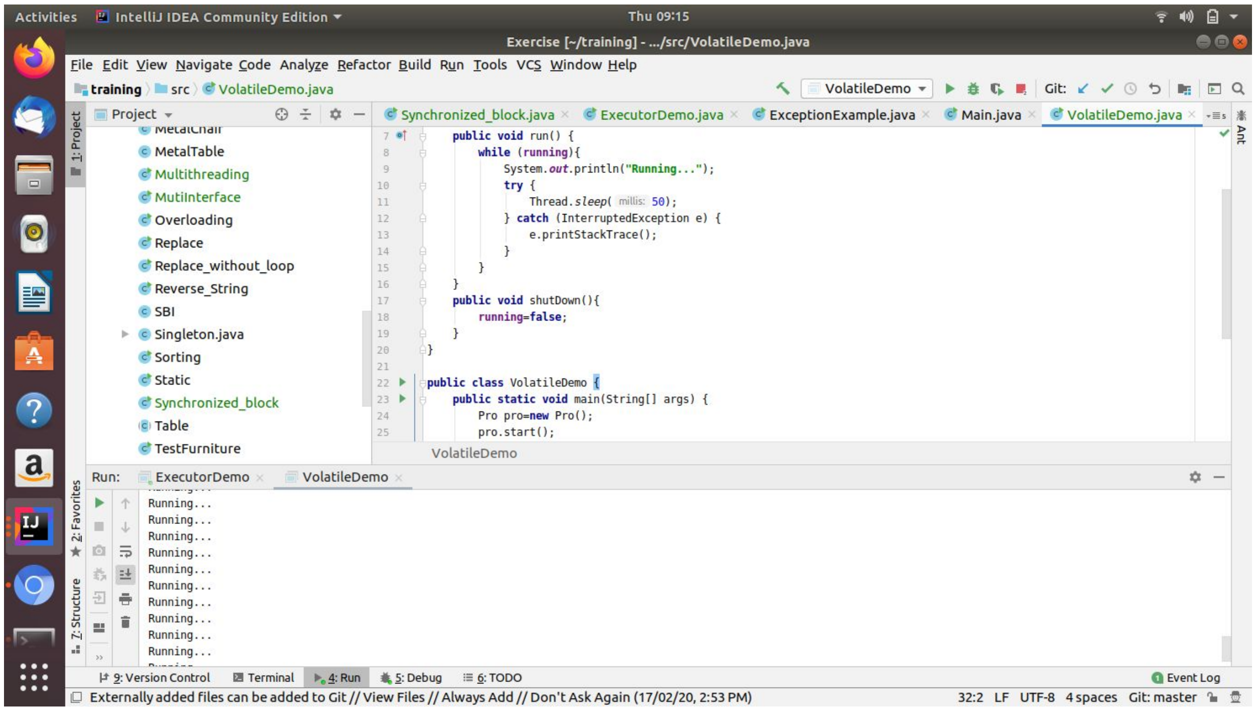
Q1 Write a program to demonstrate the use of volatile keyword.

Ans.


Exercise [~/training] - .../src/VolatileDemo.java
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Synchronized_block.java
ExecutorDemo.java
ExceptionExample.java
Main.java
VolatileDemo.java
Run: ExecutorDemo VolatileDemo

```
    public void run() {
        while (running){
            System.out.println("Running...");
            try {
                Thread.sleep( millis: 50);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
    public void shutDown(){
        running=false;
    }
}

public class VolatileDemo {
    public static void main(String[] args) {
        Pro pro=new Pro();
        pro.start();
```

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

Externally added files can be added to Git // View Files // Always Add // Don't Ask Again (17/02/20, 2:53 PM)

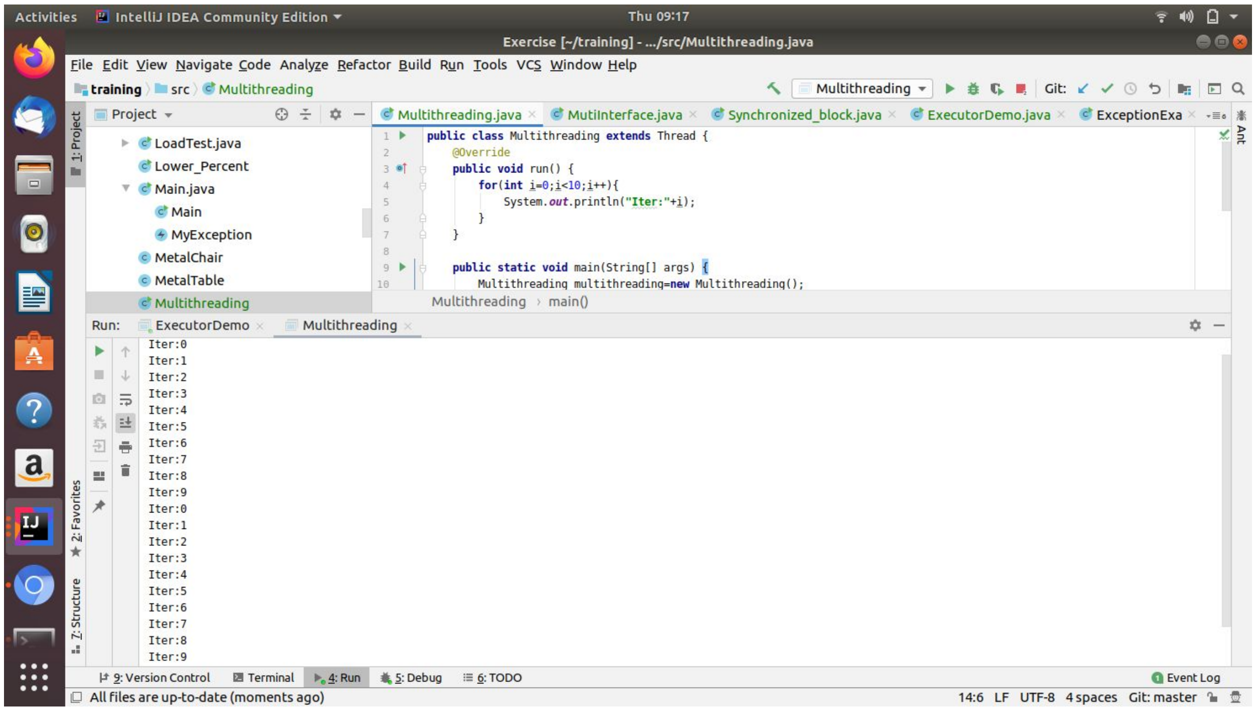
Code:-

```
import java.util.Scanner;

class Pro extends Thread{
  private volatile boolean running=true;

  @Override
  public void run() {
     while (running){
        System.out.println("Running...");
        try {
           Thread.sleep(50);
        } catch (InterruptedException e) {
           e.printStackTrace();
        }
     }
  }
  public void shutDown(){
     running=false;
  }
}

public class VolatileDemo {
  public static void main(String[] args) {
     Pro pro=new Pro();
     pro.start();
     new Scanner(System.in).nextLine();
```

```
        pro.shutDown();

    }
}
```

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

Ans.Thread Class:-



```
public class Multithreading extends Thread {
    @Override
    public void run() {
        for(int i=0;i<10;i++){
            System.out.println("Iter:"+i);
        }
    }

    public static void main(String[] args) {
        Multithreading multithreading=new Multithreading();
```

```
Iter:0
Iter:1
Iter:2
Iter:3
Iter:4
Iter:5
Iter:6
Iter:7
Iter:8
Iter:9
Iter:0
Iter:1
Iter:2
Iter:3
Iter:4
Iter:5
Iter:6
Iter:7
Iter:8
Iter:9
```

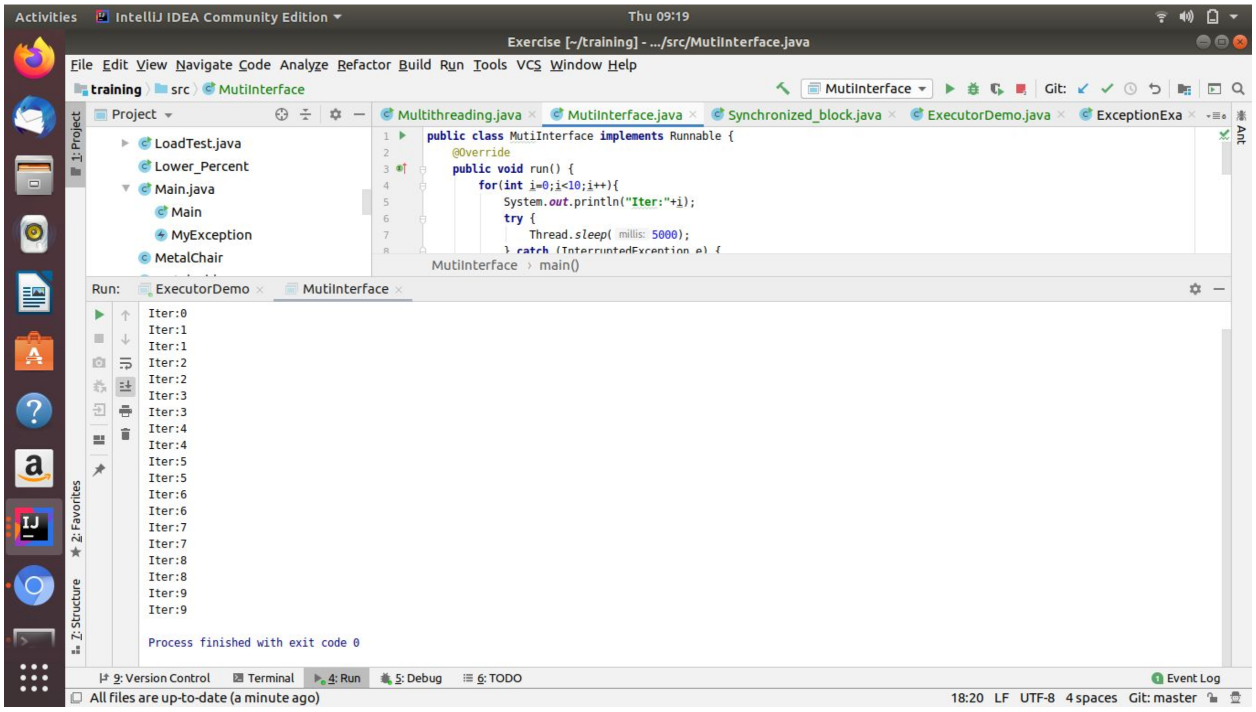
All files are up-to-date (moments ago)

Code:-

```
public class Multithreading extends Thread {

  @Override

  public void run() {

      for(int i=0;i<10;i++){

          System.out.println("Iter:"+i);

      }

  }


  public static void main(String[] args) {

      Multithreading multithreading=new Multithreading();
```

```
        Multithreading multithreading1=new Multithreading();
        multithreading.start();
        multithreading1.start();
    }
}
```

Runnable Interface:-



Exercise [~/training] - .../src/MutiInterface.java
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
training src MutiInterface
Multithreading.java
MutiInterface.java
Synchronized_block.java
ExecutorDemo.java
ExceptionExa
LoadTest.java
Lower_Percent
Main.java
Main
MyException
MetalChair
Run: ExecutorDemo MutiInterface
All files are up-to-date (a minute ago)

```
public class MutiInterface implements Runnable {
    @Override
    public void run() {
        for(int i=0;i<10;i++){
            System.out.println("Iter:"+i);
            try {
                Thread.sleep( millis: 5000);
            } catch (InterruptedException e) {
```

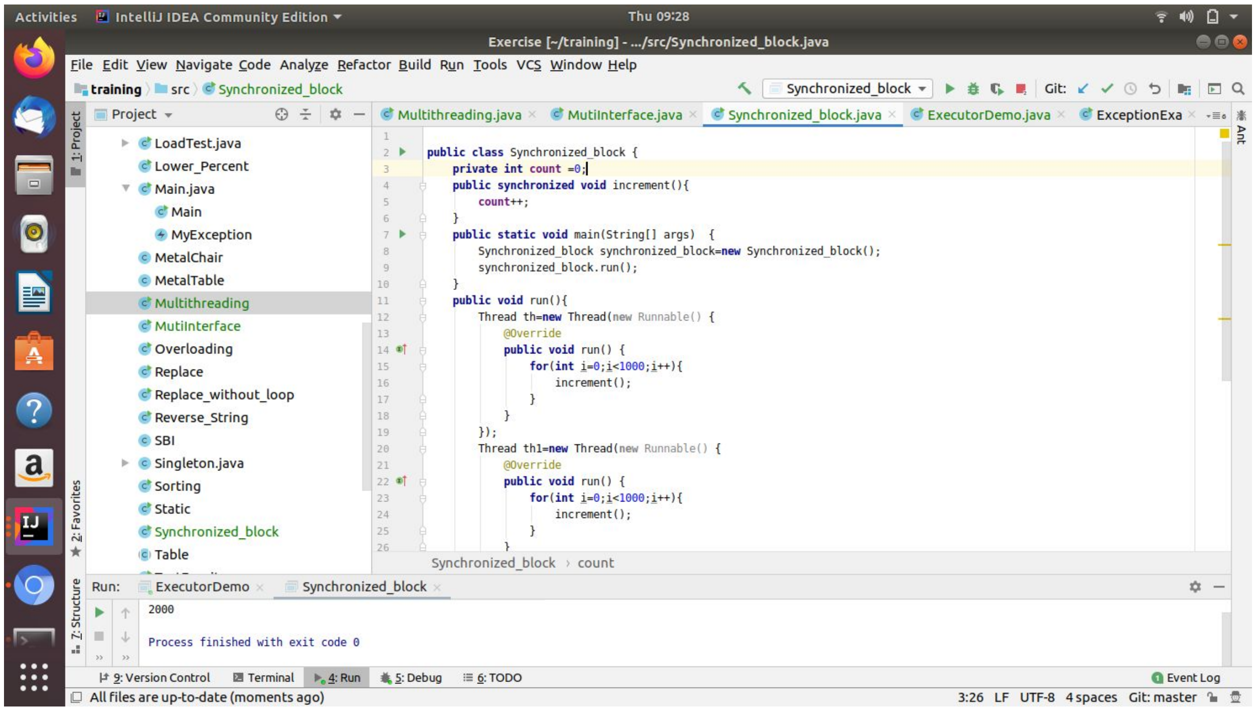
```
Iter:0
Iter:1
Iter:1
Iter:2
Iter:2
Iter:3
Iter:3
Iter:4
Iter:4
Iter:5
Iter:5
Iter:6
Iter:6
Iter:7
Iter:7
Iter:8
Iter:8
Iter:9
Iter:9

Process finished with exit code 0
```

Code:-

```
public class MutiInterface implements Runnable {
  @Override
  public void run() {
      for(int i=0;i<10;i++){
          System.out.println("Iter:"+i);
          try {
              Thread.sleep(5000);
          } catch (InterruptedException e) {
              e.printStackTrace();
          }
      }
  }

  public static void main(String[] args) {
      Thread t1=new Thread(new MutiInterface());
      Thread t2=new Thread(new MutiInterface());
      t1.start();
      t2.start();
  }
}
```

Q3.Write a program using synchronization block and synchronization method.

Ans.Synchronized Method:-


Exercise [~/training] - .../src/Synchronized_block.java

```
public class Synchronized_block {
    private int count =0;
    public synchronized void increment(){
        count++;
    }
    public static void main(String[] args)  {
        Synchronized_block synchronized_block=new Synchronized_block();
        synchronized_block.run();
    }
    public void run(){
        Thread th=new Thread(new Runnable() {
            @Override
            public void run() {
                for(int i=0;i<1000;i++){
                    increment();
                }
            }
        });
        Thread th1=new Thread(new Runnable() {
            @Override
            public void run() {
                for(int i=0;i<1000;i++){
                    increment();
                }
            }
```

```
2000

Process finished with exit code 0
```

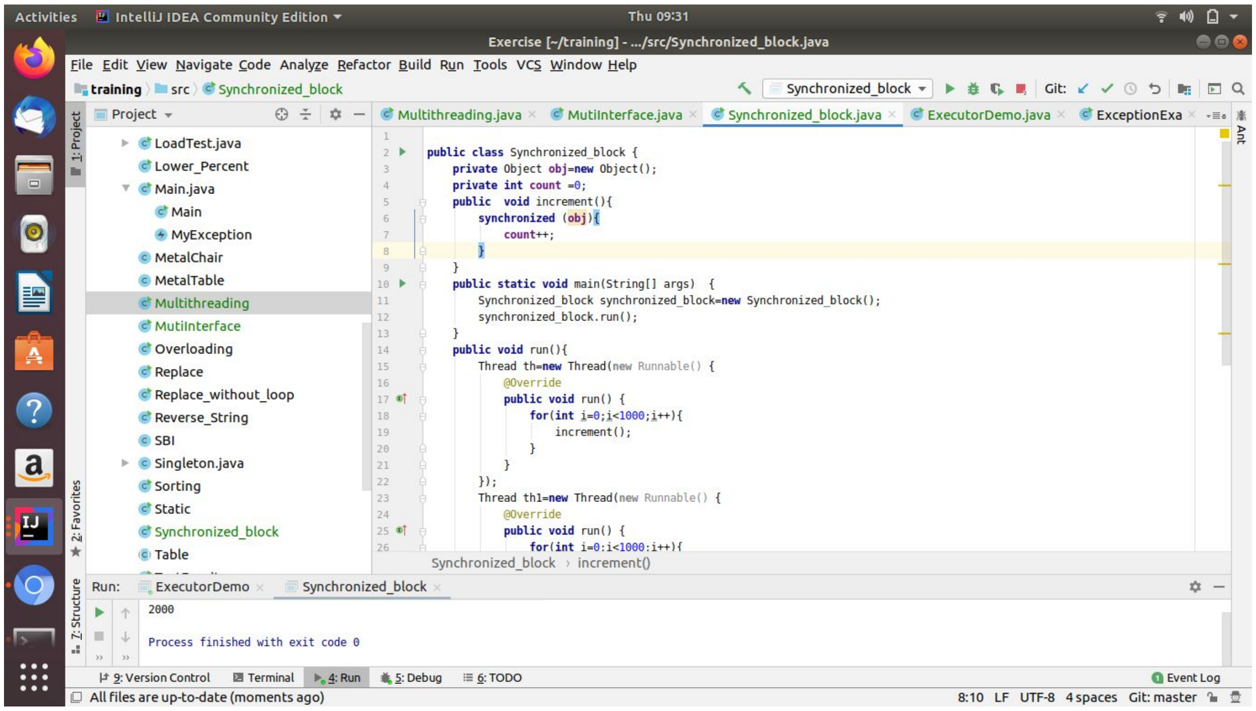
All files are up-to-date (moments ago)

Code:-

```
public class Synchronized_block {
  private int count =0;
  public synchronized void increment(){
        count++;
    }
    public static void main(String[] args)  {
        Synchronized_block synchronized_block=new Synchronized_block();
        synchronized_block.run();
    }
    public void run(){
        Thread th=new Thread(new Runnable() {
            @Override
            public void run() {
                for(int i=0;i<1000;i++){
                    increment();
                }
            }
        });
        Thread th1=new Thread(new Runnable() {
            @Override
            public void run() {
                for(int i=0;i<1000;i++){
                    increment();
                }
            }
        });
        th.start();
        th1.start();
        try {
            th.join();
            th1.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(count);
```

```
  }
}
```

Synchronized Block:-



```

public class Synchronized_block {
    private Object obj=new Object();
    private int count =0;
    public  void increment(){
        synchronized (obj){
            count++;
        }
    }
    public static void main(String[] args)  {
        Synchronized_block synchronized_block=new Synchronized_block();
        synchronized_block.run();
    }
    public void run(){
        Thread th=new Thread(new Runnable() {
            @Override
            public void run() {
                for(int i=0;i<1000;i++){
                    increment();
                }
            }
        });
        Thread th1=new Thread(new Runnable() {
            @Override
            public void run() {
                for(int i=0;i<1000;i++){
```

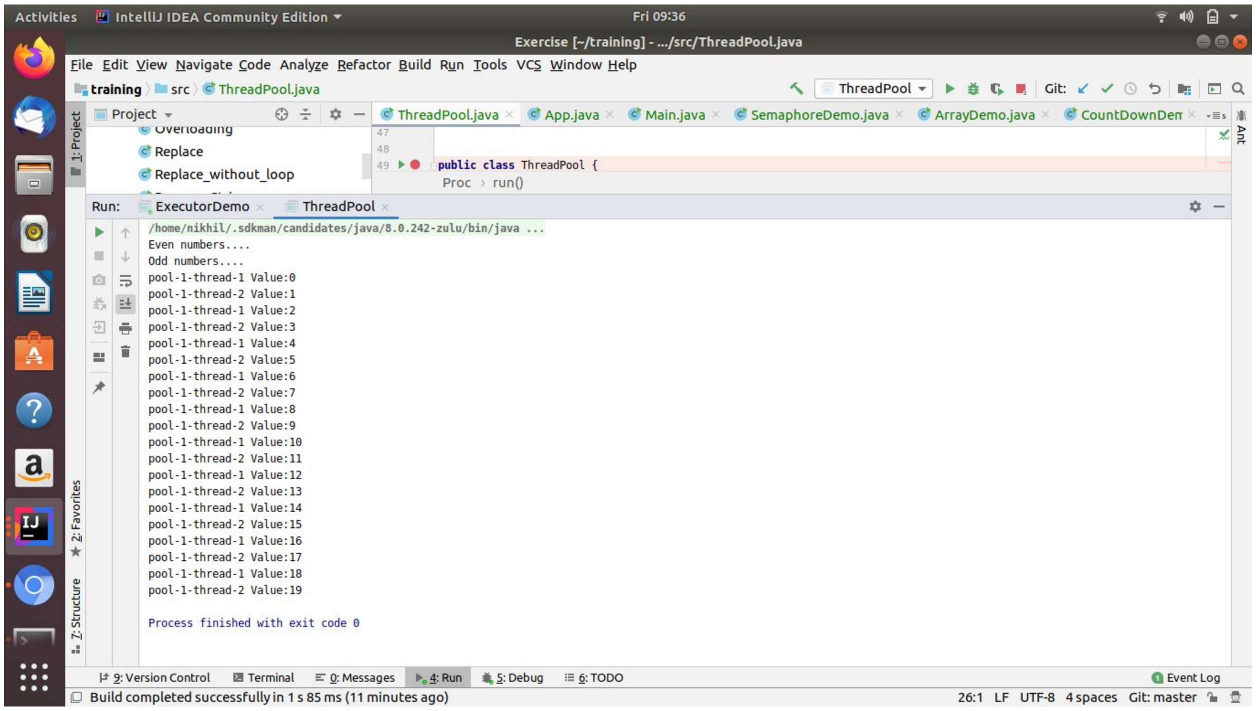
```
2000

Process finished with exit code 0
```

Code:-

```
public class Synchronized_block {
  private Object obj=new Object();
  private int count =0;
  public  void increment(){
      synchronized (obj){
          count++;
      }
  }
  public static void main(String[] args)  {
      Synchronized_block synchronized_block=new Synchronized_block();
      synchronized_block.run();
  }
  public void run(){
      Thread th=new Thread(new Runnable() {
          @Override
          public void run() {
              for(int i=0;i<1000;i++){
                  increment();
              }
          }
      });
      Thread th1=new Thread(new Runnable() {
          @Override
          public void run() {
              for(int i=0;i<1000;i++){
                  increment();
              }
          }
      });
      th.start();
      th1.start();
      try {
          th.join();
          th1.join();
      } catch (InterruptedException e) {
          e.printStackTrace();
      }
      System.out.println(count);
  }
}
```

Q4.Write a program to create a Thread pool of 2 threads where one Thread will print even numbers and other will print odd numbers.

Ans.



```
public class ThreadPool {
```

```
/home/nikhil/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Even numbers....
Odd numbers....
pool-1-thread-1 Value:0
pool-1-thread-2 Value:1
pool-1-thread-1 Value:2
pool-1-thread-2 Value:3
pool-1-thread-1 Value:4
pool-1-thread-2 Value:5
pool-1-thread-1 Value:6
pool-1-thread-2 Value:7
pool-1-thread-1 Value:8
pool-1-thread-2 Value:9
pool-1-thread-1 Value:10
pool-1-thread-2 Value:11
pool-1-thread-1 Value:12
pool-1-thread-2 Value:13
pool-1-thread-1 Value:14
pool-1-thread-2 Value:15
pool-1-thread-1 Value:16
pool-1-thread-2 Value:17
pool-1-thread-1 Value:18
pool-1-thread-2 Value:19

Process finished with exit code 0
```

Build completed successfully in 1 s 85 ms (11 minutes ago)

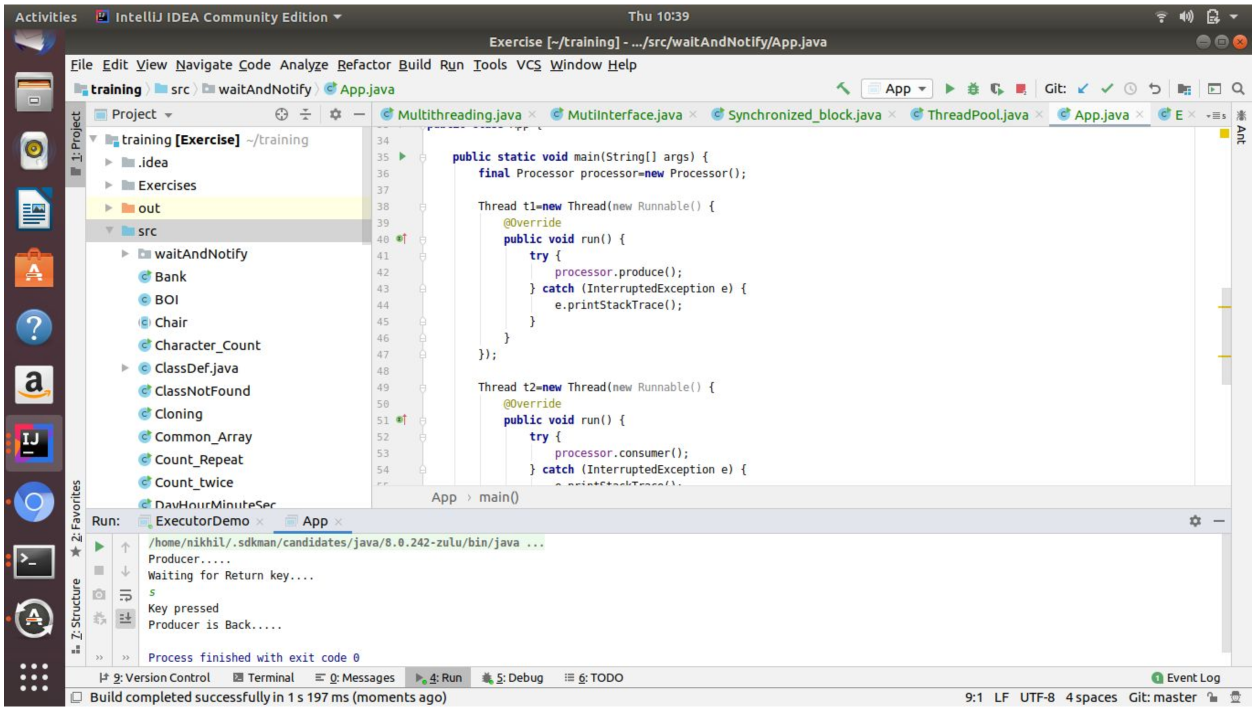
Code:-

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

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;


class Proce implements Runnable{


  @Override

  public void run() {

        System.out.println("Odd numbers....");

        try {

            Thread.sleep(500);

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

        for(int i=1;i<20;i+=2){
                System.out.println(Thread.currentThread().getName()+" Value:"+i);
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }

    }
}

class Proc implements Runnable{

    @Override
    public void run() {
            System.out.println("Even numbers....");
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            for(int i=0;i<20;i+=2){
                System.out.println(Thread.currentThread().getName()+" Value:"+i);
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
```

```
  }
}


public class ThreadPool {
  public static void main(String[] args) {
      ExecutorService executor = Executors.newFixedThreadPool(2);
      executor.submit(new Proc());
      executor.submit(new Proce());
      executor.shutdown();

      try {
          executor.awaitTermination(1, TimeUnit.DAYS);
      } catch (InterruptedException e) {
          e.printStackTrace();
      }
  }
}
```

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

Ans.



```
    public static void main(String[] args) {
        final Processor processor=new Processor();

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

        Thread t2=new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    processor.consumer();
                } catch (InterruptedException e) {
```

```
/home/nikhil/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Producer.....
Waiting for Return key....
s
Key pressed
Producer is Back.....

Process finished with exit code 0
```

Build completed successfully in 1 s 197 ms (moments ago)

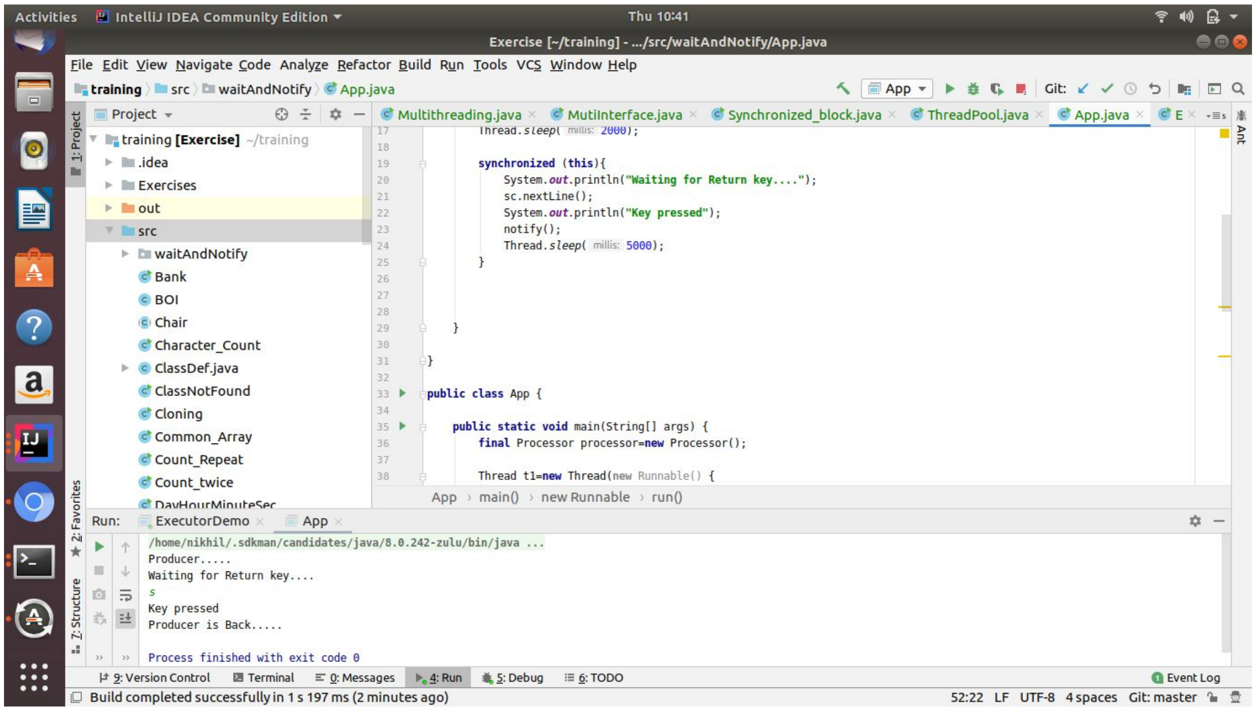
Code:-

```
package waitAndNotify;


import java.util.Scanner;


class Processor {
  public void produce() throws InterruptedException {
      synchronized (this){
          System.out.println("Producer.....");
          wait();
          System.out.println("Producer is Back.....");
      }


  }


  public void consumer() throws InterruptedException {
      Scanner sc=new Scanner(System.in);
```

```
        Thread.sleep(2000);

        synchronized (this){
            System.out.println("Waiting for Return key....");
            sc.nextLine();
            System.out.println("Key pressed");
            notify();
            Thread.sleep(5000);
        }



    }

}

public class App {

    public static void main(String[] args) {
        final Processor processor=new Processor();

        Thread t1=new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    processor.produce();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        Thread t2=new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    processor.consumer();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

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

Q6.Write a program to demonstrate sleep and join methods.

Ans.



```
        Thread.sleep( millis: 2000);

        synchronized (this){
            System.out.println("Waiting for Return key....");
            sc.nextLine();
            System.out.println("Key pressed");
            notify();
            Thread.sleep( millis: 5000);
        }



    }

}

public class App {

    public static void main(String[] args) {
        final Processor processor=new Processor();

        Thread t1=new Thread(new Runnable() {
```

```
/home/nikhil/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Producer.....
Waiting for Return key....
s
Key pressed
Producer is Back.....

Process finished with exit code 0
```

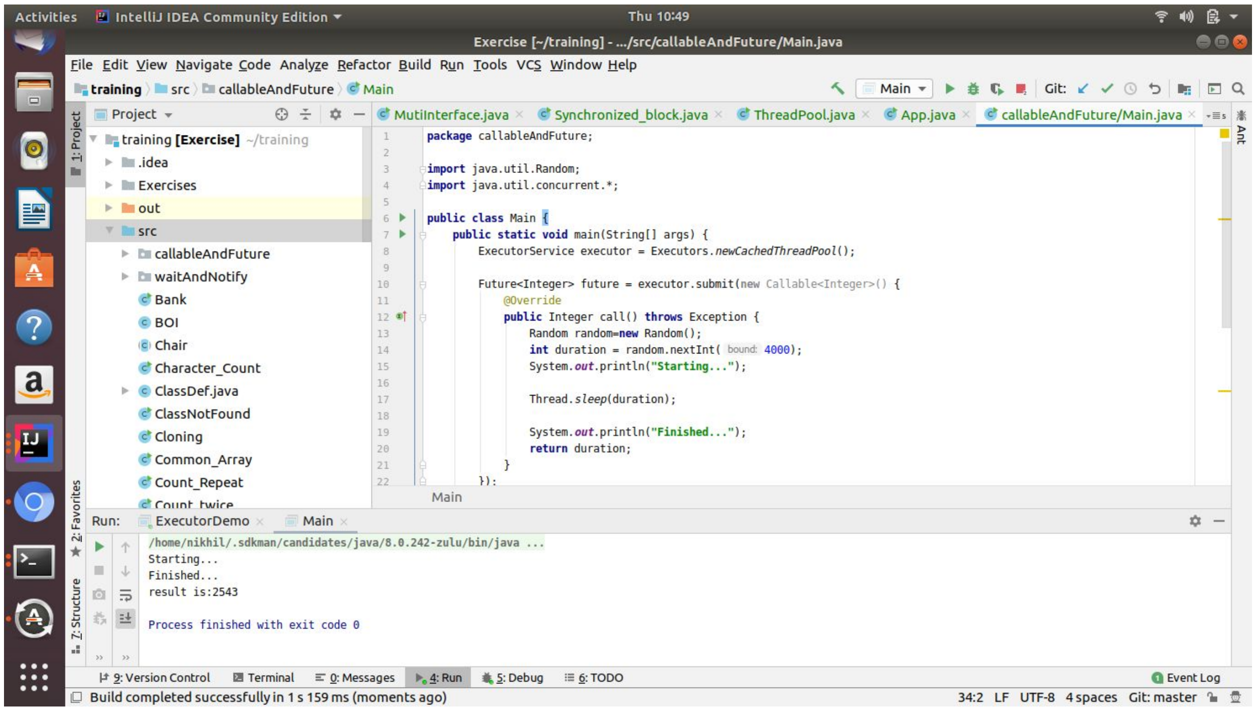
Build completed successfully in 1 s 197 ms (2 minutes ago)

```
package waitAndNotify;


import java.util.Scanner;


class Processor {
  public void produce() throws InterruptedException {
     synchronized (this){
        System.out.println("Producer.....");
        wait();
        System.out.println("Producer is Back.....");
     }


  }


  public void consumer() throws InterruptedException {
     Scanner sc=new Scanner(System.in);
     Thread.sleep(2000);
```

```
        synchronized (this){
            System.out.println("Waiting for Return key....");
            sc.nextLine();
            System.out.println("Key pressed");
            notify();
            Thread.sleep(5000);
        }




    }

}

public class App {

    public static void main(String[] args) {
        final Processor processor=new Processor();

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

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

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

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

Ans.



```java
package callableAndFuture;

import java.util.Random;
import java.util.concurrent.*;

public class Main {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newCachedThreadPool();

        Future<Integer> future = executor.submit(new Callable<Integer>() {
            @Override
            public Integer call() throws Exception {
                Random random=new Random();
                int duration = random.nextInt( bound: 4000);
                System.out.println("Starting...");

                Thread.sleep(duration);

                System.out.println("Finished...");
                return duration;
            }
        });
```

```
/home/nikhil/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Starting...
Finished...
result is:2543

Process finished with exit code 0
```

Build completed successfully in 1 s 159 ms (moments ago)

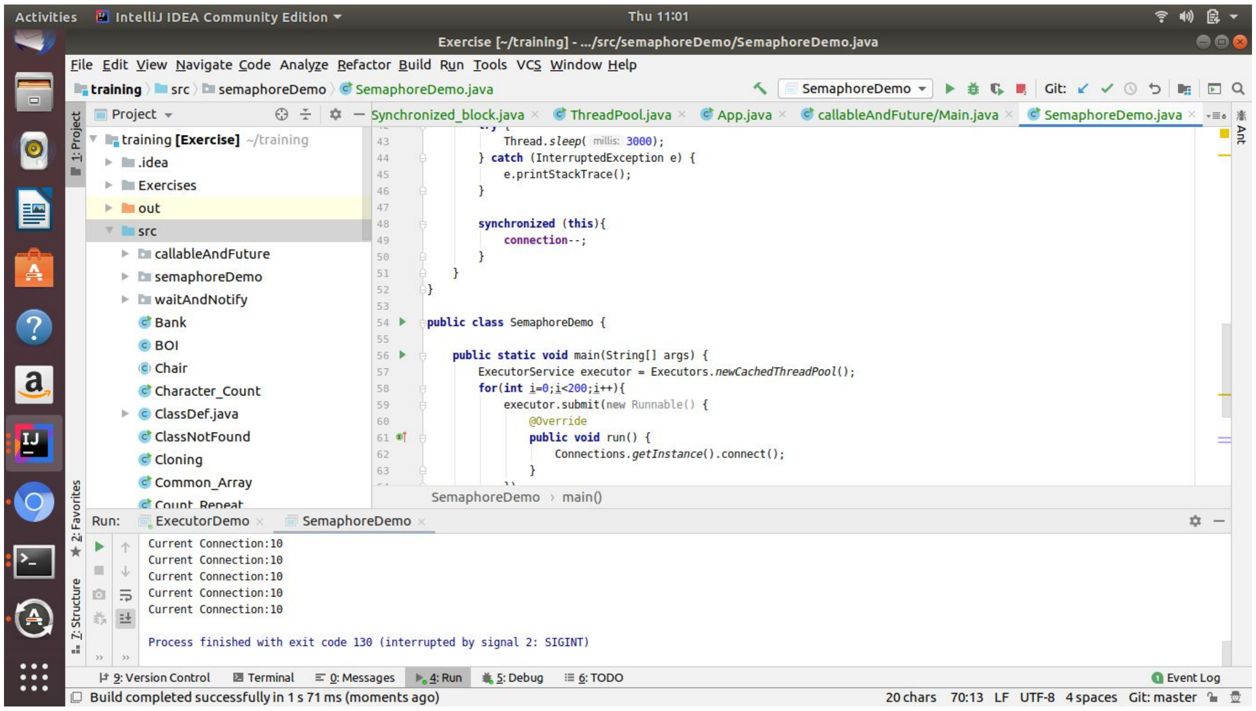
Code:-

```java
package callableAndFuture;


import java.util.Random;
import java.util.concurrent.*;


public class Main {
  public static void main(String[] args) {
      ExecutorService executor = Executors.newCachedThreadPool();


      Future<Integer> future = executor.submit(new Callable<Integer>() {
          @Override
          public Integer call() throws Exception {
              Random random=new Random();
              int duration = random.nextInt(4000);
              System.out.println("Starting...");
```

```
                Thread.sleep(duration);

                System.out.println("Finished...");
                return duration;
            }
        });

        executor.shutdown();

        try {
            System.out.println("result is:"+ future.get());
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            e.printStackTrace();
        }
    }
}
```

Q8.Write a program to demonstrate the use of semaphore.

Ans.



```
            Thread.sleep( millis: 3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        synchronized (this){
            connection--;
        }
    }
}

public class SemaphoreDemo {

    public static void main(String[] args) {
        ExecutorService executor = Executors.newCachedThreadPool();
        for(int i=0;i<200;i++){
            executor.submit(new Runnable() {
                @Override
                public void run() {
                    Connections.getInstance().connect();
                }
```

```
Current Connection:10
Current Connection:10
Current Connection:10
Current Connection:10
Current Connection:10

Process finished with exit code 130 (interrupted by signal 2: SIGINT)
```

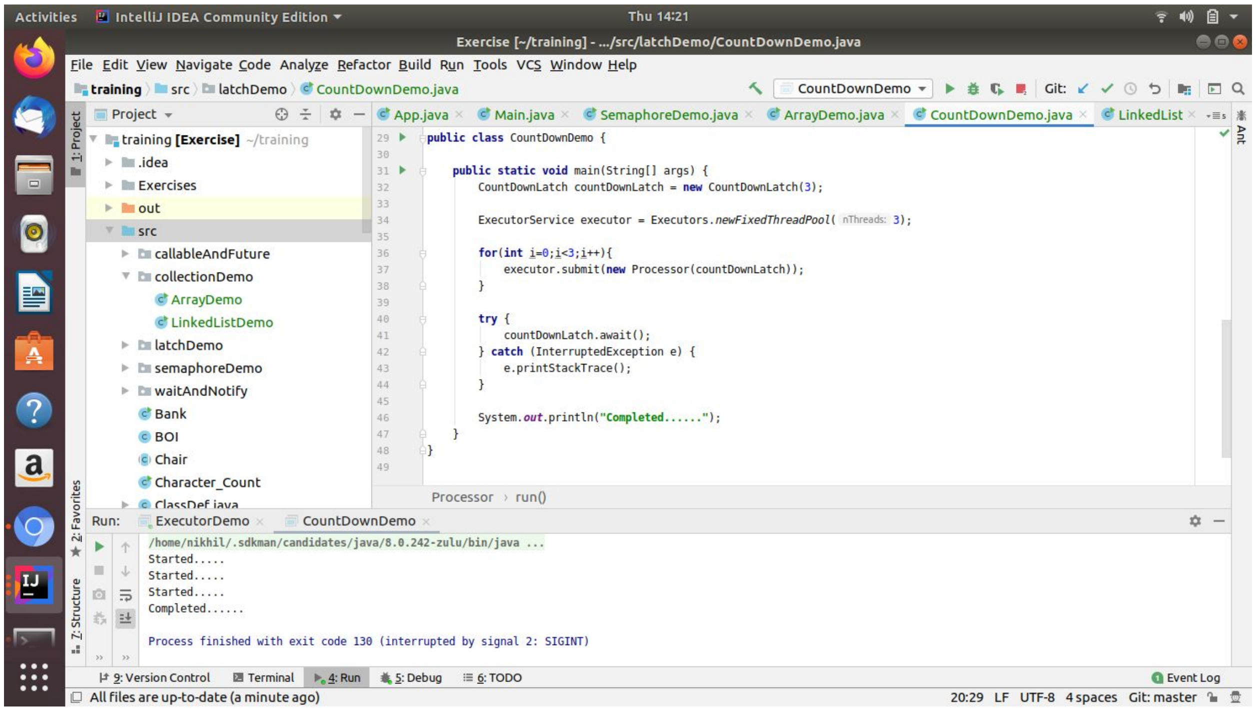
Build completed successfully in 1 s 71 ms (moments ago)

Code:-

```
package semaphoreDemo;



import java.sql.Connection;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Semaphore;

import java.util.concurrent.TimeUnit;


class Connections{

   private static Connections instance = new Connections();

   private Semaphore semaphore = new Semaphore(10);

   private int connection=0;


   private Connections(){
}

public static Connections getInstance(){
    return instance;
}
public void connect(){
    try {
        semaphore.acquire();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    try{
        doconnect();
    }
    finally {
        semaphore.release();
    }
}

public void doconnect(){

    synchronized (this){
        connection++;
        System.out.println("Current Connection:"+connection);
    }
    try {
        Thread.sleep(3000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    synchronized (this){
            connection--;
        }
    }
}

public class SemaphoreDemo {

    public static void main(String[] args) {
        ExecutorService executor = Executors.newCachedThreadPool();
        for(int i=0;i<200;i++){
            executor.submit(new Runnable() {
                @Override
                public void run() {
                    Connections.getInstance().connect();
                }
            });
        }
        executor.shutdown();
        try {
            executor.awaitTermination(1, TimeUnit.DAYS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Q9.Write a program to demonstrate the use of CountDownLatch.

Ans.



```
public class CountDownDemo {

    public static void main(String[] args) {
        CountDownLatch countDownLatch = new CountDownLatch(3);

        ExecutorService executor = Executors.newFixedThreadPool( nThreads: 3);

        for(int i=0;i<3;i++){
            executor.submit(new Processor(countDownLatch));
        }

        try {
            countDownLatch.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Completed......");
    }
}
```

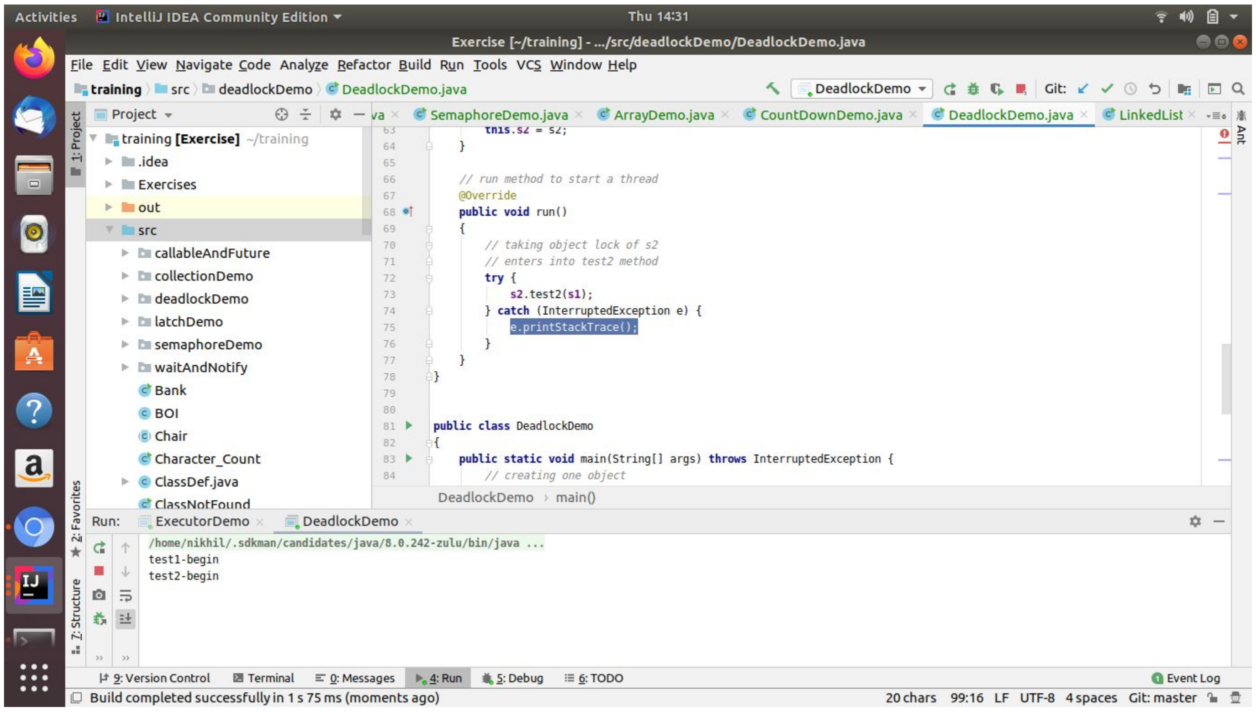
```
/home/nikhil/.sdkman/candidates/java/8.0.242-zulu/bin/java ...
Started.....
Started.....
Started.....
Completed......

Process finished with exit code 130 (interrupted by signal 2: SIGINT)
```

Code:-

```
package latchDemo;


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


class Processor implements Runnable{


  private CountDownLatch latch;


  public Processor(CountDownLatch latch){
      this.latch = latch;
  }


  @Override
  public void run() {
      System.out.println("Started.....");

      try {
          Thread.sleep(400);
      } catch (InterruptedException e) {
          e.printStackTrace();
      }

      latch.countDown();
  }
}

public class CountDownDemo {

  public static void main(String[] args) {
      CountDownLatch countDownLatch = new CountDownLatch(3);

      ExecutorService executor = Executors.newFixedThreadPool(3);

      for(int i=0;i<3;i++){
          executor.submit(new Processor(countDownLatch));
      }

      try {
          countDownLatch.await();
      } catch (InterruptedException e) {
          e.printStackTrace();
      }

      System.out.println("Completed......");
  }
}
```

Q10.Write a program which creates deadlock between 2 threads.

Ans.



```
        this.s2 = s2;
    }

    // run method to start a thread
    @Override
    public void run()
    {
        // taking object lock of s2
        // enters into test2 method
        try {
            s2.test2(s1);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}


public class DeadlockDemo
{
    public static void main(String[] args) throws InterruptedException {
        // creating one object
```

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

Build completed successfully in 1 s 75 ms (moments ago)

Code:-

```
class Shared
{
  synchronized void test1(Shared s2) throws InterruptedException {
      System.out.println("test1-begin");
      Thread.sleep(1000);
      s2.test2(this);
      System.out.println("test1-end");
  }



  synchronized void test2(Shared s1) throws InterruptedException {
      System.out.println("test2-begin");
      Thread.sleep(1000);
```

```
        s1.test1(this);
        System.out.println("test2-end");
    }
}


class Thread1 extends Thread
{
    private Shared s1;
    private Shared s2;

    public Thread1(Shared s1, Shared s2)
    {
        this.s1 = s1;
        this.s2 = s2;
    }



    @Override
    public void run()
    {

        try {
            s1.test1(s2);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```java
class Thread2 extends Thread
{
  private Shared s1;
  private Shared s2;


  public Thread2(Shared s1, Shared s2)
  {
     this.s1 = s1;
     this.s2 = s2;
  }


  @Override
  public void run()
  {

     try {
        s2.test2(s1);
     } catch (InterruptedException e) {
        e.printStackTrace();
     }
  }
}


public class DeadlockDemo
{
  public static void main(String[] args) throws InterruptedException {
     Shared s1 = new Shared();
     Shared s2 = new Shared();
```

```
        Thread1 t1 = new Thread1(s1, s2);

        t1.start();

        Thread2 t2 = new Thread2(s1, s2);

        t2.start();

        Thread.sleep(2000);

    }

}
```