**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 ques1;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

public class Ques1 {

public static void main(String[] args) {

BookItem book = new BookItem();

Person person = new Person();

}

}

abstract class Book {

private String ISBN;

private String title;

private String subject;

private String publisher;

private String language;

private int numberOfPages;

private List<Author> authors;

}

class BookItem extends Book {

private int bookId;

private Date borrowed;

private Date dueDate;

private double price;

private boolean available;

BookItem(){}

public BookItem(int bookId, Date borrowed, Date dueDate, double price, boolean available) {

this.bookId = bookId;

this.borrowed = borrowed;

this.dueDate = dueDate;

this.price = price;

this.available = available;

}

void displayBookDetails(){}

}

class Person {

private String firstName;

private String lastName;

public Person() {

}

public Person(String firstName, String lastName) {

this.firstName = firstName;

this.lastName = lastName;

}

}

class Author extends Person {

List<Book> booksPublished;

public Author(String firstName, String lastName) {

super(firstName, lastName);

}

}

class SystemUser extends Person {

private String Email;

private String phoneNumber;

private Account account;

public SystemUser(String email, String phoneNumber, Account account) {

super();

Email = email;

this.phoneNumber = phoneNumber;

this.account = account;

}

}

class Account {

private String userName;

private String password;

private int accountId;

public Account(String userName, String password, int accountId) {

this.userName = userName;

this.password = password;

this.accountId = accountId;

}

}

interface databaseOperation{

ArrayList<Book> books = new ArrayList<Book>();

void add(Book book);

void delete(Book book);

void display();

}

class Librarian extends SystemUser implements databaseOperation{

private int librarianId;

Transaction transaction;

public Librarian(String email, String phoneNumber, Account account) {

super(email, phoneNumber, account);

}

void issueBook(){}

void returnBook(){}

@Override

public void add(Book book) {

books.add(book);

}

@Override

public void delete(Book book) {

books.remove(book);

}

@Override

public void display() {

for(Book book:books)

System.out.println(book);

}

}

class Member extends SystemUser implements databaseOperation{

int memberId;

int maxBookLimit;

Address address;

Transaction transaction;

public Member(String email, String phoneNumber, Account account) {

super(email, phoneNumber, account);

}

void issueBook(){}

void returnBook(){}

@Override

public void add(Book book) {

}

@Override

public void delete(Book book) {

}

@Override

public void display() {

}

}

class Address{

String city;

String state;

String country;

int pincode;

public Address(String city, String state, String country, int pincode) {

this.city = city;

this.state = state;

this.country = country;

this.pincode = pincode;

}

}

class Transaction{

int transactionId;

int memberId;

int bookId;

Date dateOfIssue;

Date dueDate;

public Transaction(int transactionId, int memberId, int bookId, Date dateOfIssue, Date dueDate) {

this.transactionId = transactionId;

this.memberId = memberId;

this.bookId = bookId;

this.dateOfIssue = dateOfIssue;

this.dueDate = dueDate;

}

void createTransaction(){}

void deleteTransaction(){}

}

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

import java.util.Scanner;

public class Ques2 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

String str = sc.next();

char[] strarr = str.toCharArray();

System.out.println("Before Sorting: " + str);

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

for(int j=0;j<strarr.length-i-1;j++){

if(strarr[j+1] < strarr[j]){

char temp = strarr[j];

strarr[j] = strarr[j+1];

strarr[j+1] = temp;

}

}

}

System.out.print("After Sorting: ");

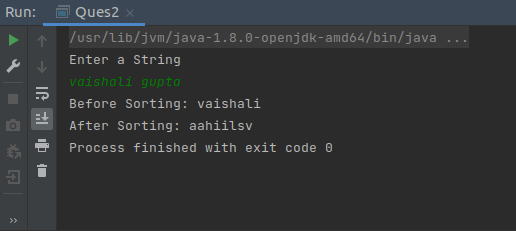
for (char c : strarr) {

System.out.print(c);

}

}

}



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

**// ClassNotFoundException**

package Ques3;

public class Ques3 {

public static void main(String[] args) {

try

{

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

}catch (ClassNotFoundException e)

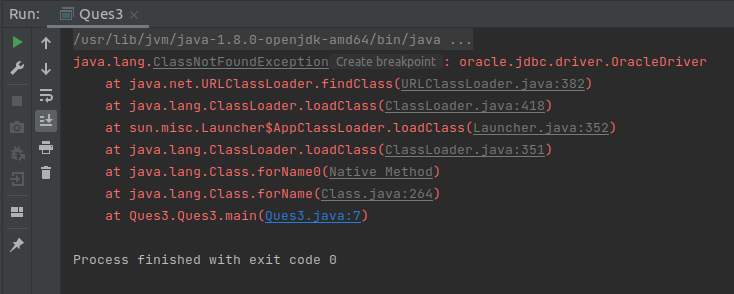
{

e.printStackTrace();

}

}

}



**// NoClassDefFoundError**

package Ques3;

class Test{

}

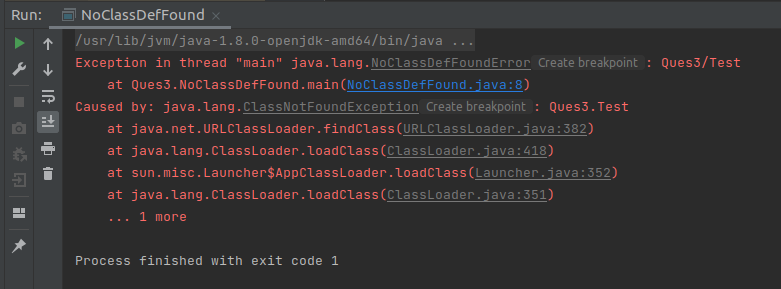
public class NoClassDefFound {

public static void main(String[] args) {

Test test = new Test();

}

}



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

public class Ques4 {

public static void main(String[] args) {

AppConfig obj1 = AppConfig.getInstance();

AppConfig obj2 = AppConfig.getInstance();

if(obj1 == obj2)

System.out.println("Both objects are same");

}

}

class AppConfig{

private AppConfig(){}

private static AppConfig obj = null;

public static AppConfig getInstance(){

if(obj == null){

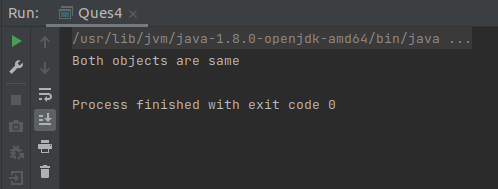
obj = new AppConfig();

}

return obj;

}

}

****

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

package Ques5;

class Employee{

private int id;

private String name;

public Employee(){}

public Employee(int id, String name){

this.id = id;

this.name = name;

}

public Employee(Employee emp){

this.id = emp.id;

this.name = emp.name;

}

public void setId(int id) {

this.id = id;

}

public void setName(String name) {

this.name = name;

}

public int getId() {

return id;

}

public String getName() {

return name;

}

@Override

public String toString() {

return "Employee{" +

"id=" + id +

", name='" + name + '\'' +

'}';

}

}

public class CopyConstructor {

public static void main(String[] args) {

Employee emp1 = new Employee(1,"vaishali");

System.out.println(emp1);

Employee emp2 = new Employee(emp1);

System.out.println("After Copying: " + emp2);

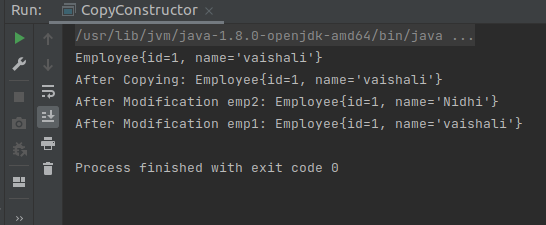
emp2.setName("Nidhi");

System.out.println("After Modification emp2: " + emp2);

System.out.println("After Modification emp1: " + emp1);

}

}



**// By Cloning**

package Ques5;

class Emp implements Cloneable{

private int id;

private String name;

public Emp(){}

public Emp(int id, String name){

this.id = id;

this.name = name;

}

public Emp(Emp emp){

this.id = emp.id;

this.name = emp.name;

}

public void setId(int id) {

this.id = id;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "Emp{" +

"id=" + id +

", name='" + name + '\'' +

'}';

}

public Object clone()throws CloneNotSupportedException{

return super.clone();

}

}

public class ObjectClone {

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

Emp emp1 = new Emp(1,"vaishali");

System.out.println("emp1 obj= " + emp1);

Emp emp2 = (Emp)emp1.clone();

System.out.println("After Copying From emp1: " + emp2);

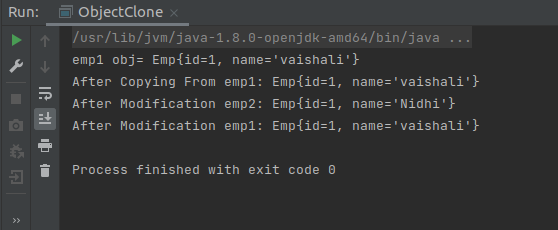
emp2.setName("Nidhi");

System.out.println("After Modification emp2: " + emp2);

System.out.println("After Modification emp1: " + emp1);

}

}

****

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

public class Ques6 {

public static void main(String[] args) {

try{

int[] a = new int[5];

int val = a[5]/0;

}catch(ArithmeticException e){

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

}catch(IndexOutOfBoundsException e){

System.out.println("Index Out Of Bound Exception occurs");

}catch(Exception e){

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

}

finally{

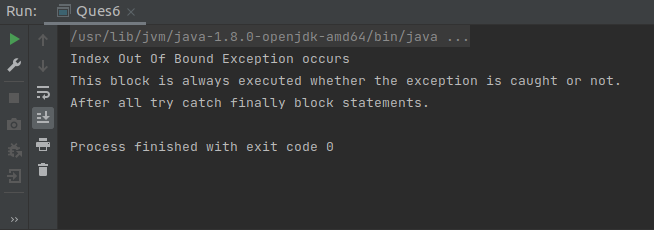
System.out.println("This block is always executed whether the exception is caught or not.");

}

System.out.println("After all try catch finally block statements.");

}

}



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

import java.util.Scanner;

public class Ques7 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Please Enter seconds");

int seconds = sc.nextInt();

SectoDayHourMinutes(seconds);

}

public static void SectoDayHourMinutes(int sec){

System.out.println("seconds " + sec);

int minutes = sec/60;

System.out.println("Minutes " + minutes);

int hours = minutes/60;

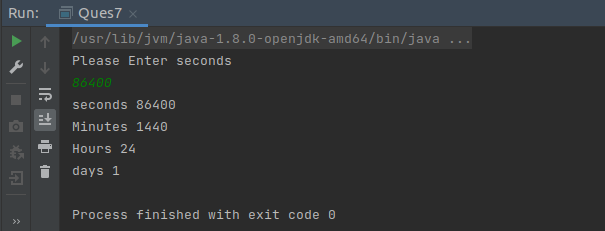
System.out.println("Hours " + hours);

int day = hours/24;

System.out.println("days " + day);

}

}



**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**

**a)using while statement**

package Ques8;

import java.util.Scanner;

public class WhileLoop {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter A Word");

String word = sc.next();

while(!word.equals("done")){

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

System.out.println("First Character is equal to its Last Character");

else

System.out.println("First character is not equal to its last character");

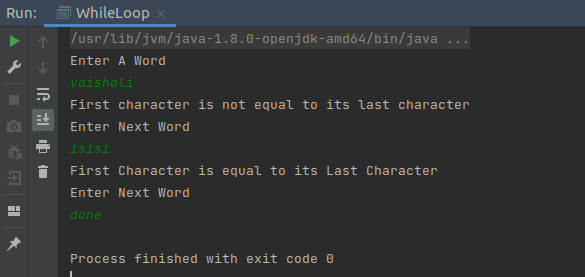
System.out.println("Enter Next Word");

word = sc.next();

}

}

}



**b) do while statement**

package Ques8;

import java.util.Scanner;

public class doWhile {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter A Word");

String word = sc.next();

do{

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

System.out.println("First Character is equal to its Last Character");

else

System.out.println("First character is not equal to its last character");

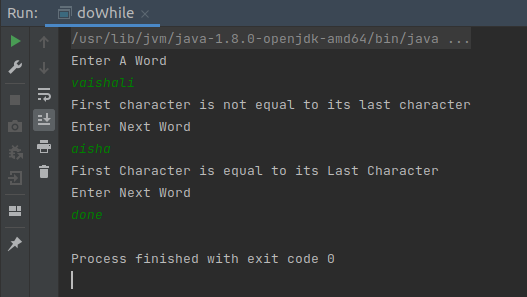
System.out.println("Enter Next Word");

word = sc.next();

}while(!word.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;

public class Ques9 {

public static void main(String[] args) {

Table table = null;

Scanner sc = new Scanner(System.in);

System.out.println("Enter type of Table [ wooden or metal ]: ");

String str = sc.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();

}

}

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() {

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

}

@Override

public void fireTest() {

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

}

@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() {

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

}

@Override

public void fireTest() {

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

}

@Override

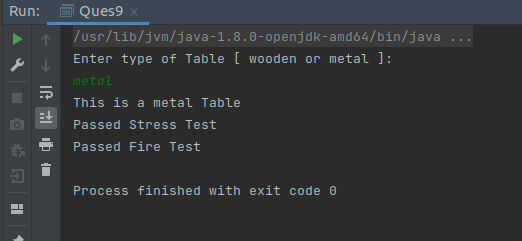
public String chairType() {

String s = "This is a wooden Chair";

return s;

}

}

****

**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**

package ques10;

class CoffeeShop {

public void menu(){};

public void coffeeShopAddress(){};

public void upcomingOffers(){};

}

class Customer extends CoffeeShop{

int cid;

String name;

public Customer(){

System.out.println("customer created");

}

public void pay(){};

public void orderToken(){};

public void orderServed(){};

}

class Cashier extends CoffeeShop{

int cid;

String name;

public Cashier(){

System.out.println("Cashier created");

}

public void collectPayment(){};

public void placeOrder(){};

public void generatePaymentToken(){};

}

class Barista extends CoffeeShop{

int cid;

String name;

public Barista(){

System.out.println("Barista created");

}

public void getNextOrder(){};

public void prepareCoffee(){};

public void orderCompleted(){};

public void generateOrderToken(){};

}

public class Ques10 {

public static void main(String[] args) {

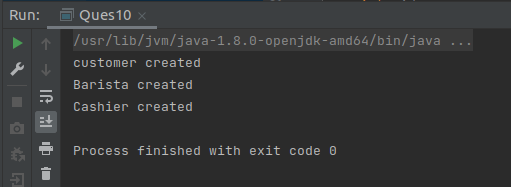
Customer cs = new Customer();

Barista bs = new Barista();

Cashier cas = new Cashier();

}

}

****

**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);**

**//nested while statement**

public class Ques11 {

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--;

}

s = s \* t;

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

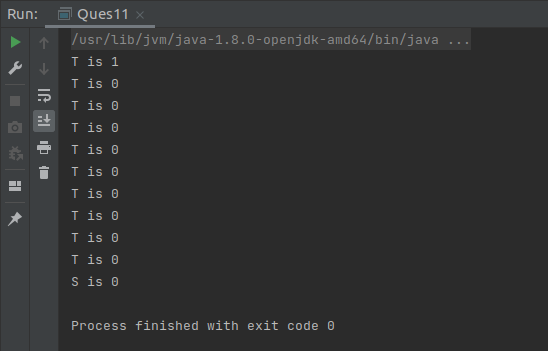
i++;

}

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");**

**}**

**}**

public class Ques12 {

public static void main(String[] args) {

Child child = 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");

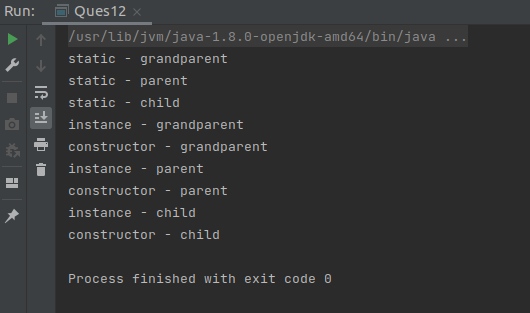
}

{

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

}

}

****

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

public class Ques13 {

public static void main(String[] args) {

try{

throw new MyException("This is my custom exception");

}catch(MyException e){

e.printStackTrace();

}

}

}

class MyException extends Exception{

private String message;

public MyException(String message){

super(message,null,false,false);

this.message = message;

}

@Override

public String getMessage() {

return message;

}

@Override

public String toString() {

return "MyException{" +

"message='" + message + '\'' +

'}';

}

}

