**CORE JAVA ASSIGNMENT 2**

**1. Create Java classes having suitable attributes for Library management system.Use OOPs concepts in your design.Also try to use interfaces and abstract classes.**

import java.io.\*;

interface library {

void addBooks(int id,String book\_name, String genre);

void bookIssue(int book\_id,String book\_name,String student\_name);

void showAllBooks();

}

class Book implements library {

int id;

String book\_name;

String student\_name;

String genre;

public void addBooks(int bid,String bname,String gen)

{

id=bid;

bname=book\_name;

genre=gen;

}

public void bookIssue(int bid,String bname,String sname)

{

id=bid;

book\_name=bname;

student\_name=sname;

}

@Override

public void showAllBooks() {

System.out.println("Book is : "+book\_name);

System.out.println("Issued by : "+student\_name);

System.out.println("Genre : "+genre);

}

}

public class Q1\_Library {

public static void main(String[] args) {

Book b1=new Book();

b1.addBooks(1,"Alchemist","Quest");

b1.bookIssue(1,"Alchemist","Gurleen");

b1.showAllBooks();

Book b2 = new Book();

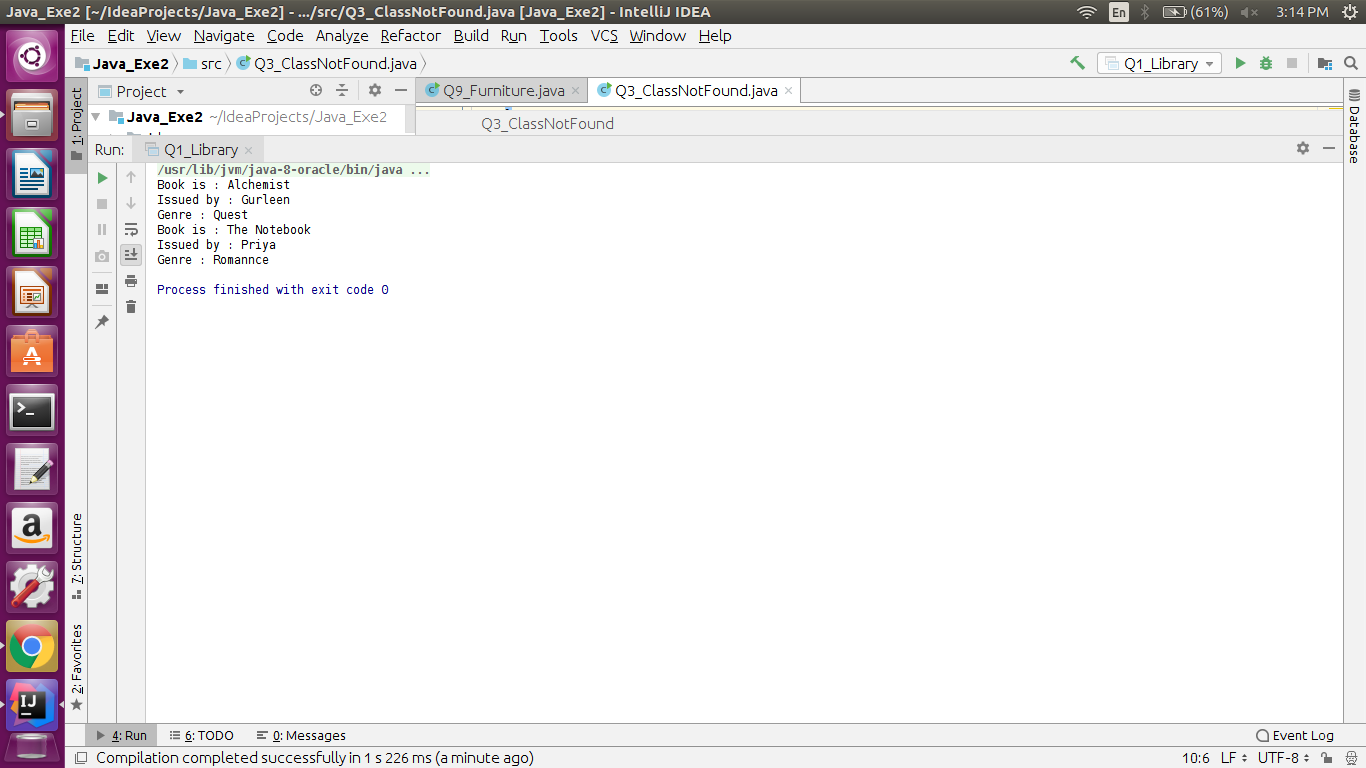
b2.addBooks(2,"The Notebook","Romannce");

b2.bookIssue(2,"The Notebook", "Priya");

b2.showAllBooks();

}

}



**2. WAP to sorting string without using string Methods.**

import java.util.\*;

public class Q2\_Sort {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter String ");

String str = scan.nextLine();

Q2\_Sort sr = new Q2\_Sort();

if (!str.equals("null")) {

sr.sort(str);

}

}

public static void sort(String s) {

int j = 0;

char temp = 0;

char[] chars = s.toCharArray();

for (int i = 0; i < chars.length; i++) {

for (j = 0; j < chars.length; j++) {

if (chars[j] > chars[i]) {

temp = chars[i];

chars[i] = chars[j];

chars[j] = temp;

}

}

}

String name="";

for (int k = 0; k < chars.length; k++) {

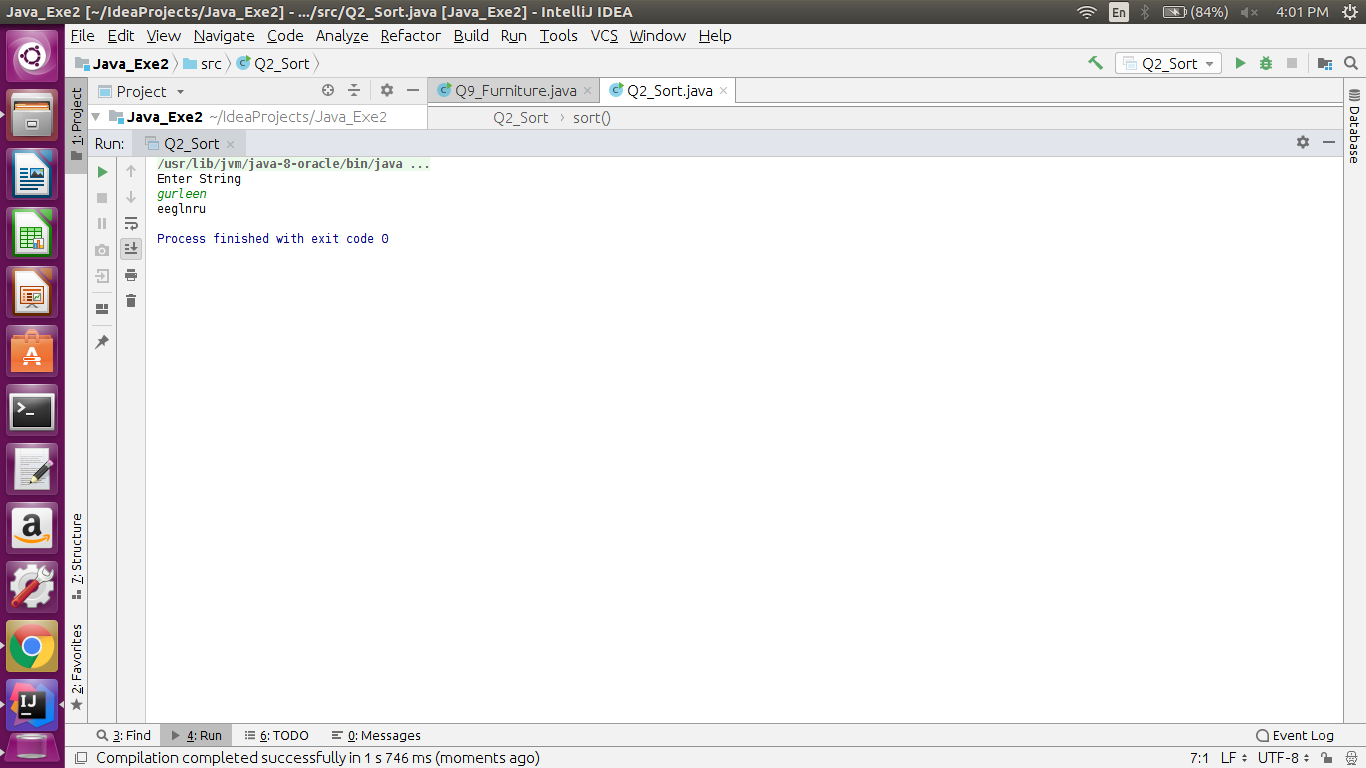
name=name+chars[k];

}

System.out.println(name);

}

}



**3. WAP to produce NoClassDefFoundError and ClassNotFoundException exception.**

**---NoClassDefFoundError**

class A

{}

public class Q3\_NoClassDef {

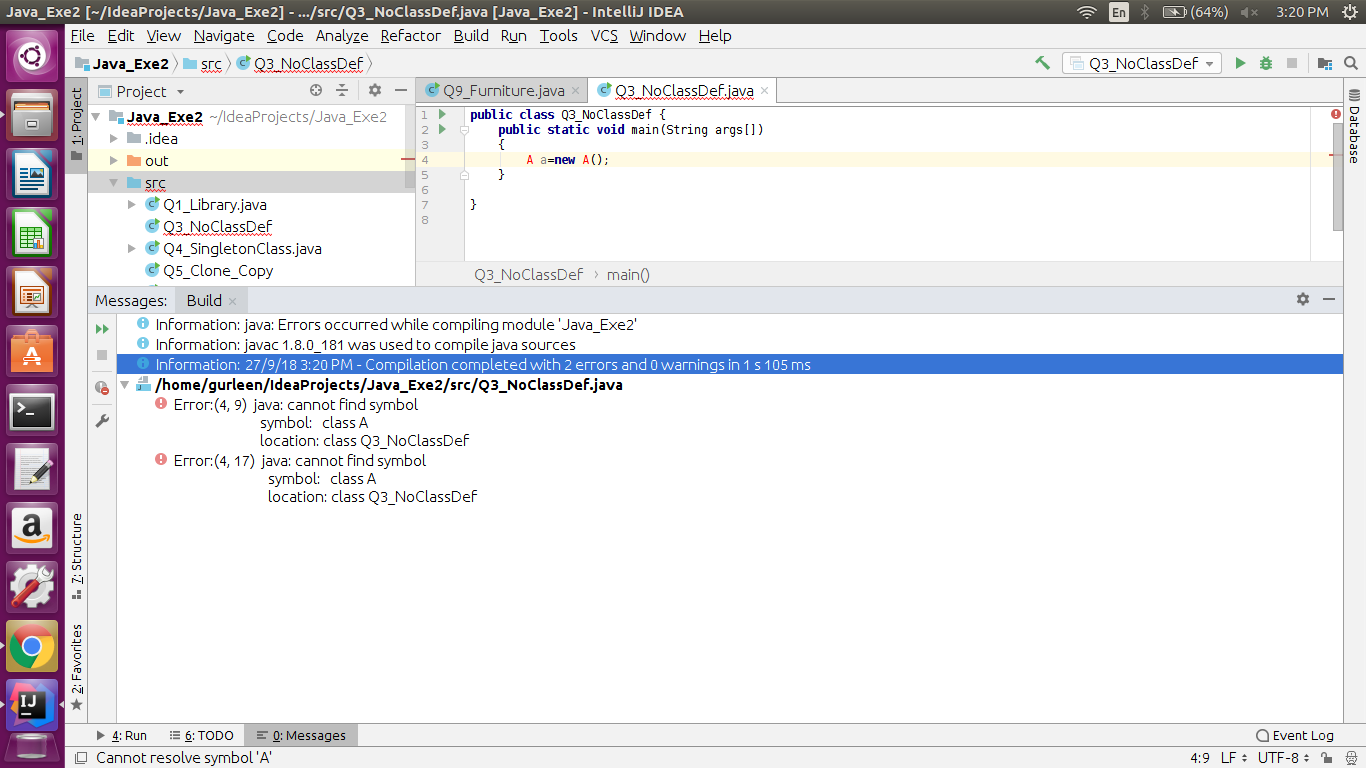
public static void main(String args[])

{

A a=new A();

}

}



**---For ClassNotFound**

public class Q3\_ClassNotFound {

public static void main(String[] args) {

try {

Class.forName("oracle.jdbc.driver.OracleDriver");

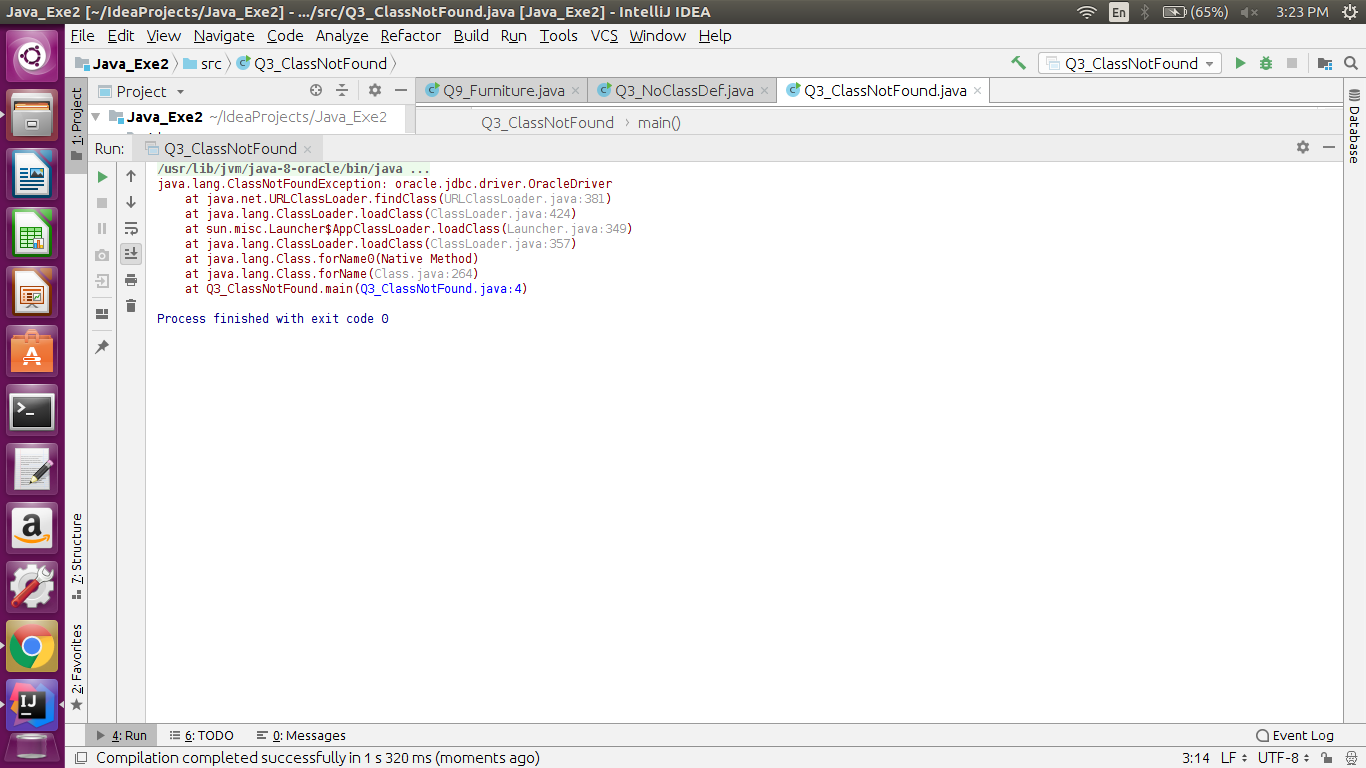
} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

}



**4. WAP to create singleton class.**

class Singleton{

private static Singleton singleton\_instance=null;

public String s;

private Singleton()

{

s="String is instantiated in the constructor";

}

public static Singleton getInstance()

{

if(singleton\_instance==null)

singleton\_instance=new Singleton();

return singleton\_instance;

}

}

public class Q4\_SingletonClass {

public static void main(String args[])

{

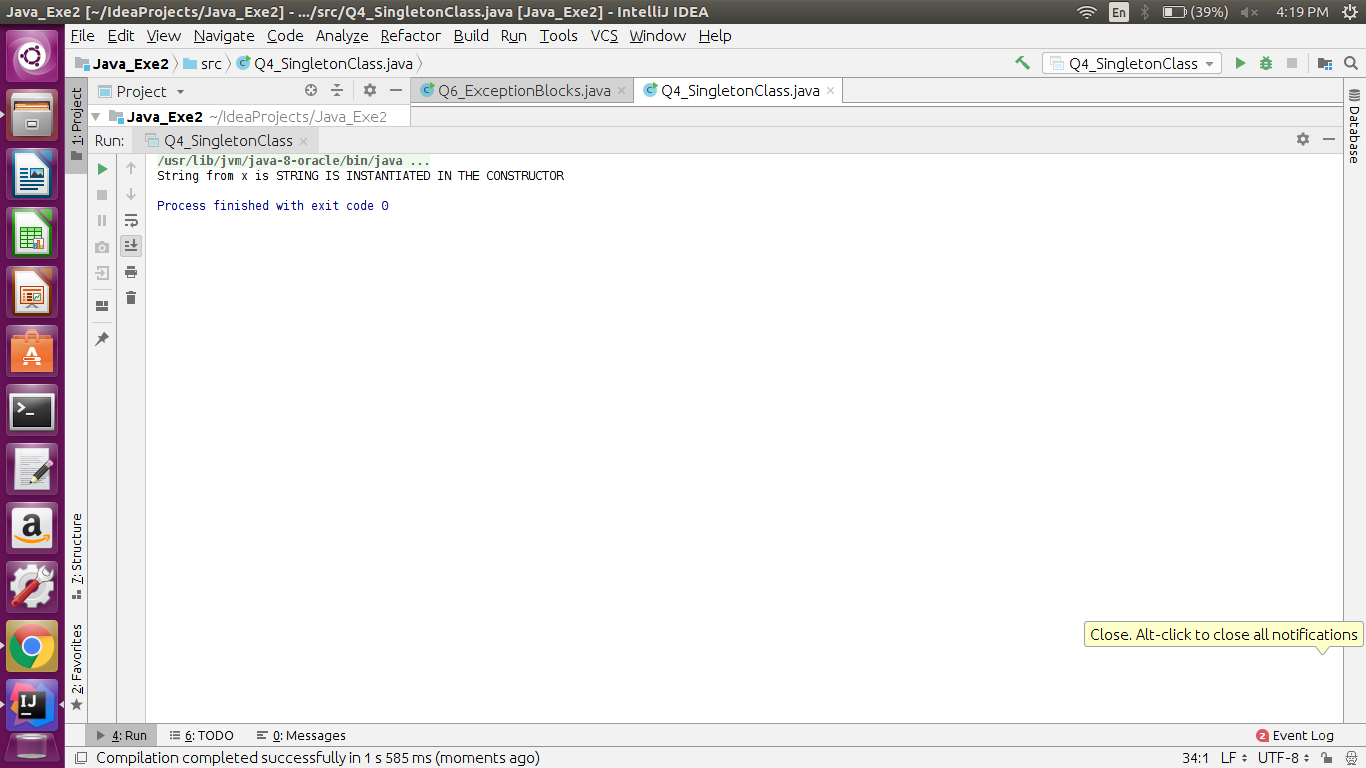
Singleton x = Singleton.getInstance();

x.s = (x.s).toUpperCase();

System.out.println("String from x is " + x.s);

}

}



**5. WAP to show object cloning in java using cloneable and copy constructor both.**

public class Q5\_Clone\_Copy implements Cloneable {

int id;

String name;

Q5\_Clone\_Copy(int i, String s)

{

id=i;

name=s;

}

Q5\_Clone\_Copy(Q5\_Clone\_Copy cc)

{

id=cc.id;

name=cc.name;

}

public Object clone()throws CloneNotSupportedException{

return super.clone();

}

public static void main(String args[])

{

try {

Q5\_Clone\_Copy obj1 = new Q5\_Clone\_Copy(1, "Gurleen");

Q5\_Clone\_Copy obj2 = (Q5\_Clone\_Copy) obj1.clone();

System.out.println("Id is: "+obj1.id +" Name is: "+obj1.name);

System.out.println("Id is: "+obj2.id +" Name is: "+obj2.name+" for cloned object ");

Q5\_Clone\_Copy obj3=new Q5\_Clone\_Copy(obj1);

System.out.println("Id is: "+obj3.id +" Name is: "+obj3.name+" for copy constructor ");

}

catch (CloneNotSupportedException ex)

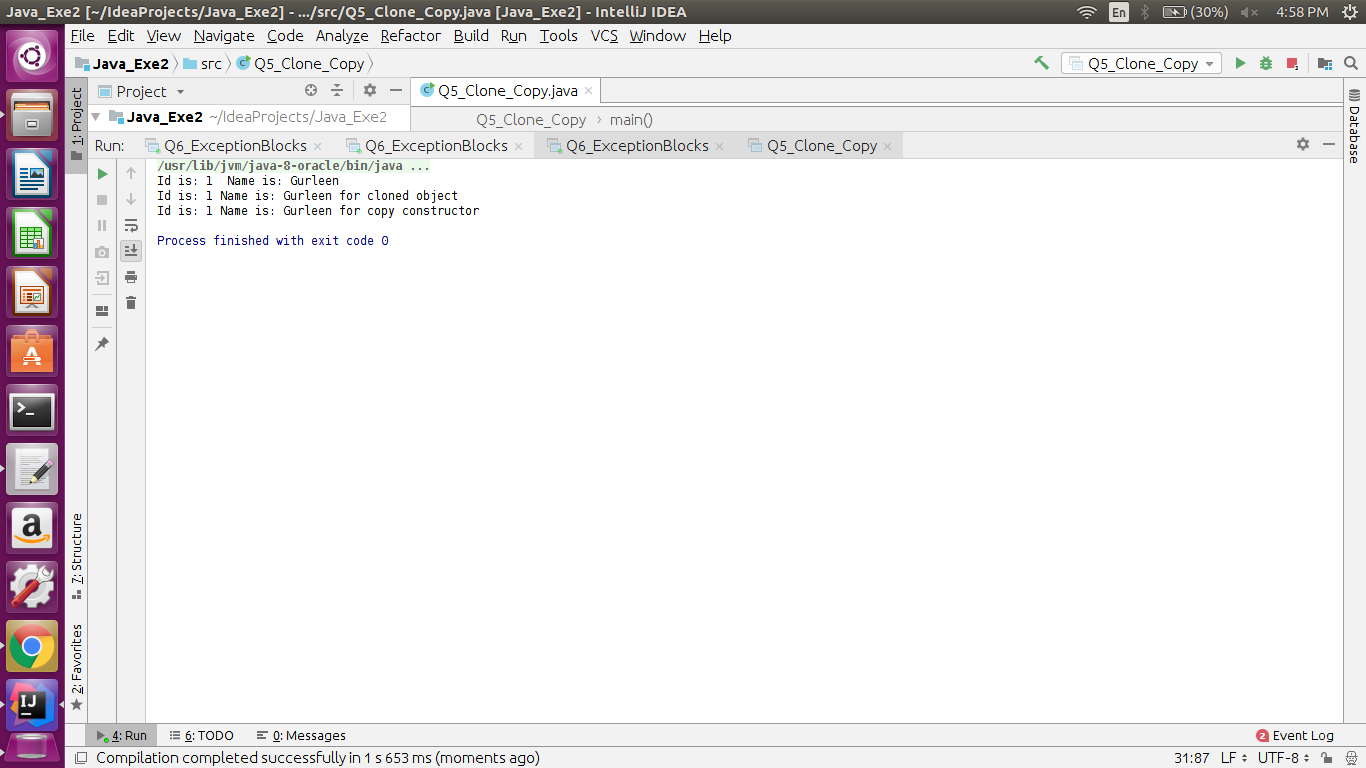
{

System.out.println("Exception arises");

}

}

}



**6. WAP showing try, multi-catch and finally blocks.**

import java.io.IOException;

import java.util.\*;

public class Q6\_ExceptionBlocks {

public static void main(String args[])

{

Scanner sc=new Scanner(System.in);

try

{

System.out.println("Enter the size of an array");

int n=sc.nextInt();

int arr[]=new int[n];

System.out.println("Enter the elements");

for(int i=0;i<arr.length;i++)

{

arr[i]=sc.nextInt();

}

System.out.println("Enter a number for division");

int no=sc.nextInt();

int div=arr[n-1]/no;

System.out.println("Answer is: "+div);

}

catch (ArithmeticException | ArrayIndexOutOfBoundsException | NullPointerException ex)

{

System.out.println("Exception arises");

}

finally

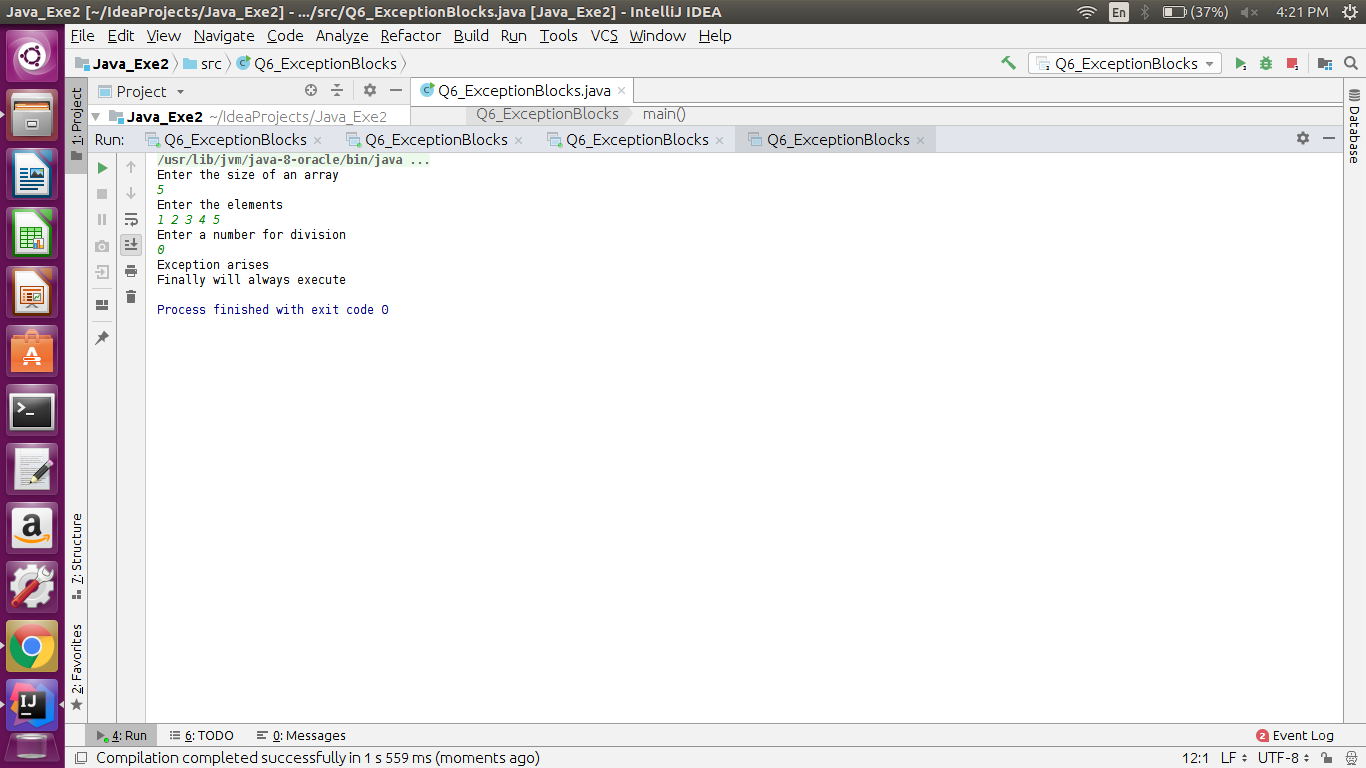
{

System.out.println("Finally will always execute");

}

}

}

**  
7. WAP to convert seconds into days, hours, minutes and seconds.**

import java.util.\*;

import java.text.\*;

public class Q7\_SecondConversion {

int hour, min, sec, day;

public void convert ( int seconds)

{

day = seconds / 86400;

seconds = seconds % (24 \* 3600);

int hour = seconds / 3600;

seconds %= 3600;

int min = seconds / 60;

seconds %= 60;

int sec = seconds;

System.out.println(+day+" "+hour+":"+min+":"+sec+" ");

}

public static void main(String args[])

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the value for seconds");

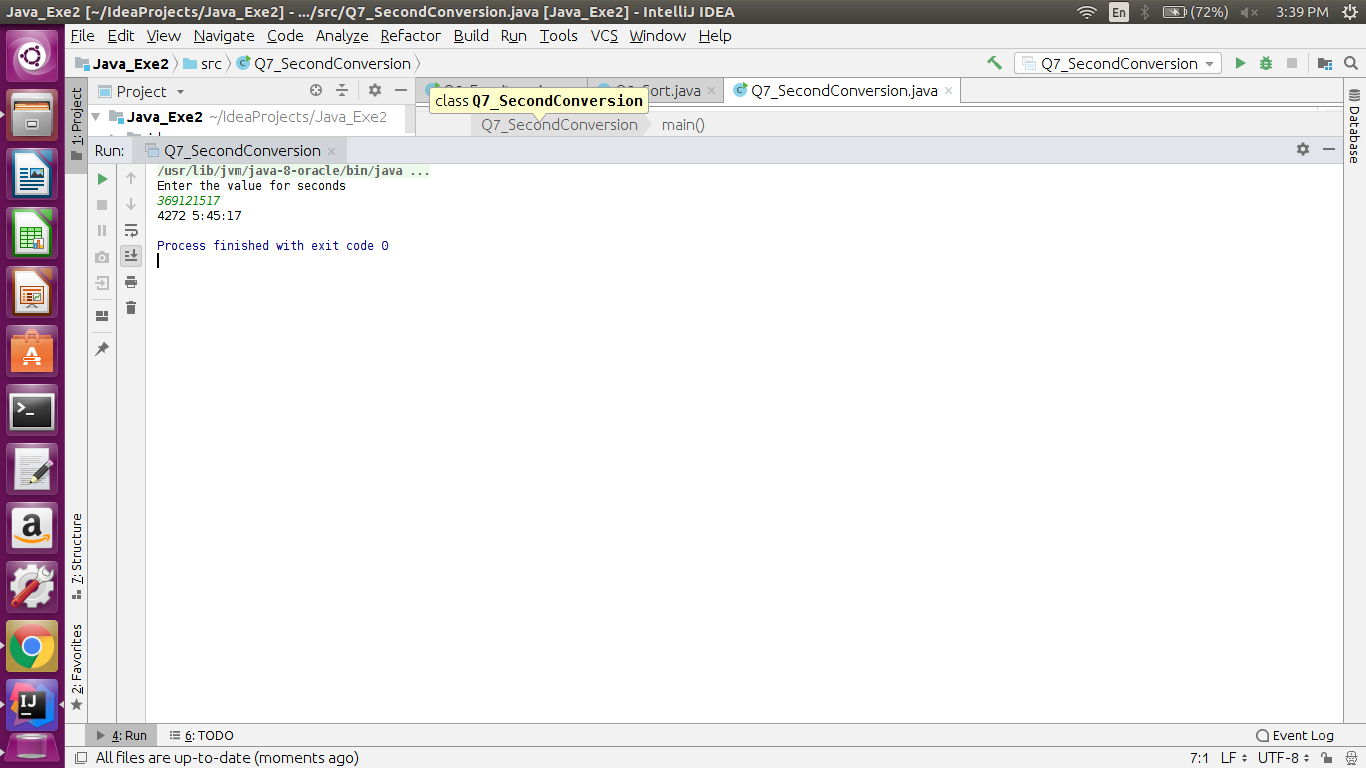
int seconds=sc.nextInt();

Q7\_SecondConversion sec\_con=new Q7\_SecondConversion();

sec\_con.convert(seconds);

}

}

**  
8. WAP to read words from the keyboard until the word done is entered. For each word except done, report whether its first character is equal to its last character. For the required loop, use a   
a)while statement   
b)do-while statement**import java.util.\*;

public class Q8\_Check\_Done {

public static void main(String args[])

{

Scanner sc=new Scanner(System.in);

System.out.println("USING WHILE LOOP");

System.out.println("Enter a string");

String str=sc.nextLine();

String demo;

String str\_do;

String demo\_do;

while(!str.equals("done"))

{

if(str.charAt(0)==str.charAt(str.length()-1))

{

System.out.println("Starting and Ending characters are same in " +str);

}

else

{

System.out.println("Starting and Ending characters are different in: "+str);

}

System.out.println("Enter another string");

demo=sc.nextLine();

str=demo;

}

System.out.println("String terminated after entering done" );

System.out.println("Using DO WHILE LOOP");

do {

System.out.println("Enter a string");

str\_do = sc.nextLine();

{

if (str\_do.charAt(0) == str\_do.charAt(str\_do.length() - 1)) {

System.out.println("Starting and Ending characters are same in " + str\_do);

} else {

System.out.println("Starting and Ending characters are different in: " + str\_do);

}

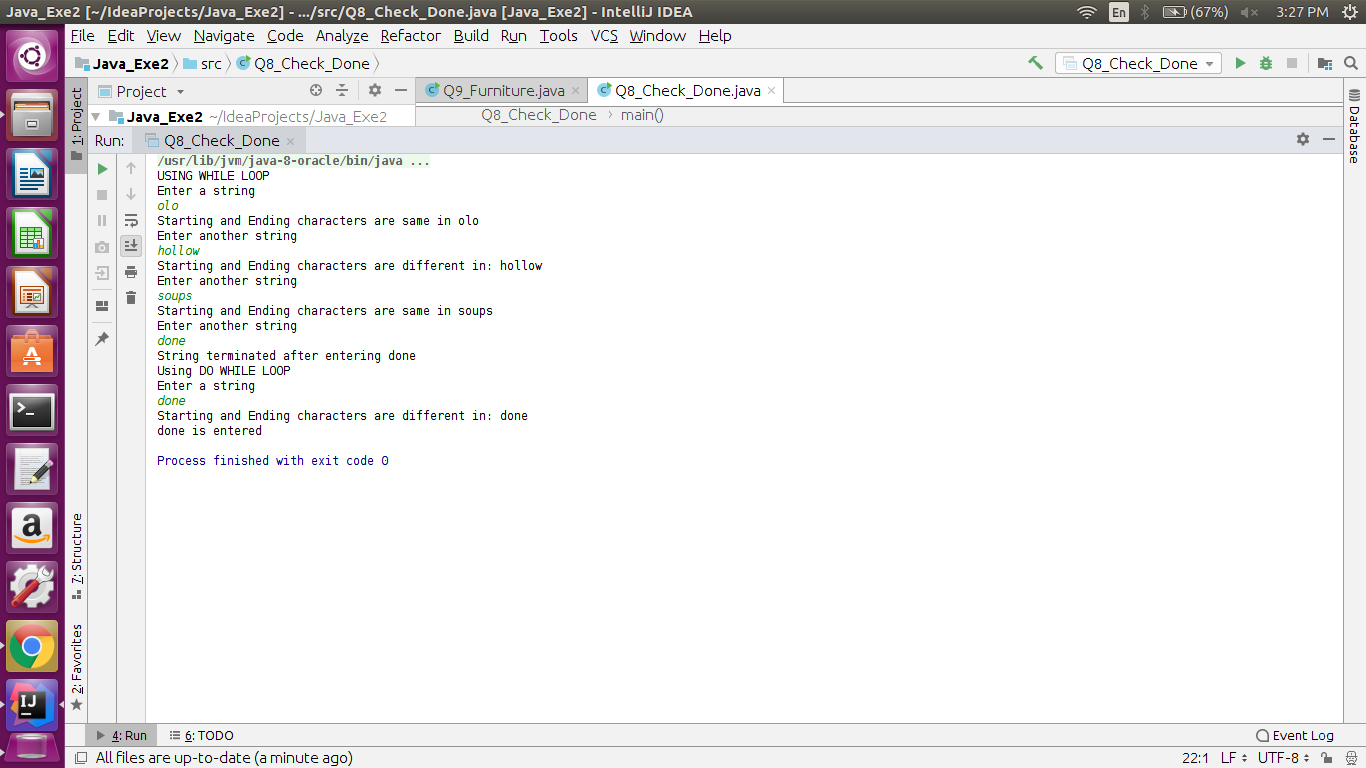
}

}while(!str\_do.equals("done"));

System.out.println("done is entered");

}

**}**

**  
9. Design classes having attributes for furniture where there are wooden chairs and tables, metal chairs and tables. There are stress and fire tests for each products.**

public class Q9\_Furniture {

public static void main(String[] args) {

}

}

abstract class Wooden {

String type;

String durability;

double price;

String finishing;

abstract public void fireTest();

}

abstract class Metal {

String Type;

double price;

String finishing;

abstract public void stressTest();

}

class Chair\_wooden extends Wooden

{

private double height;

private double width;

private String color;

private String chairType;

private double makingCost;

@Override

public void fireTest() {

}

}

class Table\_Wooden extends Wooden

{

private double height;

private double width;

private String color;

private String tableType;

private double cost;

@Override

public void fireTest() {

}

}

class ChairMetallic extends Metal

{

private double height;

private double width;

private String color;

private String chairType;

private double cost;

@Override

public void stressTest() {

}

}

class TableMetallic extends Metal

{

private double height;

private double width;

private String color;

private String chairType;

private double cost;

@Override

public void stressTest() {

}

}

**10. Design classes having attributes and method(only skeleton) for a coffee shop. There are three different actors in our scenario and i have listed the different actions they do also below  
  
\* Customer  
 - Pays the cash to the cashier and places his order, get a token number back  
 - Waits for the intimation that order for his token is ready  
 - Upon intimation/notification he collects the coffee and enjoys his drink  
 ( Assumption: Customer waits till the coffee is done, he wont timeout and cancel the order. Customer always likes the drink served. Exceptions like he not liking his coffee, he getting wrong coffee are not considered to keep the design simple.)  
  
\* Cashier  
 - Takes an order and payment from the customer  
 - Upon payment, creates an order and places it into the order queue  
 - Intimates the customer that he has to wait for his token and gives him his token  
 ( Assumption: Token returned to the customer is the order id. Order queue is unlimited. With a simple modification, we can design for a limited queue size)  
  
\* Barista  
 - Gets the next order from the queue  
 - Prepares the coffee  
 - Places the coffee in the completed order queue  
 - Places a notification that order for token is ready**

import java.util.\*;

public class Q10\_CoffeeShop {

public static void main(String[] args) {

}

}

class Customer

{

private String name;

private int orderId;

private double cashPaid;

private String orderPlaced;

}

class Cashier

{

private Queue<Integer> orderQueue;

private double paymentReceived;

private String takesOrder;

private int orderId;

void insertInOrderQueue(int orderId)

{

orderQueue.add(orderId);

}

}

class Barista extends Cashier

{

Queue<Integer> completeOrder;

int currentOrder = completeOrder.peek();

void sendNotification(int currentOrder)

{

//Completed order

}

}

**11. Convert the following code so that it uses nested while statements instead of for statements:**  **int s = 0;   
 int t = 1;   
 for (int i = 0; i < 10; i++)   
 {   
 s = s + i;   
 for (int j = i; j > 0; j−−)   
 {   
 t = t \* (j - i);   
 }   
 s = s \* t;   
 System.out.println("T is " + t);   
 }   
 System.out.println("S is " + s);**

public class Q11\_NestedWhile {

public static void main(String args[]) {

int s = 0;

int t = 1;

int i=0;

while(i<10) {

s=s+i;

int j=i;

while(j>0)

{

t=t\*(j-i);

j--;

}

i++;

s=s\*t;

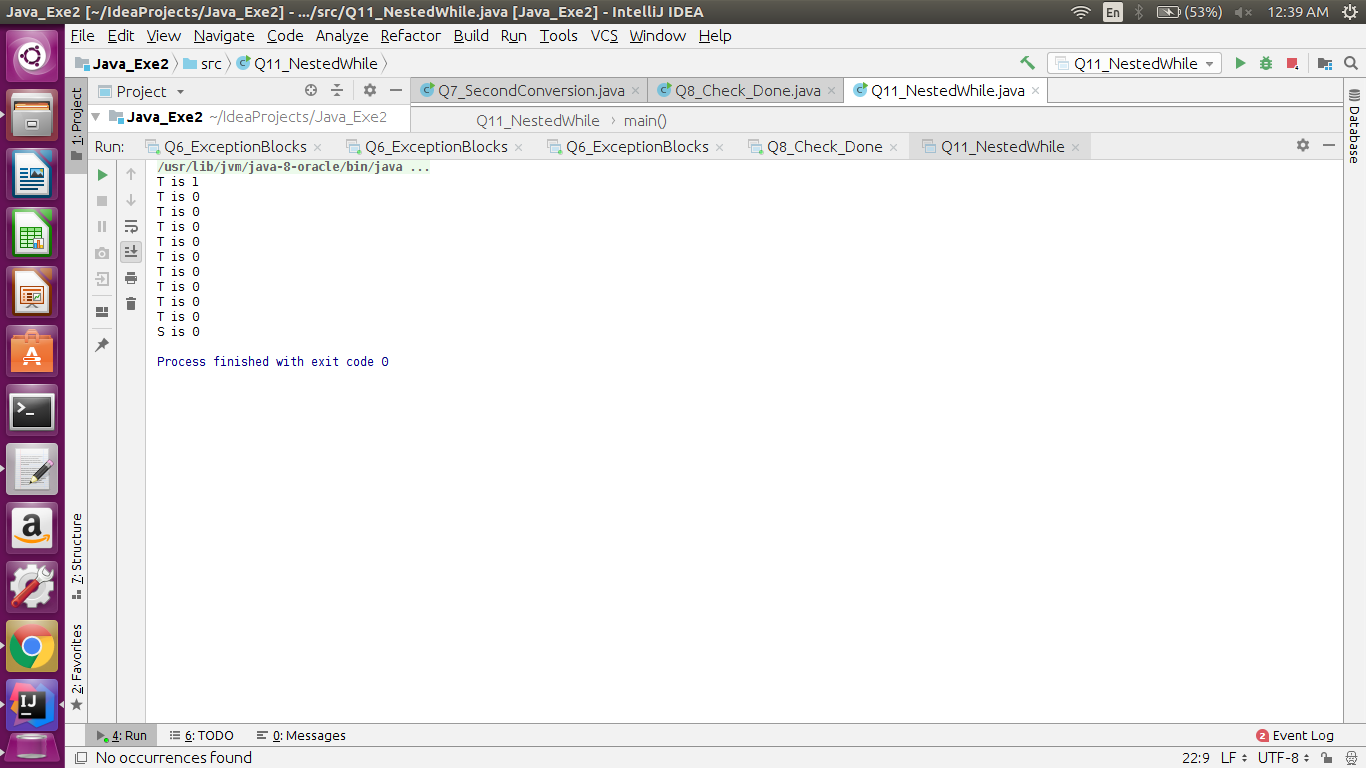
System.out.println("T is " + t);

}

System.out.println("S is " + s);

}

}

**  
12.What will be the output on new Child(); ?   
 class Parent extends Grandparent {  
   
 {  
 System.out.println("instance - parent");  
 }  
  
 public Parent() {  
 System.out.println("constructor - parent");  
 }  
  
 static {  
 System.out.println("static - parent");  
 }  
 }  
  
 class Grandparent {**   
  **static {  
 System.out.println("static - grandparent");  
 }  
  
 {  
 System.out.println("instance - grandparent");  
 }**  
 **public Grandparent() {  
 System.out.println("constructor - grandparent");  
 }  
 }  
  
 class Child extends Parent {  
  
 public Child() {  
 System.out.println("constructor - child");  
 }  
  
 static {  
 System.out.println("static - child");  
 }  
  
 {  
 System.out.println("instance - child");  
 }  
 }**

class Grandparent {

static {

System.out.println("static - grandparent");

}

{

System.out.println("instance - grandparent");

}

public Grandparent() {

System.out.println("constructor - grandparent");

}

}

class Parent extends Grandparent {

{

System.out.println("instance - parent");

}

public Parent() {

System.out.println("constructor - parent");

}

static {

System.out.println("static - parent");

}

}

class Child extends Parent {

public Child() {

System.out.println("constructor - child");

}

static {

System.out.println("static - child");

}

{

System.out.println("instance - child");

}

}

public class Q12\_ParentChild {

public static void main(String args[])

{

//Grandparent g=new Grandparent();

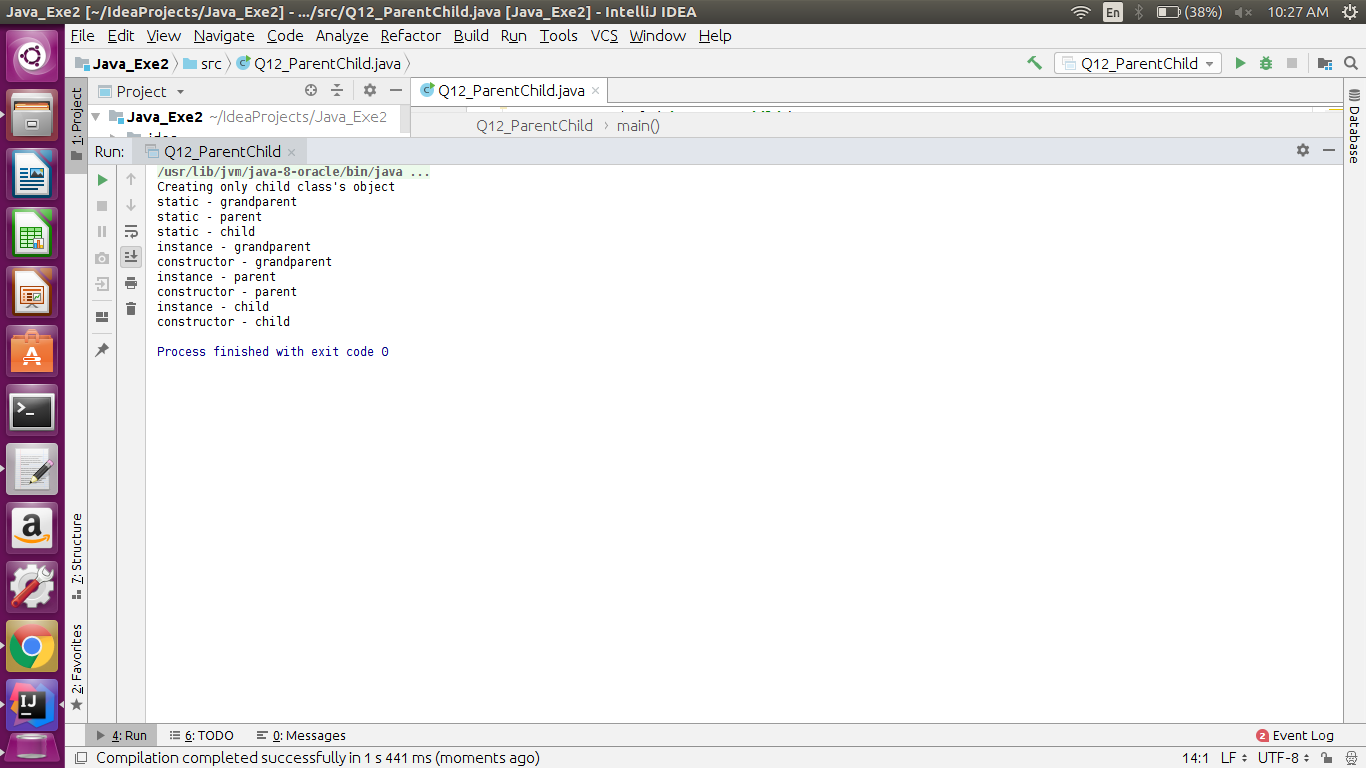
//Parent p=new Parent();

System.out.println("Creating only child class's object");

Child c=new Child();

}

}

**  
Q13. Create a custom exception that do not have any stack trace**

import java.util.\*;

class InvalidValueException extends Exception

{

InvalidValueException(String s) {

super(s);

}

}

public class Q13\_CustomException {

static void check(int i)throws InvalidValueException

{

if(i>10)

throw new InvalidValueException("Enter a value less than 10");

if(i==10)

System.out.println(+i+" is equal to 10");

else

System.out.println(+i+" is less than 10");

}

public static void main(String args[])

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter a value to check");

int val=sc.nextInt();

try{

check(val);

}catch(Exception m)

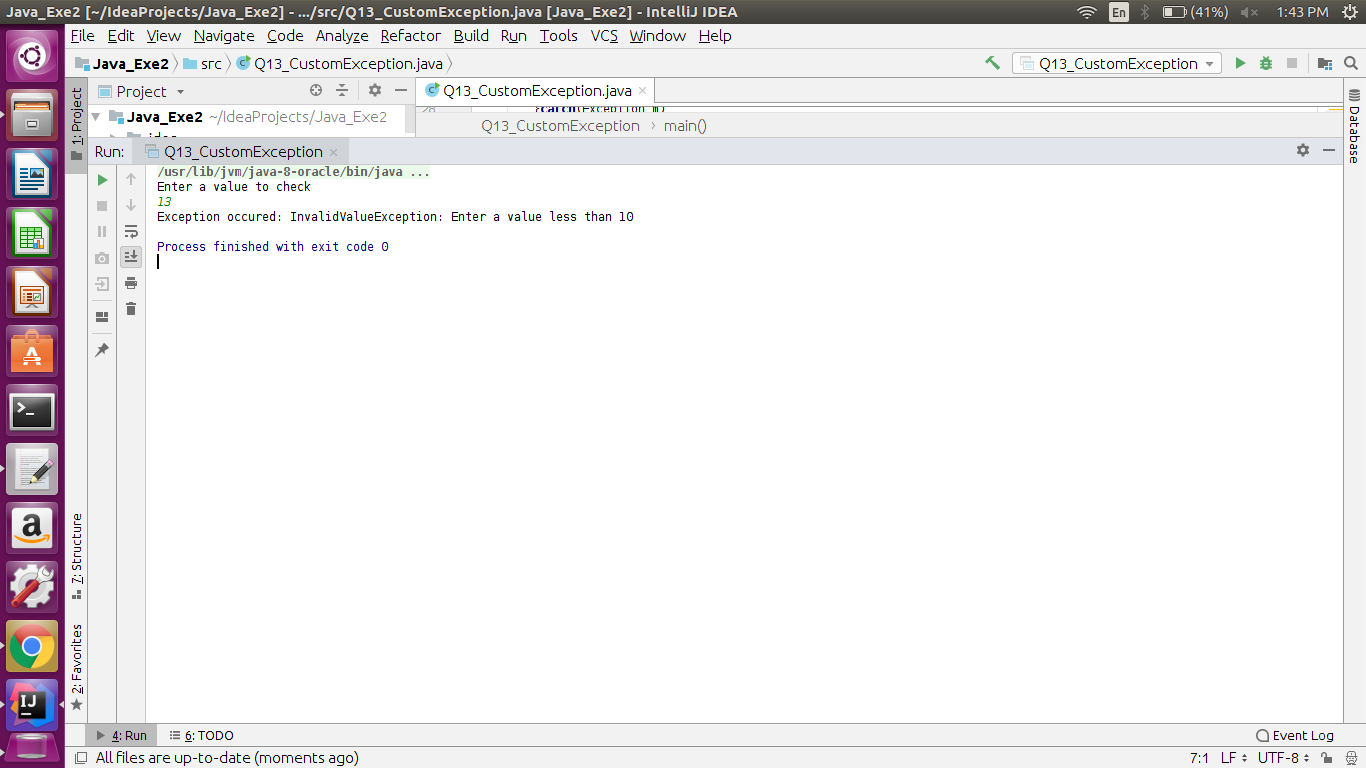
{

System.out.println("Exception occured: "+m);

}

}

}

**.**