**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 daytwo;**

**import java.util.\*;**

**abstract class Account {**

**private String id;**

**private String password;**

**}**

**class Librarian extends Account {**

**}**

**class Member extends Account {**

**private Date dateOfMembership;**

**private int totalBooksCheckedout;**

**}**

**interface issue{**

**void issueBooks();**

**void alreadyIssuedBooks();**

**}**

**interface return\_book{**

**void haveToReturn();**

**void returned\_Books();**

**void total\_Balance();**

**}**

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

**@Override**

**public void issueBooks() { }**

**@Override**

**public void alreadyIssuedBooks() {}**

**@Override**

**public void haveToReturn() {}**

**@Override**

**public void returned\_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 Q1LibraryManagementSystem extends Library{**

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

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

**l.DisplayBooks();**

**l.issueBooks();**

**}**

**}**

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

**package daytwo;**

**public class SortingString {**

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

**String str = "ddecaf";**

**System.*out*.println("Original String: "+str);**

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

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

**for (int j = i+1; j < ch.length-1; j++) {**

**if (ch[i] > ch[j]) {**

**char temp = ch[i];**

**ch[i] = ch[j];**

**ch[j] = temp;**

**}**

**}**

**System.*out*.print("Sorted String: "+ch[i]);**

**}**

**}**

**}**

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

Exceptions

package daytwo;

class school{

void hi(){

System.*out*.println("Hi. I can still run");

}

}

public class Exceptions {

public static void main(String[] args) {

try{

Class.*forName*("java.java");

}

catch(ClassNotFoundException e){

e.printStackTrace();;

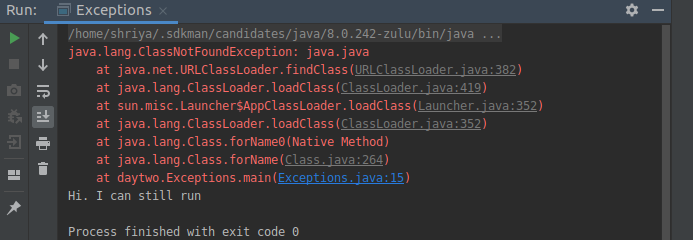
}

school emp1 =new school();

emp1.hi();

}

}



Error

package daytwo;

class excep{

void print(){}

}

class ErrorGen{

public static void main(String[] args) {

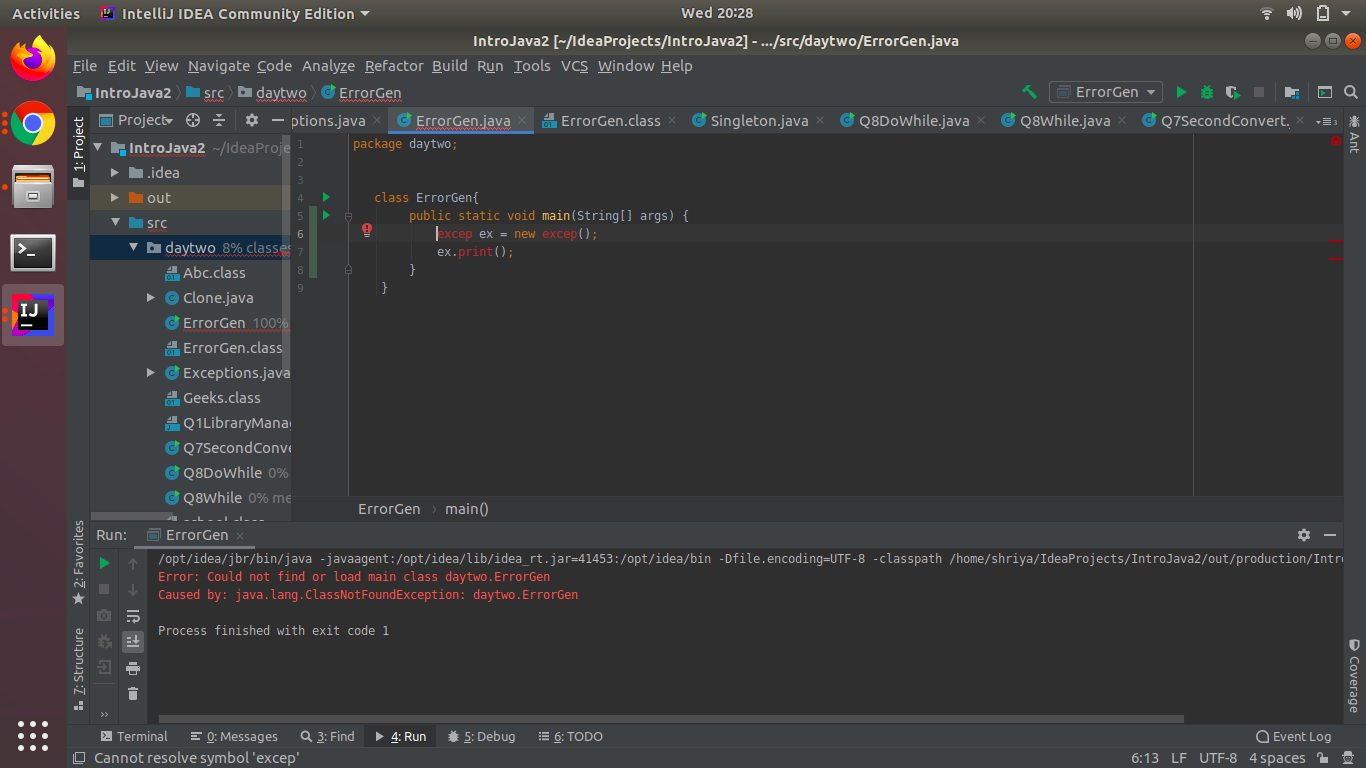
excep ex = new excep();

ex.print();

}

}

P.S: Remove class names excep. Error was visible on Java 11 complier, somehow not on 8th version



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

**package daytwo;**

**public class SortingString {**

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

**String str = "ddecaf";**

**System.*out*.println("Original String: "+str);**

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

**System.*out*.print("Sorted String : ");**

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

**for (int j = i+1; j < ch.length-1; j++) {**

**if (ch[i] > ch[j]) {**

**char temp = ch[i];**

**ch[i] = ch[j];**

**ch[j] = temp;**

**}**

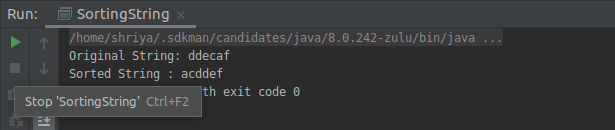
**}**

**System.*out*.print(ch[i]);**

**}**

**}**

**}**

****

**P.S : The java file uploaded on git had some error in print line. I corrected it in this document and pasted correct output.**

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

**package daytwo;**

**public class Clone {**

**int a,b,c;**

**public Clone(int a, int b, int c) {**

**this.a = a;**

**this.b = b;**

**this.c = c;**

**}**

**Clone(Clone c){**

**a=c.a;**

**b=c.b;**

**}**

**}**

**class copy{**

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

**Clone c= new Clone(1,2,3);**

**Clone c1= new Clone(c);**

**System.*out*.println("Copy of object c in c1 is : "+c1.a+" "+c1.b+ " "+ c.b);**

**}**

**}**

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

**package daytwo;**

**public class TryCatch {**

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

**int x=10,y=20,z=30;**

**try{**

**if(1==0){**

**System.*out*.println("x is 10");**

**}**

**int b=x/0;**

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

**}**

**catch (NumberFormatException e){**

**System.*out*.println("NumberFormatException has occured");**

**}**

**catch(ArithmeticException e){**

**System.*out*.println("ArithmeticException has occured");**

**}**

**finally{**

**int a=x+y+z;**

**System.*out*.println("Using finally : "+a);**

**}**

**}**

**}**

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

**package daytwo;**

**import java.util.Scanner;**

**public class Q7SecondConvert {**

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

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

**System.*out*.println("Enter seconds you want to convert into days : ");**

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

**int d=input/(60\*60\*24);**

**int dh=(input)%(60\*60\*24);**

**int h=dh/(60\*60);**

**int hm=dh%(60\*60);**

**int s=hm/(60);**

**int ss=hm%(60);**

**System.*out*.println("Converted time = "+d+"days "+h+"hours "+s+"minutes and "+ss+"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**

**package daytwo;**

**import java.util.Scanner;**

**public class Q8While {**

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

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

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

**String entry=sc.next();**

**while(!entry.equals("done|DONE|Done")){**

**char ch1=Character.*toLowerCase*(entry.charAt(0));**

**char ch2=Character.*toLowerCase*(entry.charAt(entry.length()-1));**

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

**System.*out*.println("SAME first and last char in : "+entry);**

**}**

**else{**

**System.*out*.println("DIFFERENT first and last char in : "+entry);**

**}**

**entry=sc.next();**

**}**

**}**

**}**

**b)do-while statement**

**package daytwo;**

**import java.util.Scanner;**

**public class Q8DoWhile {**

**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= Character.*toLowerCase*(word.charAt(0));**

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

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

**System.*out*.println("SAME first and last char in : " +word);**

**}**

**else {**

**System.*out*.println("DIFFERENT first and last char in : " + word);**

**}**

**word=sc.next();**

**}**

**while(!word.matches("done|DONE|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.**

**package daytwo;**

**public class Q9Furniture {**

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

**Chair c= new Chair();**

**c.stress\_test\_W();**

**c.stress\_test\_M();**

**c.fire\_test\_M();**

**c.fire\_test\_W();**

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

**Table t= new Table();**

**t.stress\_test\_W();**

**t.stress\_test\_M();**

**t.fire\_test\_M();**

**t.fire\_test\_W();**

**}**

**}**

**interface wooden {**

**void stress\_test\_W();**

**void fire\_test\_W();**

**}**

**interface Metal {**

**void stress\_test\_M();**

**void fire\_test\_M();**

**}**

**class Chair extends Q9Furniture implements wooden,Metal{**

**@Override**

**public void stress\_test\_W(){**

**System.*out*.println("wooden chair has stress");**

**}**

**@Override**

**public void stress\_test\_M(){**

**System.*out*.println("wooden chair has stress");**

**}**

**@Override**

**public void fire\_test\_M(){**

**System.*out*.println("wooden chair has stress");**

**}**

**@Override**

**public void fire\_test\_W(){**

**System.*out*.println("wooden chair has stress");**

**}**

**}**

**class Table extends Q9Furniture implements wooden,Metal{**

**@Override**

**public void stress\_test\_W(){**

**System.*out*.println("wooden TABLE table has stress");**

**}**

**@Override**

**public void stress\_test\_M(){**

**System.*out*.println("wooden TABLE has FIRE");**

**}**

**@Override**

**public void fire\_test\_M(){**

**System.*out*.println("metal TABLE has stress");**

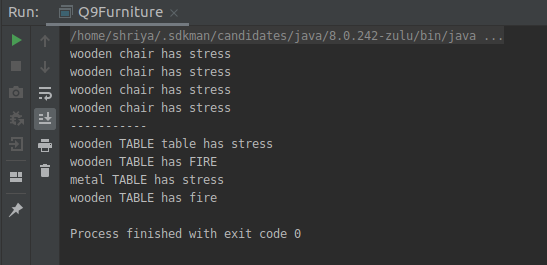
**}**

**@Override**

**public void fire\_test\_W(){**

**System.*out*.println("wooden TABLE has fire");**

**}**

**}**

**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 daytwo;**

**import java.util.ArrayList;**

**import java.util.List;**

**class Order{**

**public int OrderID;**

**public int PaymentID;**

**void OrderReady(){};**

**}**

**interface FoodCounter{**

**}**

**class Cashier extends Order{**

**public void setOrderID(int orderID) {**

**OrderID = orderID;**

**}**

**public void setPaymentID(int paymentID) {**

**PaymentID = paymentID;**

**}**

**}**

**class Customer extends Order{**

**public int getPaymentID() {**

**return PaymentID;**

**}**

**public int getOrderID() {**

**return OrderID;**

**}**

**void Displaymenu(){}**

**void OrderQueueShow(){};**

**}**

**class Barista implements FoodCounter{**

**void Displaymenu(){**

**List<String> menu=new ArrayList<String>();}**

**void OrderQueue(){};**

**void OrderReady(){};**

**}**

**public class Q10CoffeeShop extends Customer {**

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

**Customer c = new Customer();**

**c.OrderQueueShow();**

**c.Displaymenu();**

**}**

**}**

**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);**

**package daytwo;**

**public class Q11WhileInsteadofFor {**

**/\***

**Given FOR LOOP in question**

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

**int s = 0;**

**int t = 1;**

**for (int i = 0; i < 10; i++)**

**{**

**s = s + i;**

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

**for (int j = i; j > 0; j--)**

**{**

**t = t \* (j - i);**

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

**}**

**s = s \* t;**

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

**}**

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

**}**

**\*/**

**// Converted to WHILE LOOP**

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

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

**while(i<10){**

**s+=i;**

**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(); ?**

**package daytwo;**

**public class Q12Output {**

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

**Grandparent g=new Child();**

**}}**

**class Parent extends Grandparent {**

**static {**

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

**}**

**{**

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

**}**

**public Parent() {**

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

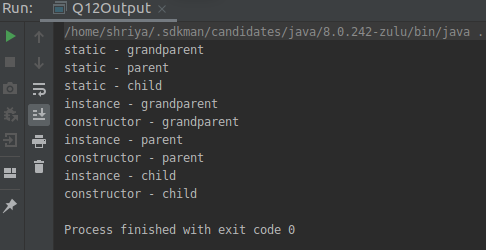
**}**

**{**

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

**}**

**}**

****

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

**package daytwo;**

**public class Q13CustomException {**

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

**try{**

**//it will throw and object of user defined exception like what i created customException**

**throw new customException("Shriya Garg");**

**}**

**catch(customException CE){**

**System.*out*.println("I caught an exception");**

**CE.printStackTrace();**

**//IT WIL SHOW WHAT HAPPEN AND WHERE**

**}**

**}**

**}**

**//To create a custom exception, we have to extend the java.lang.Exception class.**

**class customException extends Exception{**

**customException(String name){**

**super(name);**

**}**

**}**