

JAVA EXERCISE 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.

```
public enum BookFormat {
```

```
    HARDCOVER,
```

```
    PAPERBACK,
```

```
    AUDIO_BOOK,
```

```
    EBOOK,
```

```
    NEWSPAPER,
```

```
    MAGAZINE,
```

```
    JOURNAL
```

```
}
```

```
public enum BookStatus {
```

```
    AVAILABLE,
```

```
    RESERVED,
```

```
    LOANED,
```

```
    LOST
```

```
}
```

```
public enum ReservationStatus{
```

```
    WAITING,
```

```
    PENDING,
```

```
    CANCELED,  
  
    NONE  
}
```

```
public enum AccountStatus{  
  
    ACTIVE,  
  
    CLOSED,  
  
    CANCELED,  
  
    BLACKLISTED,  
  
    NONE  
}
```

```
public class Address {  
  
    private String streetAddress;  
  
    private String city;  
  
    private String state;  
  
    private String zipCode;  
  
    private String country;  
  
}
```

```
public class Person {  
  
    private String name;  
  
    private Address address;  
  
    private String email;
```

```
private String phone;  
}
```

```
public class Constants {  
  
    public static final int MAX_BOOKS_ISSUED_TO_A_USER = 5;  
  
    public static final int MAX_LENDING_DAYS = 10;  
  
}
```

// For simplicity, we are not defining getter and setter functions. The reader can

// assume that all class attributes are private and accessed through their respective

// public getter methods and modified only through their public methods function.

```
public abstract class Account {  
  
    private String id;  
  
    private String password;  
  
    private AccountStatus status;  
  
    private Person person;  
  
  
    public boolean resetPassword();  
  
}
```

```
public class Librarian extends Account {  
  
    public boolean addBookItem(BookItem bookItem);  
  
  
  
    public boolean blockMember(Member member);  
  
}
```

```
    public boolean unBlockMember(Member member);  
}
```

```
public class Member extends Account {  
  
    private Date dateOfMembership;  
  
    private int totalBooksCheckedout;  
  
    public int getTotalBooksCheckedout();  
  
    public boolean reserveBookItem(BookItem bookItem);  
  
    private void incrementTotalBooksCheckedout();  
  
    public boolean checkoutBookItem(BookItem bookItem) {  
  
    }  
  
    private void checkForFine(String bookItemBarcode) {  
  
    }  
  
    public void returnBookItem(BookItem bookItem) {  
  
    }  
}
```

```
public bool renewBookItem(BookItem bookItem) {

}

}

public class BookReservation {

    private Date creationDate;

    private ReservationStatus status;

    private String bookItemBarcode;

    private String memberId;

    public static BookReservation fetchReservationDetails(String barcode);

}

public class BookLending {

    private Date creationDate;

    private Date dueDate;

    private Date returnDate;

    private String bookItemBarcode;

    private String memberId;

    public static void lendBook(String barcode, String memberId);

    public static BookLending fetchLendingDetails(String barcode);

}
```

```
public class Fine {  
  
    private Date creationDate;  
  
    private double bookItemBarcode;  
  
    private String memberId;  
  
  
    public static void collectFine(String memberId, long days) {}  
  
}
```

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

```
import java.util.Scanner;  
  
public class sortwithoutmeth {  
  
    public static void main(String[] args){  
  
        Scanner sc = new Scanner(System.in);  
  
        System.out.println("Enter the string: ");  
  
        String str = sc.nextLine();  
  
        int j=0;  
  
        char temp=0;  
  
        char[] chars = str.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];

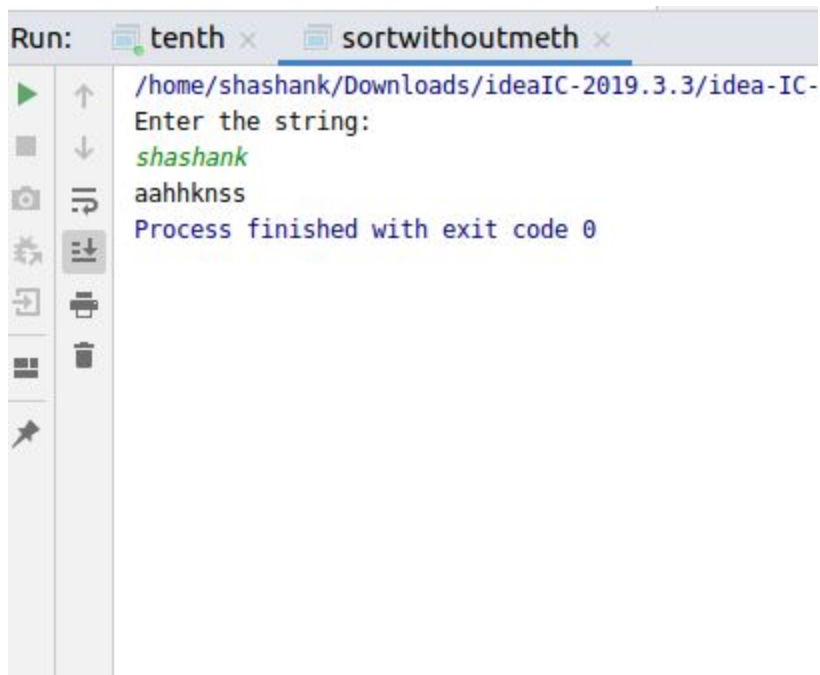
        chars[j] = temp;    }    }

    }

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

        System.out.print(chars[k]);    }    }

```



```

Run: tenth x sortwithoutmeth x
/home/shashank/Downloads/ideaIC-2019.3.3/idea-IC-
Enter the string:
shashank
aahhknss
Process finished with exit code 0

```

3. WAP to produce NoClassDefFoundError and ClassNotFoundException exception.

```

public class classnotfoundexception {

    public static void main(String args[]) {

        try

        {

            Class.forName("shashank singh");

```

```

    }

    catch (ClassNotFoundException ex)

    {

        ex.printStackTrace();

    }

}

}

```



The screenshot shows the 'Run' console of an IDE with two tabs: 'tenth' and 'noclassdeffounderr'. The active tab 'noclassdeffounderr' displays the following output:

```

/home/shashank/Downloads/ideaIC-2019.3.3/idea-IC-193.6494.35/jbr/bin/java -javaage
java.lang.ClassNotFoundException: shashank.singh <2 internal calls>
    at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:521)
    at java.base/java.lang.Class.forName0(Native Method)
    at java.base/java.lang.Class.forName(Class.java:315)
    at noclassdeffounderr.main(noclassdeffounderr.java:5)

Process finished with exit code 0

```

```

class noclassdeffounderr

```

```

{

    void greet()

    {

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

```



```

    }
}

class greeting {

    public static void main(String args[])

    {

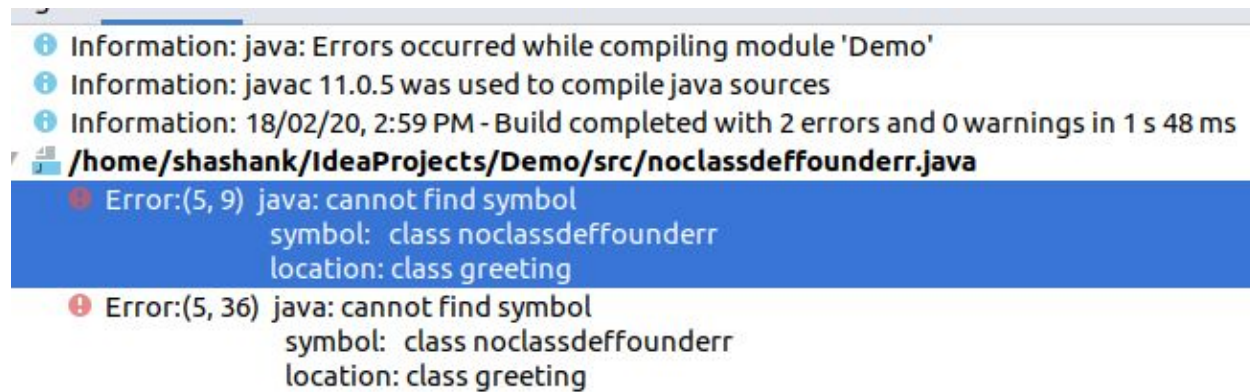
        noclassdeffounderr g = new noclassdeffounderr();

        g.greet();

    }

}

```



4. WAP to create singleton class.

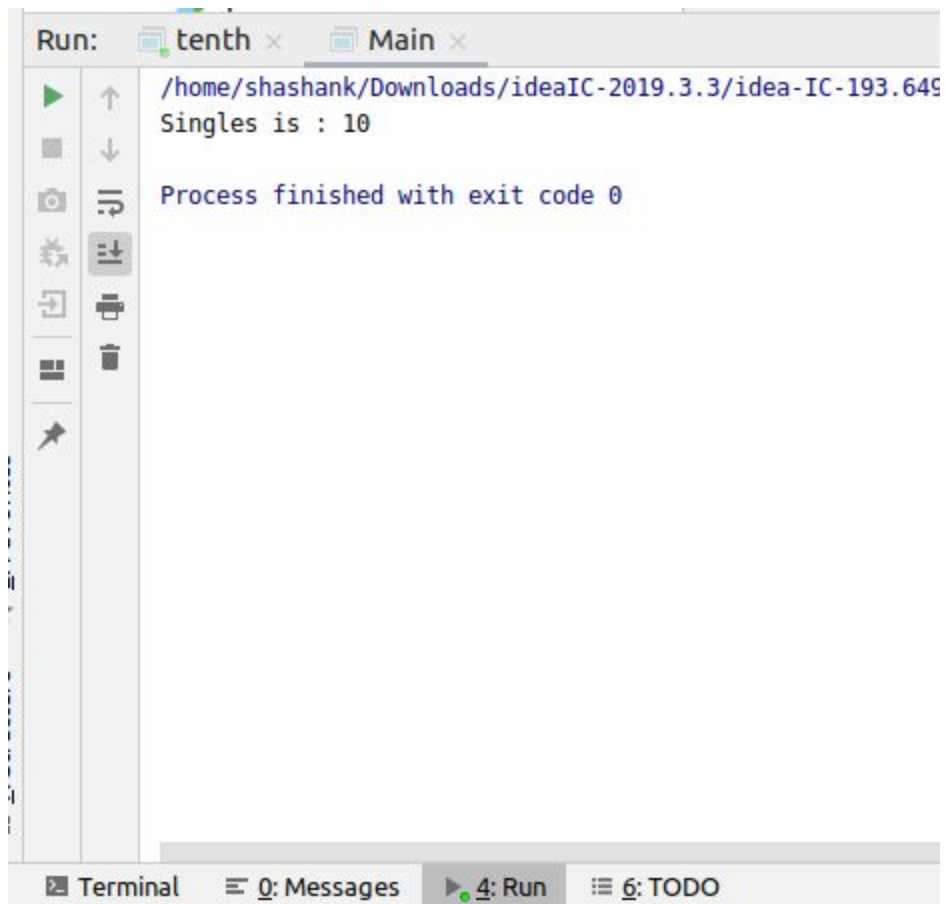
```
public class fourth {  
  
    private static fourth Single_Instance = null;  
  
    int i;  
  
    private fourth()  
  
    {  
  
        i=10;  
  
    }  
  
    public static fourth getInstance()  
  
    {  
  
        if(Single_Instance == null)  
  
            Single_Instance = new fourth();  
  
        return Single_Instance;  
  
    }  
  
}
```

```
class Main  
  
{  
  
    public static void main(String[] args) {  
  
        fourth Singles = fourth.getInstance();
```

```

        System.out.println("Singles is : "+Singles.i);
    }
}

```



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

```

class Employee implements Cloneable

```

```

{

```

```

    String name;

```

```

    int age;

```

```

Employee(String Name , int Age)

{

    this.name = Name;

    this.age = Age;

}

public String toString()

{

    return "name" + name + "\n" + "age" + age;

}

public Employee clone () throws CloneNotSupportedException {

    return (Employee) super.clone();

}

}

public class five {

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

        Employee e = new Employee("shashank" , 20);

        Employee e2 = e.clone();

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

        System.out.println(e2.hashCode());

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

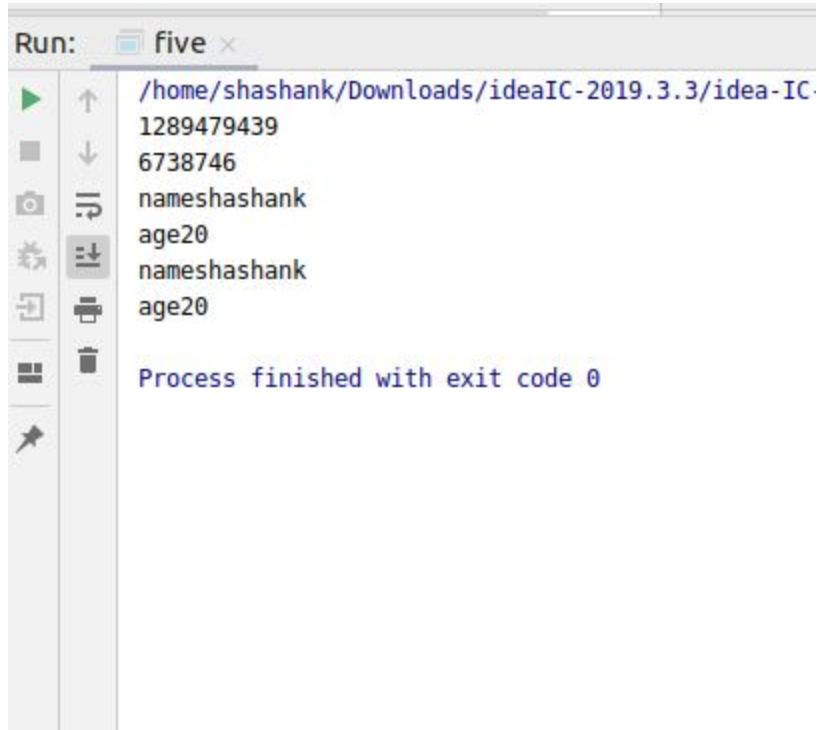
        System.out.println(e2.toString());

    }

}

```

```
}
```



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

```
public class six {  
  
    public static void main(String args[]){  
  
        try{  
  
            int arr[]=new int[7];  
  
            arr[4]=30/0;  
  
            System.out.println("Last Statement of try block");  
  
        }  
  
        catch(ArithmeticException e){  
  
            System.out.println("You should not divide a number by zero");  
  
        }  
    }  
}
```

```
}

catch(ArrayIndexOutOfBoundsException e){

    System.out.println("Accessing array elements outside of the limit");

}

catch(Exception e){

    System.out.println("Some Other Exception");

}

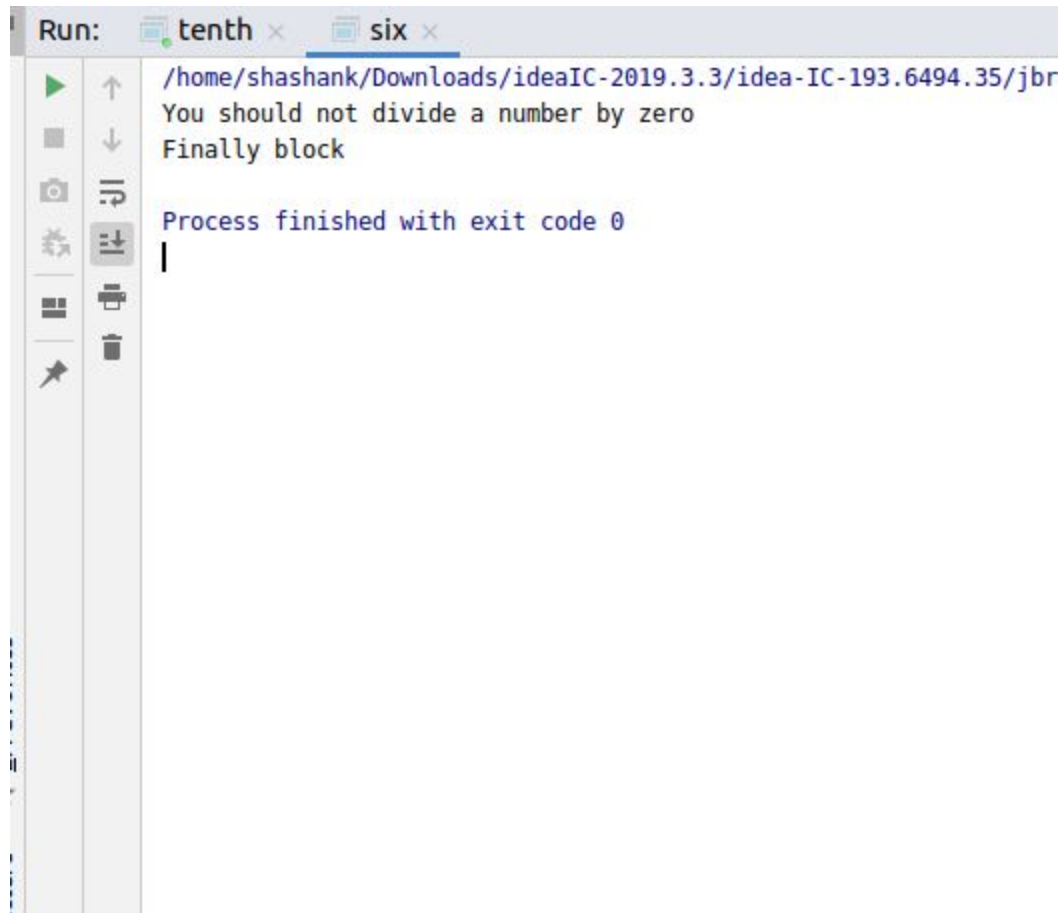
finally

{

    System.out.println("Finally block");

}

}
```



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

```
import java.util.Scanner;
```

```
public class seven {
```

```
    static void Convert(int n)
```

```
    {
```

```
        int day = n / (24 * 3600);
```

```
        n = n % (24 * 3600);
```

```
int hour = n / 3600;
```

```
n %= 3600;
```

```
int minutes = n / 60 ;
```

```
n %= 60;
```

```
int seconds = n;
```

```
System.out.println( day + " " + "days " + hour  
    + " " + "hours " + minutes + " "  
    + "minutes " + seconds + " "  
    + "seconds ");  
}
```

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

```
{
```

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

```
System.out.println("Enter the second to convert : ");
```

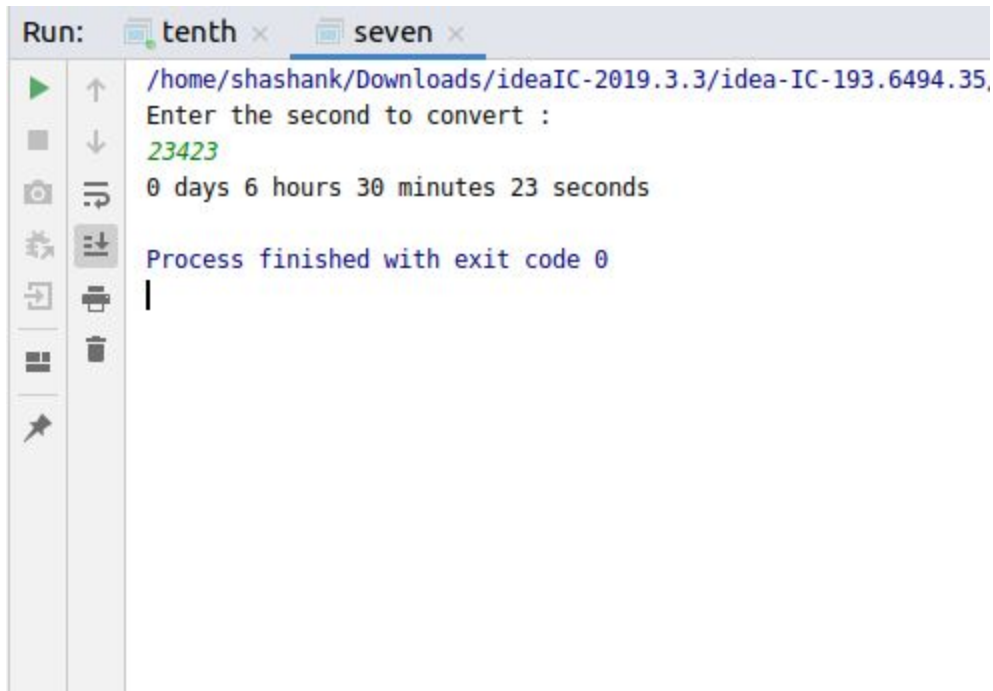
```
int n = sc.nextInt();
```

```
Convert(n);
```



```
}
```

```
}
```



```
Run: tenth x seven x
/home/shashank/Downloads/ideaIC-2019.3.3/idea-IC-193.6494.35,
Enter the second to convert :
23423
0 days 6 hours 30 minutes 23 seconds
Process finished with exit code 0
|
```

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

```
import java.util.Scanner;
```

```
public class eight{
```

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

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

```
        System.out.println("Enter a word");
```

```
        String word = keyboard.next();
```

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

```
{
```

```

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

{

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

} else

{

    System.out.println("First and last character are NOT equals for the word: " + word);

}

word = keyboard.next();

}

}

}

```

```

Run: mten x eight x
/home/shashank/Downloads/ideaIC-2019.3.3/idea-IC-193.6494.35/jbr/bin/java -
Enter a word
nirjan sir is my mentor done
First and last character are equals for the word: nirjan
First and last character are NOT equals for the word: sir
First and last character are NOT equals for the word: is
First and last character are NOT equals for the word: my
First and last character are NOT equals for the word: mentor
Process finished with exit code 0
|

```

b)do-while statement

```
import java.util.Scanner;
```

```
public class eight{

    public static void main(String[] args) {

        Scanner keyboard = new Scanner(System.in);

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

        String word = keyboard.next();

        do {

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

            {

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

            }

            else

            {

                System.out.println("First and last character are NOT equals for the word: " + word);

            }

            word = keyboard.next();

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

    }

}
```

```
Run: mten x eight x
/home/shashank/Downloads/ideaIC-2019.3.3/idea-IC-193.6494.35/jbr/bin/java -
Enter a word
nirjan sir is my mentor done
First and last character are equals for the word: nirjan
First and last character are NOT equals for the word: sir
First and last character are NOT equals for the word: is
First and last character are NOT equals for the word: my
First and last character are NOT equals for the word: mentor

Process finished with exit code 0
|
```

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 interface Furniture {

    public void stressTest();

    public void fireTest();

}

public abstract class Chair implements Furniture {

    public abstract String chairType();

}

public abstract class Table implements Furniture {

    public abstract String tableType();

}
```

```
}
```

```
public class MetalChair extends Chair {
```

```
    public void stressTest() {
```

```
    }
```

```
    public void fireTest() {
```

```
    }
```

```
}
```

```
public class MetalTable extends Table {
```

```
    public void stressTest() {
```

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

```
    }
```

```
    public void fireTest() {
```

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

```
    }
```

```
    public String tableType() {
```

```
        String s = "This is a metal Table";
```

```
        return s;
```

```
    }
```

```
}
```

```
public class WoodenTable extends Table {

    public void stressTest() {

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

    }

    public void fireTest() {

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

    }


    public String tableType() {

        String s = "This is a wooden Table";

        return s;

    }

}

public class WoodenChair extends Chair {

    public void stressTest() {

    }

    public void fireTest() {

    }

}
```

```
public class nine {

    public static void main(String[] args){

        Table table = null;
```

```

Scanner input = new Scanner(System.in);

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

```
public class Main {  
}
```

```
public class Barista implements QueueOfPendingOrder, QueueOfCompletedOrder {
```

```
    String name;
```

```
    QueueOfPendingOrder queueOfPendingOrder;
```

```
    QueueOfCompletedOrder queueOfCompletedOrder;
```

```
    @Override
```

```
    public void remove(Order order) {
```

```
        queueOfPendingOrder.remove(order);
```

```
        System.out.println(order + "removed from Pending queue");
```

```
    }
```



```

public void makeCoffee(Order order){
    System.out.println("Making coffee for " + order);
}

public void notifyAboutCompletedOrder(Customer customer,Order order) {
    System.out.println(customer + "your order " + order + " has been completed");
}

@Override
public void addToCompleteOrderQueue(Order order) {
    queueOfCompletedOrder.addToCompleteOrderQueue(order);
    System.out.println(order + " added to Completed queue");
}
}

public class Cashier extends Order implements QueueOfPendingOrder {
    String name;
    QueueOfPendingOrder queueOfPendingOrder;
    List<Customer> customerList;

    public Cashier(String name,Long id) {
        super(id);
        this.name = name;
        customerList = new ArrayList<>();
    }

    String AcceptOrderAndAddCustomerToCustomerList(Customer customer,Order order,double cash) {
        customerList.add(customer);
        System.out.println("Accepted order");

        return "token";
    }

    void addOrderInOrderQueue(Order order){
        queueOfPendingOrder.add(order);
        System.out.println(order + " added to order queue");
    }
}

public class Customer {
    private String name;
    private String token;
    Cashier cashier;
    Order order;
    double amount;

    void placeOrder() {
        token = cashier.AcceptOrderAndAddCustomerToCustomerList(this,order,amount);
        System.out.println("This is the order token: " + token);
    }

    boolean waitingState(){

```

```

        System.out.println("Customer" + this.name + "is waiting");
        return true;
    }

    boolean drinkingState() {
        System.out.println("Customer " + this.name + " has collected coffee");
        return true;
    }
}

public class Order {
    private Long id;

    public Order(Long id) {
        this.id = id;
    }
}

public interface QueueOfPendingOrder {
    Queue<Order> queue = new PriorityQueue<>();
    default void add(Order order){
        queue.add(order);
    }
    default void remove(Order order) {
        queue.remove(order);
    }
}

interface QueueOfCompletedOrder {
    default void addToCompleteOrderQueue(Order order){
        Queue<Order> queue = new PriorityQueue<>();
    }
}

```

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 seceleven {  
  
    public static void main(String [] args)  
  
    {  
  
        int i=0,j=0;  
  
        int s = 0;  
  
        int t = 1;  
  
        while(i<10)  
  
        {  
  
            s = s + i;  
  
            j=i;  
  
            while(j>0)  
  
            {  
  
                t = t * (j - i);  
  
                j=j-1;  
  
            }  
  
            s = s * t;  
  
            System.out.println("T is " + t);  
  
            i=i+1;  
  
        }  
  
    }  
  
}
```

```

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

```

```

Run: tenth x seceleven x
/home/shashank/Downloads/ideaIC-2019.3.3/idea-IC-193.1
T is 1
T is 0
T is 0
T is 0
T is 0
T is 0
T is 0
T is 0
T is 0
T is 0
S is 0

Process finished with exit code 0

```

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 twelve {  
  
    public static void main(String[] args)  
  
    {  
  
        Child obj1=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");
```

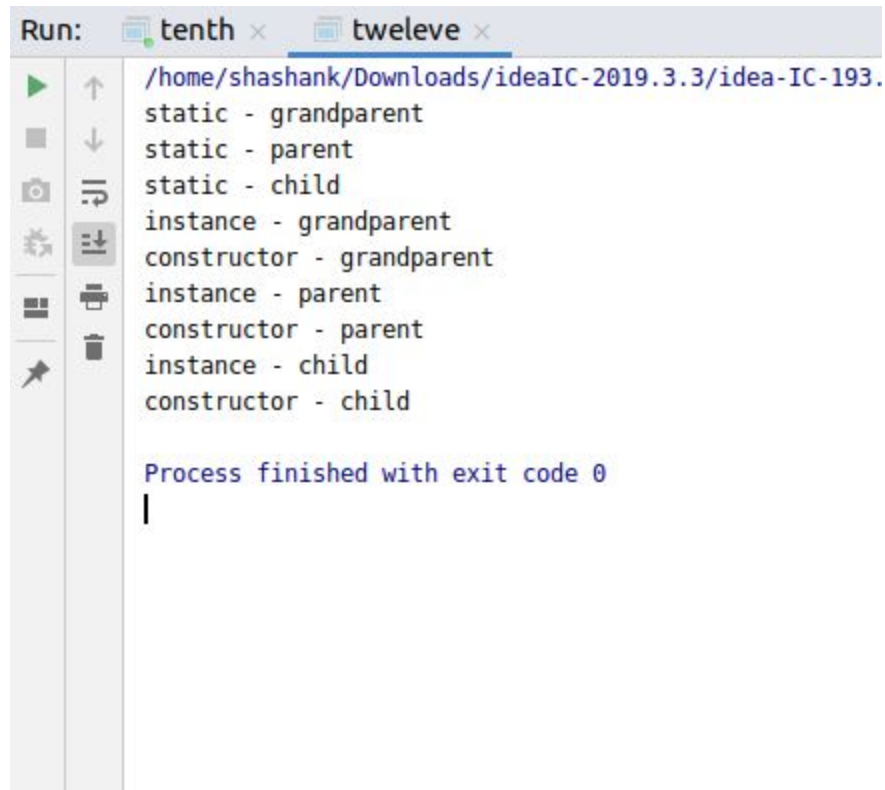
```
    }
```

```
{
```

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

```
}
```

```
}
```



```
Run: tenth x tweleve x
/home/shashank/Downloads/ideaIC-2019.3.3/idea-IC-193.
static - grandparent
static - parent
static - child
instance - grandparent
constructor - grandparent
instance - parent
constructor - parent
instance - child
constructor - child

Process finished with exit code 0
|
```

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

```
class InvalidAgeException extends Exception{
    InvalidAgeException(String s){
        super(s);
    }
}

public class thirteen {
    static void validate(int age)throws InvalidAgeException{
        if(age<18)
            throw new InvalidAgeException("not valid");
        else
            System.out.println("welcome to vote");
    }

    public static void main(String args[]){
        try{
            validate(13);
        }catch(Exception m){System.out.println("Exception occured: "+m);}

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