CS1120 – Programming Mastery Test

Problem: The Final Frontier

The year is 2092 and space travel has become commonplace. Many star systems have been discovered with multiple inhabited planets in orbit. Space cruises (like ocean cruises) have become hot vacation choices for the well to do. But how do you know what travel package to choose?

Write a program that reads in a list of star systems from an input file named *starsystems.txt*. The first line of input tells how many systems are listed. Each system follows on a separate line, its data separated by commas. The data consists of: the system name, x-, y-, and z-coordinates, and the number of inhabited planets. The coordinates represent distance in light-years from the origin (Sol, our own system). See example input below.

Read in and save the list of star systems. Display their names to the console and allow the user to choose 2 of them. Display the coordinates and the number of inhabited planets for each system. Then calculate the total distance traveled starting at Sol, visiting each system in turn, and finally returning to Sol again. Also calculate the total number of planets visited (assuming you visit all the inhabited planets in each system, NOT including the Earth or any of Sol's planets). See the example output below.

Distance formula between 2 coordinates:

$$dist = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

You must use the following interface, which your StarSystem class should implement.

```
public interface IStarSystem {
	String getName();
	double getXCoord();
	double getYCoord();
	double getZCoord();
	int getNumPlanets();
	double calculateDistanceTo(IStarSystem otherSystem);
}
```

Example input:

```
4
Sol,0,0,0,8
Alpha Centauri,3.165,-3.048,-0.0818,4
Epsilon Eridani,-6.820,-1.966,-7.733,11
Sirius,-5.745,-6.275,-1.262,1
```

Example output:

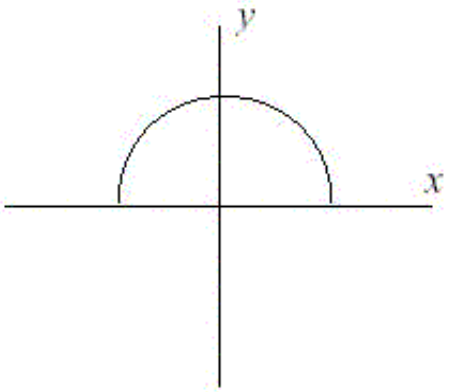
```
Sol
Alpha Centauri
Epsilon Eridani
Sirius

Please select 2 destinations:
Alpha Centauri
Sirius

Alpha Centauri: 4 planet(s)
Sirius: 1 planet(s)

Total distance: 22.545148744870257 light years
Total planets: 5
```

CS1120 – Programming Mastery Test

Problem: I Think I Need a Houseboat

Fred Mapper is considering purchasing some land in Louisiana to build his house on. In the process of investigating the land, he learned that the state of Louisiana is actually shrinking by 50 square miles each year, due to erosion caused by the Mississippi River. Since Fred is hoping to live in this house the rest of his life, he needs to know if his land is going to be lost to erosion.

After doing more research, Fred has learned that the land that is being lost forms a semicircle. This semicircle is part of a circle centered at (0,0), with the line that bisects the circle being the X axis. Locations below the X axis are in the water. The semicircle has an area of 0 at the beginning of year 1 and will grow 50 square miles in each year.



Input
The first line of input will be a positive integer (N) indicating how many remaining lines in the input file, *houseboatin.txt*. Each of the next N lines will contain the X and Y Cartesian coordinates of the land Fred is considering. These will be floating point numbers measured in miles. The Y coordinate will be non-negative.

Output
For each property, a single line of output should appear on the console. This line should take the form of: "*Property N: This property will begin eroding in year Z.*" Z is the first year (start from 1) this property will be within the semicircle at the end of year Z where Z must be an integer. After the last line, your program should print "END OF OUTPUT."

Sample Input:

```
2
1.0 1.0
25.0 0.0
```

Sample Output:

```
Property 1: This property will begin eroding in year 1.
Property 2: This property will begin eroding in year 20.
END OF OUTPUT.
```

Hints:

1. No property will appear exactly on the semicircle boundary: it will either be inside or outside.
2. All locations are given in miles.

CS1120 – Programming Mastery Test

Problem: Big Numbers and Their Addition

At times, as a programmer, you may need larger numbers than what will fit in a long data type. In this case, we switch to a data type usually called BigNum. In the real world, these are implemented several ways, the most common being Binary Coded Decimal. You will use short data type for this program.

You will define a class named BigNum which has an array of shorts of 100 elements. There is one digit (0-9) in each element of the array. This means objects of this type can handle numbers up to 100 digits. The class also needs an overflow flag and a pointer to where in the array the stored number starts or stops. It will only handle positive whole numbers. In the BigNum class, you will write a method addTo(BigNum) to add numbers stored in two BigNum objects and return a new BigNum object, a toString() method, and a constructor BigNum(String) which accepts a string of digits as input. If the stored number does not fit in 100 digits, then set the overflow flag. Note that *overflow + any number* is overflow.

Write your BigNum class such that the following code

```
public static void main(String[] args) {
		BigNum bn = new
BigNum("99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999");
		BigNum b2 = new BigNum("0");
		System.out.println(bn + " + " + b2 + " = ");
		System.out.println(bn.addTo(b2));
		b2 = new BigNum("1");
		System.out.println("\n" + bn + " + " + b2 + " = ");
		System.out.println(bn.addTo(b2));
		bn = new BigNum("123456789");
		b2 = new BigNum("123456789");
		System.out.println("\n" + bn + " +\n" + b2 + " = ");
		System.out.println(bn.addTo(b2)+"\n");
	}
```

will produce the following output:

```
99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
999999999999999 + 0 =
99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
999999999999999

99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
999999999999999 + 1 =
Overflow

123456789 +
123456789 =
246913578
```

CS1120 – Programming Mastery Test

Problem: Video Store

A video store provides a rental service to its customers. In this part, you are asked to help the store calculate and print out the charge for each customer. A customer can borrow multiple movie CDs. Each Customer has a name. Each movie CD has a movie name, the number of days rented, and a category. There are three categories for movies: regular movie, children movie, and new release. The calculation of each category of movie is based on the following:

Regular movie: $2 for each rental day,
New release movie: $3 for each rental day,
Children movie: $1.5 for each rental day.

You implement the following interfaces via some classes. **Your program should not use any if/switch statements in your class implementing ICustomer.**

```
public interface ICustomer
    {
       String get_Name();   //return the customer's name
       ArrayList<IMovie> getMovies();  //return the arraylist of the movies borrowed by the customer
  void addMovie(IMovie m);  // add the movie object to the list borrowed by the cust.
  double getCharges(); // return the total charges based on the list of movies borrowed
                                    // by cust.
    }

    public interface IMovie
    {
            String get_Name();  //return the movie's name
     String get_Category();  //return the movie's categories: Regular, New Release,
                                          //and Children.
            int get_RentedDays();   //return the number of rented days for this movie
            double calculate_RentalFee();  // return the amount of money that should be paid
```

```
                    // for this movie; the number of rented days
                                        // should be set before this method is called.
        }
```

Also, your classes should be called by the following testing class:

```
    class MainClass
    {
            public static void Main (string[] args)
            {
              Customer c = new Customer();
        IMovie m1 = new Regular_Movie("Trainspoting", 3); //Name Trainspoting, category: regular movie, # of rented days: 3
                IMovie m2 = new NewRelease_Movie("Departed", 5); //Name: Departed, category: new release, # of rented days: 5
        IMovie m3 = new Children_Movie("Toy Story",  2); //Name: Toy Story, category: Children movie, # of rented days 2

                    c.addMovie(m1);
                    c.addMovie(m2);
                    c.addMovie(m3);

                    Console.WriteLine("Total charge is " + c.getCharges()); // Total charges is 20

        IMovie m4 = new Regular_Movie("Trainspoting II", 2);//Name Trainspoting II, category: regular movie, # of rented days: 2
        IMovie m5 = new Children_Movie("Toy Story II", 5);  //Name: Toy Story II, category: Children movie, # of rented days 2

        c.addMovie(m4);
```

```
    c.addMovie(m5);


    Console.WriteLine("Total charge is " + c.getCharges()); // Total charges is 26.5


// more test cases will be added here.

            }

  }
```