**JAVA 2 EXERCISE**

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.

Code:-

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Scanner;

**interface** issue{

**void** issueBooks();

**void** alreadyIssuedBooks();

}

**interface** return\_book{

**void** haveToReturn();

**void** returend\_Books();

**void** total\_Balance();

}

**class** Library **implements** issue,return\_book{

**public void** issueBooks() {

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

Books.add(**"DBMS"**);

Books.add(**"SQL"**);

Books.add(**"JAVA"**);

Books.add(**"DAA"**);

Books.add(**"gradle"**);

Books.add(**"linux"**);

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

String input1=sc.nextLine().toUpperCase();

**boolean** ans = Books.contains(input1);

**if**(ans){

System.***out***.println(**"BOOK IS AVAILABLE YOU CAN ISSUE IT"**);

}

**else**{

System.***out***.println(**"BOOK IS NOT AVAILABLE YOU CANNOT ISSUE IT"**);

}

}

@Override

**public void** alreadyIssuedBooks() {

}

@Override

**public void** haveToReturn() {

}

@Override

**public void** returend\_Books() {

}

@Override

**public void** total\_Balance() {

}

**void** DisplayBooks(){

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

Books.add(**"DBMS"**);

Books.add(**"SQL"**);

Books.add(**"JAVA"**);

Books.add(**"DAA"**);

Books.add(**"gradle"**);

Books.add(**"linux"**);

System.***out***.println(**"Books available in library are :"**);

**for**(String x: Books){

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

}

}

}

**class** Ques1 **extends** Library{

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

Library l = **new** Library();

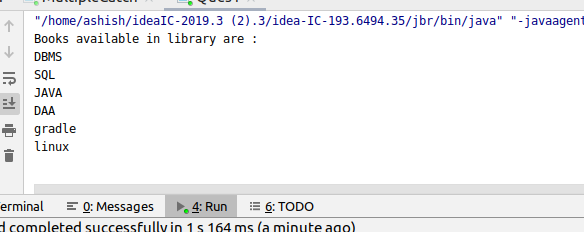
l.DisplayBooks();

l.issueBooks();

}

}

Output:-



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

CODE:-

**public class** Que2 {

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

{

String str =**"ashu"**;

**char** temp = 0;

**char**[] str1 = str.toCharArray();

**for**(**int** i=0;i<str1.**length**;i++)

{

**for**(**int** j=0;j<str1.**length**;j++)

{

**if**(str1[j] > str1[i])

{

temp = str1[i];

str1[i]=str1[j];

str1[j]=temp;

}

}

}

**for**(**int** k=0;k<str1.**length**;k++)

{

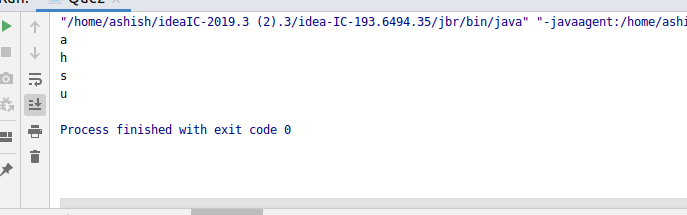
System.***out***.println(str1[k]);

}

}

}

Output:-



3. WAP to produce NoClassDefFoundError and ClassNotFoundException exception.

CODE1:-

**import** javax.naming.Name;

**public class** Que3 {

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

{

**try**{

Class.*forName*(**"Ashish jha"**);

}

**catch** (ClassNotFoundException ex)

{

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

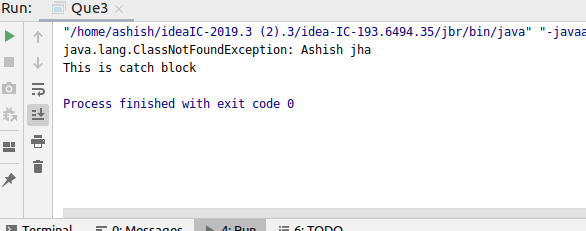
System.***out***.println(**"This is catch block"**);

}

}

}

Output:-



CODE2:-

**public class** Que3 {

**void** show()

{

System.***out***.println(**"This is ashish"**);

}

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

{

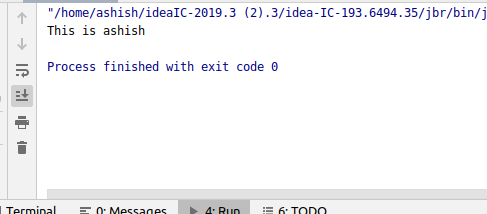
Que3 q= **new** Que3();

q.show();

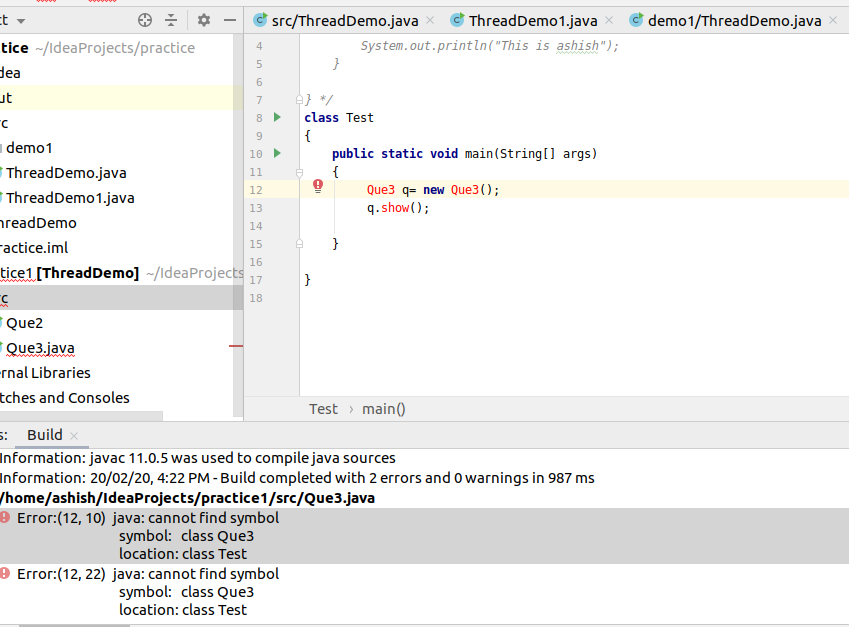
}

}

Output:-



When class is not present



4. WAP to create singleton class.

CODE:-

**public class** Practice {

**private static** Practice *myobj*;

**static**{

*myobj* = **new** Practice();

}

**private** Practice(){

}

**public static** Practice getInstance(){

**return** *myobj*;

}

**public void** testMe()

{

System.***out***.println(**"Hey......it is working"**);

}

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

{

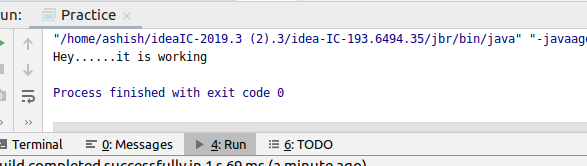
Practice ms = *getInstance*();

ms.testMe();

}

}

Output:-



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

Code:-

**public class** Test {

**int x**,**y**;Test() {

**x** = 10;

**y** = 20;

}

}

**class** main{

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

{

Test ob1=**new** Test();

System.***out***.println(ob1.**x** + **" "** + ob1.**y**);

Test ob2 =ob1;

ob2.**x** = 100;

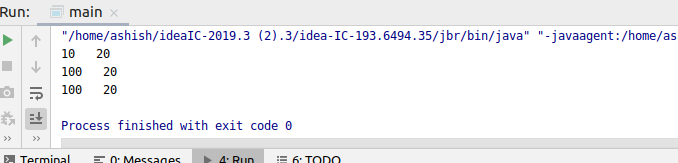
System.***out***.println(ob1.**x** + **" "** + ob1.**y**);

System.***out***.println(ob2.**x** + **" "** + ob2.**y**);

}

}

Output:-



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

Code:-

**import** java.io.\*;

**import** java.nio.channels.ScatteringByteChannel;

**import** java.util.Scanner;

**public class** MultipleCatch {

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

{

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

**try**{

**int** num = Integer.*parseInt*(input.nextLine());

**if**(500 % num == 0){

System.***out***.println(**"this is a factor of 500"**);

}

}

**catch** (ArithmeticException e){

System.***out***.println(**"catch 1"**);

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

}

**catch** (NumberFormatException e){

System.***out***.println(**"catch 2"**);

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

}

**finally**{

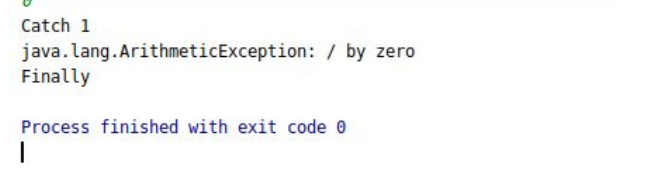
System.***out***.println(**"finally"**);

}

}

}

Output:-



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

Code:-

**import** java.util.Scanner;

**public class** Convertsecond {

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

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

System.***out***.println(**"enter second"**);

**int** sec = input.nextInt();

**int** day = sec /(24\*3600);

sec = sec % (24 \*3600);

**int** hour = sec / 3600;

sec %=3600;

**int** miniutes = sec / 60;

sec %=60;

**int** seconds = sec;

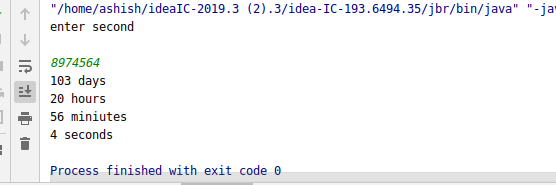
System.***out***.println(day + **" "** + **"days \n"** + hour + **" "** + **"hours \n"** + miniutes + **" "** + **"miniutes \n"**

+ seconds + **" "** + **"seconds "**);

}

}

Output:-



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

Code:-

**import** java.util.Scanner;

**public class** Ques8\_1 {

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

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

System.***out***.println(**"enter some words to check"**);

String word=sc.next();

**do**{

**char** ch1=word.charAt(0);

**char** ch2=word.charAt(word.length()-1);

**if**(ch1==ch2){

System.***out***.println(**"The first character and last character is same of the word "** +word);

}

**else** {

System.***out***.println(**"The first character and last character is not same of the word : "** + word);

}

word=sc.next();

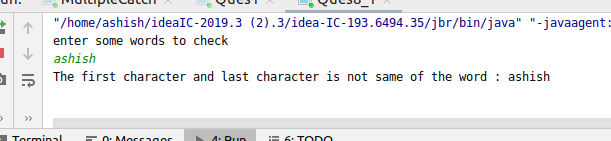
}

**while**(!word.equals(**"done"**));

}

}

Output:-



**import** java.sql.SQLOutput;

**import** java.util.Scanner;

**public class** Ques8\_2 {

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

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

System.***out***.println(**"enter a string"**);

String entry=sc.next();

**while**(!entry.equals(**"done"**)){

**if**(entry.charAt(0)==entry.charAt(entry.length()-1)){

System.***out***.println(**"first and last char is same of word: "**+entry);

}

**else**{

System.***out***.println(**"first and last char is not same of word: "**+entry);

}

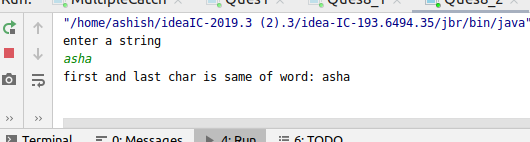
entry=sc.next();

}

}

}

Output:-



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.

Code:-

**import** ​ java.util.Scanner;

**interface** ​ Furniture {

**public void** ​ stressTest();

}

**public void** ​ fireTest();

**abstract class** ​ Table ​ **implements** ​ Furniture {

**public abstract** ​ String tableType();

}

**abstract class** ​ Chair ​ **implements** ​ Furniture {

**public abstract** ​ String chairtype();

}

**class** ​ MetalTable ​ **extends** ​ Table {

**public void** ​ stressTest() {

System.​ out ​ .println(​ **"Stress Test Passed...."**​ );

}

**public void** ​ fireTest() {

System.​ out ​ .println(​ **"Fire Test Passed...."**​ );

}

**public** ​ String tableType() {

String s = ​ **"Metal Table"**​ ;

**return** ​ s;

}

}

**class** ​ WoodenTable ​ **extends** ​ Table {

**public void** ​ stressTest() {

System.​ out ​ .println(​ **"Stress Test Passed...."**​ );

}

**public void** ​ fireTest() {

System.​ out ​ .println(​ **"Fire Test failed.... "**​ );

}

**public** ​ String tableType() {

String s = ​ **"Wooden Table"**​ ;

**return** ​ s;

}

}**class** ​ MetalChair ​ **extends** ​ Chair {

**public void** ​ stressTest() {

System.​ out ​ .println(​ **"Stress Test Passed"**​ );

}

**public void** ​ fireTest() {

System.​ out ​ .println(​ **"Fire Test Passed"**​ );

}

**public** ​ String chairtype() {

String s = ​ **"Metal Chair"**​ ;

**return** ​ s;

}

}

**class** ​ WoodenChair ​ **extends** ​ Chair {

**public void** ​ stressTest() {

System.​ out ​ .println(​ **"Stress Test Failed"**​ );

}

**public void** ​ fireTest() {

System.​ out ​ .println(​ **"Fire test Failed"**​ );

}

**public** ​ String chairtype() {

String s = ​ **"Wooden Chair"**​ ;

**return** ​ s;

}

}

**public class** ​ Q11{

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

Chair chair = ​ null​ ;

Table table = ​ null​ ;

Scanner input = ​ **new** ​ Scanner(System.​ in ) ​ ;

System.​ out ​ .println(​ **"Enter Furniture type:- table/chair"**​ );

String str1 = input.nextLine();

**if** ​ (str1.equals(​ **"chair"**​ )) {

System.​ out . ​ println(​ **"Enter Chair type:- wooden/metal"**​ );

String str = input.nextLine();

**if** ​ (str.equals(​ **"wooden"**​ )) {

chair = ​ **new** ​ WoodenChair();

System.​ out ​ .println(chair.chairtype());

chair.fireTest();

chair.stressTest();

} ​ **else if** ​ (str.equals(​ **"metal"**​ )) {

chair = ​ **new** ​ MetalChair();

System.​ out ​ .println(chair.chairtype());chair.fireTest();

chair.stressTest();

} ​ **else** ​ {

System.​ out ​ .println(​ **"Wrong input..."**​ );

}

} ​ **else if** ​ (str1.equals(​ **"table"**​ )) {

System.​ out . ​ println(​ **"Enter Table type:- wooden/metal"**​ );

String str = input.nextLine();

**if** ​ (str.equals(​ **"wooden"**​ )) {

table = ​ **new** ​ WoodenTable();

System.​ out ​ .println(table.tableType());

table.fireTest();

table.stressTest();

} ​ **else if** ​ (str.equals(​ **"metal"**​ )) {

table = ​ **new** ​ MetalTable();

System.​ out ​ .println(table.tableType());

table.fireTest();

table.stressTest();

} ​ **else** ​ {

System.​ out ​ .println(​ **"Wrong input..."**​ );

}

} ​ **else** ​ {

System.​ out . ​ println(​ **"Wrong Input..."**​ );

}

}

}

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

Code:-

public class CoffeeShop {

|  |
| --- |
| public static void main(String[] args) { |
|  |

|  |
| --- |
| Customer customer = new Customer(); |
|  |

|  |
| --- |
| customer.GiveOrder(40,"Coffee"); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Customer |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| int cash=100; |
|  |

|  |
| --- |
| int id; |
|  |

|  |
| --- |
| static Cashier cashier = new Cashier(); |
|  |

|  |
| --- |
| void GiveOrder(int cost,String order) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| cash-=cost; |
|  |

|  |
| --- |
| cashier.TakeOrders(cost, order); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void GivenOrder(int id,String order) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| this.id=id; |
|  |

|  |
| --- |
| System.out.println("Customer: Order given for "+order+" and orderid is "+id); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void TakeCoffee(String order) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| System.out.println("Customer: Customer has taken "+order); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Cashier |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| int cash=120; |
|  |

|  |
| --- |
| String order; |
|  |

|  |
| --- |
| static Customer customer = new Customer(); |
|  |

|  |
| --- |
| static Barista barista = new Barista(); |
|  |

|  |
| --- |
| void TakeOrders(int cost, String order) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| this.cash+=cost; |
|  |

|  |
| --- |
| this.order=order; |
|  |

|  |
| --- |
| System.out.println("Cashier: Order taken for "+order+" with orderid 1"); |
|  |

|  |
| --- |
| customer.GivenOrder(1,order); |
|  |

|  |
| --- |
| barista.order(order); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void GiveCoffee(String order) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| System.out.println("Cashier: "+order+ " is ready, take it"); |
|  |

|  |
| --- |
| customer.TakeCoffee(order); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Barista |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| String order; |
|  |

|  |
| --- |
| void order(String order) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| Cashier cashier = new Cashier(); |
|  |

|  |
| --- |
| System.out.println("Barista: Barista is preparing "+order); |
|  |

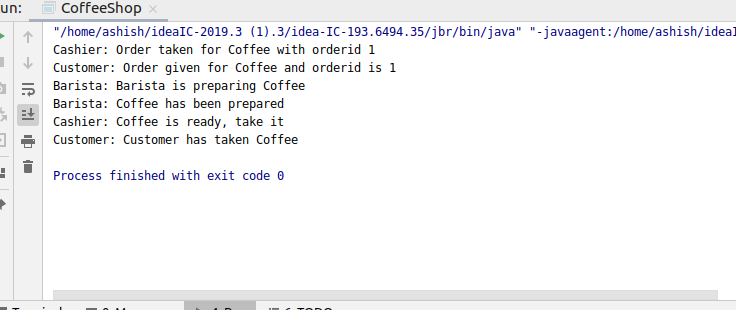
|  |
| --- |
| System.out.println("Barista: "+order+" has been prepared"); |
|  |

|  |
| --- |
| cashier.GiveCoffee(order); |
|  |

|  |
| --- |
| } |
|  |

}

Output:-



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);

Ans:-

int ​ s=​ 0 ​ ;

int ​ t=​ 1 ​ ;

int ​ i=​ 0 ​ ;

while​ (i< ​ 10​ )

{

s=s+i;

int ​ j;

j=i;

while​ (j>​ 0 ​ )

{

t=t\*(j-i);

j--;

}

s=s\*t;

System.out.println(​ "T is "​ +t);

i++;

}

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");

}

}

Output:-

1.static - grandparent

2.static-parent

3.Static-child

4.instance-grandparent

5.constructor-grandparent

6.instance-parent

7.constructor-parent

8.instance-child

9.constructor-child

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

Code:-

**public class** Ques13 {

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

{

**try**

{

**throw new** customException(**"Ashu"**);

}

**catch**(customException CE)

{

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

CE.printStackTrace();

}

}

}

**class** customException **extends** Exception

{

customException(String name)

{

**super**(name);

}

}

Output:-

