11.2

Public class Person {

Protected String name;

Protected String address;

Protected String phoneNumber;

Protected String email;

Public Person(String name) {

This.name = name;

}

Public String getName() {

Return name;

}

Public void setName(String name) {

This.name = name;

}

Public String getAddress() {

Return address;

}

Public void setAddress(String address) {

This.address = address;

}

Public String getPhoneNumber() {

Return phoneNumber;

}

Public void setPhoneNumber(String phoneNumber) {

This.phoneNumber = phoneNumber;

}

Public String getEmail() {

Return email;

}

Public void setEmail(String email) {

This.email = email;

}

@Override

Public String toString() {

Return “Name: “ + getName() + “ Class: “ + this.getClass().getName();

}

}

Public class Student extends Person {

Public static final String FRESHMAN = “Freshman”;

Public static final String SOPHOMORE = “Sophomore”;

Public static final String JUNIOR = “Junior”;

Public static final String SENIOR = “Senior”;

Protected String status;

Public Student(String name) {

Super(name);

}

Public Student(String name, String status) {

Super(name);

This.status = status;

}

@Override

Public String toString() {

Return “Name: “ + getName() + “ Class: “ + this.getClass().getName();

}

}

Public class Employee extends Person {

Protected double salary;

Protected String office;

Protected MyDate dateHired;

Public Employee(String name) {

This(name, 0, “none”, new MyDate());

}

Public Employee(String name, double salary, String office, MyDate dateHired) {

Super(name);

This.salary = salary;

This.office = office;

This.dateHired = dateHired;

}

Public double getSalary() {

Return salary;

}

Public void setSalary(double salary) {

This.salary = salary;

}

Public String getOffice() {

Return office;

}

Public void setOffice(String office) {

This.office = office;

}

Public MyDate getDateHired() {

Return dateHired;

}

Public void setDateHired(MyDate dateHired) {

This.dateHired = dateHired;

}

@Override

Public String toString() {

Return “Name: “ + getName() + “ Class: “ + this.getClass().getName();

}

}

Public class Faculty extends Employee {

Public static String LECTURER = “Lecturer”;

Public static String ASSISTANT\_PROFESSOR = “Assistant Professor”;

Public static String ASSOCIATE\_PROFESSOR = “Associate Professor”;

Public static String PROFESSOR = “Professor”;

Protected String officeHours;

Protected String rank;

Public Faculty(String name) {

This(name, “9-5PM”, “Employee”);

}

Public Faculty(String name, String officeHours, String rank) {

Super(name);

This.officeHours = officeHours;

This.rank = rank;

}

Public String getOfficeHours() {

Return officeHours;

}

Public void setOfficeHours(String officeHours) {

This.officeHours = officeHours;

}

Public String getRank() {

Return rank;

}

Public void setRank(String rank) {

This.rank = rank;

}

@Override

Public String toString() {

Return “Name: “ + getName() + “ Class: “ + this.getClass().getName();

}

}

Public class Staff extends Employee {

Protected String title;

Public Staff(String name) {

This(name, “none”);

}

Public Staff(String name, String title) {

Super(name);

This.title = title;

}

Public String getTitle() {

Return title;

}

Public void setTitle(String title) {

This.title = title;

}

@Override

Public String toString() {

Return “Name: “ + getName() + “ Class: “ + this.getClass().getName();

}

}

Public class Exercise\_02 {

Public static void main(String[] args) {

Person person = new Person(“person”);

Student student = new Student(“student”);

Employee employee = new Employee(“employee”);

Faculty faculty = new Faculty(“faculty”);

Staff staff = new Staff(“staff”);

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

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

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

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

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

}

}

11.3

Import java.util.ArrayList;

Import java.util.Date;

Public class Account {

Protected String mName;

Protected int mId;

Protected double mBalance;

Protected double mAnnualInterestRate; // AnnualInterestRate is percentage.

Protected Date mDateCreated;

Protected ArrayList<Transaction> mTransactions;

Public Account() {

mDateCreated = new Date();

mTransactions = new ArrayList<>();

}

Public Account(int id, double balance) {

This();

mId = id;

mBalance = balance;

}

Public Account(String name, int id, double balance) {

This(id, balance);

mName = name;

}

Public int getId() {

Return mId;

}

Public void setId(int id) {

mId = id;

}

Public double getBalance() {

Return mBalance;

}

Public void setBalance(double balance) {

mBalance = balance;

}

Public double getAnnualInterestRate() {

Return mAnnualInterestRate;

}

Public void setAnnualInterestRate(double annualInterestRate) {

mAnnualInterestRate = annualInterestRate;

}

Public Date getDateCreated() {

Return mDateCreated;

}

Public double getMonthlyInterestRate() {

Return mBalance \* (mAnnualInterestRate / 12) / 100;

}

Public void withdraw(double amount) {

mTransactions.add(new Transaction(‘W’, amount, mBalance, “withdraw”));

mBalance -= amount;

}

Public void deposit(double amount) {

mTransactions.add(new Transaction(‘D’, amount, mBalance, “deposit”));

mBalance += amount;

}

@Override

Public String toString() {

Return “Account name: “ + mName + “\n” + “Interest rate: “ + mAnnualInterestRate + “\n” + mTransactions;

}

Public ArrayList<Transaction> getTransactions() {

Return new ArrayList<>(mTransactions);

}

}

Public class CheckingAccount extends Account {

Protected double OVERDRAFT\_LIMIT = -100;

Public CheckingAccount(int id, double balance) {

Super(id, balance);

}

@Override

Public void withdraw(double amount) {

If (mBalance – amount >= OVERDRAFT\_LIMIT) {

Super.withdraw(amount);

}

}

@Override

Public String toString() {

Return “CheckingAccount{“+

“mBalance=” + mBalance +

‘}’;

}

}

Public class SavingsAccount extends Account {

Protected double OVERDRAFT\_LIMIT = 0;

Public SavingsAccount(int id, double balance) {

Super(id, balance);

}

@Override

Public void withdraw(double amount) {

If (mBalance – amount >= OVERDRAFT\_LIMIT) {

Super.withdraw(amount);

}

}

@Override

Public String toString() {

Return “SavingsAccount{“ +

“mBalance=” + mBalance +

‘}’;

}

}

Public class Exercise\_03 {

Public static void main(String[] args) {

Account account = new Account(111, 200);

CheckingAccount checkingAccount = new CheckingAccount(112, 250);

SavingsAccount savingsAccount = new SavingsAccount(113, 300);

System.out.println(account);

System.out.println(checkingAccount);

System.out.println(savingsAccount);

}

}

11.5

Import java.util.ArrayList;

Public class Exercise\_05 {

// modified Course class below

}

Class Course {

Private String courseName;

Private ArrayList<String> students = new ArrayList<>();

Public Course(String courseName) {

This.courseName = courseName;

}

Public void addStudent(String student) {

Students.add(student);

}

Public String[] getStudents() {

Return students.toArray(new String[students.size()]);

}

Public int getNumberOfStudents() {

Return students.size();

}

Public String getCourseName() {

Return courseName;

}

Public void dropStudent(String student) {

Students.remove(student);

}

Public void clear(){

Students.clear();

}

}

11.6

Import java.util.ArrayList;

Import java.util.Date;

Public class Account {

Protected String mName;

Protected int mId;

Protected double mBalance;

Protected double mAnnualInterestRate; // AnnualInterestRate is percentage.

Protected Date mDateCreated;

Protected ArrayList<Transaction> mTransactions;

Public Account() {

mDateCreated = new Date();

mTransactions = new ArrayList<>();

}

Public Account(int id, double balance) {

This();

mId = id;

mBalance = balance;

}

Public Account(String name, int id, double balance) {

This(id, balance);

mName = name;

}

Public int getId() {

Return mId;

}

Public void setId(int id) {

mId = id;

}

Public double getBalance() {

Return mBalance;

}

Public void setBalance(double balance) {

mBalance = balance;

}

Public double getAnnualInterestRate() {

Return mAnnualInterestRate;

}

Public void setAnnualInterestRate(double annualInterestRate) {

mAnnualInterestRate = annualInterestRate;

}

Public Date getDateCreated() {

Return mDateCreated;

}

Public double getMonthlyInterestRate() {

Return mBalance \* (mAnnualInterestRate / 12) / 100;

}

Public void withdraw(double amount) {

mTransactions.add(new Transaction(‘W’, amount, mBalance, “withdraw”));

mBalance -= amount;

}

Public void deposit(double amount) {

mTransactions.add(new Transaction(‘D’, amount, mBalance, “deposit”));

mBalance += amount;

}

@Override

Public String toString() {

Return “Account name: “ + mName + “\n” + “Interest rate: “ + mAnnualInterestRate + “\n” + mTransactions;

}

Public ArrayList<Transaction> getTransactions() {

Return new ArrayList<>(mTransactions);

}

}

Import javafx.scene.shape.Circle;

Public class Circle2D extends GeometricObject {

Private double x;

Private double y;

Private double radius;

Public Circle2D(Circle c) {

This(c.getCenterX(), c.getCenterY(), c.getRadius());

}

Public Circle2D(double x, double y, double radius) {

This.x = x;

This.y = y;

This.radius = radius;

}

Public Circle2D() {

This(0, 0, 1);

}

Public double getX() {

Return x;

}

Public void setX(double x) {

This.x = x;

}

Public double getY() {

Return y;

}

Public void setY(double y) {

This.y = y;

}

Public double getRadius() {

Return radius;

}

Public void setRadius(double radius) {

This.radius = radius;

}

@Override

Public double getArea() {

Return radius \* radius \* Math.PI;

}

@Override

Public double getPerimeter() {

Return 2 \* radius \* Math.PI;

}

Public boolean contains(Circle2D circle2D) {

Double distance = getPoint().distance(circle2D.x, circle2D.y);

If (distance <= Math.abs(this.radius – circle2D.radius)) {

Return true;

} else {

Return false;

}

}

Public static boolean c1ContainsC2(Circle c1, Circle c2) {

Circle2D cir1 = new Circle2D(c1);

Circle2D cir2 =new Circle2D(c2);

Return cir1.contains(cir2);

}

Public static boolean c1OverlapsC2(Circle c1, Circle c2) {

Circle2D cir1 = new Circle2D(c1);

Circle2D cir2 =new Circle2D(c2);

If (cir1.contains(cir2) || cir2.contains(cir1)) return false;

Return cir1.overlaps(cir2);

}

Public boolean overlaps(Circle2D circle2D) {

Double distance = getPoint().distance(circle2D.x, circle2D.y);

If (distance <= this.radius + circle2D.radius)

Return true;

Else

Return false;

}

Private MyPoint getPoint() {

Return new MyPoint(this.x, this.y);

}

@Override

Public String toString() {

Return “Circle2D{“ +

“x=” + x +

“, y=” + y +

“, radius=” + radius +

‘}’;

}

Public boolean contains(double x, double y) {

Double distance = getPoint().distance(x, y);

If (distance <= radius) return true;

Else return false;

}

}

Import java.util.ArrayList;

Import java.util.Date;

Public class Exercise\_06 {

Public static void main(String[] args) {

ArrayList<Object> list = new ArrayList<>();

List.add(new Circle2D());

List.add(“Hello I’m a string”);

List.add(new Date());

List.add(new Account(111, 230));

List.forEach(System.out::println);

}

}

11.11

Import java.util.ArrayList;

Import java.util.Scanner;

Public class Exercise\_11 {

Public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print(“Enter 5 numbers: “);

ArrayList<Integer> list = new ArrayList<>();

For (int I = 0; I < 5; i++) list.add(input.nextInt());

System.out.println(“Sorting numbers…”);

Sort(list);

System.out.println(“Displaying numbers…”);

System.out.println(list);

}

Public static void sort(ArrayList<Integer> list) {

// simple solution

//list.sort(null);

// manual solution

For (int I = 0; I < list.size() – 1; i++) {

Int currentMin = list.get(i);

Int currentIndex = I;

For (int j = I + 1; j < list.size(); j++) {

If (currentMin > list.get(j)) {

currentMin = list.get(j);

currentIndex = j;

}

}

If (currentIndex != i) {

List.set(currentIndex, list.get(i));

List.set(I, currentMin);

}

}

}

}

11.13

Import java.util.ArrayList;

Import java.util.Scanner;

Public class Exercise\_13 {

Public static void main(String[] args) {

System.out.print(“Enter 10 integers: “);

Scanner input = new Scanner(System.in);

ArrayList<Integer> list = new ArrayList<>();

For (int I = 0; I < 10; i++) list.add(input.nextInt());

System.out.println(“Removing duplicates…”);

removeDuplicate(list);

System.out.println(“Displaying list…”);

System.out.println(list);

}

Public static void removeDuplicate(ArrayList<Integer> list) {

ArrayList<Integer> temp = new ArrayList<>();

For (int I = 0; I < list.size(); i++) {

If (!temp.contains(list.get(i))) {

Temp.add(list.get(i));

}

}

List.clear();

List.addAll(temp);

}

}

11.14

​Import​ ​java.util.Scanner​;

​Import​ ​java.util.ArrayList​;

​Public​ ​class​ ​Exercise\_11\_14​ {

​        ​/\*​\* Main method ​\*/

​        ​Public​ ​static​ ​void​ ​main​(​String​[] ​args​) {

​                ​//​ Create two ArrayLists

​                ​ArrayList<​Integer​>​ list1 ​=​ ​new​ ​ArrayList<​Integer​>​();

​                ​ArrayList<​Integer​>​ list2 ​=​ ​new​ ​ArrayList<​Integer​>​();

​                ​//​ Prompt the user to enter two lists

​                ​//​ each with five integers

​                ​System​.​out​.​print(​“​Enter five integers for list1: ​“​);

​                Fill(list1);

​                ​System​.​out​.​print(​“​Enter five integers for list2: ​“​);

​                Fill(list2);

​                ​//​ Combined lists

​                ​ArrayList<​Integer​>​ list3 ​=​ union(list1, list2);

​                ​//​ Display combined list

​                ​System​.​out​.​print(​“​The combined list is: ​“​);

​                ​For​ (​int​ I ​=​ ​0​; I ​<​ list3​.​size(); i​++​) {

​                        ​System​.​out​.​print(list3​.​get(i) ​+​ ​“​ ​“​);

​                }

​                ​System​.​out​.​println();

​        }

​        ​/\*​\* Returns the union of two array lists of integers ​\*/

​        ​Public​ ​static​ ​ArrayList<​Integer​>​ ​union​(

​                ​ArrayList<​Integer​>​ ​list1​, ​ArrayList<​Integer​>​ ​list2​) {

​                ​ArrayList<​Integer​>​ list3 ​=​ list1;

​                ​For​ (​int​ I ​=​ ​0​; I ​<​ list2​.​size(); i​++​) {

​                        List3​.​add(list2​.​get(i));

​                }

​                ​Return​ list3;

​        }

​        ​/\*​\* Adds user input to a list ​\*/

​        ​Public​ ​static​ ​void​ ​fill​(​ArrayList<​Integer​>​ ​list​) {

​                ​//​ Create a Scanner

​                ​Scanner​ input ​=​ ​new​ ​Scanner​(​System​.​in);

​                ​For​ (​int​ I ​=​ ​0​; I ​<​ ​5​; i​++​) {

​                        List​.​add(input​.​nextInt());

​                }

​        }

​}

11.15

Public class MyPoint {

Public double x;

Public double y;

Public MyPoint(double x, double y) {

This.x = x;

This.y = y;

}

Public MyPoint() {

This(0,0);

}

Public double x() {

Return x;

}

Public void setX(double x) {

This.x = x;

}

Public double y() {

Return y;

}

Public void setY(double y) {

This.y = y;

}

Public double distance(double x, double y) {

Return Math.sqrt((this.x – x) \* (this.x – x) + (this.y – y) \* (this.y – y));

}

Public double distance(MyPoint point) {

Return distance(point.x, point.y);

}

Public MyPoint getCenterPoint(MyPoint p) {

Return new MyPoint((p.x + this.x) / 2, (p.y + this.y) / 2);

}

Public static MyPoint getCenterPoint(double x1, double y1, double x2, double y2) {

Return new MyPoint((x1 + x2) / 2, (y1 + y2) / 2);

}

/\*\* Return true if this point is on the left side of the

* Directed line from p0 to p1 \*/

Public boolean leftOfTheLine(MyPoint p0, MyPoint p1) {

Return leftOfTheLine(p0.x, p0.y, p1.x, p1.y, x, y);

}

/\*\* Return true if this point is on the same

* Line from p0 to p1 \*/

Public boolean onTheSameLine(MyPoint p0, MyPoint p1) {

Return onTheSameLine(p0.x, p0.y, p1.x, p1.y, x, y);

}

/\*\* Return true if this point is on the right side of the

* Directed line from p0 to p1 \*/

Public boolean rightOfTheLine(MyPoint p0, MyPoint p1) {

Return rightOfTheLine(p0.x, p0.y, p1.x, p1.y, x, y);

}

/\*\* Return true if this point is on the

* Line segment from p0 to p1 \*/

Public boolean onTheLineSegment(MyPoint p0, MyPoint p1) {

Return onTheLineSegment(p0.x, p0.y, p1.x, p1.y, x, y);

}

/\*\* Return true if point (x2, y2) is on the left side of the

* Directed line from (x0, y0) to (x1, y1) \*/

Public static boolean leftOfTheLine(double x0, double y0, double x1, double y1, double x2, double y2){

Return (x1 – x0) \* (y2 – y0) – (x2 – x0) \* (y1 – y0) > 0;

}

/\*\* Return true if point (x2, y2) is on the same

* Line from (x0, y0) to (x1, y1) \*/

Public static boolean onTheSameLine(double x0, double y0, double x1, double y1, double x2, double y2) {

Return (x1 – x0) \* (y2 – y0) – (x2 – x0) \* (y1 – y0) == 0;

}

/\*\* Return true if point (x2, y2) is on the

* Line segment from (x0, y0) to (x1, y1) \*/

Public static boolean onTheLineSegment(double x0, double y0, double x1, double y1, double x2, double y2) {

Double position = (x1 – x0) \* (y2 – y0) – (x2 – x0) \* (y1 – y0);

Return position <= 0.0000000001 && ((x0 <= x2 && x2 <= x1) || (x0 >= x2 && x2 >= x1));

}

/\*\* Return true if point (x2, y2) is on the right side of the

* Directed line from (x0, y0) to (x1, y1) \*/

Public static boolean rightOfTheLine(double x0, double y0, double x1, double y1, double x2, double y2){

Return (x1 – x0) \* (y2 – y0) – (x2 – x0) \* (y1 – y0) < 0;

}

@Override

Public String toString() {

Return “(“ + x + “, “ + y + “)”;

}

}

Import java.util.ArrayList;

Import java.util.Scanner;

Public class Exercise\_15 {

Public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print(“Enter the number of the points: “);

Int numOfPoints = input.nextInt();

System.out.print(“Enter the coordinates of the points: “);

ArrayList<MyPoint> points = new ArrayList<>();

For (int I = 0; I < numOfPoints; i++) {

Points.add(new MyPoint(input.nextDouble(), input.nextDouble()));

}

System.out.println(“The total area is “ + getConvexPolygonArea(points));

}

// Area of a Convex Polygon

// <http://www.mathwords.com/a/area_convex_polygon.htm>

Public static double getConvexPolygonArea(ArrayList<MyPoint> points) {

// points must be counter clockwise

Double sum1 = 0;

Double sum2 = 0;

For (int I = 0; I < points.size(); i++) {

Int limitIndex = (I + 1) % points.size();

MyPoint p1 = points.get(i);

MyPoint p2 = points.get(limitIndex);

System.out.println(“P1 index = “ + i);

System.out.println(“P2 index =” + limitIndex);

Sum1 += (p1.x \* p2.y);

Sum2 += (p1.y \* p2.x);

}

Double area = 0.5 \* (sum1 – sum2);

Return (area > 0) ? area : -area;

}

}

11.19

Import java.util.ArrayList;

Import java.util.Scanner;

Public class Exercise20 {

Public static void main(String[] args) {

@SuppressWarnings(“resource”)

Scanner input = new Scanner(System.in);

System.out.print(“Enter the number of objects: “);

Int numberOfObjects = input.nextInt();

ArrayList<Integer> objects = new ArrayList<>();

System.out.print(“Enter the weights of the objects: “);

For (int I = 0; I < numberOfObjects; i++) {

Objects.add(input.nextInt());

}

Int container = 1;

While(!objects.isEmpty()) {

System.out.println(“Container “ + container++ + “ contains objects with weight “ + getConteiner(objects, 10));

}

}

Static String getConteiner(ArrayList<Integer> objects, int max) {

String result = “”;

For (int I = 0; I < objects.size(); i++) {

If(objects.get(i) <= max) {

Int tmp = objects.get(i);

Result += tmp + “ “;

Objects.remove(i);

Return result + getConteiner(objects, max – tmp);

}

}

Return result;

}

}

Mini project

Public class Triangle extends GeometricObject {

Private double side1;

Private double side2;

Private double side3;

Public Triangle(double side1, double side2, double side3) throws IllegalTriangleException {

This.side1 = side1;

This.side2 = side2;

This.side3 = side3;

isValidTriangle();

}

Public Triangle() {

This.side1 = 1;

This.side2 = 1;

This.side3 = 1;

}

@Override

Public double getArea() {

Double s = (side1 + side2 + side3) / 2.0;

Return Math.pow(s \* (s – side1) \* (s – side2) \* (s – side3), 0.5);

}

@Override

Public double getPerimeter() {

Return side1 + side2 + side3;

}

@Override

Public String toString() {

Return “Triangle{“ +

“side1=” + side1 +

“, side2=” + side2 +

“, side3=” + side3 +

‘}’;

}

Public static boolean isTriangle(double side1, double side2, double side3) {

Return ((side1 + side2 > side3) &&

(side1 + side3 > side2) &&

(side3 + side2 > side1));

}

Public double getSide1() {

Return side1;

}

Public void setSide1(double side1) throws IllegalTriangleException {

This.side1 = side1;

isValidTriangle();

}

Public double getSide2() {

Return side2;

}

Public void setSide2(double side2) throws IllegalTriangleException{

This.side2 = side2;

isValidTriangle();

}

Public double getSide3() {

Return side3;

}

Public void setSide3(double side3) throws IllegalTriangleException {

This.side3 = side3;

isValidTriangle();

}

Private void isValidTriangle() throws IllegalTriangleException{

If (!isTriangle(side1, side2, side3)) {

Throw new IllegalTriangleException(side1, side2, side3);

}

}

Public class IllegalTriangleException extends IllegalArgumentException {

Private double s1;

Private double s2;

Private double s3;

Public IllegalTriangleException(double s1, double s2, double s3) {

Super(“Not a real triangle:” + “ side1 = “ + s1 + “ side2 = “ + s2 + “ side3 = “ + s3);

This.s1 = s1;

This.s2 = s2;

This.s3 = s3;

}

Public double getS1() {

Return s1;

}

Public double getS2() {

Return s2;

}

Public double getS3() {

Return s3;

}

}

}

Import java.util.Scanner;

Public class Exercise\_01 {

Public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print(“Enter three sides of a triangle: “);

Double[] sides = new double[3];

For (int I = 0; I < sides.length; i++) sides[i] = input.nextDouble();

System.out.print(“Enter a triangle color: “);

String color = input.next();

System.out.print(“Is the triangle filled? True/false: “);

String isFilledString = input.next();

Boolean isFilled = (isFilledString.equals(“true”));

Triangle t1 = null;

Try {

T1 = new Triangle(sides[0], sides[1], sides[2]);

T1.setColor(color);

T1.setFilled(isFilled);

System.out.println(“Triangle 1:”);

System.out.println(“Area = “ + t1.getArea());

System.out.println(“Perimeter = “ + t1.getPerimeter());

System.out.println(“Color = “ + t1.getColor());

System.out.println(“Is filled? “ + t1.isFilled());

} catch (IllegalTriangleException e) {

e.printStackTrace();

}

}

}