Object Oriented Programming

Lab 2 - 11/02/2022

A massive outbreak involving several bacterial and viral infections has taken place in the state of New Jersey. Hundreds of people have been infected with these new mutations of diseases that humans were previously immune to. The administration of the Princeton Plainsboro Hospital has been swamped with the sudden influx of new patients, and since there are only two Doctors available on call - Doctor House and Doctor Wilson, they are having trouble managing it, as their older systems are not prepared to handle such a situation.
For this lab, you are tasked with creating the necessary classes and objects for the Princeton Plainsboro Hospital's administration to help them manage and coordinate the new diseases and patients, and divide them among the two available doctors.

Instructions:

1. Ensure that the name of the file is the exact same as the class contained within it, with the appropriate file extension.
2. Ensure that the name, return-type, and parameter-types of the methods are the exact same as given in the JavaDoc. In case of any mismatch, your test cases may fail, causing you to lose marks despite implementing the correct logic.
3. You are provided with a folder containing the following files:
    a. *EclipseSteps.pdf*: Contains all the steps you need to follow in order to import the project archive and test cases into Eclipse. The steps are the same as from previous labs.
    b. *Lab2_Student.zip*: Project archive file that you're supposed to import into Eclipse. It includes all the starter code you will need for this lab.
    c. *Lab2_TestCases.jar*: Test cases archive file that you're supposed to import into your Eclipse project and run as JUnit Test. There are a total of ten test cases within this archive.
4. A brief description of the classes and their fields and methods is outlined in this document, however you are advised to refer to the Javadoc for the detailed descriptions.

5. You will only be given 75 minutes to complete the lab. No requests for extensions in deadline will be entertained under any circumstances. Make sure you upload your submission sufficiently before the deadline to avoid any last minute server issues.

Class - Disease

Fields:

1) ***String* name** - To store the name of the disease.
2) ***String* treatment** - To store the name of the treatment used for this disease.
3) ***int* communicability** - To store an integer between 1 and 100 which indicates the infectiousness of the disease.
4) ***String[]* symptoms** - A String array to store the list of symptoms that this disease presents with.

Methods:

1) **Disease(String name, String treatment, int communicability, String[] symptoms)** - Constructor to initialize the object.
2) ***String* getName()** - To get the name of the disease.
3) ***String* getTreatment()** - To get the treatment of the disease.
4) ***boolean* isInfectious()** - Returns *true* if communicability > 42
5) ***boolean* symptomsInclude(String symptom)** - Returns *true* if the symptom passed as parameter is a member of the symptoms[] array of the object.

Class - Patient

Fields:

1) ***String* name** - To store the full name of the patient.
2) ***int* ID**- To store the ID of the patient. This value will be used to assign a doctor to the patient while writing the Hospital class.
3) ***String[]* symptoms** - To store the symptoms that the patient has presented with.
4) ***Disease* disease** - To store the disease that the patient has been diagnosed with. Initially set to *null*.

Methods:

1) **Patient(String name, int category, String[] symptoms)** - Constructor to initialize the fields of the Patient object with the corresponding values. The field "*Disease* disease" is initialized as *null*.
2) ***String* getName()** - To get the name of the patient.
3) ***String[]* getSymptoms()** - To get the symptoms that the patient has presented with.
4) **int getID()** - To get the ID of the patient. This value will be used to assign a doctor to the patient.
5) ***Disease* getDisease()** - To get the diagnosed disease.
6) ***void* setDisease(Disease d)** - To finalize the diagnosis of a patient and set their disease.
7) ***boolean* hasBeenDiagnosed()** - Returns whether the patient has been diagnosed with a disease or not, based on whether the *disease* field has a value or not.

Class - Doctor

Fields:

1) ***String* name** - To store the full name of the Doctor.

Methods:

1) **Doctor(String name, int specialization)** - Constructor to initialize the name and specialization fields of the Doctor object.
2) ***String* getName()** - To get the name of the Doctor.
3) ***Disease* differentialDiagnosis(Patient p, Disease[] d)** - This function receives a Patient object *p*, and an array of Disease objects *d[]* as its parameters. It extracts the list of symptoms from *p* by using the appropriate getter function, and then it loops through each disease in d[]. Within the loop, it matches each symptom from *p* with the symptoms of the disease,, and maintains a score.
   For example, if *p* has the symptoms ["cough", "fever", "headache"], and d[0] has the symptoms ["cough", "fatigue", "fever"], then the score associated with d[0] will be 2. The reason is that - out of the 3 symptoms that the patient has presented with, "cough" and "fever" match with the symptoms within the Disease object at d[0], hence score = 2.
   Score for all Disease objects in d[] will be calculated, and the disease with the maximum score - that is, the maximum number of symptom-matches with the symptoms in p - is evaluated and returned by this function.

Class - Hospital

Fields:

1) ***Patient[]* patients** - To store the patients currently admitted into the Hospital.
2) ***int* numPatients**- To store the current number of patients added to the array. Initially set to 0, but when a Patient object gets added to the patients[] array, it gets incremented.
3) ***Disease[]* diseases** - To store the list of diseases that can be treated by the Doctors available on call.
4) ***int* numDiseases** - To store the current number of diseases added to the array. Initially set to 0, but when a Disease object gets added to the diseases[] array, it gets incremented.
5) ***Doctor* House** - To store an instance of the Doctor object. House is a doctor of category 1.
6) ***Doctor* Wilson** - To store an instance of the Doctor object. Wilson is a doctor of category 2.

Methods:

1) **Hospital() -** Initialize the two arrays with size 10 each, and initialize House and Wilson as Doctor objects such that House has the name: “Gregory House” and category 1, while Wilson has the name: “James Wilson” and category 2.
2) ***Patient[]* getPatients()** - To get the patients array.
3) ***int* getNumPatients()** - To get the numPatients value.
4) ***Disease[]* getDiseases()** - To get the diseases array.
5) ***int* getNumDiseases()** - To get the numDiseases value.
6) ***void* addPatient(String name, int category, String[] symptoms)** - To create a new Patient object and store it in the patients[] array.
7) ***void* addDisease(String name, String treatment, int communicability, String[] symptoms)** - To create a new Disease object and store it in the diseases[] array.
8) ***Doctor* assignDoctor(Patient p)** - This function receives a Patient object *p* as a parameter. It checks the *ID* field of p, and if it is odd, then this function returns a reference to the global object *House*, and if the ID is even, then this function returns a reference to *Wilson*.
9) ***void* diagnose(Patient p)** - This function receives a Patient object *p* as a parameter. Firstly, it checks if the patient has been diagnosed already or not. If yes, then it exits the function. If the patient has not been diagnosed, then it uses the *assignDoctor* function to get a reference to the *Doctor* object which will be

used for this patient. This assigned doctor's object is now used to execute its method *differentialDiagnosis(Patient p)*, and the return value of that function is stored in the *disease* field of *p*.

10) **String treatment(Patient p)** - It checks whether the patient *p* has been diagnosed or not by calling the corresponding function.
    If the patient hasn't been diagnosed, it returns the string "Not diagnosed yet."
    If the patient has been diagnosed, then it looks at the *Disease* object in the disease field of p, and returns its *treatment* field.

Hints

1) To check for equality between two String values, for example String s1 and String s2, use the following function:

```
boolean equality = s1.equals(s2);
```

2) If you ever encounter a NullPointerException while testing, check if there is any part of your code that involves accessing a null object. This could be due to improper initialization or not having appropriate null checks.

Marking Scheme

	Method	**Marks**
1.	Patient - getName()	0.25
2.	Patient - getSymptoms()	0.50
3.	Patient - getID()	0.25
4.	Disease - symptomsInclude()	1.25
5.	Disease - isInfectious()	0.25
6.	Hospital - getNumPatients()	0.25
7	Hospital - getPatients()	0.50
8	Hospital - getNumDiseases()	0.25
9	Hospital - getDiseases()	0.50
10	Hospital - diagnose()	2.5
11	Hospital - treatment()	1
12	Doctor - differentialDiagnosis()	2.5