CPSC 233: Assignment 4 (*Due March 26 at 4 PM)*

New learning concepts: Problem solving using object-oriented programming. Aside from **`main()`** you cannot implement other static methods. Also you should not be including any static variable attributes in your class definitions (class constants may be okay on an as needed basis). You will be given the code for the **`Driver`** and **`Oracle`** classes; and you must implement the methods in the **`Detective`**, **`Person`**, **`Location`**, and **`Time`** classes. Don't change the **`Driver`** and **`Oracle`** classes because they will be used by the markers to test your program

Problem Statement:

In this assignment, you will complete a program that simulates a murder mystery game. There are two entities in the simulation: the Oracle and the detective. The Oracle knows the exact description of the murderer, the location, and the time, which is the information read from an input file (*e.g.*, *oracle.txt*). The detective gets the information of all suspects from another input file (*e.g.*, *suspects.txt*). You may assume that there will be no more than 20 suspects in the file. The detective may query the Oracle for each suspect, but the Oracle will only response with a numerical score indicating how much the information provided matches the actual murder case. The goal of the detective is to identify the murderer, the location, and the time by going through a minimum number of suspects and query the Oracle as little as possible. (The fewer the queries, the higher will be your grade).

Program description:

In this assignment, you are given the **`Driver`** class and the implementation of the Oracle. The **`Driver`** class creates the Oracle and passes a string containing the actual details of the murder. The string is read from a file (*e.g.*, *oracle.txt*). The Driver then creates the detective and asks the detective to check each of the suspects (*e.g.*, information coming from *suspects.txt*) until the actual murder case is identified. In the input files for both the Oracle and the detective, the murder case and the suspect cases are presented in a single line in the following format:

`<gender>, <age range>, <height>; <x>, <y>; <hh>:<mm>`

, where a person is identified by gender ('M' for male or 'F' for female), age range (an integer), and height in meters (a real number), a location is identified by the (x, y) coordinate in a 2D map (with x as the x-coordinate and y as the y-coordinate), and the time is defined by the hour (in 24-hour format) and the minute. You may assume that the input files are error-free and follows the exact format as defined here.

In the input file for the Oracle (*e.g.*, *oracle.txt*), there will be one and only one line presenting the actual information about the murder: the actual combination of "person", "location" and "time". This information will be parsed and stored by the Oracle so it can answer queries from the detective on whether the suspected cases (other combinations of person, location, and time) are correct or how close were the cases. In the input file for the detective (*e.g.*, *suspects.txt*), there will be a list of suspected cases: one suspect per line following the same format as the other file. The detective reasons about the murder by processing these cases and querying the Oracle.

The implementation of the **`Oracle`** class includes the following members:

- **`private Person murderer;`**
  The murderer description, an instance of class **`Person`** with the attributes for gender, age range, and height.
- **`private Location murderLocation;`**
  The location of the murder case, an instance of class **`Location`** with the attributes for the x-y coordinates of the city in a 2D map.
- **`private Time murderTime;`**
  The time of the murder case, an instance of class **`Time`** with the attributes for hour and minute.
- **`private int count;`**
  The number of times the Oracle is queried.
- **`public Oracle (String line);`**
  The constructor first parses (separates the line into parts) the murder information presented in the string **`line`**. The **`line`**, the one and only one line from the input file for the Oracle (*e.g.*, *oracle.txt*), is passed here by the **`Driver`** class. This constructor then initializes the **`murderer`**, **`murderLocation`**, and **`murderTime`** according to the information presented in the string **`line`**.
- **`public double checkPerson(Person suspect);`**
  A method that matches the description of the suspect with the murderer's and returns a score indicating the difference between the **`suspect`** and the **`murderer`**. The score is determined by **`match()`** from the **`Person`** class.[1]
- **`public double checkLocation(Location suspectLocation);`**
  A method that matches the suspect's location information with the murder location and returns the distance between the **`suspectLocation`** and the **`murderLocation`**. The distance is determined by **`match()`** from the **`Location`** class.[1]
- **`public int checkTime(Time suspectTime);`**
  A method that matches the suspect's time with the murder time and returns a score indicating how close the **`suspectTime`** is to the **`murderTime`**. The score is determined by **`match()`** from the **`Time`** class.[1]
- **`public void getCount();`**
  A method returns the number of times the Oracle is queried.

Partial implementation of the **`Person`**, **`Location`**, and **`Time`** classes are also provided to you. Below is the definition of the data member of each class. You will complete the methods (defined later) in these classes.

- **`public class Person:`**
  - **`private char gender;`**
    Defines the gender of the person: 'M' as male, 'F' as female, and 'U' as defined.
  - **`private int ageRange;`**
    Defines the age range of a person: 10 means teenagers, 20 means 20's (20, 21, 22…29), 30 means 30's, and so on. Valid age ranges are 10's, 20's, and up to 90's.

[1] See the detailed description of the **`match()`** method on the next page.

  - **`private double height;`**
    Defines the height of a person in meters.
- **`public class Location:`**
  - **`private double x;`**
    The x coordinate of the location in a 2D map.
  - **`private double y;`**
    The y coordinate of the location in a 2D map.
- **`public class Time:`**
  - **`private int hour;`**
    Defines the hour in the 24-hour format. Valid values include 0, 1, 2, …, 23.
  - **`private int min;`**
    Defines the minute. Valid values include 0, 1, 2, …, 59.

Your first task is to complete the implementation of the **`Person`**, **`Location`**, and **`Time`** classes, including:

- completing the **`match()`** method, as defined below, in each class;
  - **`public double match(Person individual);`** // in the **`Person`** class
    This method matches the given **`individual`** with the person defined in **`this`** object, and returns a real number N to indicate the difference. The number N is calculated as:
    $$N = N_1 + N_2 + N_3$$
    , where N_1 equals to 0 if the gender matches and 100 otherwise, N_2 is the absolute difference between the age ranges, and N_3 is the absolute different between the heights of the **`individual`** with the person defined in **`this`** object.[2]
  - **`public double match(Location loc);`** // in the **`Location`** class
    Assume that the locations are defined by the (x, y) coordinate in a 2D map, this method determines the distance between the given **`loc`** with the location defined in **`this`** object using the following formula:
    $$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$
    , where (x_1, y_1) is the location defined in **`loc`** and (x_2, y_2) is the location defined in **`this`** object. Please be careful when comparing variables of type **`double`**, as they might have many points of precisions after arithmetic calculations.[2]
  - **`public int match(Time t);`** // in the **`Time`** class
    This method computes and returns a score indicating how close is the time **`t`** to the time defined in **`this`** object. The score is defined as follows:
    $$score = \begin{cases} 0, & d = 0 \\ 1, & 0 < d \leq 1 \\ 2, & 1 < d \leq 4 \\ 3, & d > 4 \end{cases}$$
    , where d is the absolute time difference in hours.[2]
- adding necessary accessors (**`get`** methods) and mutators (**`set`** methods);

[2] These methods may be called by both the Detective and the Oracle classes, potentially with data from either file.

- adding other methods or data members on an as needed basis

You next task is to design and implement the **`Detective`** class, including:

- adding necessary data members.
- completing the following public methods and the constructor;
    - **`public Detective (Oracle ora);`**
      This method initializes all data members as well as the reference to the Oracle, **`theOracle`**.
    - **`public boolean check (String line);`**
      This method checks a suspect and analyzes the murder case based on the response from the Oracle. The information of the suspect, **`line`**, in the string format is provided by the **`Driver`** class. For each suspect, the method may query the oracle with an analysis based on information from previously examined suspects. The goal is to determine the murderer, the location, and the time by checking as few suspects as possible and querying the Oracle as little as possible.
      Hint: knowing the time difference returned by **`Oracle.checkTime()`**, the detective can rule out certain suspect times. Similarly, the detective can rule out certain cases or derive clues based on the score and the distance returned by **`Oracle.checkPerson()`** and **`Oracle.checkLocation()`**, respectively.
    - **`public String toString ();`**
      This method provides the string representation of the murder information identified by the detective. Please see the sample output for the format.
- adding necessary private methods to support the algorithm implemented in **`check()`**.

Sample input files are provided for testing purpose. In the input file for the Oracle (*oracle.txt*), the murder case is described as:

```
M, 20, 1.75; 50, -144; 19:30
```

Given the suspects in *suspects.txt*, the output of your program should be the following:

```
Number to times the Oracle is queried: 13
Number of suspects checked: 6
Detective found:
Person: male, 20's, 1.75m
Location: 50, -144
Time: 19:30
```

Note the numbers in the above sample output are based on a version of our solution. The actual number of queries and suspects that your program displays depends on your algorithm and the input file you are using. Although the information found by the detective is the same as the information provided to the Oracle in *oracle.txt*, the detective must find this information from examining the suspects from the *suspects.txt* and querying the Oracle, not by reading the answer from *oracle.txt*. You may assume that there will be no more than 20 suspects in the input file. **Your program will receive higher score for checking less number of suspects and querying Oracle less. Please see the marking key for specific details.**

Programming Requirements:

In this assignment, you **may**:

- modify the **`Detective`**, **`Person`**, **`Location`**, and **`Time`** classes, including adding new data members and methods
- create your own input files to test difference scenarios
- implement the bodies of the existing methods in **`Detective`**, **`Person`**, **`Location`**, and **`Time`** classes
- write your own version of the **`Driver`** class to test different parts of the program

You **may NOT**:

- change the method signature of the given public methods in the **`Detective`**, **`Person`**, **`Location`**, and **`Time`** classes
- change the **`Driver`** and **`Oracle`** classes provided. Your submissions will be marked using these classes with different input files.

External libraries/methods that can be used (you don't write yourself)

You can use pre-created code for console input and output (*e.g.*, **`print`** and **`Scanner`**), classes for file input and to methods convert from **`String`** to numeric types. You may also find the java **`Math`** library useful.

Important reminders

- **Assignment Guidelines:** When working on your assignments, please adhere strictly to the generic assignment submission guidelines (which apply to all assignments) as well as the ones listed in this assignment.
- **Handing in your assignment**: Use the UNIX 'submit' program. Make sure you submit *Detective.java*, *Person.java*, *Location.java*, and *Time.java*, with the exact file names. The submit command is:
  `submit –c 233 –a 4 Detective.java Person.java Location.java Time.java`
- **Collaboration**: Assignments must reflect individual work, **group work is not allowed in this class nor can you copy the work of others.**