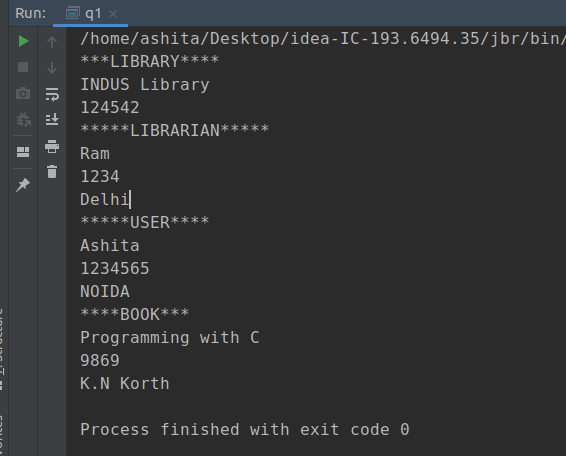
**Exercise**

**Note: Use Java keywords and concepts in problem solving.**

**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.**

****

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

**public class q2 {**

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

**String str="Ashita";**

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

**char temp;**

**for(int character=0;character<str.length();character++){**

**for(int nxt=0;nxt<str.length();nxt++){**

**if(ch[character]<ch[nxt]){**

**temp=ch[character];**

**ch[character]=ch[nxt];**

**ch[nxt]=temp;**

**}**

**}**

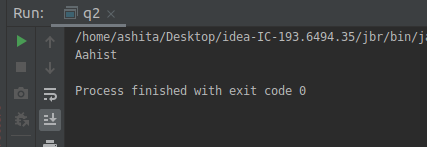
**}**

**String sortarr=String.*valueOf*(ch);**

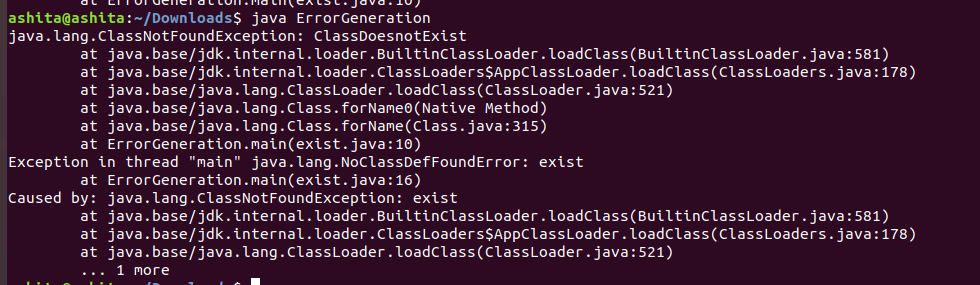
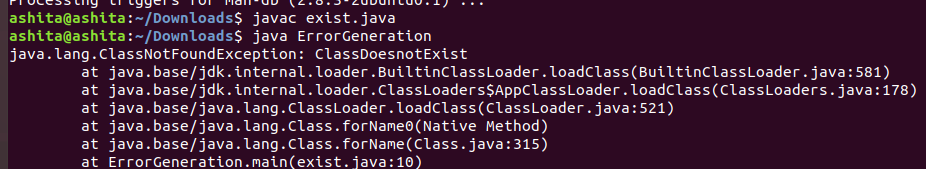
**System.*out*.println(sortarr);**

**}**

**}**

****

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

****

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

**package ashita.java2;**

**public class q4 {**

**private static q4 *myObj*;**

**static{**

***myObj* = new q4();**

**}**

**private q4(){**

**}**

**public static q4 getInstance(){**

**return *myObj*;**

**}**

**public void testMe(){**

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

**}**

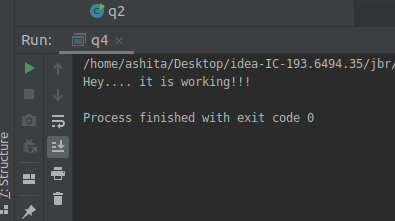
**public static void main(String a[]){**

**q4 ms = *getInstance*();**

**ms.testMe();**

**}**

**}**

****

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

**package ashita.java2;**

**class CopyContructer{**

**int rollno;**

**String name;**

**CopyContructer(int rollno,String name)**

**{**

**this.rollno=rollno;**

**this.name=name;**

**}**

**CopyContructer(CopyContructer obj)**

**{**

**System.*out*.println("Copy Constructor Invoked");**

**rollno = obj.rollno;**

**name= obj.name;**

**}**

**}**

**public class q5 implements Cloneable {**

**int rollno;**

**String name;**

**q5(int rollno, String name){**

**this.rollno=rollno;**

**this.name=name;**

**}**

**public Object clone()throws CloneNotSupportedException{**

**return super.clone();**

**}**

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

**try{**

**q5 s1=new q5(101,"ashita");**

**q5 s2=(q5) s1.clone();**

**System.*out*.println(s1.rollno+" "+s1.name);**

**System.*out*.println(s2.rollno+" "+s2.name);**

**CopyContructer cp=new CopyContructer(121,"smriti");**

**CopyContructer cp2=new CopyContructer(cp);**

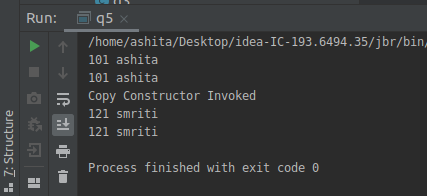
**System.*out*.println(cp.rollno+" "+cp.name);**

**System.*out*.println(cp2.rollno+" "+cp2.name);**

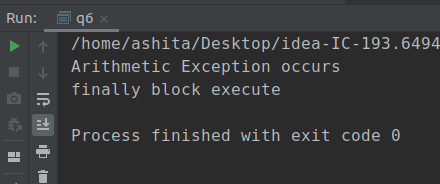
**}catch(CloneNotSupportedException c){}**

**}**

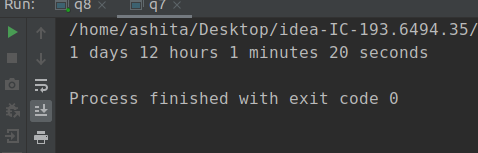
**}**

****

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

****

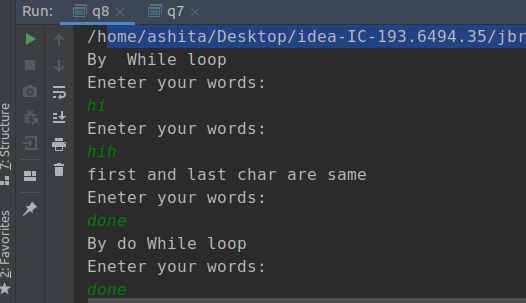
**7. WAP to convert seconds into days, hours, minutes and seconds.**

****

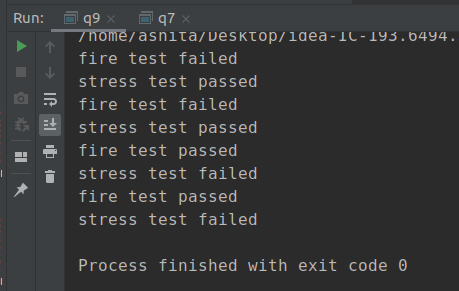
**8. WAP to read words from the keyboard until the word done is entered. For each word except done, report whether its first character is equal to its last character. For the required loop, use a**

**a)while statement**

**b)do-while statement**

****

**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.**

****

**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**

**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);**

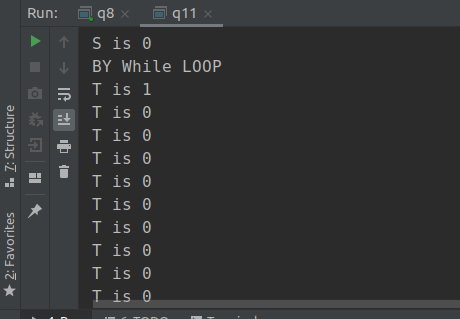
**}**

**s = s \* t;**

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

**}**

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

****

**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");**

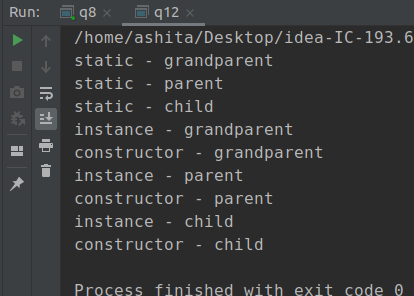
**}**

**{**

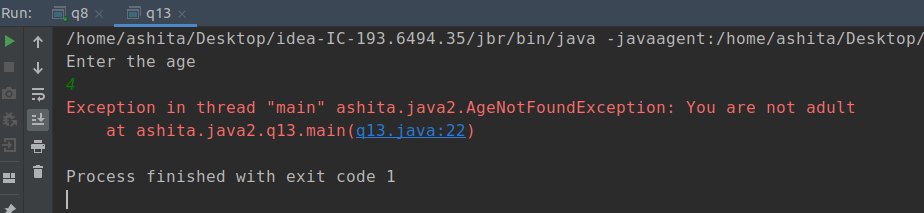
**System.out.println("instance - child");**

**}**

**}**

****

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

****