EECS 2030: Programming Exam 2
Winter 2022

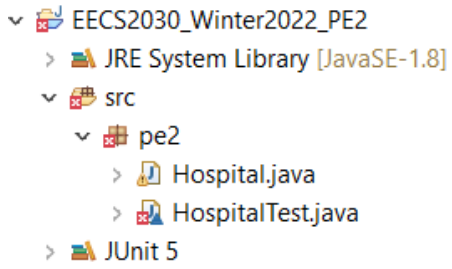
Deadline: Sunday, April 10, 2021 11:59 PM

In this programming exam, we are going to assess your understanding of

- Objects relationships (such as inheritance, composition, and aggregation).
- The difference between interfaces, abstract classes, and classes.
- Polymorphism
- Exception handling
- Encapsulation
- Java documentation API and the ability to create JavaDoc

Setting up the environment:

- Download the attached zip file.
- Open eclipse.
- Click on File and select Import
- Choose Existing Projects into Workspace and click Next
- Click on Select Archive File and then Browse.
- Find the zip file that you have already downloaded and click Finish
- Please make sure that you do not already have a project called EECS2030_Winter2022_PE2
- Now you should see the following settings in the package explorer.



Before you start, I would like to remind you that this assignment is different from the labs you have done until now, as this assignment is an **exam** that you take home, which means you can neither discuss your solution with peers nor get help from them. Therefore, you need to carefully read the statement at the top of **`Hospital.java`** before starting the programming test2 (PE2). When you finish solving PE2, you need to sign this statement by writing your information (full name and student number and section) in the gap provided. Please be aware that your PE2 solution **WILL NOT** be marked if you do not sign the declaration statement. Also, I would like to inform you that your code will be checked by both a plagiarism detection tool and us to make sure your solution to this PE2 is unique and not similar to others.

Refusing to follow the policy stated above is a violation of academic integrity that will have serious consequences. Please read about the definition of academic integrity and the penalty you may face if you violate academic integrity https://lassonde.yorku.ca/student-life/academic-support/academic-honesty-integrity.

Important Reminders

- You can submit your **PE2** work in eClass any time before 23:59 on Sunday (**April 10, 2022**). Your last submission will overwrite the previous ones, **and only the last submission will be graded**.
- The deadline is strict with no excuses: **you receive 0 for not making your electronic submission in time**. Emailing your solutions to the instructors or TAs will not be acceptable
- To submit your work, you need to use the York eClass.
- Do not copy or look at specific solutions from the net.
- Academic honesty will be strictly enforced in this course. Specifically, direct collaboration (e.g., sharing code or answers) is not permitted, and plagiarism detection software will be employed. Students are expected to read the Senate Policy on Academic Honesty. See also the EECS Department Academic Honesty Guidelines.
- Please note that encapsulating data is essential in **PE2**. Your code will be evaluated on the correct implementation of encapsulation. Since the testing is done **automatically**, improper encapsulation implementation might affect the successful execution of other test cases.
- **Your code should include Javadoc comments.**
- The **PE2** will be graded **not only by JUnit tests given to you** but also by additional tests covering some other input values. This is to encourage you to take more responsibility for the correctness of your code by writing your tests using custom testers that can handle the boundary cases and error conditions.
- *It would be best to run the JUnit tester* **HospitalTest.java** *after completing all the required classes to check your work. Nonetheless, passing all given tests does not guarantee full marks for this* ***PE2****. Therefore, you are required to write additional tests to ensure the correctness of your implementations.*

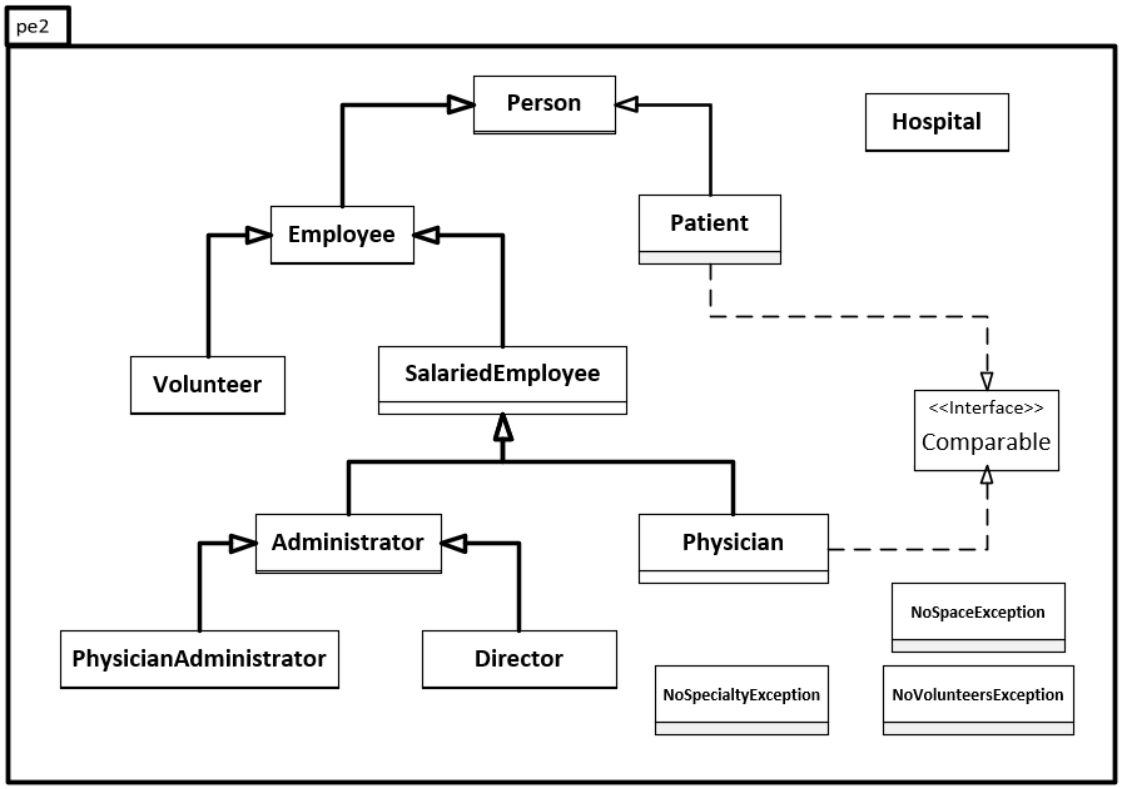
Marking Scheme:

- **[80 points]:** Your code will be tested with test cases. All these 80 points are lost if the code has compilation errors. Hence, Please note that you should not submit a code that contains compiler errors.
- **[20 points]:** For inner documentation, Javadoc and proper code style.

Problem Description:

For this programming test2, you must write an object-oriented program that implements a virtual hospital scenario. The following UML diagram shows an overview architecture of the virtual hospital. The UML diagram shows the inherence relationship among the classes. **It should be noted here that this UML diagram is not complete**. Hence, **you need to read the problem description carefully to determine the attributes and methods for each class. Also, you need to read the JUnit test file `HospitalTest.java`.**
Moreover, you need to determine if the class is **abstract** or **concrete** based on the problem description. Therefore, if you look at the startup PE2 project, you will find the package **pe2** with `Hospital.java` file, which does not contain any functional code, as you are supposed to read this diagram and implement the code in `Hospital.java`. **You must create your classes inside the `Hospital.java file` after reading the problem description.**

Remember that you need to follow the best practice while implementing the solution for the virtual hospital scenario (i.e., object-oriented encapsulation, abstraction, information hiding, ... etc.)



General information and Configuration about the Virtual Hospital:

- A Hospital has a team of **Physicians**, **Administrators**, **Volunteer** workers.
- A Hospital admits and provides treatment for **Patients**.
- A **volunteer** who works in the hospital is considered an **employee**, but volunteers are not paid for their work.
- The hospital stores the following information about any **volunteer**: ***First name***, ***Last name***, ***Age***, ***Gender***, ***Employee ID***, and ***Address***
- The hospital stores the following information about any **employee**, **physician** or **administrator**, ***First Name***, ***Last name***, ***Age***, ***Gender***, ***Employee ID***, ***Address*** and ***Salary***.
- The hospital assigns a ***unique*** employee ID to each ***physician***, ***administrator*** or ***volunteer*** worker starting from 100.
- The hospital stores the following information about any **patient**: ***First Name, Last name***, ***Age***, ***Gender***, ***Patient ID*** and ***Address***.
- The hospital assigns a ***unique*** Patient ID to each patient starting from 1000.
- The hospital is not allowed to have duplicate patient records.
- A Patient has a designated Physician assigned.
- A Physician can have several Patients assigned at the same time.
- Any volunteer worked in the hospital can be assigned to any physician.
- There are two types of administrator roles: **director** and **physician administrator**.
- There is only one director for the hospital. The director manages all the physician administrators.
- There are three types of physician specialties in the hospital: **Immunology**, **Dermatology**, and **Neurology**.
- The number of physician administrators in the hospital is equal to the number of specialties, and a physician administrator manages only physicians with the same specialty.

It is essential to know that the hospital has the following restrictions:

- Number of Directors: 1
- Number of Physician Administrators: 3
- Number of Physicians assigned to a Physician Administrator: Up to a maximum of 25
- Number of Physicians: Up to a Maximum of 70
- Number of Volunteers: Up to a Maximum of 150
- Number of Patients: Up to a Maximum of 500
- Number of Volunteers assigned to a Physician: Up to a maximum of 5
- Number of Patients assigned to a Physician: Up to a maximum of 8

Task1:

- Read the problem description and check the Junit test cases to identify all the implicit and explicit entities stated in the scenario and design a suitable class hierarchy for the virtual hospital. Add appropriate constructors, accessor/getter, mutator/setter methods to provide basic functionality to the classes. Also, appropriately redefine equals and toString methods while using the class attributes.

Task2:

Implement following operations relevant to the appropriate classes you have defined.

- **AdmitPatient**: when the hospital admits a new patient
    a. You need to add the patient information to the hospital patient record
    b. You need to assign a physician to every newly admitted patient, and then the patient will be added to the corresponding physician record.
    c. The patients are assigned to physicians on a **first-come-first-serve** basis.
    d. The hospital should not admit the same patient twice (i.e., no duplicate patient in the hospital patient record)
- **DischargePatient**: when the hospital discharges a patient (patient discharged from the hospital when patient treatment is completed), you need to clear/remove the patient information from the hospital patient record.
- **HireVolunteer**: when the hospital hires a new volunteer, you need to store volunteer information in the hospital volunteer record and then assign the volunteer to a physician who has not exceeded the volunteer limit yet.
    a. The hospital cannot hire the same volunteer twice. Hence, there is no duplicate.
    b. The volunteers are assigned to physicians on a **first-come-first-serve** basis.
- **ResignVolunteer**: when a volunteer is resigning from the hospital, you need to clear/remove the volunteer information from the hospital volunteer record.
- **ExtractAllPatientsDetails**: you need to extract and return all the patient information stored in the hospital patient record as a sorted list of patients according to the patient's full name.
    a. The patient's full name is defined as a concatenation of first name and last name as follows: if a patient's first name is "**Mark**" and the last name is "**Smith**", the patient's full name is "**Mark, Smith**".
- **ExtractAllPhysicianDetails**: you need to extract and return all the physician information stored in the hospital physician record as a sorted list of physicians according to the physician's full name.
    a. The physician's full name is defined as a concatenation of first name and last name as follows: if a physician's first name is "**Chris**" and the last name is "**Johnny**", the physician's full name is "**Chris, Johnny**".

- **ExtractAllVolunteersDetails**: you need to extract and return all the volunteer information stored in the hospital volunteer record as a list of volunteers.

- **ExtractPatientDetails:** you need to extract and return all the patient information assigned to a particular physician as a sorted list of patients according to the patient's full name.
    a. The patient's full name is defined as a concatenation of first name and last name as follows: if a patient's first name is "**Mark**" and the last name is "**Smith**", the patient's full name is "**Mark, Smith**".
- **ExtractVolunteerDetails:** you need to extract and return all the volunteer information assigned to a particular physician as a list of volunteers

- **ExtractPhysician**: you need to extract and return all the physician information belonging to a particular specialty as a sorted list of physicians according to the physician's full name.
    a. The physician's full name is defined as a concatenation of first name and last name as follows: if a physician's first name is "**Chris**" and the last name is "**Johnny**", the physician's full name is "**Chris, Johnny**".
- **HirePhysician**: when the hospital hires a new physician
    a. You need to add the physician information to the hospital physician record.
    b. The newly hired physician is assigned to the designated physician administrator, and then the physician will be added to the corresponding physician administrator record with the same specialty.
    c. The hospital cannot hire the same physician twice. Hence, there is no duplicate.

- **ResignPhysician**: when a physician is resigning from the hospital
    a. You need to remove the Physician information from the hospital physician record.
    b. The designated physician administrator will assign the patients of the resigned physician to the next available physician (or physicians).
    c. Reassign the volunteers for the next available physician (or physicians).
    d. When reassigning patients and volunteers, there can be a scenario where all the physicians in the hospital are not available to handle the remaining patients, or no volunteers are needed. At this point, ignore these specific cases and implement this operation, assuming these steps can be achieved successfully.

- **Optional**: It is highly recommended that you write and add your Junit test cases to test and show the output of the above implementations.

Task 3: Handle Special Cases

In the virtual hospital scenario, there can be exceptional cases/situations that rarely happen. Instead of handling them with logical conditions, you need to define specific exceptions and use them within the program.

a. **Case1**: if the hospital reaches the maximum capacity for patients (i.e., you can not admit any new patient), you need to handle this situation by using exceptions.
b. **Case2**: if a physician is resigning from the hospital, but no other physician is available with the same specialty, you need to handle this situation by using exceptions.
c. **Case3**: if a volunteer is resigning from the hospital, but no other volunteer is available under supervision/work with a particular physician, you need to handle this situation by using exceptions.

- **Optional**: It is highly recommended that you write and add your Junit test cases to test and show the output of the above implementations.

Submit your work by using the course eClass

Check List:

Before submitting your files for PE2, you need to make sure you completed the following

	I have signed the statement on academic honesty.
	TASK1 implemented and tested
	TASK2 implemented and tested
	TASK3 implemented and tested
	There is No compilation error generated from your implementation
	The **Hospital.java** file contains the implementation for all PE2 tasks and Classes.

What and how to Submit:

- You only submit one file, which is the completed **Hospital.java** file.
- Submit your solution in eClass, where you have downloaded these instructions.