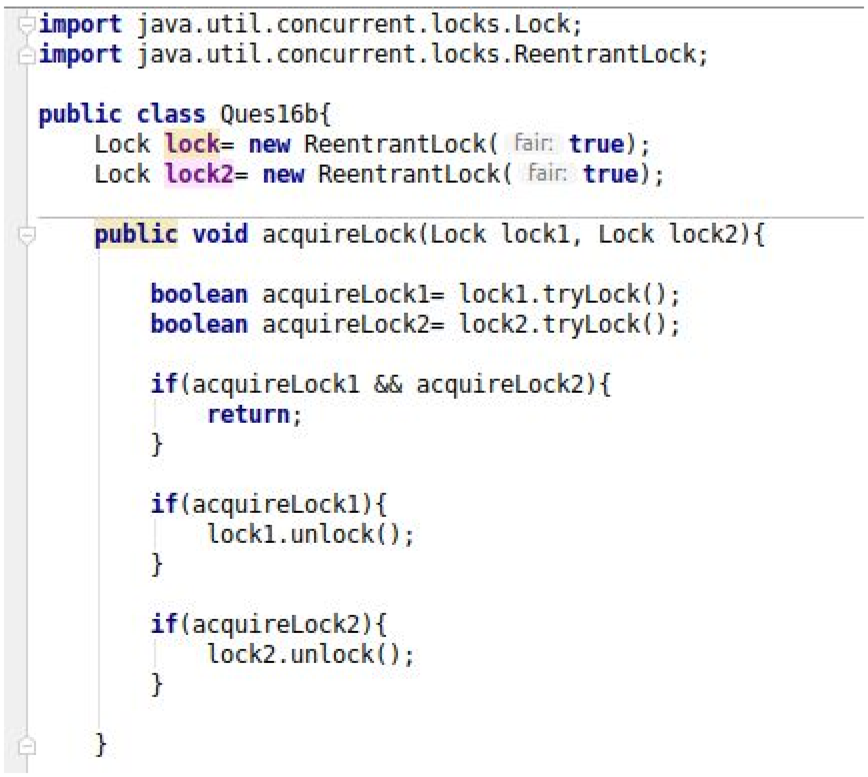


```
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class Ques16{
    Lock lock= new ReentrantLock( fair: true);
    Lock lock2= new ReentrantLock( fair: true);

    public void method1(){
        lock.lock();
        System.out.println("method-1");
        lock2.lock();
        System.out.println("again method-1");
        lock2.unlock();
        lock.unlock();
    }

    public void method2(){
        lock2.lock();
        lock.lock();
        System.out.println("method-2");
        lock2.unlock();
        lock.unlock();
    }

    public static void main(String[] args) throws InterruptedException {
        Ques16 obj= new Ques16();
        Thread thread1= new Thread(()->obj.method1());
        Thread thread2= new Thread(()->obj.method2());
        thread1.start();
        thread2.start();
        thread1.join();
        thread2.join();
    }
}
```

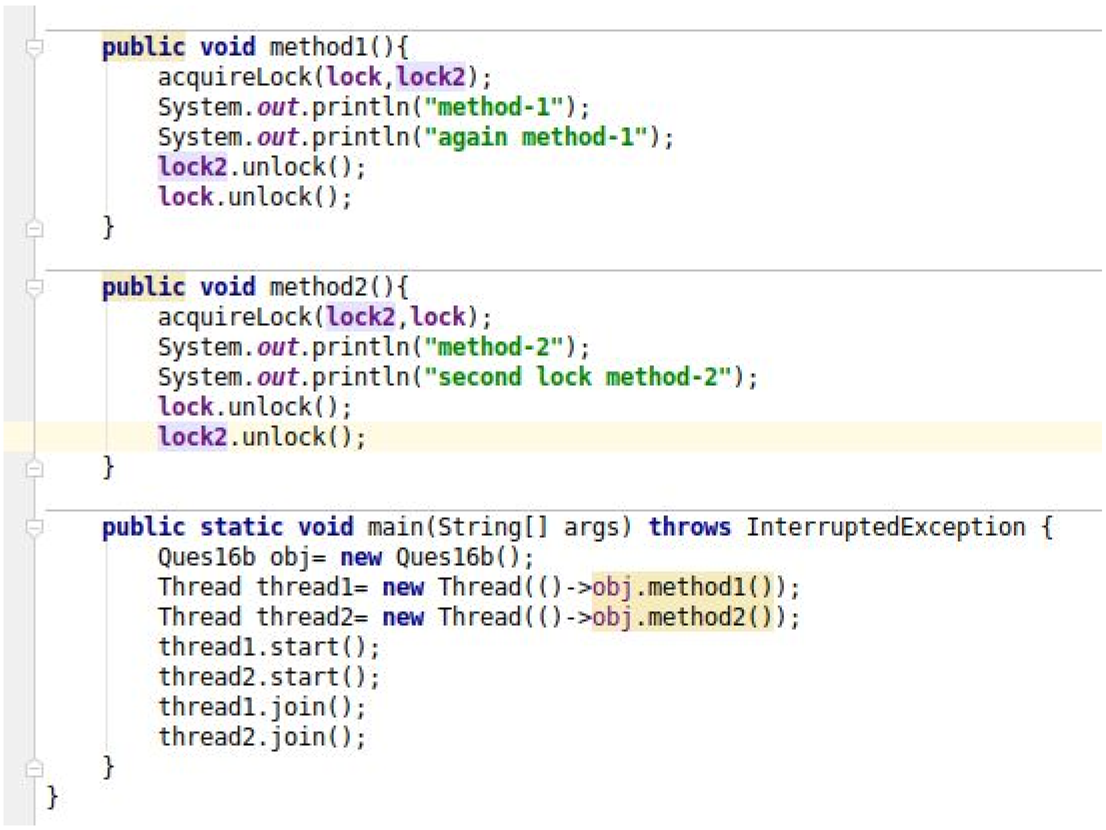


```
/home/yukti/.sdkman/candidates/java/8.0.202-amzn/bin/java ...
method-1
```

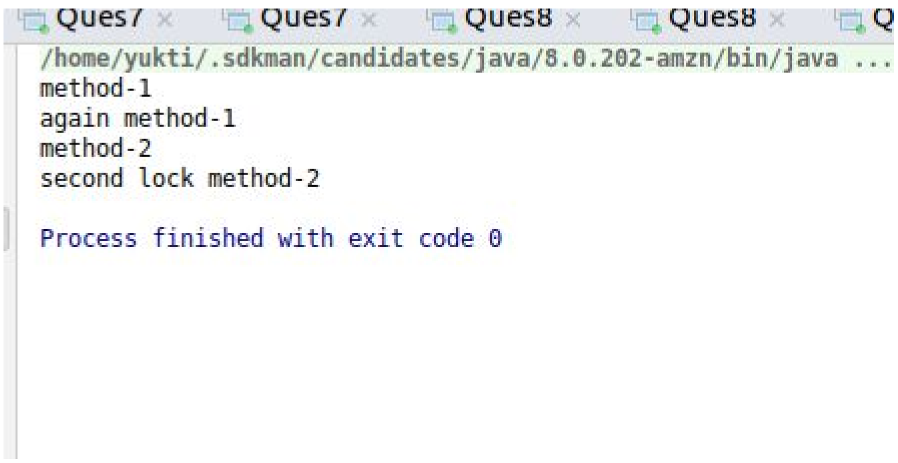
Solving deadlock-



```java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class Ques16b{
    Lock lock= new ReentrantLock( fair: true);
    Lock lock2= new ReentrantLock( fair: true);

    public void acquireLock(Lock lock1, Lock lock2){

        boolean acquireLock1= lock1.tryLock();
        boolean acquireLock2= lock2.tryLock();

        if(acquireLock1 && acquireLock2){
            return;
        }

        if(acquireLock1){
            lock1.unlock();
        }

        if(acquireLock2){
            lock2.unlock();
        }

    }
```



```java
    public void method1(){
        acquireLock(lock,lock2);
        System.out.println("method-1");
        System.out.println("again method-1");
        lock2.unlock();
        lock.unlock();
    }

    public void method2(){
        acquireLock(lock2,lock);
        System.out.println("method-2");
        System.out.println("second lock method-2");
        lock.unlock();
        lock2.unlock();
    }

    public static void main(String[] args) throws InterruptedException {
        Ques16b obj= new Ques16b();
        Thread thread1= new Thread(()->obj.method1());
        Thread thread2= new Thread(()->obj.method2());
        thread1.start();
        thread2.start();
        thread1.join();
        thread2.join();
    }
}
```



```
/home/yukti/.sdkman/candidates/java/8.0.202-amzn/bin/java ...
method-1
again method-1
method-2
second lock method-2

Process finished with exit code 0
```