1.Write a Java program to accept IA marks obtained by five students in three subjects. The program should accept marks obtained by each student and display the total marks and the average marks. The average marks is computed using a method as the average of best two marks obtained.

import java.util.Scanner;

public class TW1 {

public static void main(String[] args){

int[][] marks=new int[5][3];

int[] total = {0,0,0,0,0};

int[] avg = new int[5];

Scanner in = new Scanner(System.in);

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

{

System.out.println("Enter the marks of student "+(i+1));

for(int j=0; j<3; j++)

{

marks[i][j]=in.nextInt();

total[i]=total[i]+marks[i][j];

}

}

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

avg[i] = computeAvg(marks[i][0], marks[i][1],marks[i][2]);

System.out.println("Student results:");

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

System.out.println("For student " +(i+1));

System.out.println("Total marks:" +total[i]);

System.out.println("Avreage marks:" +avg[i]);

}

}

static int computeAvg(int m1,int m2,int m3){

int min = m1;

if(m2<min)

min = m2;

if(m3<min)

min = m3;

int total = (m1+m2+m3-min);

return(int)Math.ceil(total/2.0);

}

}

TW 2: Write a program to demonstrate the implementation of class and its member methods.

Design a class by name myTriangle to model a triangle geometrical object with three sides. Include functions to:

* Initialize the three sides of triangle.
* Determine the type of triangle represented by the three sides (Equilateral/ Isosceles/ Scalene triangle**).**
* Compute and return the area of the triangle.

**Note:**

When three sides are given we use the following formula:

s=(a+b+c)/2;

area=sqrt(s\*(s-a)\*(s-b)\*(s-c));

import static java.lang.System.exit;

import java.util.Scanner;

class Triangle {

double a,b,c;

void getSides(){

Scanner in=new Scanner(System.in);

System.out.println("Enter 3 sides of a triangle:");

a=in.nextDouble();

b=in.nextDouble();

c=in.nextDouble();

}

void checkTriangle(){

if((a+b)>c && (b+c)>a && (a+c)>b){

if(a==b && b==c && c==a)

System.out.println("Triangle is equilateral");

else if(a==b || b==c || c==a)

System.out.println("Triangle is isosceles");

else

System.out.println("Triangle is scalene");

}

else{

System.out.println("Triangle cannot be formed");

exit(0);

}

}

double computeArea(){

double s=(a+b+c)/2;

double area=Math.sqrt(s\*(s-a)\*(s-b)\*(s-c));

return area;

}

}

public class tw2 {

public static void main(String[] args){

Triangle t=new Triangle();

t.getSides();

t.checkTriangle();

if((t.computeArea())!=0){

System.out.println("Area is "+t.computeArea());

}

}

}

3.Write a program to demonstrate the implementation of parameterized:

1. Methods.
2. Constructor.

3.1) A certain small bank intends to automate few of its banking operations for its customers. Design a class by name mybankAccount to store the customer data having following details:

1.accountNumber 2. acctType 3. Name 4. Address 5. accountBalance

The class must have both default and parameterized constructors. Write appropriate method to compute interest accrued on accountBalance based on accountType and time in years. Assume 5% for S/B account 6.5% for RD account and 7.65 for FD account. Further, add two methods withdrawAmount/depositAmount with amount as input to withdraw and deposit respectively. The withdrawAmount method must report in-sufficient balance if accountBalance falls below Rs. 1000.

TASK 1 : Build the class with appropriate member variables, constructors and methods.

TASK 2 : Instantiate three objects of above type and perform different operations on the same.

TASK 3 : Write a function to display all the three customer details in a tabular form with appropriate column headings

import java.util.Scanner;

class BankAcc{

int accNumber;

String name, address, accType;

double accBal;

static int count = 0;

BankAcc(){

accNumber = ++count;

Scanner in = new Scanner(System.in);

System.out.println("ENTER NAME: ");

name = in.nextLine();

System.out.println("ENTER ADDRESS: ");

address = in.nextLine();

System.out.println("ENTER ACC TYPE: ");

accType = in.nextLine();

System.out.println("ENTER ACC BALANCE: ");

accBal = in.nextDouble();

}

BankAcc(String name, String address, String accType, double accBal){

accNumber = ++count;

this.name = name;

this.address = address;

this.accType = accType;

this.accBal = accBal;

}

void computeInterest(int time){

double interest;

if("sb".equals(accType)){

interest = 0.05 \* accBal \* time;

System.out.println("INTEREST EARNED for SB account is "+ interest);

}

else if ("rd".equals(accType)){

interest = 0.063 \* accBal \* time;

System.out.println("INTEREST EARNED for RD account is "+ interest);

}

else if ("fd".equals(accType)){

interest = 0.0765 \* accBal \* time;

System.out.println("INTEREST EARNED for FD account is "+ interest);

}

else

System.out.println("Invalid account type");

}

void deposit(double amount){

accBal = accBal + amount;

System.out.println("ACC BALANCE IS "+ accBal);

}

void withdraw(double amount){

if((accBal-amount)<1000)

System.out.println("INSUFFICIENT BALANCE!!!!");

else

{

accBal = accBal - amount;

System.out.println("ACC BALANCE IS "+ accBal);

}

}

}

public class TW3 {

public static void main(String[] args){

BankAcc b1 = new BankAcc();

BankAcc b2 = new BankAcc("Namitha","xyz","fd",20000);

BankAcc b3 = new BankAcc();

b1.computeInterest(1);

b2.computeInterest(1);

b3.computeInterest(1);

b1.deposit(500);

b1.withdraw(1500);

b2.deposit(10000);

b2.withdraw(1000);

b3.deposit(2000);

b3.withdraw(3000);

}

}

4.1) A company has two types of employees – FullTime and Partime. The company records for each employee his/her name, age, address, salary and gender. Given the basic salary of the FullTime employee the components of his/her gross salary are: Dearness allowance – 75% of basic salary, HRA – 7.5% of basic salary, IT – 10% of basic. The salary of a Partime employee is dependent on the qualification, experience, number of working hours and the rate per hour, as below:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Qualification | | |
| Experience | BE | MTech | Ph.D |
| 1-5 years | 300 Rs. | 500 Rs. | 800 Rs. |
| 6-10 years | 400 Rs. | 700 Rs. | 1200 Rs. |
| >10 years | 500 Rs. | 1000 Rs. | 1500 Rs. |

Model this as a problem of hierarchical inheritance by:

1) Identifying the super class with its data members and member functions.

2) Identify the sub-class/sub-classes and their associated data members and member functions.

Test the program by creating objects of the classes that are so identified.

class Employ{

String name,address,gender;

int age;

double sal;

Employ(String name,int age,String address,String gender){

this.name=name;

this.age=age;

this.address=address;

this.gender=gender;

}

void show(){

System.out.println("Name:"+name);

System.out.println("Age:"+age);

System.out.println("Address:"+address);

System.out.println("Gender:"+gender);

System.out.println("Salay:"+sal);

}

}

class FTEmploy extends Employ{

int basSal;

FTEmploy(String name,int age,String address,String gender,int basSal){

super(name,age,address,gender);

this.basSal=basSal;

}

void calSal(){

sal=(basSal+basSal\*0.75+basSal\*0.075-basSal\*0.1);

}

}

class PTEmploy extends Employ{

String qual;

int exp,numHour;

PTEmploy(String name,int age,String address,String gender,String qual,int exp,int numHour){

super(name,age,address,gender);

this.qual=qual;

this.exp=exp;

this.numHour=numHour;

}

void calSal(){

switch(qual){

case "BE":

if(exp<=5) sal=numHour\*300;

else if(exp<=10) sal=numHour\*400;

else sal=numHour\*500;

break;

case "MTech":

if(exp<=5) sal=numHour\*500;

else if(exp<=10) sal=numHour\*700;

else sal=numHour\*1000;

break;

case "PhD":

if(exp<=5) sal=numHour\*800;

else if(exp<=10) sal=numHour\*1200;

else sal=numHour\*1500;

break;

}

}

}

public class Tw4{

public static void main(String []args){

FTEmploy f1=new FTEmploy("Arun",25,"Vijayapur","Male",10000);

f1.calSal();

System.out.println("Details of Full time Employ");

f1.show();

PTEmploy e1=new PTEmploy("Rohit",30,"Belgum","Male","BE",6,10);

e1.calSal();

System.out.println("\nDetails of Part time Employ 1:");

e1.show();

PTEmploy e2=new PTEmploy("Rohini",26,"Mysore","Female","PhD",10,8);

e2.calSal();

System.out.println("\nDetails of Part time Employ 2:");

e2.show();

}

}

5a) Write a program to demonstrate the implementation of method overloading.

5.1) Create a Stack class having an integer array say elem and top\_of\_stack as instance variables. Define three overloaded methods having the following signatures:

a. initStack(int size) to create an array of specified size and initialize the top\_of\_stack

b. initStack(Stack another) to intialize the Stack object with state of the Stack object "another"

c. initStack(int [] a) to initialize contents of a[] to the instance variable elem.

Write following methods:

a. push(): Pushes the element onto the stack,

b. pop(): Returns the element on the top of the stack, removing it in the process, and

c. peek(): Returns the element on the top of the stack, but does not remove it.

Also write methods that check whether stack is full and stack is empty and return boolean value true or false appropriately.

class Stack{

int[] ele;

int top;

void initStack(int size){

ele=new int[size];

top=-1;

}

void initStack(Stack another){

ele=new int[another.ele.length];

top=-1;

for(int item:another.ele)

push(item);

}

void initStack(int[] a){

ele=new int[a.length];

top=-1;

for(int item:a)

push(item);

}

void push(int item){

if(top<ele.length){

ele[++top]=item;

System.out.println("Pushed element is "+item);

}

else

System.out.println("Stack overflow");

}

int pop(){

if(top==-1){

System.out.println("Stack underflow");

return -1;

}

else{

int item=ele[top--];

return item;

}

}

int peek(){

return ele[top];

}

}

public class TW5a {

public static void main(String[] args) {

Stack s1=new Stack();

Stack s2=new Stack();

s1.initStack(5);

s1.push(10);

s1.push(20);

s1.push(30);

s1.push(40);

s1.push(50);

s2.initStack(s1);

int[] array={1,2,3,4};

Stack s3=new Stack();

s3.initStack(array);

System.out.println("Popped element in S1 object is "+s1.pop());

System.out.println("Element on top of the stack of object s1 is "+s1.peek());

System.out.println("Element on top of the stack of object s2 is "+s2.peek());

}

}

The code is implementing a stack data structure using the Java programming language. A stack is a collection of elements in which the last element added is the first one to be removed (Last-In-First-Out).

The Stack class has the following instance variables:

• ele is an integer array that will hold the stack elements.

• top is an integer that will keep track of the index of the top element in the stack.

The Stack class has the following methods:

• initStack(int size): This method initializes the stack with a specific size. It creates an integer array of the specified size and sets the top variable to -1 (indicating that the stack is empty).

• initStack(Stack another): This method initializes the stack with the elements of another stack. It creates a new integer array of the same size as the input stack and copies all its elements into the new array. It then sets the top variable to -1 and pushes all the elements onto the new stack.

• initStack(int[] a): This method initializes the stack with the elements of an integer array. It creates a new integer array of the same size as the input array and copies all its elements into the new array. It then sets the top variable to -1 and pushes all the elements onto the new stack.

• push(int item): This method adds an element to the top of the stack. It first checks if there is space in the ele array, and if there is, it increments top and adds the element to the ele array at the new top index. If there is no space, it prints an error message.

• pop(): This method removes and returns the top element from the stack. It first checks if the stack is empty (by checking if top is equal to -1), and if it is, it prints an error message and returns -1. If the stack is not empty, it retrieves the top element, decrements top, and returns the element.

• peek(): This method returns the top element of the stack without removing it.

The main() method of the TW5a class creates three Stack objects and perfo rms various operations on them. It initializes s1 with a size of 5 and adds five elements to it using the push() method. It then initializes s2 with the elements of s1 using the initStack(Stack another) method. It initializes s3 with an integer array and adds all its elements to the stack using the initStack(int[] a) method. Finally, it retrieves and prints the top element of s1 and s2 using the peek() method, and removes and prints the top element of s1 using the pop() method.

**5 b) Overriding.**

5 b.1 )Implement the following class hierarchy. In the Cuboid class, override the method computeArea() and computePerimeter() of Rectangle class to compute the surface area and perimeter of a rectangle cuboid. Add a method computeVolume() in Cuboid class to compute volume of the cuboid. Assuming length, width and height as l, w and h respectively,

* formula to find the surface area = 2(lw) + 2(hl) + 2(hw)
* formula to find the perimeter = 2l + 2w
* formula to find the volume = l x w x h

Rectangle

length:double = 1.0

width:double = 1.0

Rectangle()

Rectangle(length, width)

computeArea():double

computePerimeter():double

Cuboid

height:double = 1.0

Cuboid()

Cuboid(length, width, height)

computeArea():double

computePerimeter():double

computeVolume():double

class Rectangle {

double length;

double width;

Rectangle() {

length = 1.0;

width = 1.0;

}

Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

double computeArea() {

return length \* width;

}

double computePerimeter() {

return 2 \* (length + width);

}

}

class Cuboid extends Rectangle {

double height;

Cuboid() {

super();

height = 1.0;

}

Cuboid(double length, double width, double height) {

super(length, width);

this.height = height;

}

@Override

double computeArea() {

return 2 \* ((length \* width) + (width \* height) + (length \* height));

}

@Override

double computePerimeter() {

return 4 \* (length + width + height);

}

double computeVolume() {

return length \* width \* height;

}

}

public class TW5b{

public static void main(String[] args) {

Rectangle r1 = new Rectangle();

System.out.println("Rectangle 1:");

System.out.println("Area:" + r1.computeArea());

System.out.println("Perimeter:" + r1.computePerimeter());

Rectangle r2 = new Rectangle(10,30);

System.out.println("\nRectangle 2:");

System.out.println("Area:" + r2.computeArea());

System.out.println("Perimeter:" + r2.computePerimeter());

Cuboid c1=new Cuboid();

System.out.println("\nCuboid 1:");

System.out.println("Area:" + c1.computeArea());

System.out.println("Perimeter:" + c1.computePerimeter());

System.out.println("Volume:" + c1.computeVolume());

Cuboid c2=new Cuboid(10,30,40);

System.out.println("\nCuboid 2:");

System.out.println("Surface area:" + c2.computeArea());

System.out.println("Perimeter:" + c2.computePerimeter());

System.out.println("Volume:" + c2.computeVolume());

}

}

1. Write a program to demonstrate Run-time Polymorphism.

6.1) Design an abstract class Car to have carName, chassiNum, modelName as member variables and add two abstract methods, startCar and operateSteering . Inherit MarutiCar and BmwCar from Car class and override the two abstract methods in their own unique way. Design a driver class to have driver name, gender and age as data members and add a method driveCar with abstract class reference variable as argument and invoke the two basic operations namely, startCar and operateStearing and demonstrate run-time polymorphism.

abstract class Car{

String carName,modelName;

int chassiNum;

Car(String carName,int chassiNum,String modelName){

this.carName=carName;

this.chassiNum=chassiNum;

this.modelName=modelName;

}

abstract void startCar();

abstract void operateSteering();

void display(){

System.out.println("Car Name:"+carName);

System.out.println("Chassi number:"+chassiNum);

System.out.println("Model Name:"+modelName);

}

}

class MarutiCar extends Car{

MarutiCar(String carName, int chassiNum, String modelName) {

super(carName, chassiNum, modelName);

}

void startCar(){

System.out.println("Starting a Maruti car....");

}

void operateSteering(){

System.out.println("This car is manually steered.....");

}

}

class BmwCar extends Car{

BmwCar(String carName, int chassiNum, String modelName) {

super(carName, chassiNum, modelName);

}

void startCar(){

System.out.println("Starting a BMW car....");

}

void operateSteering(){

System.out.println("This car is automatically steered.....");

}

}

class Driver{

String name,gender;

int age;

Driver(String name,int age,String gender){

this.name=name;

this.age=age;

this.gender=gender;

}

void driveCar(Car obj){

System.out.println("Driver:"+name);

System.out.println("Age:"+age);

System.out.println("Gender:"+gender);

obj.display();

obj.startCar();

obj.operateSteering();

}

}

public class Tw6 {

public static void main(String []args){

MarutiCar m=new MarutiCar("Suzuki",1253,"A21s");

BmwCar b=new BmwCar("BMW5",4596,"S5");

Driver d1=new Driver("Vishal",25,"Male");

d1.driveCar(m);

System.out.println();

Driver d2=new Driver("Priya",23,"Female");

d2.driveCar(b);

}

}

1. Write a program to demonstrate the implementation of interfaces.

7.1) Write a Java application to implement the following UML diagram.

* PrimeTester class implements isPrime() method by iterating from 2 to n-1 for a given number n
* ImprPrimeTester class implements isPrime() method by iterating from 2 to n/2
* FasterPrimeTester class implements isPrime() method by iterating from 2 to
* FastestPrimeTester class implements isPrime() method using Fermat’s Little theorem.
  + Fermat’s Little Theorem:
  + If n is a prime number, then for every a, 1 < a < n-1,
    - an-1 % n = 1

IPrime <<interface>>

isPrime(n:int):bool

PrimeTester

isPrime(n:int):bool

FastestPrimeTester

isPrime(n:int):bool

ImprPrimeTester

isPrime(n:int):bool

FasterPrimeTester

isPrime(n:int):bool

interface IsPrime{

boolean isPrime(int n);

}

class PrimeTester implements IsPrime{

@Override

public boolean isPrime(int n){

boolean flag=true;

for(int i=2; i<=n-1; i++){

if((n%i)==0)

{

flag=false;

break;

}

}

return flag;

}

}

class ImprPrimeTester implements IsPrime{

@Override

public boolean isPrime(int n){

boolean flag=true;

for(int i=2; i<=n/2; i++){

if((n%i)==0)

{

flag=false;

break;

}

}

return flag;

}

}

class FasterPrimeTester implements IsPrime{

@Override

public boolean isPrime(int n){

boolean flag=true;

for(int i=2; i<=Math.sqrt(n); i++){

if((n%i)==0)

{

flag=false;

break;

}

}

return flag;

}

}

class FastestPrimeTester implements IsPrime{

@Override

public boolean isPrime(int n){

int a=2;

return Math.pow(a,n-1)%n==1;

}

}

public class TW7 {

public static void main(String[] args) {

PrimeTester p1=new PrimeTester();

ImprPrimeTester p2=new ImprPrimeTester();

FasterPrimeTester p3=new FasterPrimeTester();

FastestPrimeTester p4=new FastestPrimeTester();

System.out.println("32 is Prime? "+p1.isPrime(32));

System.out.println("17 is Prime? "+p1.isPrime(17));

System.out.println("32 is Prime? "+p2.isPrime(32));

System.out.println("17 is Prime? "+p2.isPrime(17));

System.out.println("32 is Prime? "+p3.isPrime(32));

System.out.println("17 is Prime? "+p3.isPrime(17));

System.out.println("32 is Prime? "+p4.isPrime(32));

System.out.println("17 is Prime? "+p4.isPrime(17));

}

}

1. Write a program to demonstrate the implementation of **customized exception handling.**

8.1) Assume that you have received a request from the transport authority for automating the task of issuing the permanent license for two wheelers. The mandatory condition to issue the license are: 1) the applicant must over 18 years of age and 2) holder of a valid learner’s license and 3) no accident cases in the last one year.

Write a Java program that reads user details as required (use the Scanner class). Create user defined exceptions to check for the three conditions imposed by the transport authority. Based on the inputs entered by the user, decide and display whether or not a license has to be issued or an error message as defined by the user exception.

Exception

getMessage()

AccidentCasesException

@Override

getMessage()

ValidLearnerLicenseException

@Override

getMessage()

UnderAgeException

@Override

getMessage()

License

name: String

age: int

gender : String

valid\_LL : char/String

no\_of\_cases : int

import java.util.Scanner;

class UnderAgeException extends Exception{

UnderAgeException(String s){

super(s);

}

@Override

public String toString(){

return "Sorry. You are too young for the license";

}

}

class ValidLLR extends Exception{

ValidLLR(String s){

super(s);

}

@Override

public String toString(){

return "Sorry. You do not hold a valid LLR";

}

}

class NumAccidents extends Exception{

NumAccidents(String s){

super(s);

}

@Override

public String toString(){

return "Sorry. There are accidents in last one year";

}

}

class License{

String name;

int age, no\_of\_cases;

char gender;

char validLLR;

void readData(){

Scanner in=new Scanner(System.in);

System.out.println("Enter the name: ");

name = in.nextLine();

System.out.println("Enter the age: ");

age = in.nextInt();

System.out.println("Enter the gender: ");

gender = in.next().charAt(0);

System.out.println("Do you have Valid LLR (Y/N)? ");

validLLR = in.next().charAt(0);

System.out.println("How many number of cases in past one year? ");

no\_of\_cases = in.nextInt();

}

}

public class TW8 {

public static void main(String[] args) {

License applicant =new License();

applicant.readData();

validateApplicant(applicant);

}

static void validateApplicant(License a){

try{

if(a.age<18)

throw new UnderAgeException("Underageexception:");

if(a.validLLR!='Y')

throw new ValidLLR("ValidLLRexception:");

if(a.no\_of\_cases>0)

throw new NumAccidents("Numberofaccidentsexception:");

System.out.println("Congrats!! Your license is being posted");

}

catch(UnderAgeException e){

System.out.println(e.getMessage()+e);

}

catch(ValidLLR e){

System.out.println(e.getMessage()+e);

}

catch(NumAccidents e){

System.out.println(e.getMessage()+e);

}

catch(Exception e){

System.out.println(e.getMessage()+e);

}

}

}

This Java program is an example of how to handle exceptions using the try-catch block. The program creates three custom exception classes, namely UnderAgeException, ValidLLR, and NumAccidents, which inherit from the built-in Exception class.

The License class contains fields for the applicant's name, age, gender, validLLR, and the number of cases in the past year. It also has a readData() method that reads the values for these fields from the user using the Scanner class.

The TW8 class is the main class of the program. It creates a new License object and calls its readData() method to read the values for the fields. Then, it calls the validateApplicant() method, which takes the License object as its argument.

The validateApplicant() method uses the try-catch block to catch exceptions that may occur during the validation process. If the applicant is underage, does not have a valid LLR, or has had accidents in the past year, then the corresponding exception is thrown and caught in the catch block. If any other exception occurs, it is caught by the generic Exception catch block.

If none of the exceptions are caught, then the program prints a congratulatory message to the user, indicating that their license is being posted. If an exception is caught, the program prints an error message to the user, indicating the reason for the failure, as defined by the custom exception classes.

1. **Write a program to demonstrate the implementation of string handling.**

9.1) Read a string containing 3\_4 words using Scanner class object. Split it into words and for each word check if it's palindrome by writing a function isPalindrome(String the myWord, int s, int e) which return true if its palindrome else return false. Where s is start index and e is end index of the input myWord. Print it in uppercase if it is palindrome else reverse the string and print it in lowercase.  Use appropriate string functions to implement the above problem statement.

import java.util.Scanner;ii

public class tw9 {

@SuppressWarnings("empty-statement")

public static void main(String args[])

{

String s1,s2;

Scanner in=new Scanner(System.in);

System.out.println("Enter the String #1 :");

s1=in.next().toLowerCase();

System.out.println("Enter the String #2 :");

s2=in.next().toLowerCase();

check\_anagrams(s1,s2);

}

static void check\_anagrams(String s1,String s2)

{

char c1[] = s1.toCharArray();

char c2[] =s2.toCharArray();

String s3 =sort(c1);

String s4 =sort(c2);

System.out.println(s1 +" and " +s2 + " are anagrams--->"+s3.equalsIgnoreCase(s4));

}

static String sort(char arr[])

{

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

for(int j=0; j<arr.length-1; j++)

{

if(arr[j]>arr[j+1])

{

char temp =arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

return new String(arr);

}

}

This Java program checks if two input strings are anagrams or not. An anagram is a word or phrase formed by rearranging the letters of another word or phrase. In this program, the check\_anagrams() method takes two strings as input and checks if they are anagrams.

The main() method initializes two string variables s1 and s2 and reads the values from the user using the Scanner class. It then calls the check\_anagrams() method, passing the two input strings as arguments.

The check\_anagrams() method converts the input strings to character arrays c1 and c2 using the toCharArray() method. It then calls the sort() method to sort the characters in each character array and return a new string representing the sorted characters. The sort() method uses the bubble sort algorithm to sort the characters in ascending order.

Finally, the check\_anagrams() method compares the two sorted strings using the equalsIgnoreCase() method to check if they are equal, indicating that the two input strings are anagrams of each other. The program prints the result to the console, indicating whether the two input strings are anagrams or not.

10a. Write a Java program to implement ArrayList of Strings using Collection framework. Add the

following entries to the array list.

”Alpha”, “Beta”, “Gamma”, “Delta”, “Epsilon”, “Zeta”, “Eta”

Use Iterator to display the contents of the list, to remove Gamma from the list. Display the contents

of the list after deletion. Use ListIterator to add Gamma back and to modify the objects being

iterated, and display the list backwards. Sort and display the ArrayList elements in ascending and

descending order. Find whether the key element is present in the list or not using linear search

import java.util.\*;

public class IteratorDemo {

public static void main(String[] args) {

ArrayList<String> al = new ArrayList<>();

al.add("Alpha");

al.add("Beta");

al.add("Gamma");

al.add("Delta");

al.add("Epsilon");

al.add("Zeta");

al.add("Eta");

System.out.println("Original Contents");

Iterator<String> i = al.iterator();

while(i.hasNext())

System.out.print(i.next() + " ");

System.out.println("\n");

//to remove Gamma from the list

i = al.iterator();

while(i.hasNext()){

if(i.next().equals("Gamma"))

i.remove();

}

System.out.println("Contents after deletion");

i=al.iterator();

while(i.hasNext())

System.out.print(i.next() + " ");

System.out.println("\n");

//to add Gamma back to the list

ListIterator<String> li = al.listIterator();

while(li.hasNext()){

if(li.next().equals("Beta"))

li.add("Gamma");

}

System.out.println("Contents after addition ");

li = al.listIterator();

while(li.hasNext())

System.out.print(li.next() + " ");

System.out.println("\n");

//to modify the objects

String str;

li = al.listIterator();

while(li.hasNext()){

str = li.next();

switch (str) {

case "Eta":

li.set("Omega");

break;

case "Zeta":

li.set("Psi");

break;

case "Epsilon":

li.set("Chi");

break;

case "Delta":

li.set("Dlt");

break;

default:

break;

}

}

System.out.println("Contents after changes ");

li= al.listIterator();

while(li.hasNext())

System.out.print(li.next() + " ");

System.out.println("\n");

//ListIterator to display the backwards

System.out.println("Modified list backwards ");

while(li.hasPrevious()){

System.out.print(li.previous() + " ");

}

System.out.println("\n");

System.out.println("Array elements before sorting ");

for(String s:al)

System.out.print(s+" ");

System.out.println("\n");

System.out.println("Array elements after sorting in ascending order ");

Collections.sort(al);

for(String s:al)

System.out.print(s+" ");

System.out.println("\n");

System.out.println("Array elements after sorting in descending order ");

Collections.sort(al,Collections.reverseOrder());

for(String s:al)

System.out.print(s+" ");

System.out.println("\n");

Scanner sc = new Scanner(System.in);

System.out.println("Enter the element to be searched in the list: ");

String key = sc.nextLine();

for(String s:al){

if(s.equals(key)){

System.out.println("Element found");

return;

}

}

System.out.println("Element not found");

}

}

/\* TW10b

Use lambda expressions to design a TaxCalculator interface

which defines a single method calculateTax that takes in an

income and returns a double.

The tax amount based on the following criteria:

Income <=50000 tax = income \* 0.1

Income >=50000 & <= 100000 tax = income \* 0.2

Income >=50000 tax = income \* 0.1

Determine if the tax amount is less than 1000.

\*/

package tw10b;

import java.util.function.Predicate;

interface TaxCalculator {

double calculateTax(double income);

}

public class TW10b {

public static void main(String[] args) {

TaxCalculator taxCalculator = (double income) -> {

if (income <= 50000) {

return income \* 0.1;

} else if (income > 50000 && income <= 100000) {

return income \* 0.2;

} else {

return income \* 0.3;

}

};

Predicate<Double> isTaxLessThan1000 = (tax) -> tax < 1000;

double[] incomes = {30000, 60000, 80000, 100000, 150000};

for (double income : incomes) {

double tax = taxCalculator.calculateTax(income);

System.out.println("Income: " + income + ", Tax: " + tax + ", Less than 1000: " + isTaxLessThan1000.test(tax));

}

}

}