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.

package que1;

public abstract class Person {

String name;

public Person()

{

name="";

}

public Person(String name) {

this.name = name;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

package que1;

public interface Issuable {

public boolean issueBook(Reader r,Book b);

public boolean returnBook(Reader r,Book b);

}

package que1;

public interface Verifiable {

public boolean verifyReader( Reader r);

}

package que1;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Driver {

private static BufferedReader *br*;

static BookList *bl*;

static ReaderRecord *rr*;

static Librarian *librarian*;

static void librarianOptions() throws IOException

{

int i=0;Book b;

Reader r;

while(i!=9)

{

System.*out*.println("Enter 1 to add book");

System.*out*.println("Enter 2 to add reader");

System.*out*.println("Enter 3 to issue book");

System.*out*.println("Enter 4 to return book");

System.*out*.println("Enter 5 to verify reader");

System.*out*.println("Enter 6 to remove user");

System.*out*.println("Enter 7 for Reader List");

System.*out*.println("Enter 8 for Book List");

i=Integer.*parseInt*(*br*.readLine());

switch (i)

{

case 1:

System.*out*.println("Enter book Name: ");

Book book = new Book();

book.setName(*br*.readLine());

*bl*.addBook(book);

break;

case 2:

System.*out*.println("Enter reader name: ");

Reader reader=new Reader(*br*.readLine());

*rr*.addReader(reader);

break;

case 3:

System.*out*.println("Enter reader name: ");

r=*rr*.getReaderByName(*br*.readLine());

System.*out*.println("Enter book name: ");

b=*bl*.getBookByName(*br*.readLine());

if(r==null || b==null)

System.*out*.println("Reader or Book Not found");

else {

*librarian*.issueBook(r, b);

}

break;

case 4:

System.*out*.println("Enter reader name: ");

r=*rr*.getReaderByName(*br*.readLine());

System.*out*.println("Enter book name: ");

b=*bl*.getBookByName(*br*.readLine());

if(r==null || b==null)

System.*out*.println("Reader or Book Not found");

else

*librarian*.returnBook(r,b);

break;

case 5:

System.*out*.println("Enter reader name: ");

r=*rr*.getReaderByName(*br*.readLine());

if (r != null)

*librarian*.verifyReader(r);

else

System.*out*.println("Reader not found ");

break;

case 6:

System.*out*.println("Enter reader name: ");

r=*rr*.getReaderByName(*br*.readLine());

if (r != null)

*rr*.removeReader(r);

else

System.*out*.println("Reader not found");

break;

case 7:

*rr*.printList();

break;

case 8:

*bl*.showList();

break;

default:

i=9;

break;

}

}

}

static void readerOptions() throws IOException

{

int i=0;

Reader r;

while(i!=9)

{

System.*out*.println("Enter 1 to show details");

System.*out*.println("Enter 2 to show list ");

i=Integer.*parseInt*(*br*.readLine());

switch (i)

{

case 1:

System.*out*.println("Enter reader name: ");

r=*rr*.getReaderByName(*br*.readLine());

if (r != null)

r.showDetails();

else

System.*out*.println("Reader not found");

break;

case 2:

System.*out*.println("Enter reader name: ");

r=*rr*.getReaderByName(*br*.readLine());

if (r != null)

r.showBook();

else

System.*out*.println("Reader not found");

break;

default:

i=9;

break;

}

}

}

static void mainMenu()

{

System.*out*.println("Enter 1 for Librarian");

System.*out*.println("Enter 2 for Reader");

System.*out*.println("Enter else to exit");

}

public static void main(String[] args) throws IOException

{

System.*out*.println("Enter Librarian Name: ");

*br*=new BufferedReader(new InputStreamReader(System.*in*));

int i=1;

*bl*=new BookList();

*rr*=new ReaderRecord();

*librarian*=new Librarian(*br*.readLine());

while(i==1 || i==2) {

*mainMenu*();

System.*out*.println("Enter an option");

i=Integer.*parseInt*(*br*.readLine());

//System.out.println(i);

switch (i)

{

case 1:

System.*out*.println("Librarian options");

*librarianOptions*();

break;

case 2:

*readerOptions*();

System.*out*.println("Reader options: ");

break;

default:

i=3;

break;

}

}

}

}

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

package que2;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class q2 {

static int compare(String str1,String str2)

{

int a=0;

int l1 = str1.length();

int l2 = str2.length();

int lmin = (l1<l2)?l1:l2;

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

int str1\_ch = (int)str1.charAt(i);

int str2\_ch = (int)str2.charAt(i);

if (str1\_ch != str2\_ch) {

return str1\_ch - str2\_ch;

}

}

if (l1 != l2) {

return l1 - l2;

}

else {

return 0;

}

}

public static void main(String[] args) {

BufferedReader br=null;

try{

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

br=new BufferedReader(new InputStreamReader(System.*in*));

//read line

String str=br.readLine();

String[] arr=str.split(" ");

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

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

if(*compare*(arr[i],arr[j])<0){

String temp=arr[j];

arr[j]=arr[i];

arr[i]=temp;

}

}

}

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

System.*out*.print(arr[i]+" ");

}

}

catch (IOException e)

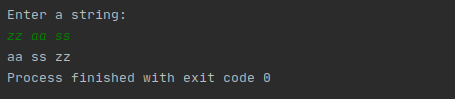
{

e.printStackTrace();

}

}

}



3. WAP to produce NoClassDefFoundError and ClassNotFoundException exception.

// Java program to illustrate

// ClassNotFoundException

public class q3{

public static void main(String args[]) {

try

{

Class.*forName*("GeeksForGeeks");

}

catch (ClassNotFoundException ex)

{

ex.printStackTrace();

}

}

}

//To show NoclassDeffoundError delete class1.class file the run the code

class class1

{

void greeting()

{

System.*out*.println("hello!");

}

}

class q3\_2 {

public static void main(String args[])

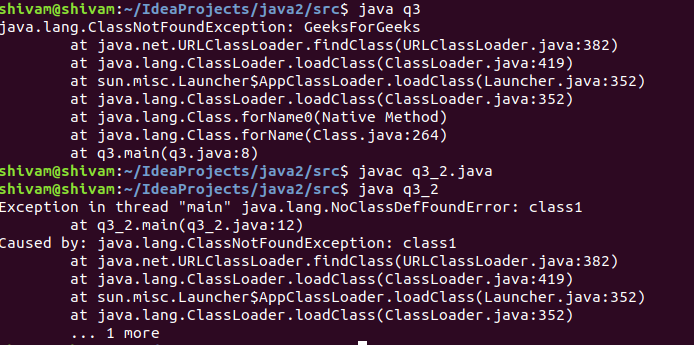
{

class1 c1 = new class1();

c1.greeting();

}

}



4. WAP to create singleton class.

class Singleton {

static Singleton *single* ; // Everytime this object will be called.

public String str;

private Singleton() {

str=" This is singleton class";

/\* hinder other classes to create an object of this class. \*/

}

public static Singleton getInstance()

{

if(null==*single*)

{

*single* = new Singleton();

}

return *single*;

}

}

public class q4 {

public static void main(String[] args) {

Singleton s1 = Singleton.*getInstance*();

Singleton s2 = Singleton.*getInstance*();

s1.str= (s1.str).toUpperCase();

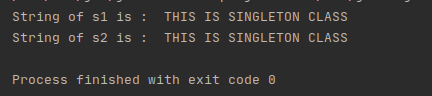
System.*out*.println("String of s1 is : "+s1.str);

System.*out*.println("String of s2 is : "+s2.str);

// Singleton s2= new Singleton();/\* Can't be created because Singleton() has private access in Singleton.\*/

}

}



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

class CheckClone implements Cloneable{

int i;

int j;

@Override

public String toString() {

return "CheckClone{" + "i=" + i + ", j=" + j + '}';

}

@Override

public Object clone() throws CloneNotSupportedException{

return super.clone();

}

}

public class q5 {

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

CheckClone clone= new CheckClone();

clone.i=5;

clone.j=7;

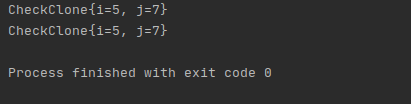
CheckClone clone1= (CheckClone) clone.clone();

System.*out*.println(clone);

System.*out*.println(clone1);

}

}



public class q5two {

int one, two,one1,two1;

q5two()

{

one =5;

two =6;

}

q5two(q5two sc )

{ one1 =sc.one;

two1 =sc.two;

}

public static void main(String[] args) {

q5two obj = new q5two();

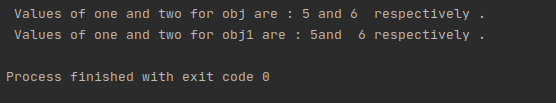
q5two obj2 =new q5two(obj);

System.*out*.println(" Values of one and two for obj are : "+(obj.one) +" and "+(obj.two)+" respectively .");

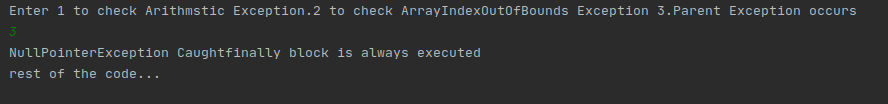
System.*out*.println(" Values of one and two for obj1 are : "+(obj2.one1) +"and "+(obj2.two1)+" respectively .");

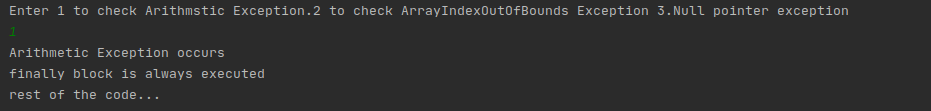
}

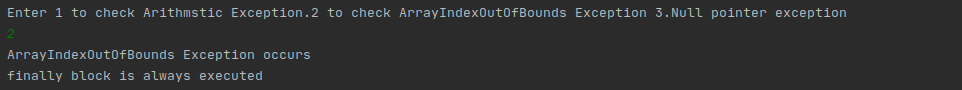
}



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







import java.io.\*;

import java.util.Scanner;

public class q6 {

public static void main(String[] args) {

try {

System.*out*.println("Enter 1 to check Arithmstic Exception.2 to check ArrayIndexOutOfBounds Exception 3.Null pointer exception");

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

int input = sc.nextInt();

String ptr = null;

int a[] = new int[5];

switch (input) {

case 1:

a[5] = 30 / 0;

case 2:

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

{ a[i] = i;}

case 3:

if (ptr.equals("gfg"))

System.*out*.print("Same");

else

System.*out*.print("Not Same");

default:

System.*out*.println("Enter right choice");

}

} catch (ArithmeticException e) {

System.*out*.println("Arithmetic Exception occurs");

} catch (ArrayIndexOutOfBoundsException e) {

System.*out*.println("ArrayIndexOutOfBounds Exception occurs");

} catch (NullPointerException e) {

System.*out*.print("NullPointerException Caught");

} catch (Exception e) {

System.*out*.println("Parent Exception occurs");

} finally {

System.*out*.println("finally block is always executed");

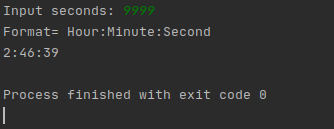
}

System.*out*.println("rest of the code...");

}

}

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



import java.util.Scanner;

public class q7 {

public static void main(String[] args)

{

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

System.*out*.print("Input seconds: ");

int seconds = in.nextInt();

int p1 = seconds % 60;

int p2 = seconds / 60;

int p3 = p2 % 60;

p2 = p2 / 60;

System.*out*.println("Format= Hour:Minute:Second");

System.*out*.print( p2 + ":" + p3 + ":" + p1);

System.*out*.print("\n");

}

}

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.sql.SQLOutput;

import java.util.Scanner;

public class q8 {

public static void main(String[] args) {

System.*out*.println("Using while loop :: ");

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

System.*out*.println("Enter a word : ");

String str = keyboard.next();

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

{

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

{

System.*out*.println(" First and last character are equal in the str: " + str);

}

else

{

System.*out*.println("First and last character are not equal in the str: " + str);

}

System.*out*.println("Enter more to check....else enter 'done' to do same using DO-WHILE loop....");

str = keyboard.next();

}

System.*out*.println("Now Using do-while loop :: ");

Scanner keyboard1= new Scanner(System.*in*);

System.*out*.println("Enter a word : ");

String str1 = keyboard.next();

do {

if(str1.charAt(0) == str1.charAt(str1.length() - 1))

{

System.*out*.println("First and last character are equals for the str: " + str1);

}

else

{ System.*out*.println("First and last character are NOT equals for the str: " + str1);

}

System.*out*.println("Enter more to check....else enter 'done' to exit....");

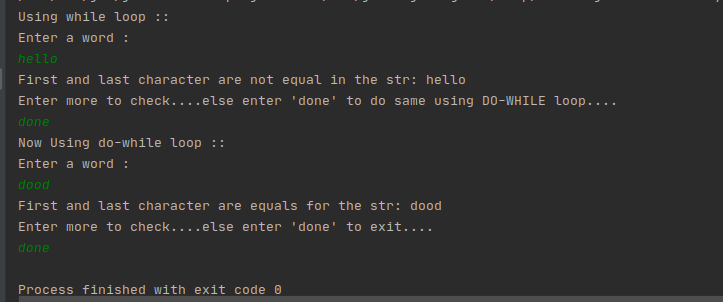
str1 = keyboard.next();

}

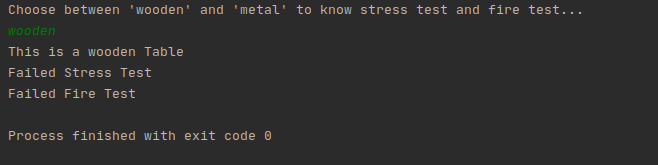
while(!str1.equals("done"));

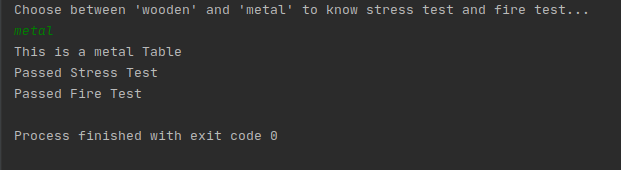
}

}



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.





import java.util.Scanner;

interface Furniture {

public void stressTest();

public void fireTest();

}

abstract class Chair implements Furniture {

public abstract String chairType();

}

abstract class Table implements Furniture {

public abstract String tableType();

}

class MetalChair extends Chair {

@Override

public void stressTest() {

}

@Override

public void fireTest() {

}

@Override

public String chairType() {

String s = "This is a metal Chair";

return s;

}

}

class MetalTable extends Table {

@Override

public void stressTest() {

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

}

@Override

public void fireTest() {

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

}

@Override

public String tableType() {

String s = "This is a metal Table";

return s;

}

}

class WoodenTable extends Table {

@Override

public void stressTest() {

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

}

@Override

public void fireTest() {

System.*out*.println("Failed Fire Test");

}

@Override

public String tableType() {

String s = "This is a wooden Table";

return s;

}

}

class WoodenChair extends Chair {

@Override

public void stressTest() {

}

@Override

public void fireTest() {

}

@Override

public String chairType() {

String s = "This is a wooden chair";

return s;

}

}

public class q9{

public static void main(String[] args){

Table table = null;

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

System.*out*.println("Choose between 'wooden' and 'metal' to know stress test and fire test... ");

String str = input.next();

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

table = new WoodenTable();

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

table = new MetalTable();

}

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

table.stressTest();

table.fireTest();

}

}

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

**//File name Q10.java**

package java2;

//10. Design classes having attributes and method(only skeleton) for a coffee shop.

public class Q10 {

public static void main(String[] args) {

Cashier cashier = new Cashier();

Bistro bistro = new Bistro();

Customer customer = new Customer("Shivam",3999);

customer.PlaceOrder();

cashier.takeOrder(customer.getName());

int token=cashier.giveTokenNo(customer.getName());

customer.getTokeno(token);

cashier.receivePayment(customer.getTokeno(token));

cashier.addItToPendingQueue(customer.getTokeno(token));

customer.checkOrderStatus(customer.getTokeno(token));

bistro.getOrderFromPendingQueue();

bistro.prepareOrder();

bistro.insertOrderIntoCompletedOrderQueue();

}

}

package java2;

public class Bistro

{

public void getOrderFromPendingQueue()

{

//take order from pending queue

}

public void prepareOrder()

{

// called when order is preparing

}

public void insertOrderIntoCompletedOrderQueue()

{

//output to completed order queue

}

}

package java2;

public class Cashier

{

String cashierName;

int CashierId;

public String getCashierName() {

return cashierName;

}

public void setCashierName(String cashierName) {

this.cashierName = cashierName;

}

public int getCashierId() {

return CashierId;

}

public void setCashierId(int cashierId) {

CashierId = cashierId;

}

public void takeOrder(String customerName)

{

//cashier receives the order

}

public int giveTokenNo(String customerName)

{

int token=0;

//cashier provides tokeno to the customer

return token;

}

public void receivePayment(int tokenno)

{

// receives payment

}

public void addItToPendingQueue(int tokeno)

{

//called when there is a queue of orders

}

}

package java2;

public class Customer

{

private String name;

private double phone;

private int tokeno;

Customer(String name,double phone)

{

this.name=name;

this.phone=phone;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getPhone() {

return phone;

}

public void setPhone(double phone) {

this.phone = phone;

}

public int getTokeno(int t) {

tokeno=t;

return tokeno;

}

public void setTokeno(int tokeno) {

this.tokeno = tokeno;

}

public void PlaceOrder()

{

//customer places order on the basis of his name and phone number

}

public void checkOrderStatus(int tokeno)

{

// customer checks order status on the basis of tokeno given him

}

public void collectOrder(int tokeno)

{

// customer can collect the coffee after entering tokenno

}

public void payment()

{

//payment is done through this method

}

public void wrongOrder(){

//not liking his coffee, he getting wrong coffee are not considered to keep the design simple.

}

}

11. Convert the following code so that it uses nested while statements instead of for statements:

class q11{

public static void main(String[] args) {

int i=0,t=1,s=0;

while(i<10)

{

s+=i;

int j=i;

while(j>0)

{

t\*=(j-i);

j--;

}

s\*=t;

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

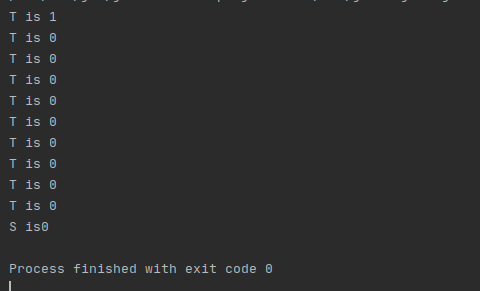
i++;

}

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

}

}



12.What will be the output on new Child(); ?

class 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{

public static void main(String[] args) {

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

Grandparent gp = new Child();

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

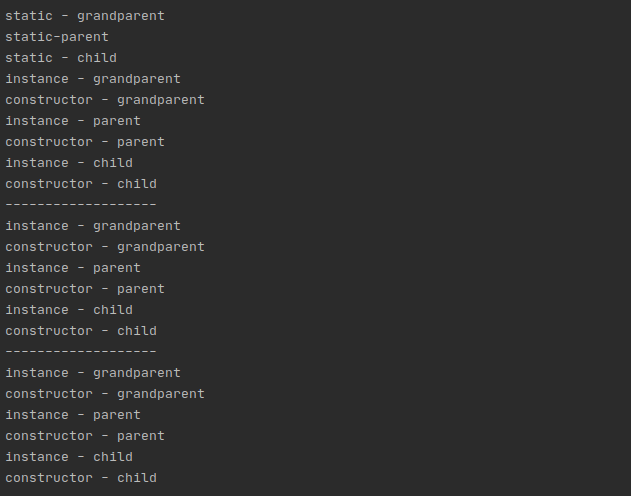
Parent parent= new Child();

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

Child ch= new Child();

}

}



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

class NewException extends Exception{

String str;

NewException(String str)

{

this.str=str;

}

@Override

public String toString() {

return "NewException without stack trace{" +

"str='" + str + '\'' +

'}';

}

}

public class q13 {

public static void main(String[] args) {

try{

throw new NewException("New exception found ");

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

