**Introduction-To-Multithreading-Assingment**

Ques 1:Name of method in Thread class to pause execution?

Ans:**JAVA-CODE**

Thread. sleep(long millis) method causes the currently executing thread to sleep for the specified number of milliseconds.

public class Ques1 extends Thread{

public void run(){

for(int i=0; i<5; i++){

try{

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

Thread.sleep(200);

}

catch (Exception e){

System.out.println("Exception in thread "+e.getMessage());

}

}

}

public static void main(String[] args) {

Ques1 q1 = new Ques1();

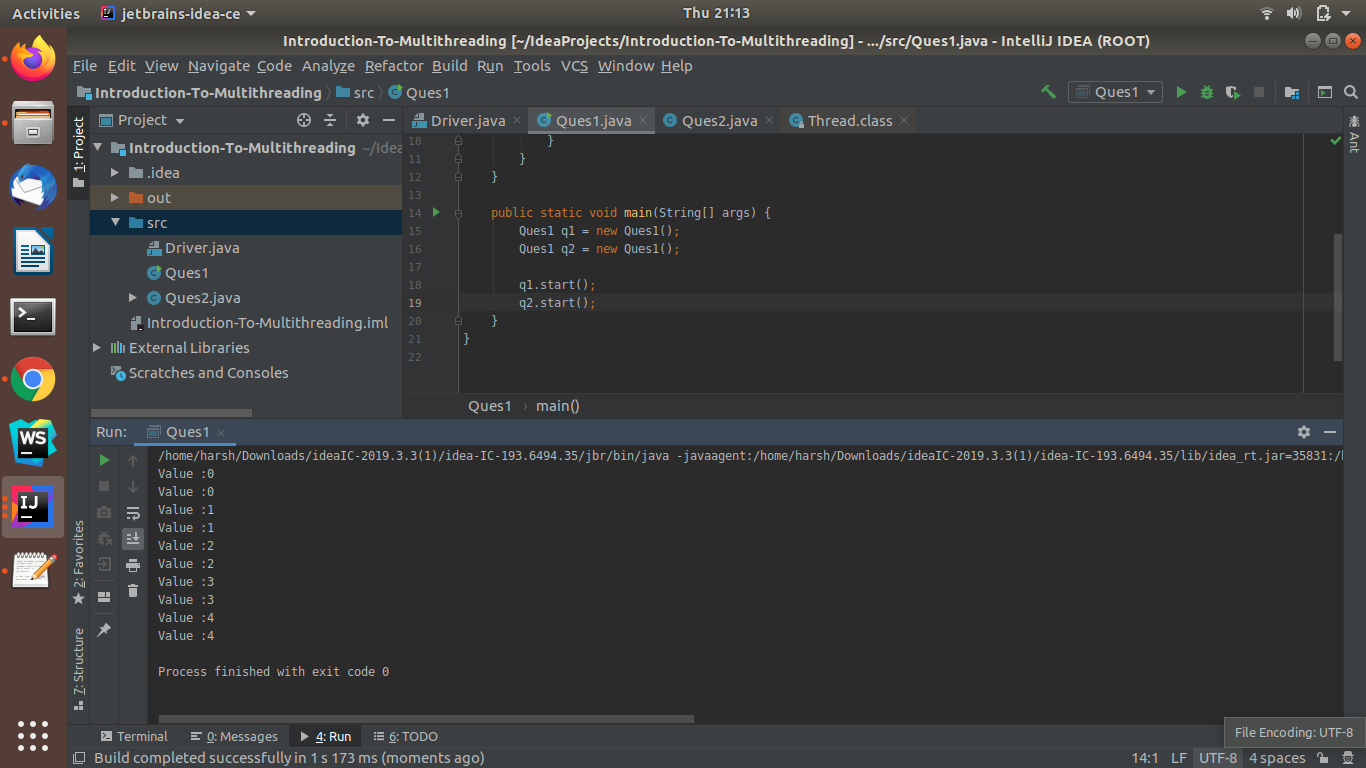
Ques1 q2 = new Ques1();

q1.start();

q2.start();

}

}



Ques 2:Role of "volatile" keyword.

Ans:

**Volatile** Keyword in **Java**. **Volatile** keyword is used to modify the value of a variable by different threads. It is also used to make classes thread safe. It means that multiple threads can use a method and instance of the classes at the same time without any problem.

**JAVA-CODE**

class Ques2 extends Thread{

private volatile boolean flag=true;

@Override

public void run() {

while(flag){

System.out.println("Thread is running");

try{

Thread.sleep(200);

}

catch (Exception e){

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

}

}

}

public void stoprun(){

flag=false;

}

public boolean show(){

return flag;

}

}

class Driver1 {

public static void main(String[] args) {

Ques2 q2 = new Ques2();

q2.start();

Scanner sc = new Scanner(System.in);

sc.nextLine();

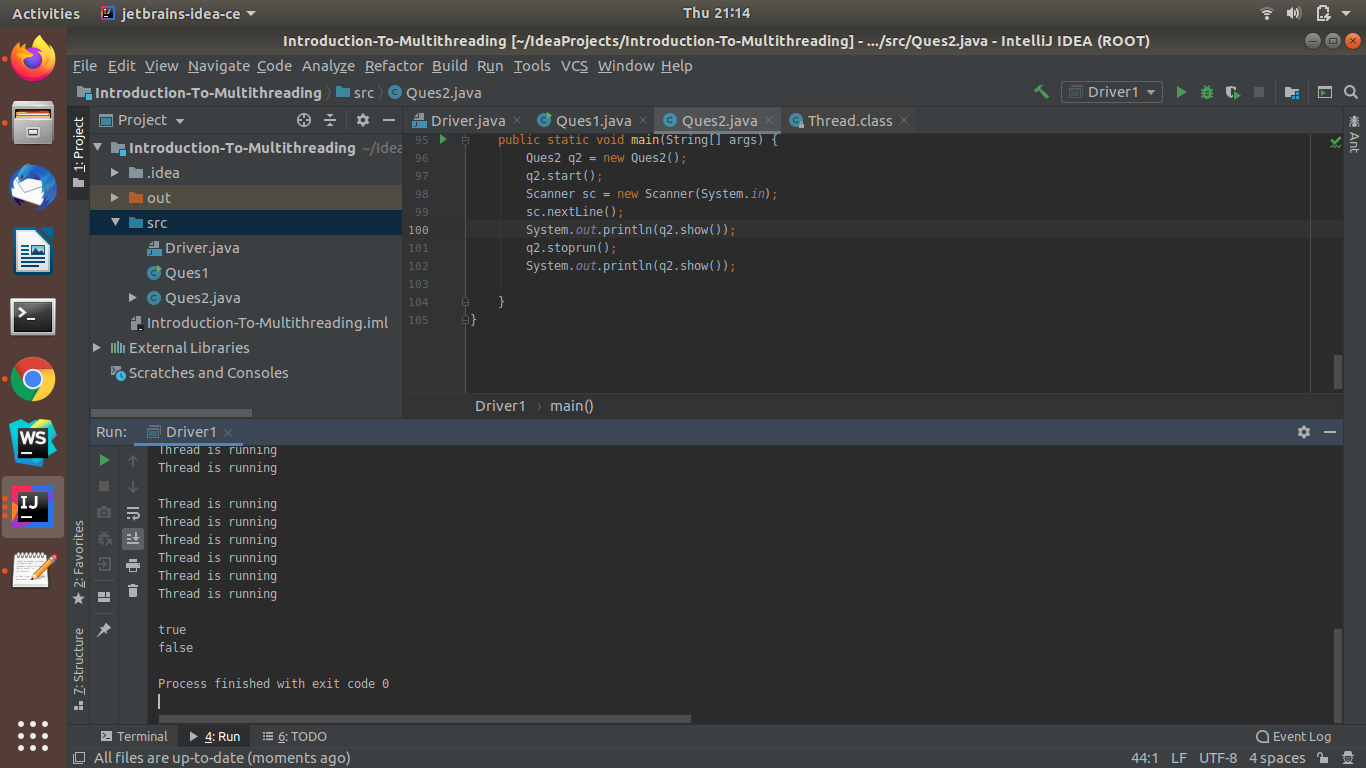
System.out.println(q2.show());

q2.stoprun();

System.out.println(q2.show());

}

}



Ques 3:Write a program to create a thread using Thread class and Runnable interface each.

Ans: **JAVA-CODE**

//using thread class

public class Ques3 extends Thread{

public void run(){

for(int i=0; i<5; i++){

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

}

}

public static void main(String[] args) {

Ques1 q1 = new Ques1();

q1.start();

}

}

//using Runnable interface

public class Ques3 implements Runnable{

public void run(){

for(int i=0; i<5; i++){

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

}

}

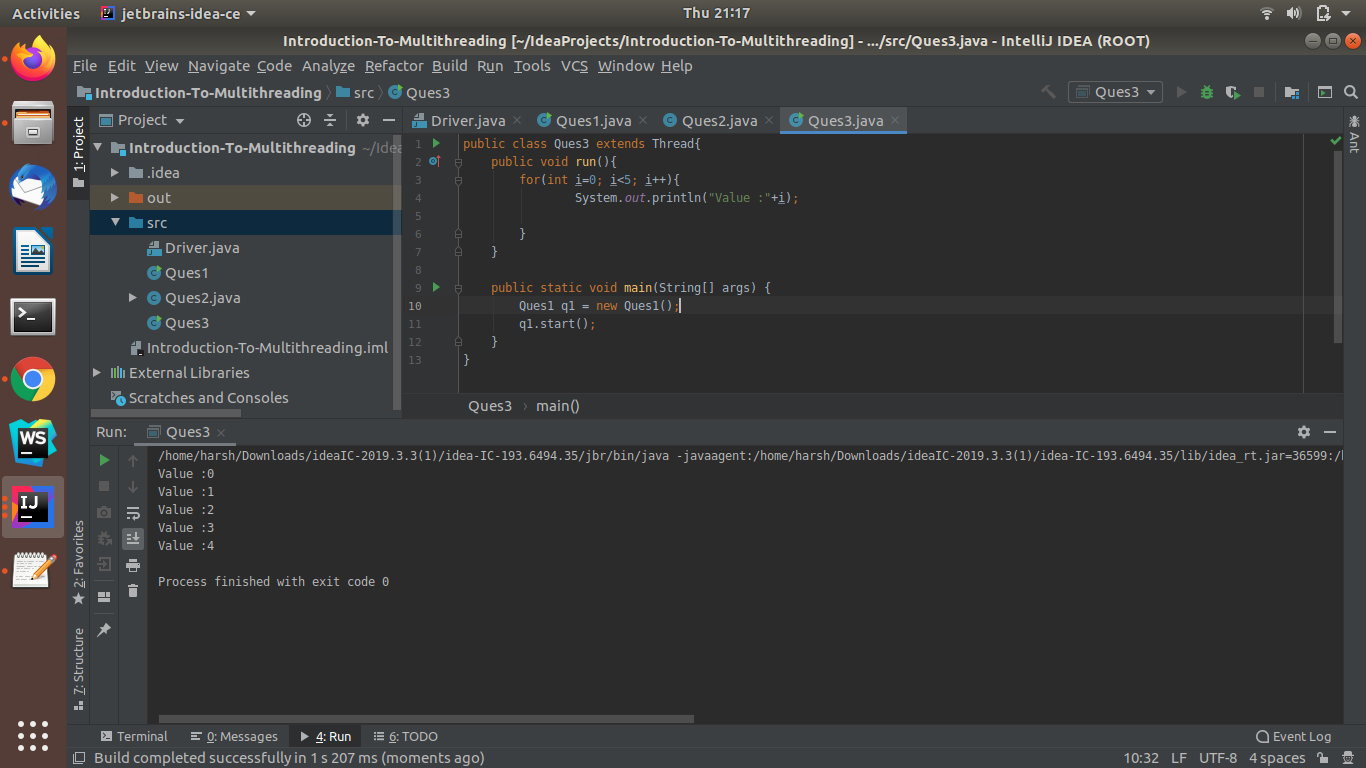
public static void main(String[] args) {

Thread t1 = new Thread(new Ques3());

t1.start();

}

}



Ques 4:Write a program using synchronization block

Ans: **JAVA-CODE**

class NamePrint{

String name;

void print(String name1){

synchronized (this){

System.out.println("Given name is printing....");

name=name1;

try{

Thread.sleep(200);

}

catch (Exception e){

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

}

System.out.println(name);

}

}

}

class Client1 extends Thread{

NamePrint m;

Client1(NamePrint n){

m=n;

}

@Override

public void run() {

m.print("CLIENT 1");

}

}

class Client2 extends Thread{

NamePrint m;

Client2(NamePrint n){

m=n;

}

@Override

public void run() {

m.print("CLIENT 2");

}

}

public class Ques4 {

public static void main(String[] args) {

NamePrint obj = new NamePrint();

Client1 c1 = new Client1(obj);

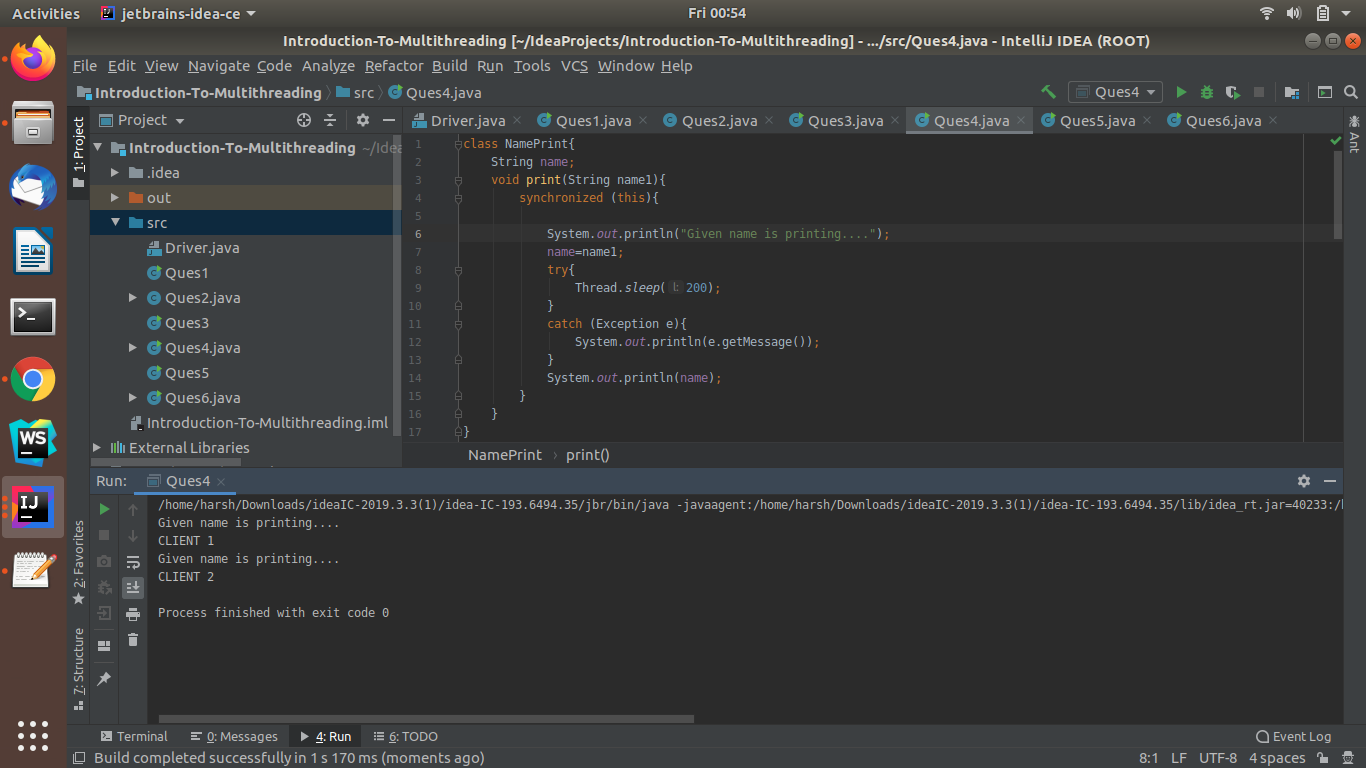
Client2 c2 = new Client2(obj);

c1.start();

c2.start();

}

}



Ques 5:Write a program using synchronization method

Ans: **JAVA-CODE**

class Ques5{

private int counter=0;

public synchronized void increment(){

counter++;

}

public static void main(String[] args) {

Ques5 q5 = new Ques5();

q5.synchronisedmethodex();

}

public void synchronisedmethodex(){

Thread t1 = new Thread(new Runnable() {

@Override

public void run() {

for(int i=0; i<100; i++){

increment();

}

}

});

Thread t2 = new Thread(new Runnable() {

@Override

public void run() {

for(int i=0; i<100; i++){

increment();

}

}

});

t1.start();

t2.start();

try{

t1.join();

t2.join();

}

catch (Exception e){

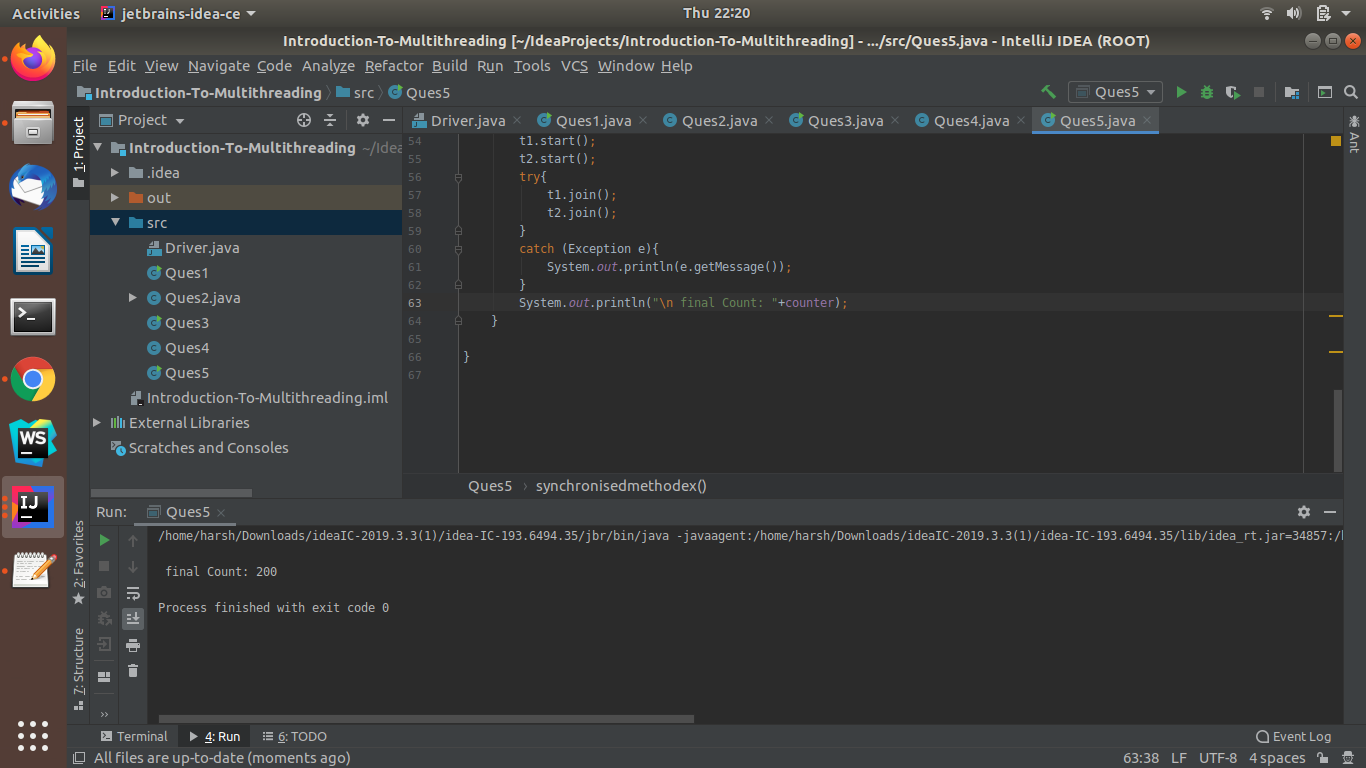
System.out.println(e.getMessage());

}

System.out.println("\n final Count: "+counter);

}

}



Ques 6: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: **JAVA-CODE**

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

class even implements Runnable{

@Override

public void run() {

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

System.out.println("Even number: "+i);

}

}

}

class odd implements Runnable{

@Override

public void run() {

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

System.out.println("Odd number: "+i);

}

}

}

public class Ques6 {

public static void main(String[] args) {

Runnable r1 = new even();

Runnable r2 = new odd();

ExecutorService pool = Executors.newFixedThreadPool(2);

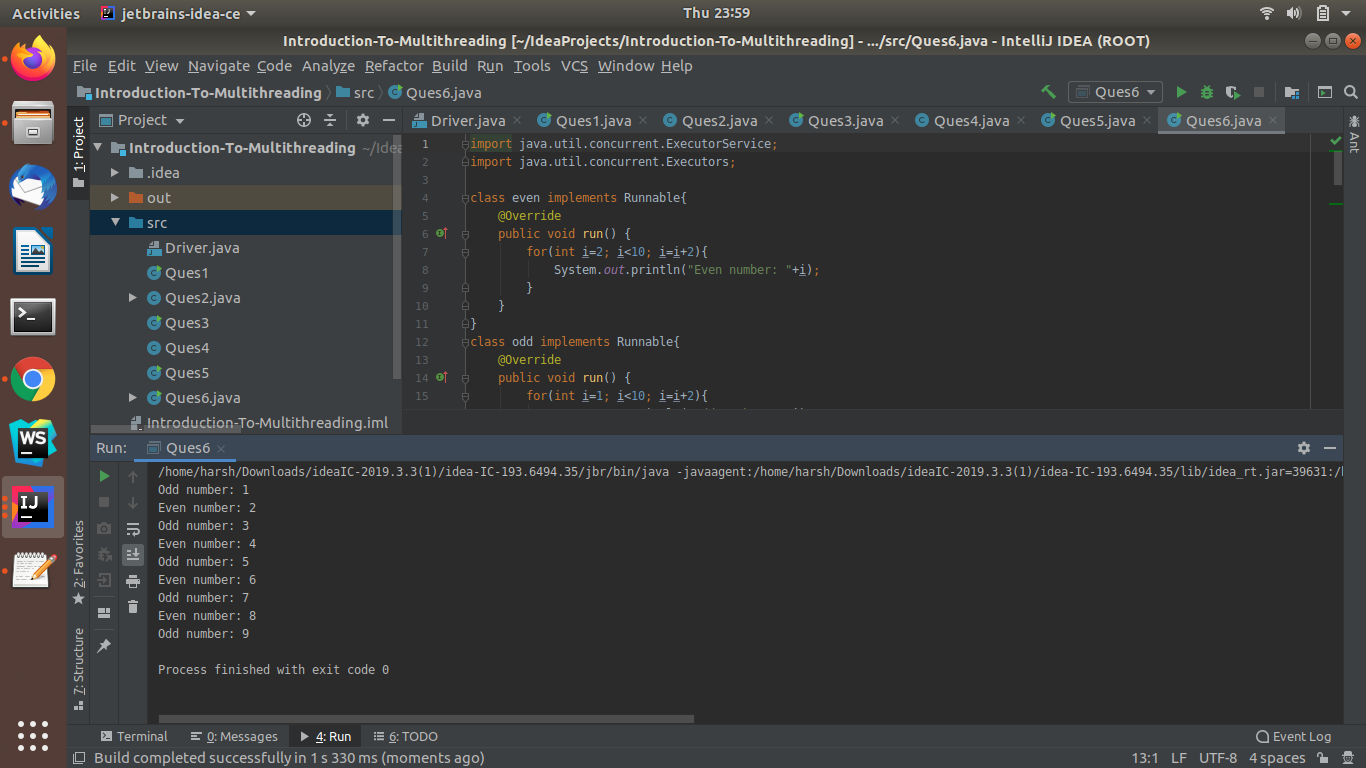
pool.execute(r1);

pool.execute(r2);

pool.shutdown();

}

}



Ques 7:Write a program to demonstrate wait and notify methods.

Ans: **JAVA-CODE**

class Shared{

int flag=0,data=0;

synchronized void submit(){

flag=1;

try{

Thread.sleep(200);

}

catch (Exception e){

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

}

data =1000;

System.out.println("Value Submitted");

notify();

}

synchronized int withdraw(){

if(flag ==0){

try{

System.out.println("Wait Block insufficient balance");

wait();

}

catch (Exception e){

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

}

}

return data;

}

}

class Customer extends Thread{

Shared s;

Customer(Shared s1){

s=s1;

}

@Override

public void run() {

System.out.println(s.withdraw());

System.out.println("Customer withdraw amount");

}

}

class Cashier extends Thread{

Shared s;

Cashier(Shared s1){

s=s1;

}

@Override

public void run() {

System.out.println("Cashier takes value ready to submit...");

s.submit();

}

}

public class Ques7 {

public static void main(String[] args) {

Shared obj = new Shared();

Cashier cs = new Cashier(obj);

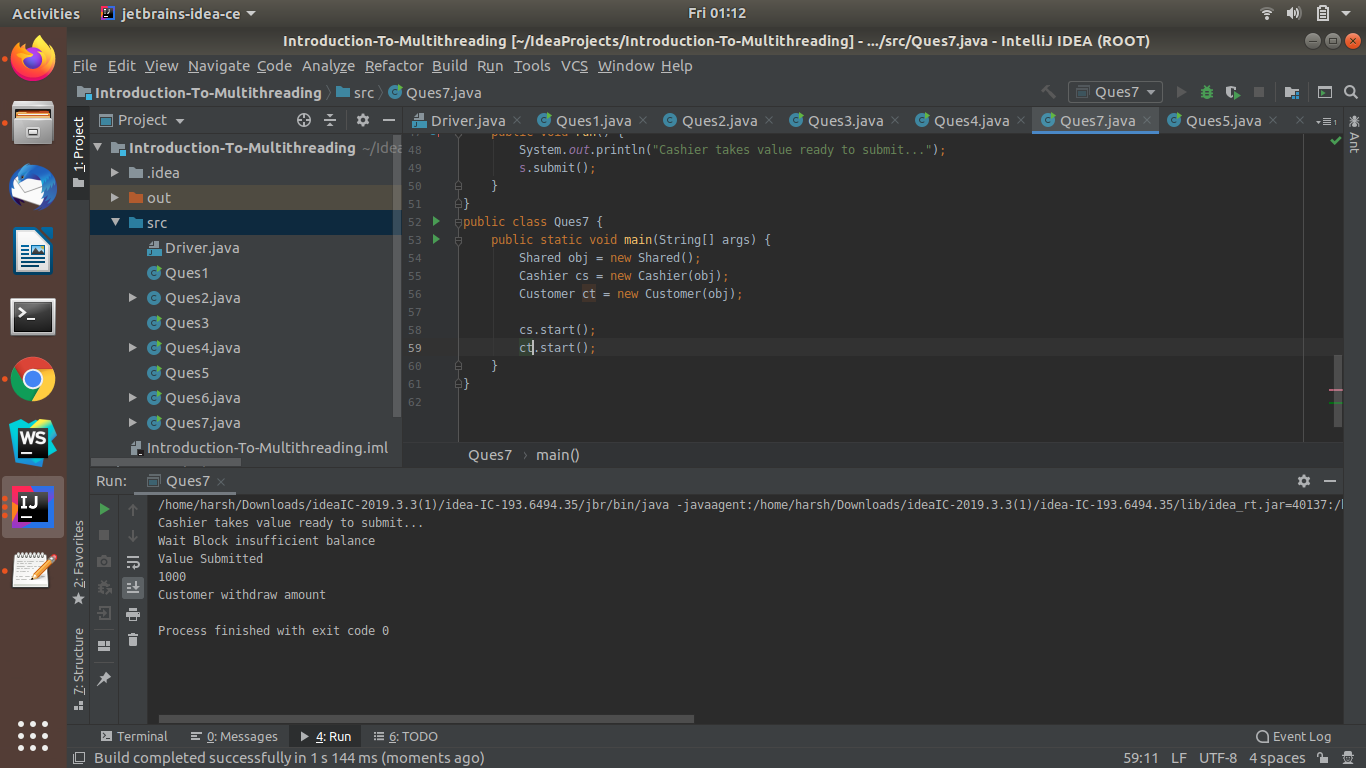
Customer ct = new Customer(obj);

cs.start();

ct.start();

}

}



Ques 8: Write a program to demonstrate sleep and join methods.

Ans: **JAVA-CODE**

public class SleepAndJoin {

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

Thread t1 = new Thread(new Runnable() {

@Override

public void run() {

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

try {

Thread.sleep(500);

System.out.println("Thread one is running"+i);

} catch (Exception e) {

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

}

}

}

});

Thread t2 = new Thread(new Runnable() {

@Override

public void run() {

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

try{

Thread.sleep(500);

System.out.println("Thread Two is running"+i);

}

catch (Exception e){

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

}

}

}

});

t1.start();

t2.start();

try{

t1.join();

t2.join();

}

catch (Exception e){

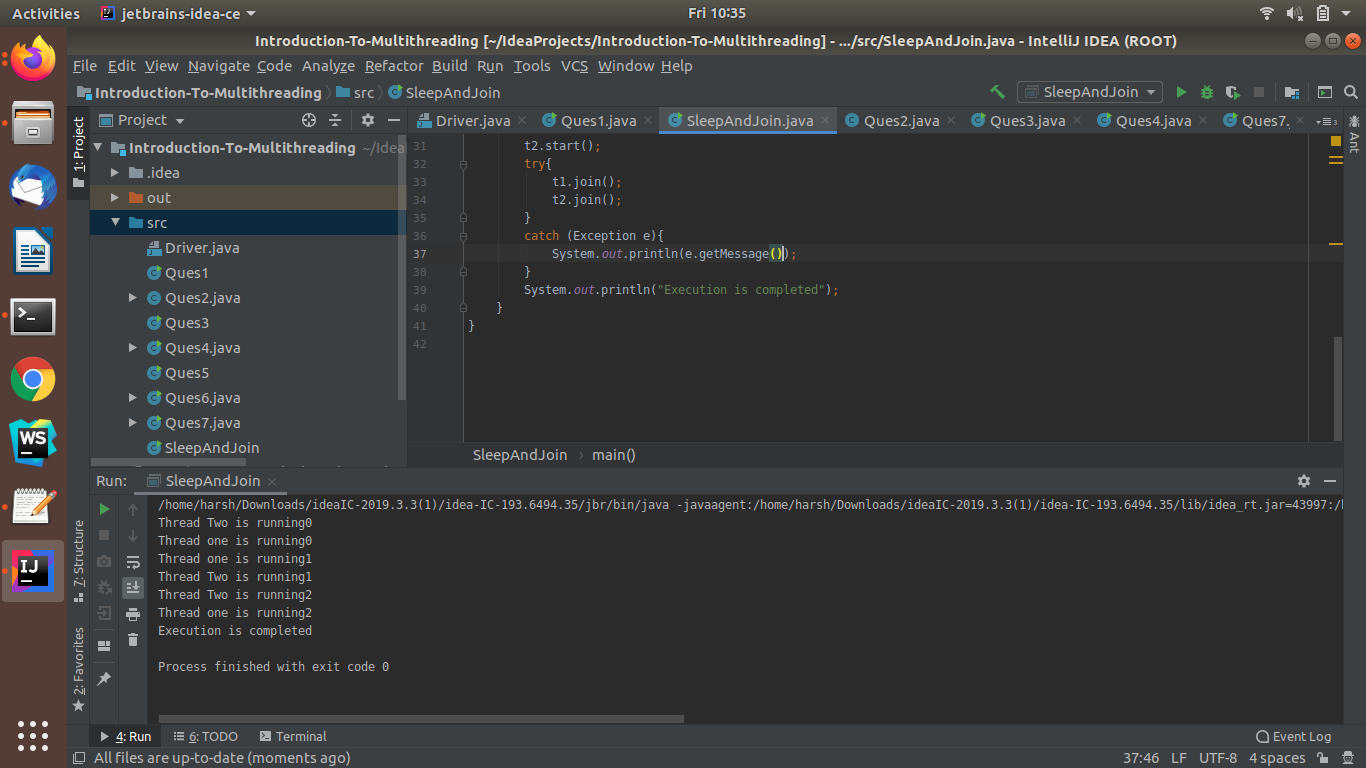
System.out.println(e.getMessage());

}

System.out.println("Execution is completed");

}

}



Ques 9: Run a task with the help of callable and store it's result in the Future.

Ans: **JAVA-CODE**

import java.io.IOException;

import java.util.Scanner;

import java.util.concurrent.\*;

public class CollabeAndFuture {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

ExecutorService executor = Executors.newCachedThreadPool();

Future<Integer> future = executor.submit(new Callable<Integer>() {

@Override

public Integer call() throws Exception {

// Random random = new Random();

System.out.println("Enter your age");

int age = sc.nextInt();

if(age >100){

throw new IOException("Invalid age");

}

System.out.println("sleeping for too long");

try{

Thread.sleep(2000);

}

catch (InterruptedException e){

//System.out.println(e.getMessage());

e.printStackTrace();

}

System.out.println("Finished ....");

return age;

}

});

executor.shutdown();

try{

System.out.println("Entered age is: "+future.get());

}

catch (InterruptedException e){

e.printStackTrace();

}

catch(ExecutionException e){

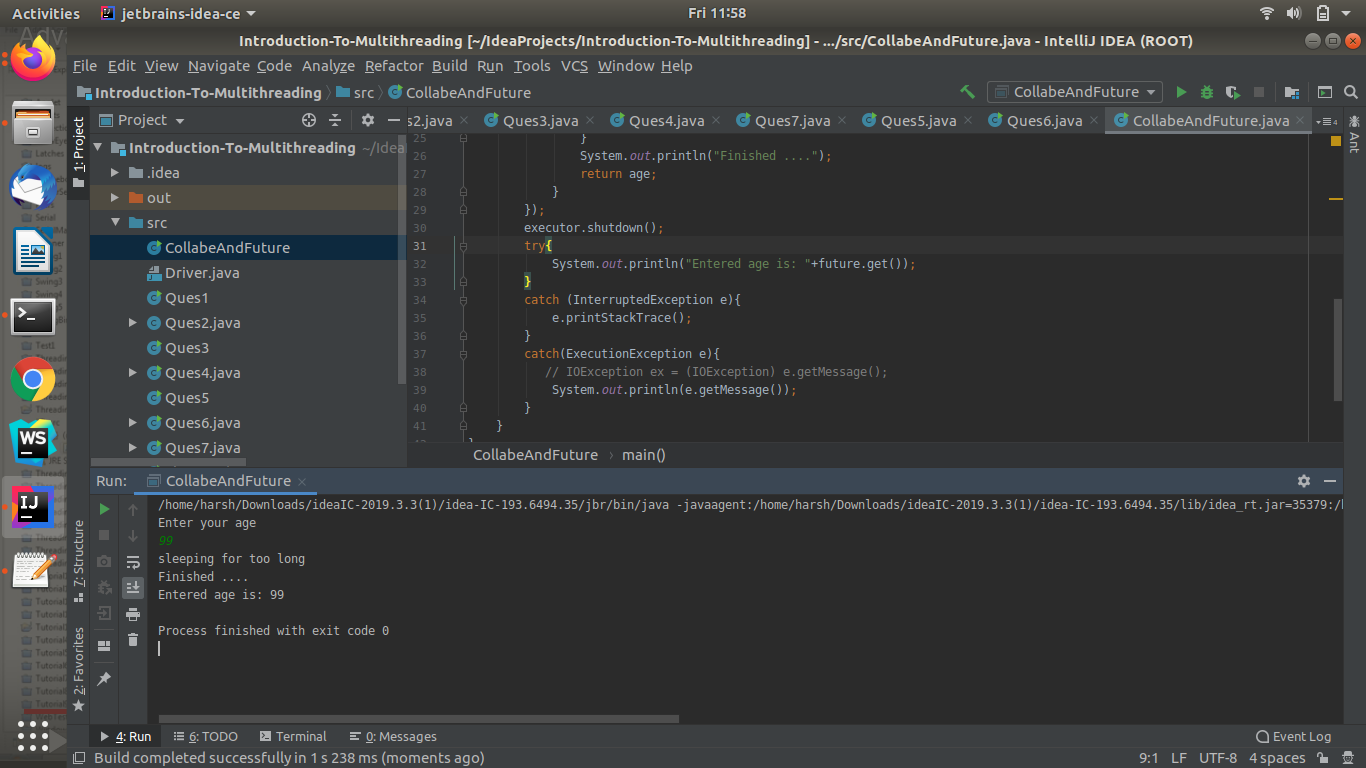
// IOException ex = (IOException) e.getMessage();

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

}

}

}



Ques 10:Write a program to demonstrate the use of semaphore

Ans: **JAVA-CODE -OF-MAIN-CLASS**

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

public class SemaphoreEx { //this class is used to run thread

public static void main(String[] args) {

ExecutorService execute = Executors.newCachedThreadPool();

for(int i=0; i<200; i++){

execute.submit(new Runnable() {

@Override

public void run() {

SingletonEx.getInstance().connect();

//new Semaphore1().connect();

}

});

}

execute.shutdown();

try {

execute.awaitTermination(1, TimeUnit.DAYS);

}

catch(Exception e){

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

}

}

}

**JAVA-CODE-TO-USE OF SEMAPHORE**

import java.util.concurrent.Semaphore;

public class SingletonEx { // this class will return singleton class object

private static SingletonEx obj = new SingletonEx();

private Semaphore sem = new Semaphore(10);

private int value=0;

private SingletonEx(){

}

public static SingletonEx getInstance(){

return obj;

}

public void connect(){

try{

sem.acquire();

}

catch(InterruptedException e){

e.printStackTrace();

}

try{

doconnect();

}

finally {

sem.release();

}

}

public void doconnect(){

synchronized (this){

value++;

System.out.println("Current value: "+value);

}

try{

Thread.sleep(2000);

}

catch (InterruptedException e){

e.printStackTrace();

}

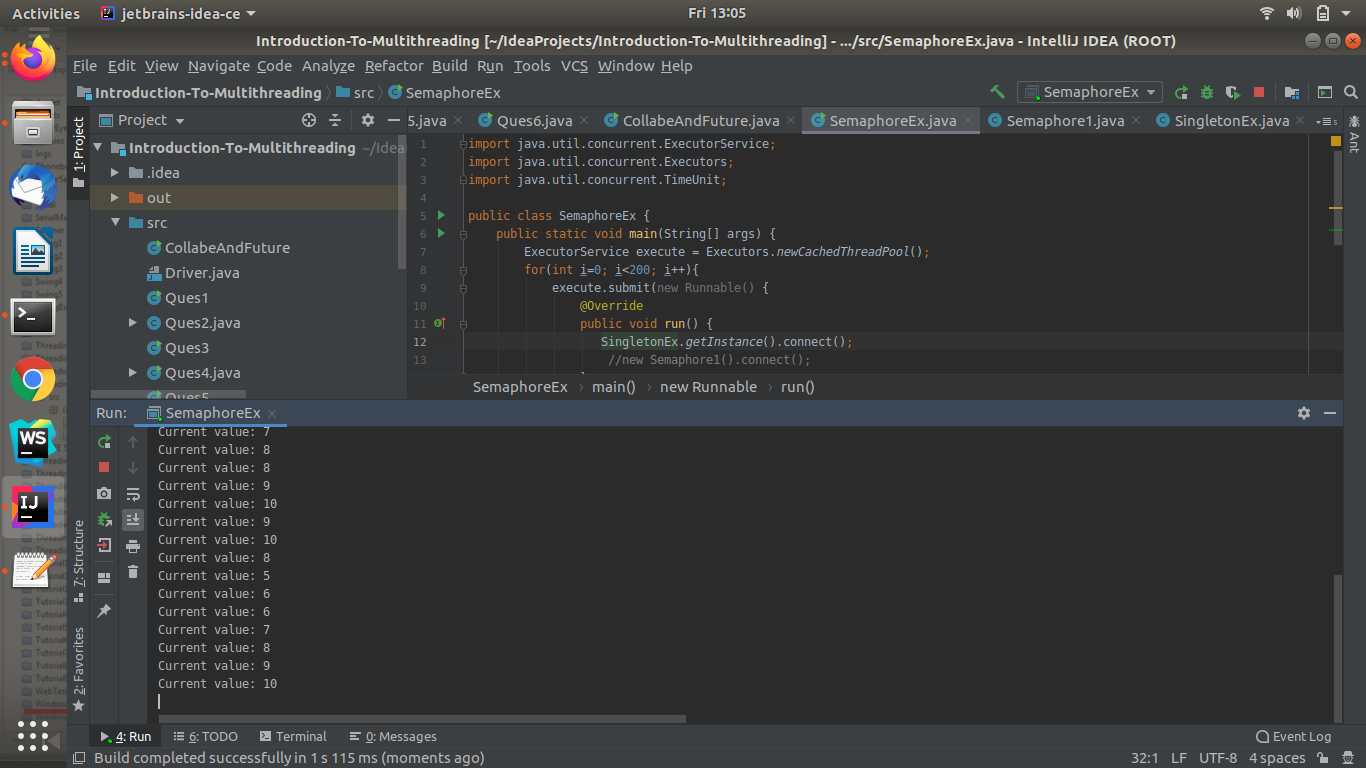
synchronized (this){

value--;

}

}

}



Ques 10: Write a program to demonstrate the use of CountDownLatch

Ans: **JAVA-CODE**

import java.util.concurrent.CountDownLatch;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

class UseOfCountDownLatch implements Runnable {

private CountDownLatch latch;

public UseOfCountDownLatch( CountDownLatch latch){

this.latch=latch;

}

@Override

public void run() {

System.out.println("Latch is Started...");

try{

Thread.sleep(1000);

}

catch (InterruptedException e){

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

}

}

}

class DriverLatch{

public static void main(String[] args) {

CountDownLatch latch = new CountDownLatch(3);

ExecutorService execute = Executors.newScheduledThreadPool(3);

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

execute.submit(new UseOfCountDownLatch(latch));

}

try{

Thread.sleep(100);

}

catch (InterruptedException e){

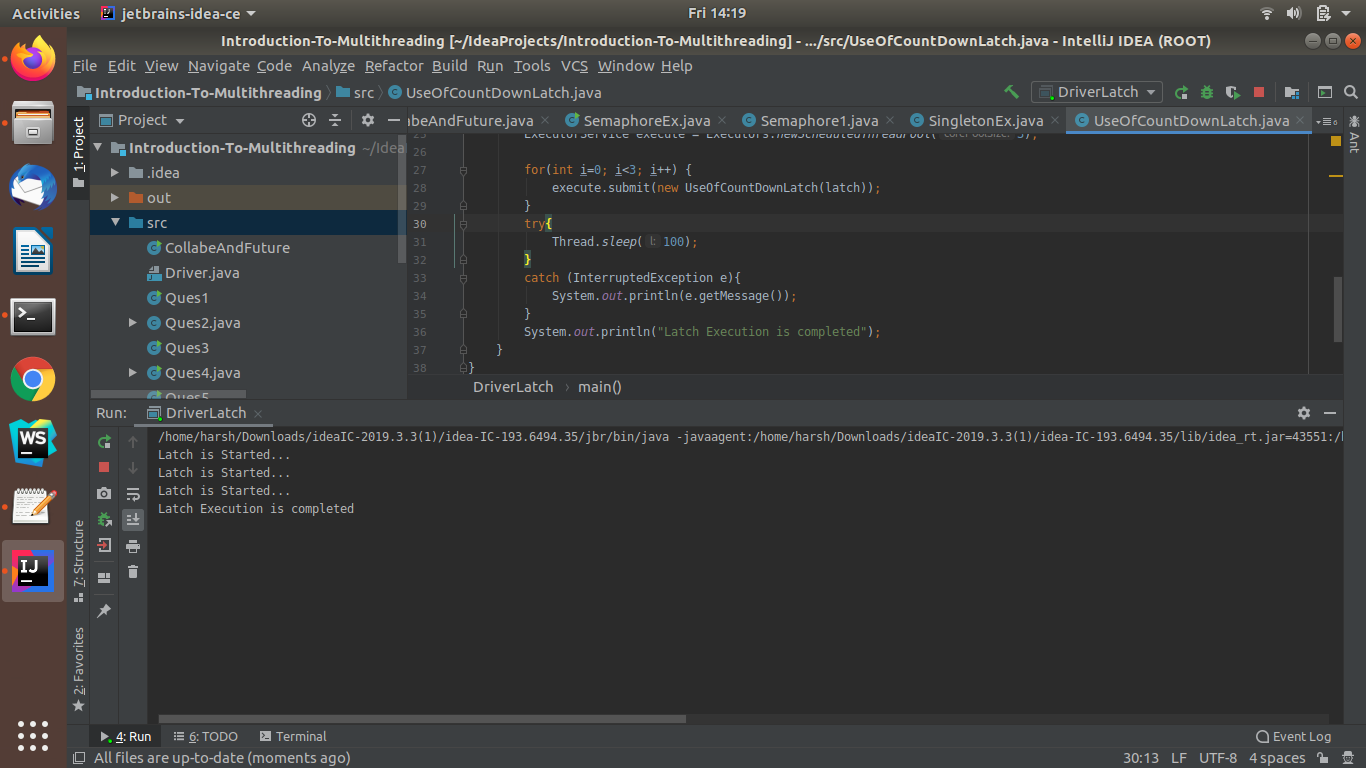
System.out.println(e.getMessage());

}

System.out.println("Latch Execution is completed");

}

}



Ques 11:Write a program which creates deadlock between 2 threads

Ans: **JAVA-CODE**

public class DeadlockEX {

public static void main(String[] args) {

final Integer value1 =10;

final Integer value2 =20;

Thread t1 = new Thread(new Runnable() {

@Override

public void run() {

synchronized (value1){

System.out.println("Thread1 have value1 "+value1);

try{

Thread.sleep(500);

}

catch (InterruptedException e){

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

}

synchronized (value2){

System.out.println("Thread1 requesting for value2"+value2);

}

}

}

});

Thread t2 = new Thread(new Runnable() {

@Override

public void run() {

synchronized (value2){

System.out.println("Thread2 have value2 "+value2);

try{

Thread.sleep(500);

}

catch (InterruptedException e){

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

}

synchronized (value1){

System.out.println("Thread2 requesting for value2 "+value2);

}

}

}

});

t1.start();

t2.start();

}

}

