*//Ques1: Write a programe do to demonstrate the use of volatile keyword.*

**import** java.util.Scanner;

**public class** Ques1 {

**public static void** main(String[] args) {

NewThread obj = **new** NewThread();

obj.start();

Scanner sc = **new** Scanner(System.***in***);

sc.nextLine();

obj.setFlagFalse();

}

}

**class** NewThread **extends** Thread{

**public volatile boolean flag** = **true**;

**public void** run(){

**while**(**flag**){

System.***out***.println(**"Flag is "**+**flag**);

**try**{

Thread.*sleep*(500);

}

**catch** (InterruptedException e) {

e.printStackTrace();

}

}

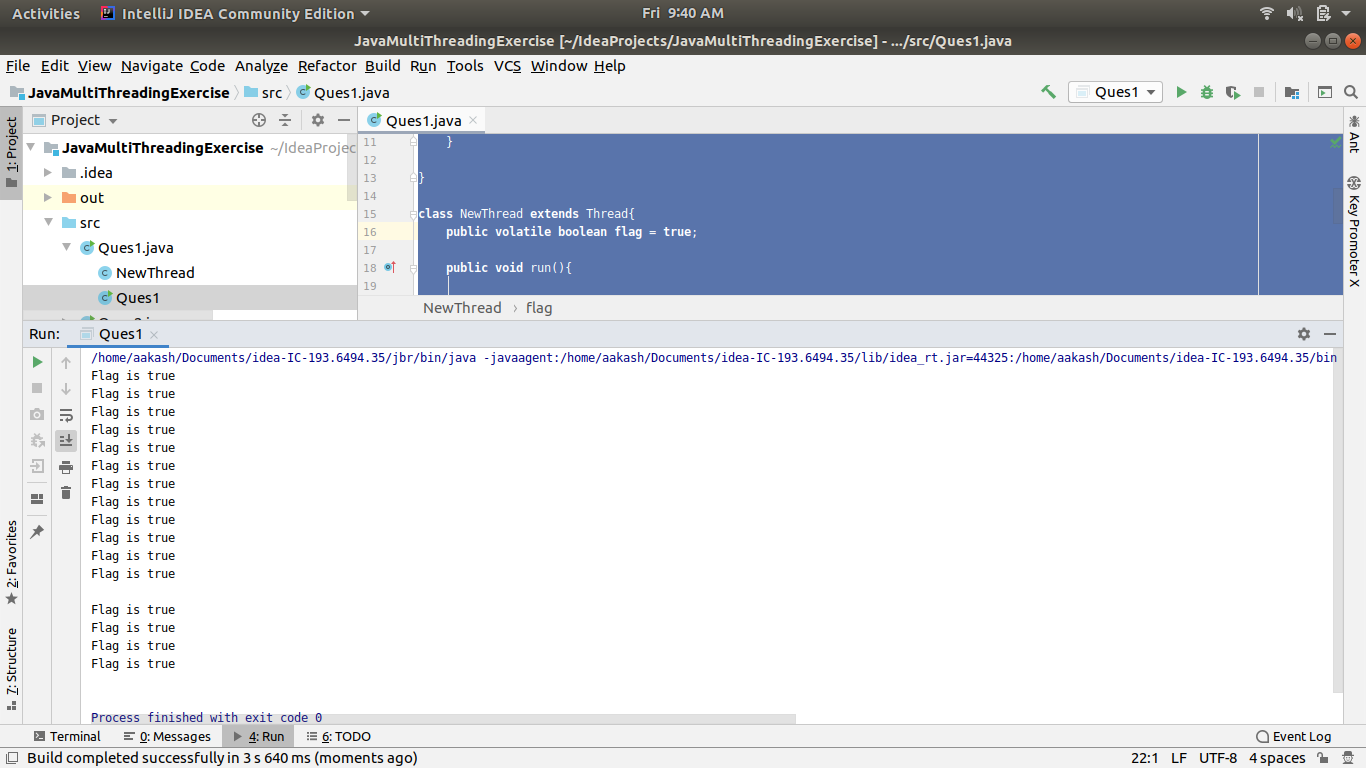
}

**public void** setFlagFalse(){

**flag** = **false**;

}

}



*//Ques2: Write a program to create a thread using Thread class and Runnable interface each.*

**public class** Ques2 {

**public static void** main(String[] args) {

System.***out***.println(**"This is Main Thread "**);

System.***out***.println(**"Current Thread: "**+ Thread.*currentThread*().getName());

UsingRunnableInterface threadFromRunnable = **new** UsingRunnableInterface();

Thread T = **new** Thread(threadFromRunnable);

T.start();

UsingThreadClass threadFromThreadClass = **new** UsingThreadClass();

threadFromThreadClass.start();

}

}

**class** UsingThreadClass **extends** Thread{

**public void** run(){

**try** {

Thread.*sleep*(4000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println(**"This Thread is spun using Thread class"**);

System.***out***.println(**"Current Thread: "**+ Thread.*currentThread*().getName());

}

}

**class** UsingRunnableInterface **implements** Runnable{

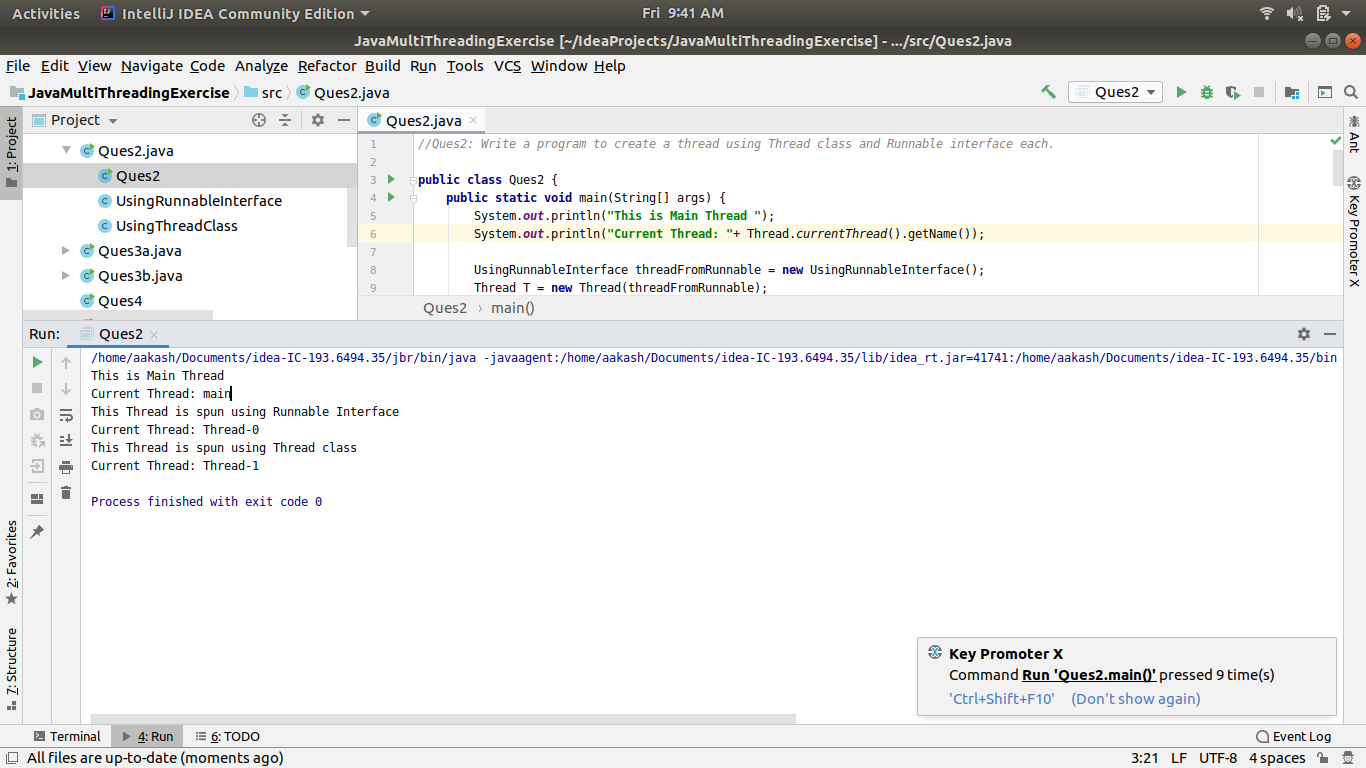
**public void** run(){

System.***out***.println(**"This Thread is spun using Runnable Interface"**);

System.***out***.println(**"Current Thread: "**+ Thread.*currentThread*().getName());

}

}



*//Ques 3a: Write a program using synchronization method*

**public class** Ques3a {

**public static void** main(String[] args)

{

Line obj = **new** Line();

Train train1 = **new** Train(obj);

Train train2 = **new** Train(obj);

train1.start();

train2.start();

}

}

**class** Line

{

**synchronized public void** getLine()

{

**for** (**int** i = 0; i < 3; i++)

{

System.***out***.println(i);

**try**

{

Thread.*sleep*(400);

}

**catch** (Exception e)

{

System.***out***.println(e);

}

}

}

}

**class** Train **extends** Thread

{

Line **line**;

Train(Line line)

{

**this**.**line** = line;

}

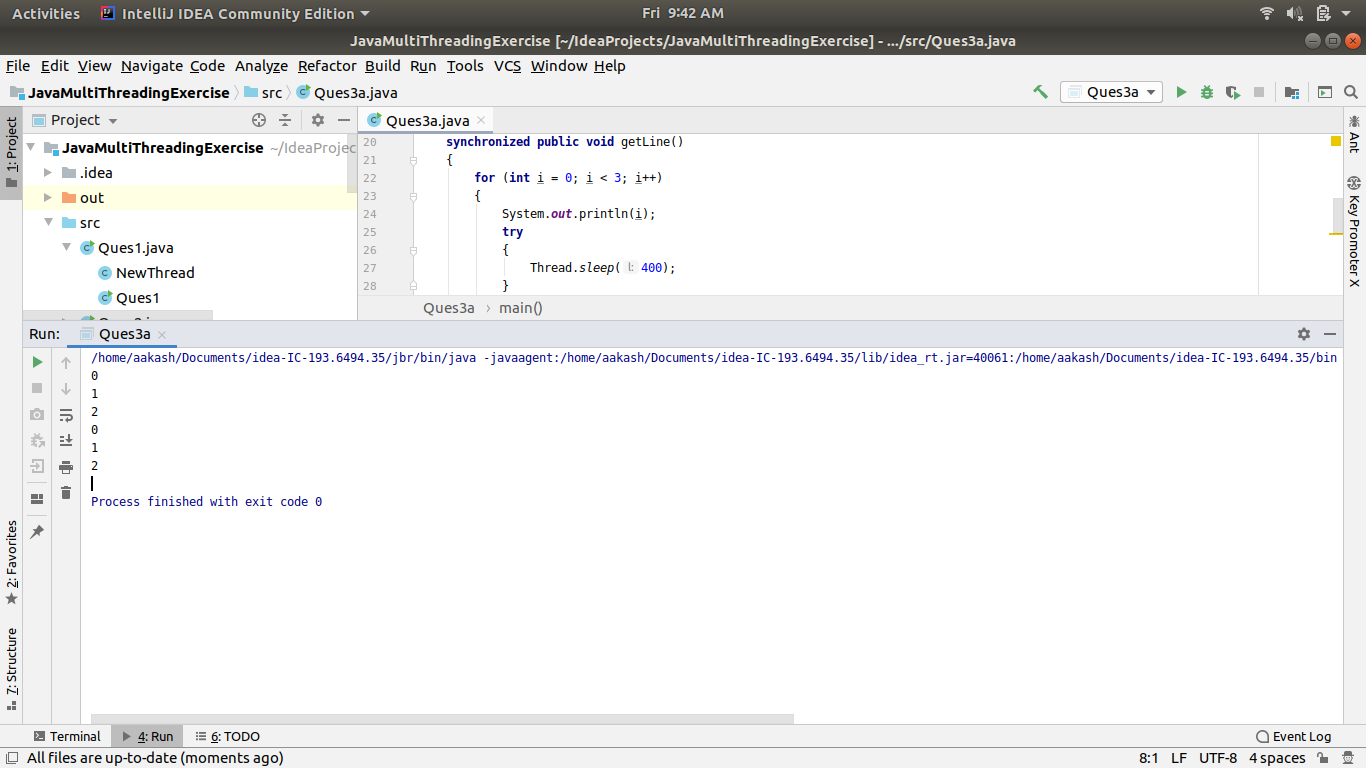
@Override

**public void** run()

{

**line**.getLine();

}}



*//Ques 3b: Write a program using synchronization block*

**import** java.util.\*;

**class** Student

{

String **name** = **""**;

**public int count** = 0;

**public void** StudentName(String student1, List<String> list)

{

**synchronized**(**this**)

{

**name** = student1;

**count**++;

}

list.add(student1);

}

}

**public class** Ques3b {

**public static void** main (String[] args)

{

Student student = **new** Student();

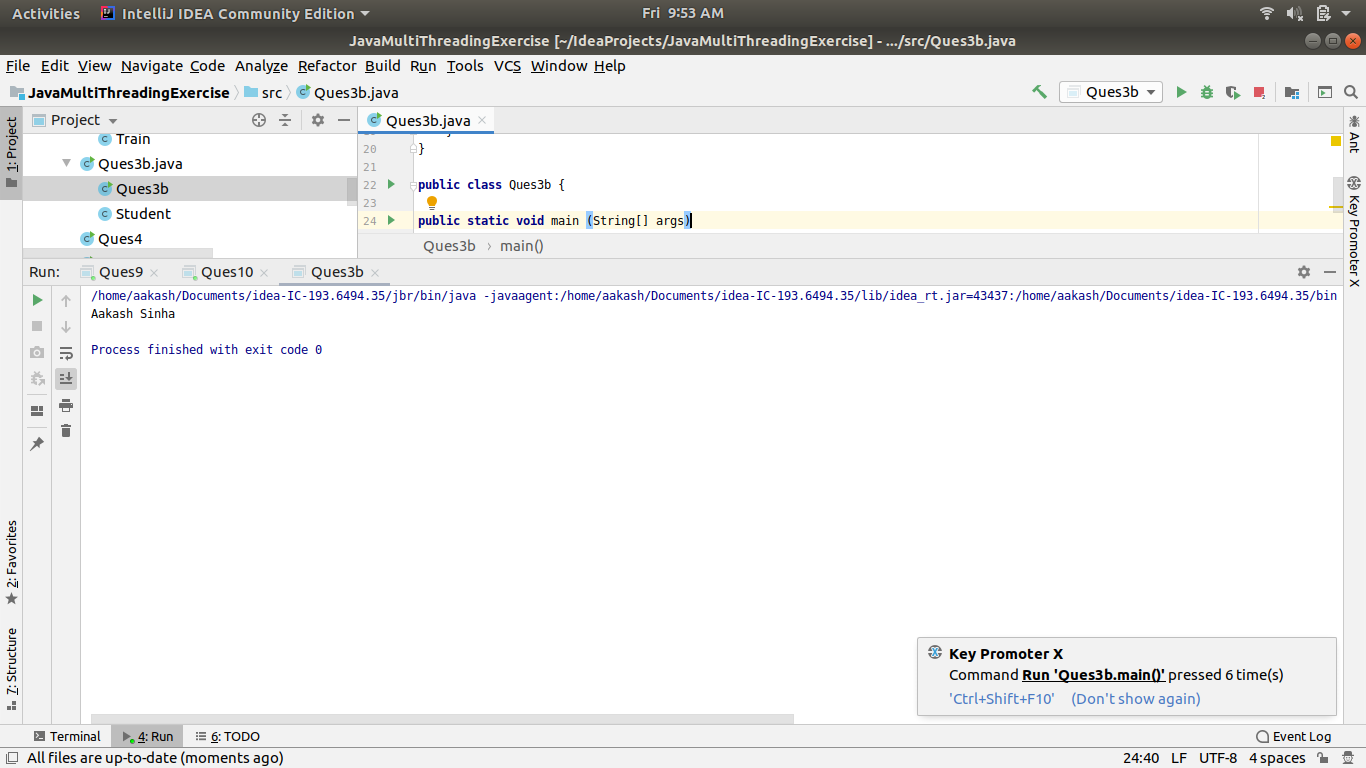
List<String> list = **new** ArrayList<String>();

student.StudentName(**"Aakash Sinha"**, list);

System.***out***.println(student.**name**);

}

}



*//Ques 4: Write a program to create a Thread pool of 2 threads where one Thread will print even numbers*

*// and other will print odd numbers.*

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

**public class** Ques4 {

**public static void** main(String[] args) {

ExecutorService executorService = Executors.*newSingleThreadScheduledExecutor*();

executorService.submit(**new** Runnable() {

@Override

**public void** run() {

System.***out***.println(**"Thread 1 to print Even Number is running"**);

**for**(**int** i=0;i<10;i+=2){

System.***out***.println(i);

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

});

executorService.submit(**new** Runnable() {

@Override

**public void** run() {

System.***out***.println(**"Thread 1 to print Odd Number is running"**);

**for**(**int** i=1;i<10;i+=2){

System.***out***.println(i);

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

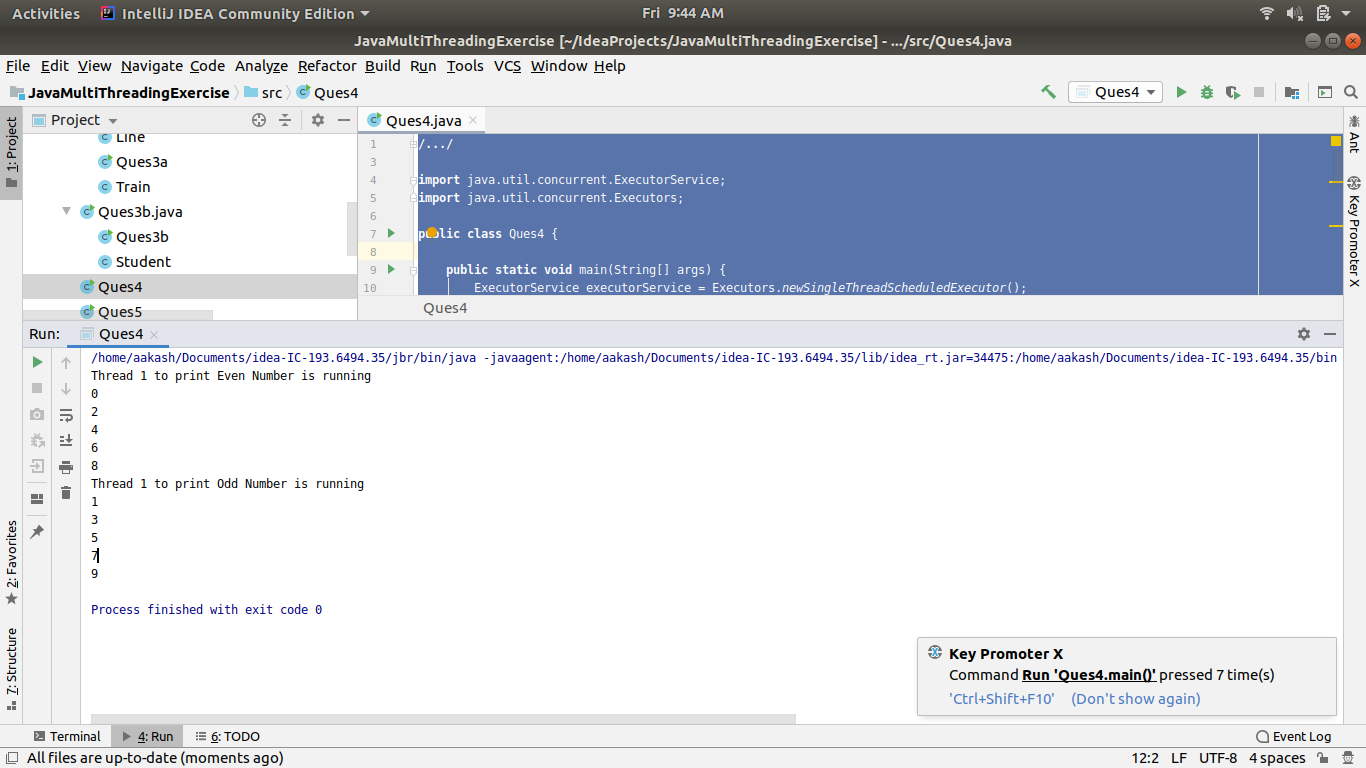
}

});

executorService.shutdown();

}

}



*//Ques 5: Write a program to demonstrate wait and notify methods.*

**import** java.util.Scanner;

**public class** Ques5 {

**public void** produce() **throws** InterruptedException

{

**synchronized** (**this**)

{

System.***out***.println(**"Producer thread running.."**);

wait();

System.***out***.println(**"Resumed.."**);

}

}

**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(**"Return Key pressed..."**);

notify();

Thread.*sleep*(5000);

}

}

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

{

Ques5 processor = **new** Ques5();

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();

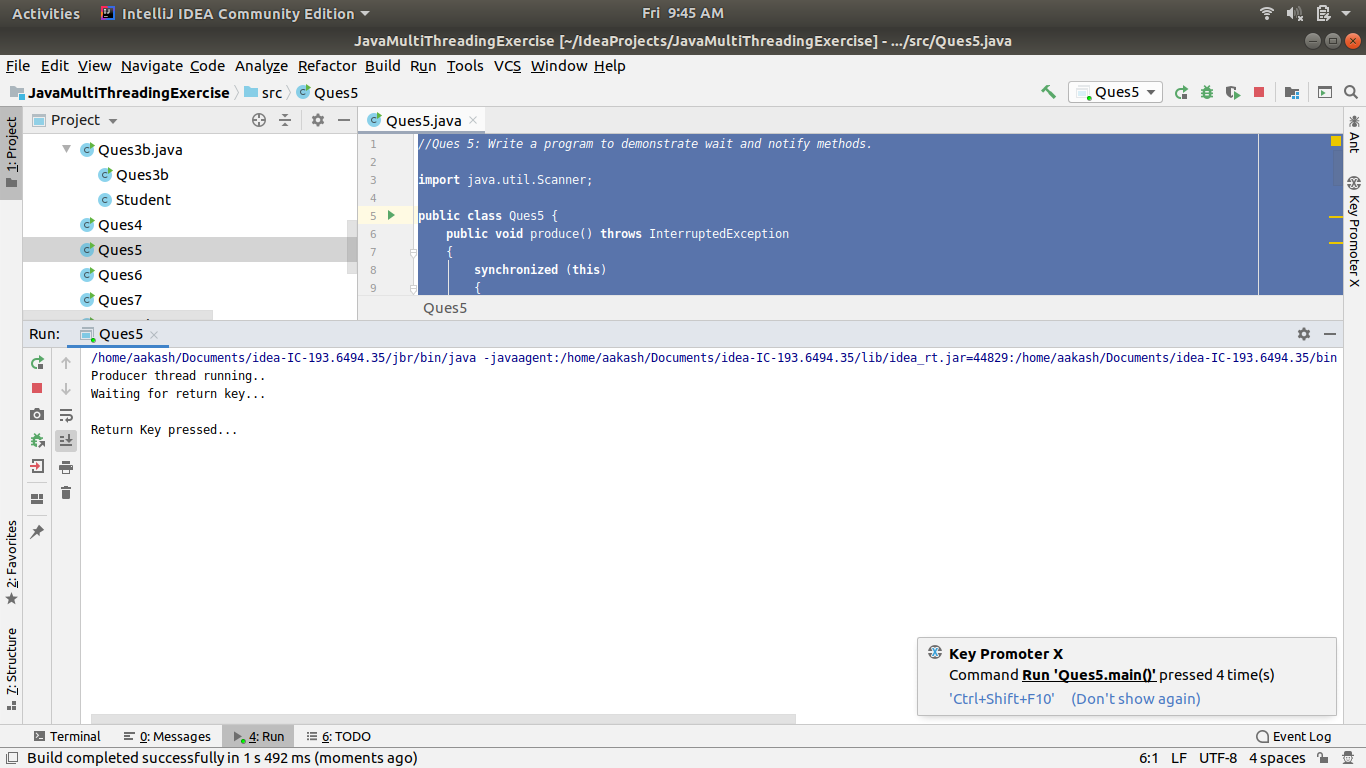
t2.start();

t1.join();

t2.join();

}

}



*//Ques 6: Write a program to demonstrate sleep and join methods.*

**public class** Ques6 **extends** Thread {

**public void** run(){

**for**(**int** i=1;i<=5;i++){

**try**{

Thread.*sleep*(500);

}**catch**(Exception e){System.***out***.println(e);}

System.***out***.println(i);

}

}

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

Ques6 t1=**new** Ques6();

Ques6 t2=**new** Ques6();

Ques6 t3=**new** Ques6();

t1.start();

t2.start();

**try**{

t1.join();

t2.join();

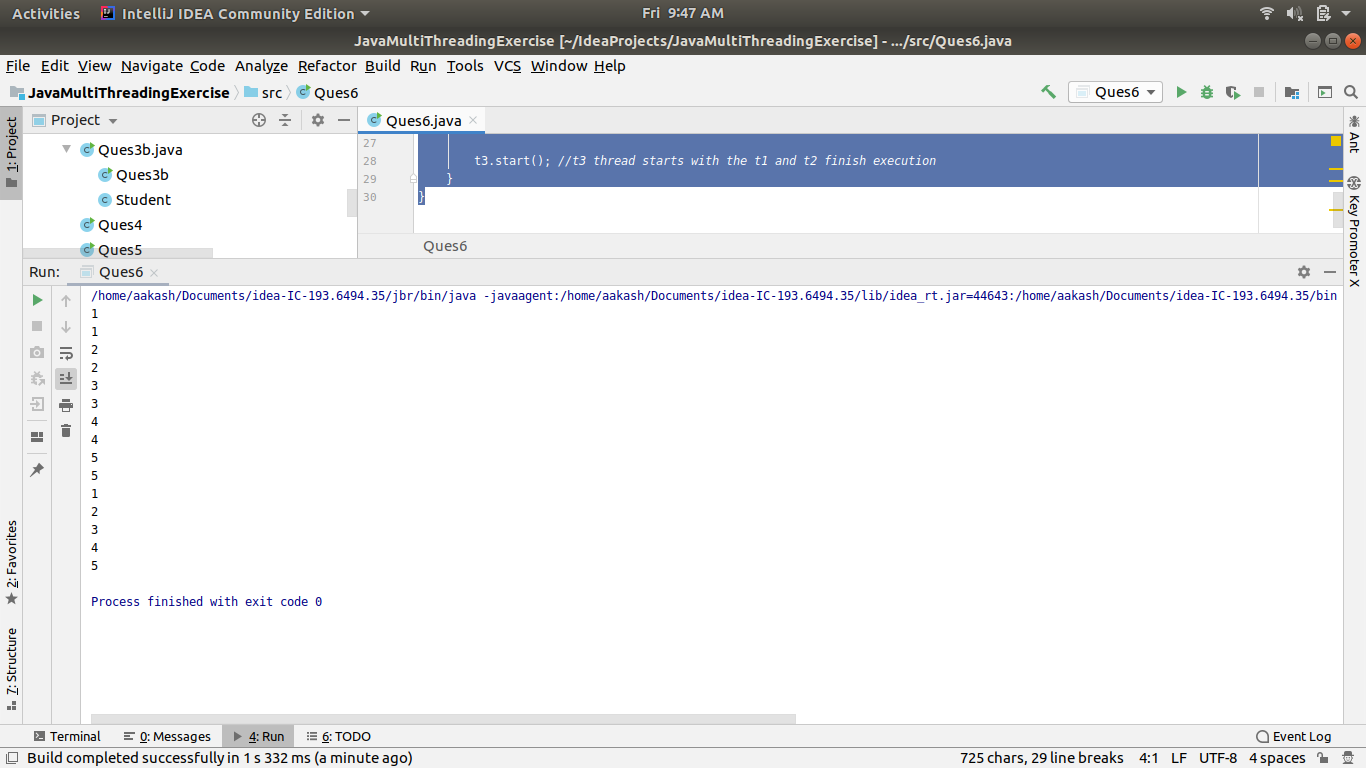
}

**catch**(Exception e){System.***out***.println(e);}

t3.start(); *//t3 thread starts with the t1 and t2 finish execution*

}

}



*//Ques7: Run a task with the help of callable and store it's result in the Future.*

**import** java.util.Random;

**import** java.util.concurrent.\*;

**public class** Ques7 {

**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 | ExecutionException e)

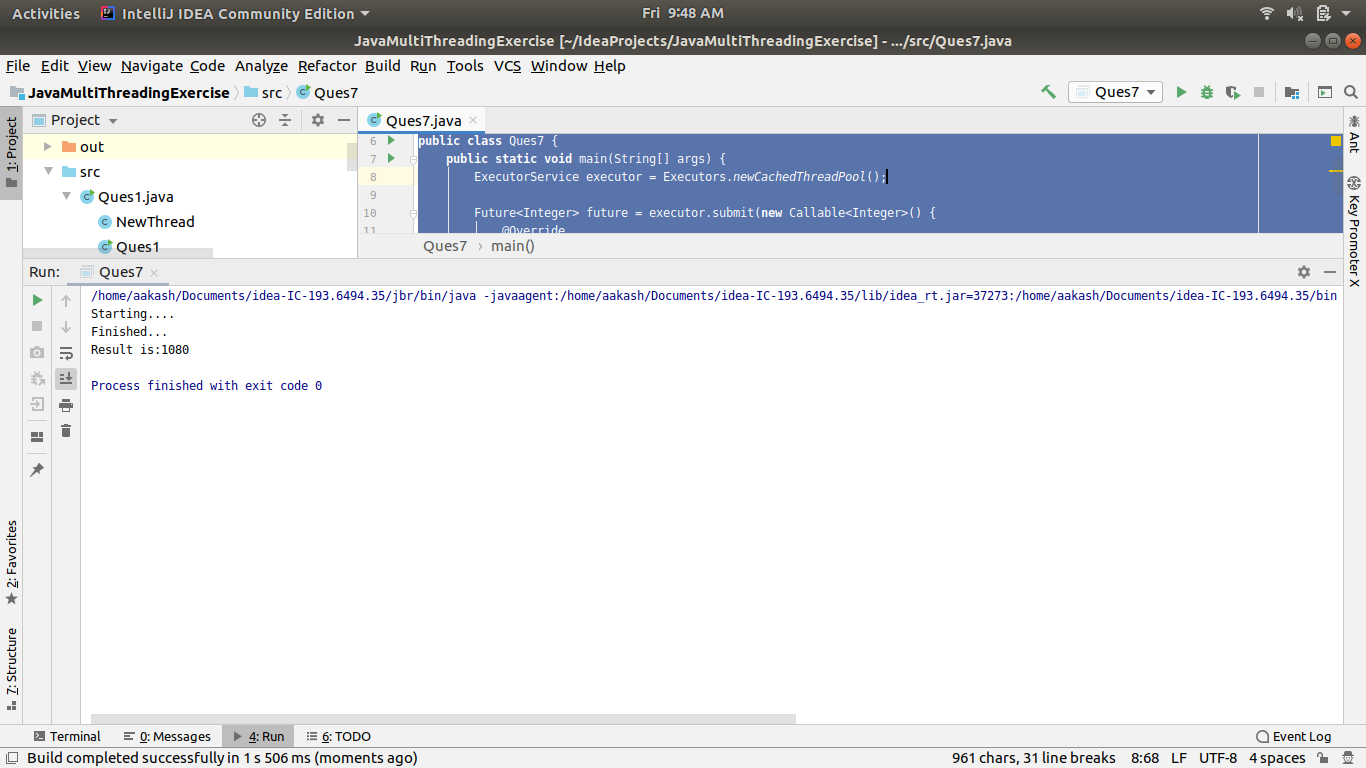
{

e.printStackTrace();

}

}

}



*//Ques 8: Write a program to demonstrate the use of semaphore*

**import** java.util.concurrent.\*;

**class** connection{

**private** Semaphore **sem** = **new** Semaphore(2);

**private int connections**=0;

**public void** connect() **throws** InterruptedException {

**sem**.acquire();

**synchronized** (**this**)

{

**connections**++;

System.***out***.println(**"Current connections:"**+ **connections**);

}

Thread.*sleep*(2000);

**synchronized** (**this**)

{

**connections**--;

}

**sem**.release();

}

}

**public class** Ques8 {

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

ExecutorService executor = Executors.*newCachedThreadPool*();

connection obj = **new** connection();

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

{

executor.submit(**new** Runnable() {

@Override

**public void** run() {

**try** {

obj.connect();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

});

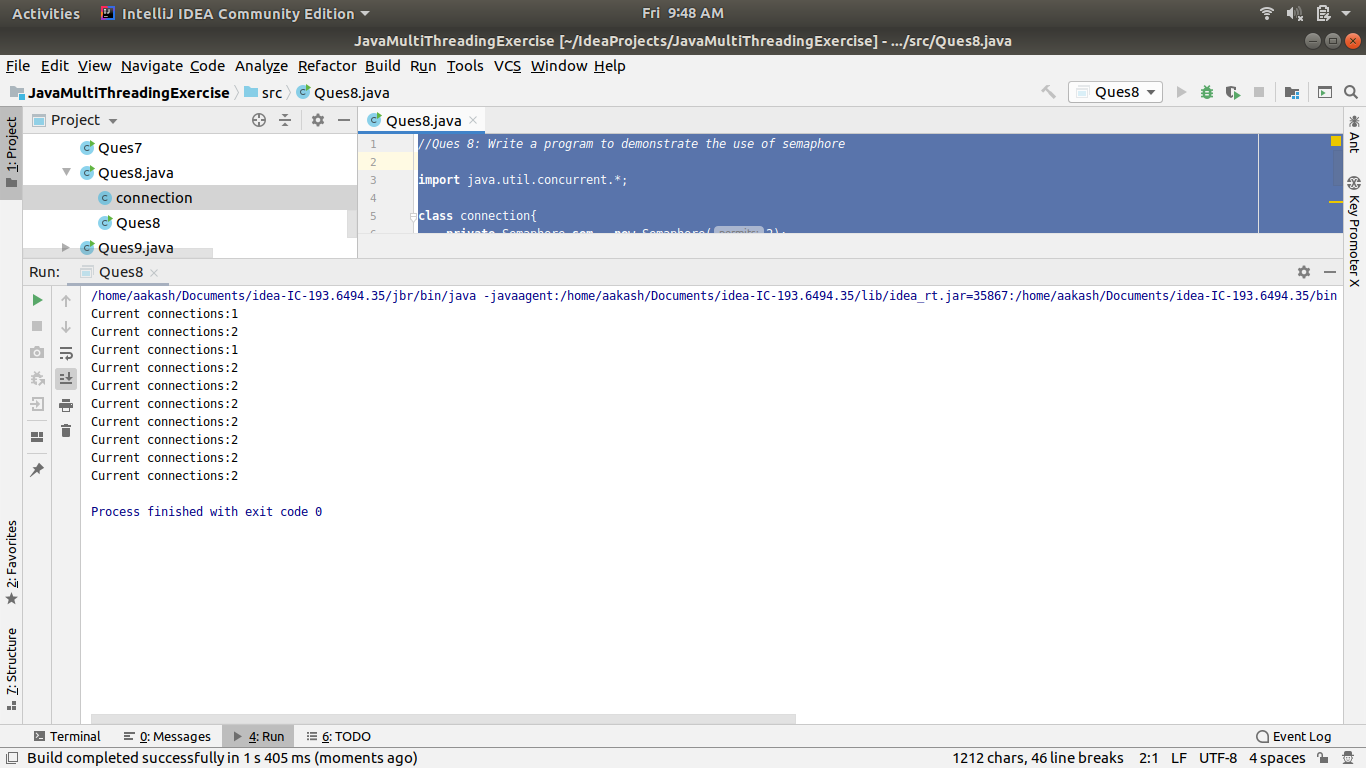
}

executor.shutdown();

executor.awaitTermination(1, TimeUnit.***DAYS***);

}

}



*//Ques9: Write a program to demonstrate the use of CountDownLatch*

**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*(3000);

} **catch** (InterruptedException ex) {

ex.printStackTrace();

}

**latch**.countDown();

}

}

**public class** Ques9 {

**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** Processor(latch));

}

**try**{

latch.await();

}

**catch** (InterruptedException ex)

{

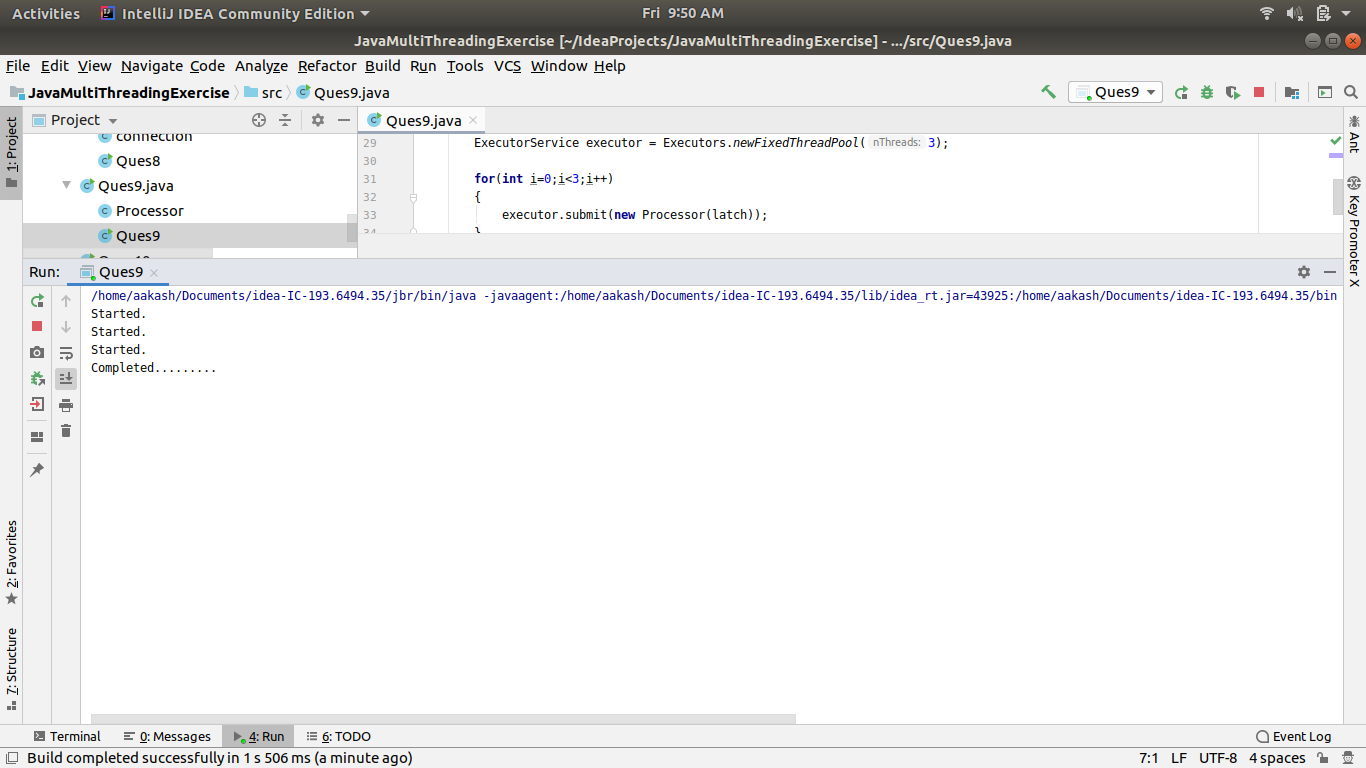
ex.printStackTrace();

}

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

}

}



*//Ques 10: Write a program which creates deadlock between 2 threads*

**public class** Ques10 {

**public static void** main(String[] args) {

**final** String resource1 = **"Aayushi"**;

**final** String resource2 = **"Pragya"**;

*// t1 tries to lock resource1 then resource2*

Thread t1 = **new** Thread() {

**public void** run() {

**synchronized** (resource1) {

System.***out***.println(**"Thread 1: locked resource 1"**);

**try** { Thread.*sleep*(100);} **catch** (Exception e) {}

**synchronized** (resource2) {

System.***out***.println(**"Thread 1: locked resource 2"**);

}

}

}

};

*// t2 tries to lock resource2 then resource1*

Thread t2 = **new** Thread() {

**public void** run() {

**synchronized** (resource2) {

System.***out***.println(**"Thread 2: locked resource 2"**);

**try** { Thread.*sleep*(100);} **catch** (Exception e) {}

**synchronized** (resource1) {

System.***out***.println(**"Thread 2: locked resource 1"**);

}

}

}

};

t1.start();

t2.start();

}

}

