Object-Oriented Programming. September 2021

NOTES FOR THE EXERCISE

- It is assumed that students have verified whether they meet the requirements to take advantage of the continuous assessment modality. If not, in addition to the practical exercise, you must take an additional questionnaire. If this is your case, please, contact the teaching staff.
- During the exam, you can consult the class notes (in the CV), the Java API, and, of course, all the help provided by the programming environment.
- It is not allowed to add variables, constants, or non-private methods to the classes, nor to modify the visibility of variables, constants and methods that appear in the UML diagram.
- It is allowed to add private methods to classes.
- Once the exercise is finished, you must upload to the task created in the virtual campus a compressed file with the 'src' folder of your project.
- The evaluation will take into account the clarity of your algorithms and code code, and the correct choice of data structures, as well as the design criteria that favors reuse.

In this exercise we will create an application to manage patients admitted to a hospital in a `prHospital` project. To do this, we will create a `hospital` package that will include a `Criterion` interface and classes `HospitalException`, `Patient`, `Room`, `BornBefore`, `SameFloor`, `Hospital` and `HospitalPlus`, and classes `MainPatient`, `MainHospital` and `MainHospitalPlus` in the anonymous package.

A hospital will have a number of floors, and on each floor a number of rooms. The management of occupied and unoccupied rooms will be done as follows: Each patient will be assigned a room, and we will have a set with all the free rooms.

Class `HospitalException` (0.25 pts.)

All exceptional situations in this exercise will be handled using a *checked* `HospitalException` class.

Class `Patient` (1.5 pts.)

A patient has a first name (of type `String`), a surname (of type `String`), a social security number (SSN, of type `String`) and a year of birth (of type `int`). An SSN is a ten digit number. The first two digits of these ten are the code of the province where the number has been requested. The next six digits are an assigned sequential number. The last two digits are control digits, which are calculated as the remainder of dividing the number formed by the first eight digits by 97. For example, if we consider the number 08 123456 xx, the remainder of dividing 08123456 by 97 is 94, which will be the control digits. That is, the

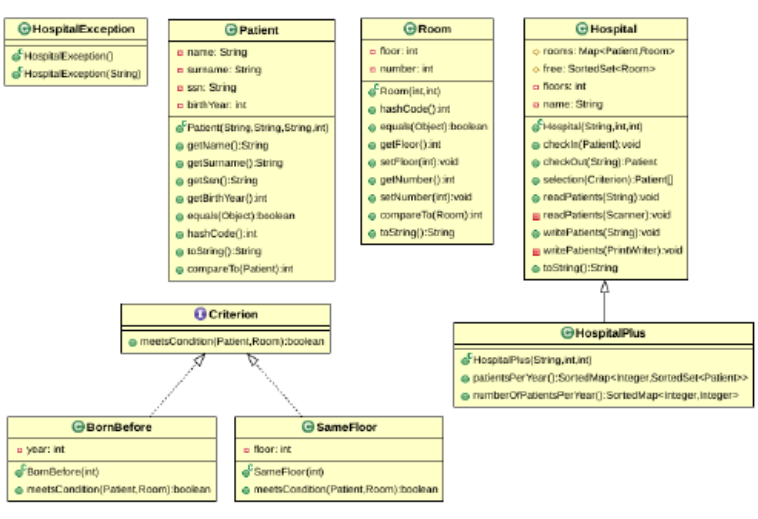

Figure 1: UML class diagram

number would definitely remain as 08 123456 94. If the remainder is less than 10, a 0 is placed before it to convert the check digits into a two-digit figure. For example, if the remainder were 3, it would be 03; if it were 0, it would be 00.

The class must provide:

1. **(0.5 pts.)** A constructor `Patient(String name, String surn, String numb, int year)` that receives as arguments, in this order, the patient's first name, last name, SSN, and year of birth. The constructor will make the necessary checks to ensure that every instance meets the following conditions, throwing an exception otherwise. The first name, surname and SSN of a patient cannot be `null`, the first name and surname cannot be empty, and the SSN will be given by a 10-digit string. The SSN will be verified to be valid (using the control digits as indicated above).
2. The class provides methods to obtain the values of each of the attributes of a patient.
3. **(0.5 pts.)** Two objects of the `Patient` class are considered *equal* if they have the same first name and surname (case insensitive), the same SSN, and the same year of birth.
4. **(0.5 pts.)** A patient is considered older than another if his/her year of birth is earlier; if they were born in the same year, if their SSN is greater; if their SSNs are also equal, then their surnames will be compared, and finally their first names.

5. A redefinition of the `toString()` method that represents the state of the object as a character string. The format will be as in the following example:

   ```
   Patient [name=Alejandro, surname=Dumas, ssn=6503158367, birthYear=180
   ```

Class `MainPatient`

Use the provided `MainPatient` class to test the basic functionality of the `Patient` class implemented in the previous exercise. The expected output can be found as a comment at the end of the file.

Class `Room` (0.25 pts.)

An instance of the `Room` class has a floor (floor where the room is located) and a number (number of the room within that floor). The class will have:

- A constructor `Room(int floor, int number)` to create rooms.
- Methods to obtain and modify the values of each of its attributes.
- Two rooms are equal if they have the same number and are on the same floor.
- The `Room` class will provide a natural order: A room will be greater than another if it is on a higher floor, or if it has a higher number if they are on the same floor.
- The class will also provide a redefinition of the `toString()` method that represents the state of the object that receives the message as a character string. The format will be as in the following example:

  ```
  Room [floor=12, number=34]
  ```

Interface `Criterion` and Classes `BornBefore` and `SameFloor` (0.5 pts.)

The `Criterion` interface provides a single `boolean meetsCondition(Patient p, Room r)` method that provides a predicate for testing a condition given an instance of the `Patient` class and one of the `Room` class.

The `BornBefore` and `SameFloor` classes implement the `Criterion` interface and provide implementations of the `meetsCondition(Patient p, Room r)` method.

Specifically, the `BornBefore` class provides:

1. A constructor `BornBefore(int y)` with the reference year as an argument.
2. A method `boolean meetsCondition(Patient p, Room r)` that returns `true` if the patient passed as an argument was born in a year previous to the one of reference, or `false` otherwise.

Similarly, the `SameFloor` class provides:

1. A constructor `SameFloor(int f)` with the reference floor as an argument.

2. A method `boolean meetsCondition(Patient p, Room r)` that returns `true` if the room passed as argument is on the reference floor, or `false` otherwise.

Class `Hospital` (4.75 pts.)

An instance of the `Hospital` class will have a `name` (of type `String`), a number of floors (of type `int`) and patients in rooms. Specifically, a `rooms` variable of type `Map<Patient, Room>` associates a room to each of the patients in the hospital. The unoccupied rooms will be kept in an ordered set of `free` rooms (of type `SortedSet<Room>`).

The class will provide:

1. **(0.75 pts.)** A constructor `Hospital(String name, int floors, int rooms)` that takes as arguments the name of the hospital, and the number of floors and rooms per floor with which the hospital is created. The name cannot be `null` or empty, and the number of floors and rooms per floor must be positive. The creation of a hospital implies the creation of each of its rooms (according to the number of floors and rooms per floor). Initially, there are no patients in the hospital, and all rooms will be free.
2. **(0.75 pts.)** A method `void checkIn(Patient p)` that admits a new patient to the hospital. The patient cannot be `null`, nor already in the hospital, and of course the hospital cannot be full. In any of these situations an exception will be thrown. When a patient is admitted, he/she is assigned a free room. Obviously, when assigning a free room to a new patient, that room is no longer free.
3. **(0.75 pts.)** A method `Patient checkOut(String ssn)` frees the room occupied by the patient with the passed SSN as the argument. The patient that leaves the hospital is returned as a result. If there is no patient with that SSN, nothing is done, and `null` is returned as a result.
4. **(1 pt.)** The method `Patient[] selection(Criterion c)` returns an array with the patients that meet the criterion provided as an argument. The method will return a complete array (without `null` positions) with all the hospital patients that meet the condition established by the criterion given as an argument.
5. **(1 pt.)** The methods `readPatients(String filename)` and `readPatients(Scanner sc)` will read information about patients, respectively, from a file and from a scanner, and will load it into the hospital structures. Please note the format of the provided `patients.txt` file. In short, each line of the file contains information about a patient. For instance:

   ```
   Antonio,Machado,0673932455,1875
   ```


   The different information elements for each patient are separated by commas (the regular expression to use to specify the separators could be `"\\s*[,]\\s*"`). If the information on a patient is incorrectly formatted or contains incorrect data, that patient's information will be discarded and the process will continue with the rest of the input.
6. **(0.5 pt.)** The methods `writePatients(String f)` and `writePatients(PrintWriter pw)` will write the information about the hospital patients, respectively, to a text file and to a `PrintWriter` object. Please note the format of the provided `output.txt` file. In summary, the information of each of the patients will be written in one line, grouped by plants. First all the patients on floor 0, then those on floor 1, etc.
7. A redefinition of the `toString()` method that represents the state of the object as a string. The format will be similar to that of the `Patient` and `Room` classes:

   ```
   Hospital [name=<name>, rooms=<toString-of-rooms>, free=<toString-of-free>]
   ```

Class `MainHospital` (0.75 pts.)

Implement the `MainHospital` class to test the basic operation of the `Hospital` class implemented in the previous exercise. Specifically, the program must create a hospital, load in it the patients from the file `patients.txt`, and use the method `writePatients(String)` to create a file `output.txt` like the one provided, where the hospital patients are grouped by floors. In addition, the program will show the status of the hospital on the console, the array with patients born before 1800, and the array with patients on the 2nd floor of the hospital.

Class `HospitalPlus` (2 pts.)

The `HospitalPlus` class implements hospitals with the functionality of any hospital, but also provides the following:

1. **(0.25 pts.)** A constructor `HospitalPlus(String name, int floors, int rooms)` that initializes the hospital.
2. **(1 pt.)** The `patientsPerYear()` method returns a map of type `SortedMap<Integer, SortedSet<Patient>>` that associates to each year an ordered set with the patients born in that year.
3. **(0.75 pts.)** The `numberOfPatientsPerYear()` method returns a map of type `SortedMap<Integer, Integer>` that associates to each year the number of hospital patients born in that year.

Class `MainHospitalPlus`

Use the provided `MainHospitalPlus` class to test the basic functionality of the `HospitalPlus` class implemented in the previous exercise.