SimErgy - Simulator for Emergency Department

Overview:

In this project, we are going to develop a Java application to simulate the functioning of an Emergency Department.

The implementation uses the patterns and methods we have learned during our oriented object programming courses.

1. Introduction:

Emergency department (ED) is the service within hospitals responsible for providing un-scheduled care to a wide variety of patients (life-threatening and other emergency cases) over 24 hours daily, 7 days a week. It plays a key role in patient safety and public health. The project SimErgy consists in implementing a simulator which reproduces the ED in the best way.

In the first part, the core implementation will be presented, with the design choices, the global architecture and, eventually, the different classes’ structures. Our choices and difficulties encountered will also be detailed.

In the second part, we will show how we implemented the user interface.

**Part I**

**SimErgy core**

1. Design
   1. Design of the packages

To implement this application, the object oriented programming requires a lot of classes, making the global structure messy if it isn’t well organized. We then decided to put classes relative to the same functionality in a same package.

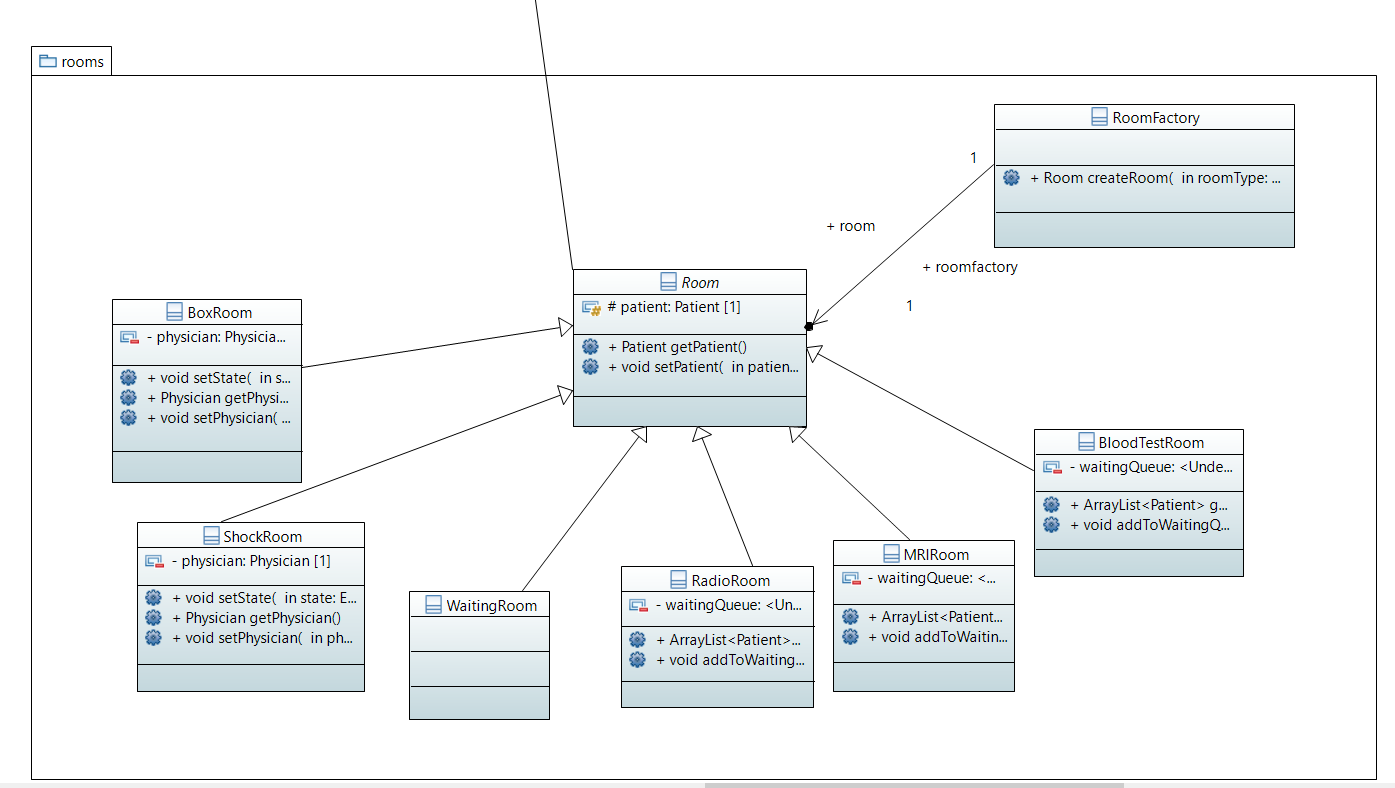
After we decided what classes we needed to implement, we regrouped classes in package depending on what they are used for.

The human package contains the human resources’ classes (nurse, physician and transporter) as well as the patient class. The rooms’ package contains the different rooms, the events’ package contains the events and so on. The package “others” contains all the functionalities that are necessary for the functioning of the system, but don’t belong to other packages. The last package contains all the tests.

* 1. Design Patterns

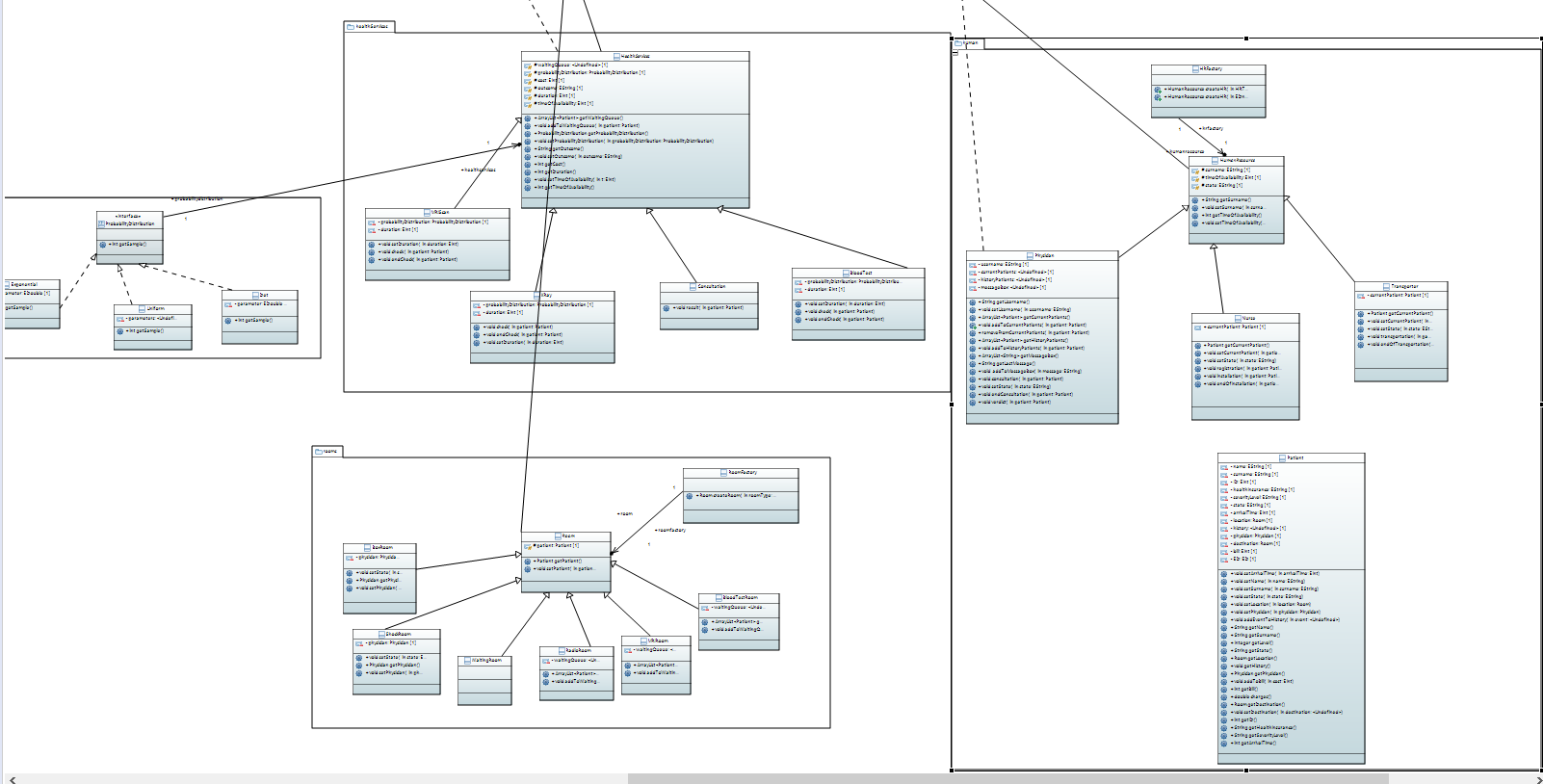
To make future code improvements and features implementations easier, we used some of the patterns studied in class:

* First of all, we used the Factory Pattern for the resources that are instanced by a string. This is why we decided to use this pattern for the human resources, the rooms and the events. This pattern mad a lot of thing easier afterwards, since every time we needed and instance of any resource or event we just had to type the type of the resource or the event. It also made the system of events following a queue possible.



* We also considered that using a *Strategy Pattern* for the probability distribution was a really useful. Indeed, we had each time to choose between the probability distribution, and in order to do that, the best way was to implement three different classes which implement an interface with a unique method “getSample()”.
* We also thought about using the *Observer Pattern* for the notification on the message boxes of the physicians. However, we found out that this wasn’t necessary and that it wouldn’t help us that much afterwards.
  1. UML diagram

To visualize the main the global structure of our code, we made an UML Diagram which gives a good overview of the core system of our simulator. Since it is too big to get on a page, we just put here a significant part, and the real one can be found inside the project.



1. Class Structure

To implement and test these structures, our organization was to split the implementation into three tasks: designing, implementing and testing. We shared the tasks as follow:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Humans | Rooms | Health services | Probabilities | Events | Others |
| Design | Léopold | Wassim & Hamza | Wassim & Hamza | Léopold | Hamza | Wassim & Hamza |
| Implementation | Wassim | Hamza | Hamza | Wassim | Hamza & Léopold | Hamza |
| Test | Wassim | Hamza | Léopold | Léopold | Wassim | Hamza |

* 1. Human package

The human package is constituted by two categories: he human resources’ classes and the patient’s class.

Human resources:

The human resources have many things in common, and are classes that realize actions in the ED. For instance, physicians can give a verdict and consult a patient, whereas nurses can install a patient in an empty room. These classes are instanced by a factory.

Patients:

The patient class contains a lot of information about the patient. We can have access to his severity level, his arrival time, and his health insurance and so on.

* 1. Room package

This package implements the different rooms we have access to in an Emergency Department. All the rooms are instanced by a factory. There are two types of rooms: the health service rooms and the consultation rooms.

Consultation rooms:

They are the place where the consultations happen after a patient is installed. They need a physician in order to assure the consultation.

Health Service rooms:

These rooms are pretty different because they have their own queue. These rooms are used to provide a health service, patients requiring it need to wait in the room waiting queue before receiving the service.

* 1. Health Services package

This package contains all the classes for the health services, that is to say consultation, MRI, Blood test and Radio. These services has random durations given by the probability distribution that is associated to them. They also implement a method which allow them to check a patient. As for the rooms and the human resources, they are instanced by a factory.

* 1. Events package

In order to have our simulator working, we had to implement a functionality that allow the events to happen.

We decided that we had to make a class for each event that changes the state of something inside the ED. So let’s take and example, when a patient arrives it triggers the *patientArrival* class, then if there is an idle nurse, this patient will be registered thanks to the *partientRegistration* class, and so on. The most important part here, is that we needed every event to be clear and to make a basic action, which is done when we launch the execute method.

Once we had created all our events, we had to find a way to make them actually happen when they are needed. So we implemented two static methods in Database, *updateEventQueue* and *updateEnabledEvents*. The second one checks the state of the ED and returns a list of authorized events, whereas the first one, after considering what events are possible, creates instances of these events and adds them to the event queue before sorting this queue.

* 1. Others package

This package contains two main classes for our simulator to work. First, there is the ED class. It contains a lot of attributes which are lists. These lists gives us an in-depth view of our ED, it contains information about all the system at any time. Thanks to this class, we can access to every resource we need in an ED, it also allows us to follow every patient from the moment they arrive until the moment they leave. Finally, it contains information on the events and which events are the one to come

The second most important class, is the Database class. We thought about this one after we noticed that we needed to be able to work on several EDs at the same time. This class only contains static information, especially the list of the generated EDs. It also implements a method that gets us the ED when we only have its name.

These two classes are the one that make our core system operational, since they’re the ones that regroup all the information and put links between it.

1. Use Case Scenario

Once we had our core system, we implemented a use case scenario to test our functionalities as a whole.

**Part 2**

**Simergy interface**

**To Launch the simergy interface :**

**Run “java –jar ED.jar“**

The main class that handles the Simergy is “CLI”. it loads the my\_simergy.ini or the testScenario.txt files if the user wants to.

We needed to implement a few more classes to make the CLI meet the project requirements.

**ReadFile class:**

The class that reads my\_symergi.ini file and creates the EDs with an initial configuration.

**PatientGeneration class:**

This class has a method that generates an arrival time for a patient with a given severity level.

When an ED is created, 5 patients (one for each severity level) are created and put in an array that contains the patients that will be generated by the command “**addPatient**”.

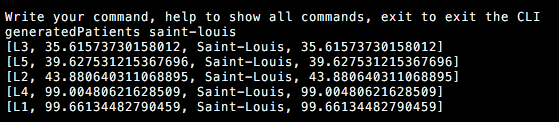
This array is sorted in the increasing order, so that the patient with the shorter arrival time arrives first. When **“addPatient**”.is executed, the patient is generated, and removed from to be generated patients array. Then the event PatientArrival is generated and added to the event queue, and another patient of the same severity level is added to the array of the patients that are waiting to be generated with a random arrival time depending on his severity level arrival distribution.

Obviously the array is updated after every event execution, this way the time of occurrence of a Patient arrival event is never smaller than the actual time of the simulation.

When the user sets a new arrival distribution for a severity level, the array is also updated with a new arrival time randomly generated following the new probability distribution.

*Example:*

We first display the list of the patients that are waiting to be generated. This list has been generated at the creation of the ED.

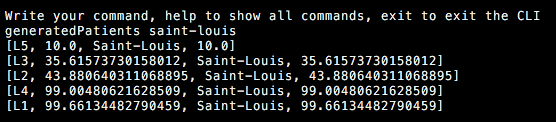
****

We then change the arrival distribution of the L5 patients.

The new proba distrubtion is now Det (10).

**../../Desktop/Capture%20d’écran%202017-12-24%20à%2015.08.28.png**

We display the new list of the patients that are waiting to be generated.

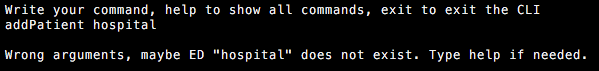
The list is updated, the next L5 patient will be generated following the Det (10) probability distribution

**Errors handling :**

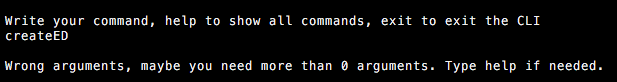
Syntax errors are correctly handled.

*Examples:*

Adding a patient to an non-existing hospital:



Trying to create an ED without a name :



Giving a string argument when it should be a number :

