Health Care System

02160 Agile Object-oriented Software Development

Report

Group members:

Pieter O'Hearn
Jack Rodman
Asger Conradsen
Kun Zhu
Anna Hogan
Karoline Østergaard

September 23, 2021

Professors: Kati Kuusinen, Andrea Burattin

Contents

1 Introduction

The purpose of this project was to create a hospital management system that allows users to preform basic tasks, such as admitting a patient, discharging a patient, adding a staff member, etc. The inspiration for this project was a project created in 2012 by the Finnish Christian Medical Society, that aims to help the Government of Tanzania to increase the quality of health services.

The hospital management system that was developed has four main functions. Those functions are patient registration, staff management, health facility management, and patient admission. The group also created a user interface, persistency layer, advanced query mechanism, and a user management login system that help make the program more user friendly.

The group created this project using an Agile mindset. Specifically, the LEAN framework was used. In the project management section, the choice of using LEAN is discussed thoroughly.

It is also discussed how the system is structured and designed. Furthermore, a detailed user manual is included. The user manual is a guide for the user, so that they can be successful when using the program.

2 System Design and Structure

2.1 UML Class Diagram

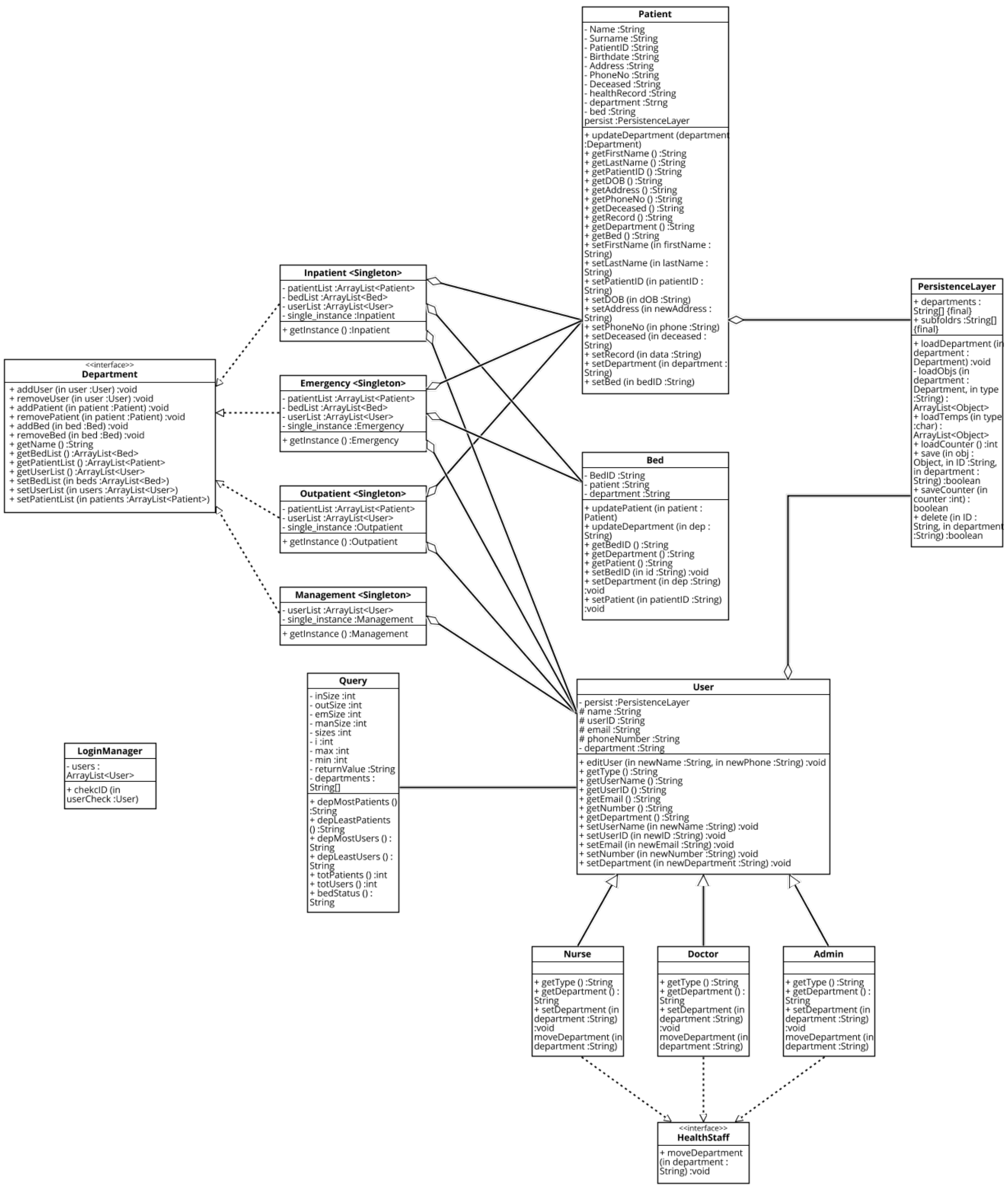
Patient
- Name :String
- Surname :String
- PatientID :String
- Birthdate :String
- Address :String
- PhoneNo :String
- Deceased :String
- healthRecord :String
- department :Strng
- bed :String
persist :PersistenceLayer
+ updateDepartment (department :Department)
+ getFirstName () :String
+ getLastName () :String
+ getPatientID () :String
+ getDOB () :String
+ getAddress () :String
+ getPhoneNo () :String
+ getDeceased () :String
+ getRecord () :String
+ getDepartment () :String
+ getBed () :String
+ setFirstName (in firstName : String)
+ setLastName (in lastName : String)
+ setPatientID (in patientID : String)
+ setDOB (in dOB :String)
+ setAddress (in newAddress : String)
+ setPhoneNo (in phone :String)
+ setDeceased (in deceased : String)
+ setRecord (in data :String)
+ setDepartment (in department : String)
+ setBed (in bedID :String)
PersistenceLayer
+ departments : String[] {final}
+ subfoldrs :String[] {final}
+ loadDepartment (in department : Department) :void
- loadObjs (in department : Department, in type :String) : ArrayList<Object>
+ loadTemps (in type :char) : ArrayList<Object>
+ loadCounter () :int
+ save (in obj : Object, in ID :String, in department : String) :boolean
+ saveCounter (in counter :int) : boolean
+ delete (in ID : String, in department :String) :boolean
Inpatient <Singleton>
- patientList :ArrayList<Patient>
- bedList :ArrayList<Bed>
- userList :ArrayList<User>
- single_instance :Inpatient
+ getInstance () :Inpatient
<<interface>>
Department
+ addUser (in user :User) :void
+ removeUser (in user :User) :void
+ addPatient (in patient :Patient) :void
+ removePatient (in patient :Patient) :void
+ addBed (in bed :Bed) :void
+ removeBed (in bed :Bed) :void
+ getName () :String
+ getBedList () :ArrayList<Bed>
+ getPatientList () :ArrayList<Patient>
+ getUserList () :ArrayList<User>
+ setBedList (in beds :ArrayList<Bed>)
+ setUserList (in users :ArrayList<User>)
+ setPatientList (in patients :ArrayList<Patient>)
Emergency <Singleton>
- patientList :ArrayList<Patient>
- bedList :ArrayList<Bed>
- userList :ArrayList<User>
- single_instance :Emergency
+ getInstance () :Emergency
Bed
- BedID :String
- patient :String
- department :String
+ updatePatient (in patient : Patient)
+ updateDepartment (in dep : String)
+ getBedID () :String
+ getDepartment () :String
+ getPatient () :String
+ setBedID (in id :String) :void
+ setDepartment (in dep :String) :void
+ setPatient (in patientID :String) :void
Outpatient <Singleton>
- patientList :ArrayList<Patient>
- userList :ArrayList<User>
- single_instance :Outpatient
+ getInstance () :Outpatient
Management <Singleton>
- userList :ArrayList<User>
- single_instance :Management
+ getInstance () :Management
Query
- inSize :int
- outSize :int
- emSize :int
- manSize :int
- sizes :int
- i :int
- max :int
- min :int
- returnValue :String
- departments : String[]
+ depMostPatients () :String
+ depLeastPatients () :String
+ depMostUsers () : String
+ depLeastUsers () : String
+ totPatients () :int
+ totUsers () :int
+ bedStatus () : String
LoginManager
- users : ArrayList<User>
+ chekcID (in userCheck :User)
User
- persist :PersistenceLayer
# name :String
# userID :String
# email :String
# phoneNumber :String
- department :String
+ editUser (in newName :String, in newPhone :String) :void
+ getType () :String
+ getUserName () :String
+ getUserID () :String
+ getEmail () :String
+ getNumber () :String
+ getDepartment () :String
+ setUserName (in newName :String) :void
+ setUserID (in newID :String) :void
+ setEmail (in newEmail :String) :void
+ setNumber (in newNumber :String) :void
+ setDepartment (in newDepartment :String) :void
Nurse
+ getType () :String
+ getDepartment () : String
+ setDepartment (in department :String) :void
moveDepartment (in department :String)
Doctor
+ getType () :String
+ getDepartment () : String
+ setDepartment (in department :String) :void
moveDepartment (in department :String)
Admin
+ getType () :String
+ getDepartment () : String
+ setDepartment (in department :String) :void
moveDepartment (in department :String)
<<interface>>
HealthStaff
+ moveDepartment (in department : String) :void

Figure 1: *The Main classes used in the system and their attributes and methods*

2.2 User Stories

In order to ensure that the program complied with the requirements of the future users. The code was tested against the following user stories:

Admission

As a User
I want to register a new patient
So that I can save them in the HMS
so they can be admitted

As a Nurse
I want to admit a patient to a bed
So that the patient is admitted to a bed
and the department's number of free beds is changed

As a Nurse
I want to discharge a patient
from my department
So that the patient is discharged
and the bed they are occupying becomes free

As a Nurse
I want to move a patient
to a different department
So that their bed becomes free
and they occupy a new one in the other department

As a Nurse
I want to move a patient
within my department
So that their old bed is freed
and they are assigned to a new bed

Advanced Query

As a User
I want to choose a statistic to see
So that the result of the
statistic is shown

As a User
I want to see which department has the most patients
So that I can make sure enough
staff members are working

Available Beds

As a Department
I want to know how many beds are available
So that I know if a new bed can be allocated

As a Department
I want see a list of beds in my department
So that I'm given a list of beds

Creating a profile

As a User
I want to register a patient with their non-medical data
So that a new patient is registered at the hospital

As an Admin
I want to register a new user

So that the user is added to the hospital

Editing Information

As a Nurse
I want to edit a patients information
So that health data is up to date

Persistency Layer

As a nurse
I want to move a previously admitted patient
So that all their information is already in the HMS

User Management

As a User
I want to change a users info
So that all information is saved in the HMS

As a User
I want to try to access a patients medical info
So that the system denies access

As an Admin
I want to access a patients info
So that I am allowed access

As a Nurse
I want to access an users info
So that I am denied Access

As a nurse
I want to access another departments patient
So that the system denies access

As a Admin
I want to access a users info
So that I am allowed access

User Interface

As a User
I want to Log in
So that I can view patients data

3 Description of Software Architecture and Design

3.1 Overall Design

Overall the system is built up around the User, Department and Patient classes, which allow you to add patients or users to a department. A patient can also be allocated to a bed. (The exact program structure can be seen in the UML Class Diagram on page 3). The LoginManager class allows the user to log in, and their level of access depends on the type of user. Furthermore, the persistency layer class saves all data that is entered and the Query class allows you to see various statistics about the hospital.

3.2 Object IDs

Users, Patients, and Beds created in the program are all initialised with an ID string to provide uniqueness and more organised indexing. An ID string consists of a letter specifying which type of object it is, followed by a number. As as example, the first user created would have the ID string: `"U1"`. These IDs are used to ensure that every object is uniquely identifiable, even if they have the same name. Furthermore, they are used by the persistence layer as file names when storing their information.

Object	ID letter
Patient	P
Bed	B
User	U

Table 1: *The ID letters assigned to each department in the HMS.*

3.3 Access and Data

Access level works as shown in the following hierarchy of implemented staff types:

$User < Nurse < Doctor < Admin$

In other words, Admin access level gives a user complete access to the system and its data including both user and patient info. This is important because at any time, a staff member of a hospital may need to edit their information(such as a changed phone number). Generally, it is important to have a more executive access level such as an Admin to ensure that issues can ultimately be resolved by authorised individuals. All other access levels do not have access to Staff information. Doctors and Nurses have very similar access levels. Doctors and Nurses are able to register/admit/discharge patients as well as view and edit patient information including health records. Lastly, the User access level represents a clerk or secretary who would not need access to a patient's health record. Users can register patients and edit a patient's registration info such as address and phone number, but they do not have access to more sensitive health data.

Patient Data	**User Data**
Patient ID	User ID
First Name	Name
Surname	Department
Department	Phone Number
Phone Number	User Email
Address	
Date of Birth	
Alive or Deceased	
Bed ID	

Table 2: *The data that has been chosen to display for each patient/user in the HMS.*

3.4 Departments

The system has 4 departments where the Inpatient and Emergency departments have beds that the patients can be assigned to, whereas the Outpatient department has no beds. The Management department has no patients assigned to it, and it simply works to hold the Admin users. **Departments:**

- Inpatient
- Outpatient
- Emergency
- Management (has no patients)

3.5 GUI design

The GUI of the Health Management System was designed to be intuitive and simple, yet effective in fulfilling the necessary functionalities of a Health Management System. The GUI is previewed in the screencast of the running software which can be found in Section 05. Furthermore, the User Manual section (Section 05) provides some insight on how to use the GUI. The GUI utilises Java's Swing toolkit to allow user interaction with the software. The GUI consists of a login process and separate interface for administrators, Health Care Staff and non-administrative management users. The main interfaces contain a table of Patients and a second tab with the Users in the administrators interface. In the toolbar buttons for registering, admitting/discharging and editing patient information can be found. The Health record can be accessed in a second interface by all except the non-administrative management users.

3.6 Persistence Layer

The main design goal of the persistence layer was to give it as simple of an interface as possible. As such, it was initially intended to only include three methods: save, loadDepartment, and delete. These three essentially covers all the basic needs of a persistence layer (Note that save overwrites any previous version, which removes the need for an update method). However, due to later changes to the program, it was necessary to add saveCounter, loadCounter, and loadTemps. SaveCounter and loadCounter are used to persist the counter that keeps track of which ID number the system is currently at, while loadTemps loads objects stored in a temporary folder. This temporary folder is usually used for registered patients or users who have not yet been assigned a department. Folders for the regular departments, including folders for their possible objects, are created when the PersistenceLayer class constructor is called.

All saving is done to XML files via the XMLEncoder[1] and loading of these files are done via XMLDecoder[2]. Using these ensures objects are saved and loaded in a consistent way. Furthermore it is future-proof in the sense that it allows for loading old versions of the objects even if their classes have been changed since they were saved.

The reason it was important to keep the interface simple is that the persistence layer must be used properly to ensure consistency between data in memory and the saved files while also avoiding loss of data. Proper use can be described in a few simple rules:

- Every time a new object, which should be saved, is created, it must also be saved.
- Every time one of these objects are modified, it must be saved again.
- loadDepartment should be called whenever a department is initialised.
- Every time a saved object is deleted in memory it must also be deleted from the files, otherwise any deleted objects would reappear once the program was reloaded.

3.7 Advanced Query Mechanism

The advanced query mechanism consists of the class "Query" and is accessed through a button labelled "query", which is visible throughout the program. Upon opening it, an instance of the class will be initialised and the user will be presented with several different options of querying. These include queries such as "Department with most users", "Department with least patients", and so on.

When a query is chosen, the relevant information is fetched from departments, processed, and formatted into something that can be presented to the user.

3.8 Unit tests

To ensure the correctness of the code, unit testing with JUnit was used. The goal was to cover all relevant classes at or very close to 100%. By relevant it is meant that classes in the GUI were excluded from the unit tests, as they do not contain much of the program logic and would be better covered by other forms of testing.

The program ended up at just under 100% where the only instructions not covered are due to catching FileNotFoundExceptions in the PersistenceLayer class. These were especially challenging to test, as the filename was guaranteed to be correct, which means the exception would only be thrown, if something outside of our control went wrong in the file system.

4 Project Management

4.1 Choice of LEAN Framework

To our group, an agile mindset is represented by an emphasis on planning and efficiency in a team to maximise results and quality. In the context of this project, it is important to be dedicated to the established plan and expectations. The agile mindset by being as deliberate as possible will be present with the emphasis on planning/organisation rather than just diving right into coding. It is understood that the project will unquestionably be easier and higher quality with proper organisation.

The LEAN framework consists of seven main principles:

- Eliminate waste
- Create Knowledge
- Defer Commitment
- Deliver Fast
- Build Quality In
- Respect People
- Optimise the Whole

The group felt that these principles suit their chosen working style very well and would contribute to improving the quality of the project overall. LEAN was chosen over other methods such as SCRUM, because of the structure of the group, where less frequent meetings and a greater independence and distribution of work between the members was chosen.

4.2 Team management

A flat team hierarchy with shared leadership implemented among group members. This allows the group to discuss and reach a common understanding on how to create most value in the finished product, in accordance with the **Create Knowledge** principle of LEAN.
The group agreed that the entire project is better conducted in a sequential order from class definition to user stories to coding and so on, as it is the best that everyone understands each step, and only then continue to next step. During this time the team can learn from each other in accordance with the **Create Knowledge** principle of the LEAN framework.

Major decisions are made at group meetings, which are held at least once per week. Here the tasks are also divided between the group members to complete on their own time to reduce the unnecessary motions for members. This also adheres to the **Respect People** principle of LEAN. By making these decisions together, it allows the group members to become self-directing and **eliminates waste**, as each member knows exactly what to do and how to do it.

A major focus in the beginning of the project was to carefully plan the structure of the software. Much time was spent creating the UML class diagram and CRC cards, thereby ensuring that all members agreed exactly on how the software should be built up. This closely relates to the **Build Quality in** principle of LEAN. It also **eliminates waste**, as it allows members to write the components of the project correctly the first time, and it ensures that different members' work will be compatible with each other. Furthermore, the group has decided to follow the Google Java style guide, so all sections of code are formatted the same. It is also agreed that clearly commenting code is very important. These considerations all add to the quality of the written code.

Furthermore, the **Defer Commitment** principle was used in the form of refactoring by building a strong basic code that could easily tolerate change and then adding the more complex functionalities on top of it later. Furthermore, automated testing was used throughout, by testing each individual class as it was written.

4.3 Version control systems

Git-hub (Pieter as the admin) is used for version control, for the team to support parallelization. This also works to **eliminate waste** as members will not do overlapping work and different work will be compatible. Each member has an individual branch to work with. In order to verify a pull requests to the master branch, it must be approved by at least one other group member if the entire group is not present. A task will be considered done when it is pushed to the master branch.

The KANBAN method is used for task tracking, assignments as well as overall progress through Trello. A deadline for each assigned task is also stated on Trello to ensure less waiting and **fast delivery**. Delays are reported to the group before a stated deadline on Trello in accordance with the **Respect People** principle. Overall, this communication between group members is very important to keep the deadline and ensure the quality of the project.

4.4 Backlog

The backlog is kept in Trello, as it allows each member to mark the task they are working on in the backlog and their progress with it. This ensures that only one person is working on each task. As the members are working on various tasks they will also update the backlog regularly to remove unnecessary items. Both of these measures serve to further **eliminate waste** in terms of double work and removing unnecessary functionalities from the program. As a general rule, the uncompleted tasks should be reported to the group in order to avoid delays. The task(s) should be switched or worked with by the entire group before moving on to next step(s).

The user stories will be considered satisfied and accepted when all requirements are met and when relevant tests are passed. All tasks need to be tested and approved by several group members before it can be marked as officially done. The aim is to achieve at least 95 % unit test coverage.

5 User Manual

This section provides a guide for using the hospital management system and describes its functionalities. For a visual explanation of the software, see the **screencast** at the link https://youtu.be/Dxp7s8k8ZX8

5.1 Adding a Patient

A patient must first be registered to the system. To register a patient, Select "**Register**" in the management toolbar and type in all required information. Note: For administrative access levels, it is necessary to be in the Patients list, which can be seen by pressing the "**Patients**" button just under the search toolbar.

Next, the registered patient must be admitted to a department to begin care service. Once a patient is registered, click the "**Admit**" button in the management toolbar to select a department. Now it is time to assign a patient to a bed. In order to do this, a bed will need to be added to a department. To add or remove beds from a department, click "**Edit Beds**" in the toolbar. Once a bed is added and its unique bed ID comes up in the window (in the form of B#), a patient can now be assigned to a bed with the "**Assign Bed**" button in the toolbar.

5.2 Editing Patient Info

Patient registration info (Name, Address, etc.) can be edited using the "**Edit**" button in the toolbar. Furthermore, for Nurse, Doctor, and Admin access levels, a patient's medical record can be seen and edited using the "Record" button in the toolbar. Once a patient is done with care and is ready to be released, select that patient and click "**Discharge Patient**"

5.3 Staff Management

Admin access is required to add a staff member to the health management system. First, click "**Add User**" on the management toolbar and enter in prompted information. Staff information can be changed by Admin users using the "**Edit**" button in the management toolbar.

5.4 Using Search Features

Patients and Users can be found by their distinct info fields using the search bar below the management toolbar. Furthermore, the health management system also includes an advanced query feature, where more specific details of the system can be looked up, such as the total number of patients or the department with the most admitted patients. To use this feature, click the **"Query"** button in the management toolbar and select which type of information is required.

6 Conclusion

Using an Agile mindset and the LEAN framework, our group created a project that has the following four functions, patient registration, staff management, health facility management, patient admission. We also created an user interface, persistency layer, advanced query mechanism, and an user management login system in order to make the program easier to handle for the user.

By using LEAN, we were able to make sure that everyone was on the same page and had something to work on, even though we were only able to meet one or two times a week. LEAN allowed us to easily work on the project without having to meet multiple times a week.

References

[1] *Class XMLEncoder (https://docs.oracle.com/javase/7/docs/api/java/beans/XMLEncoder.html) - 01/05/2019.*

[2] *Class XMLDecoder (https://docs.oracle.com/javase/7/docs/api/java/beans/XMLDecoder.html) - 01/05/2019.*

[3] [Disabled World 2018, Disabled World, accessed 20 March 2019, <https://www.disabled-world.com/definitions/hospital-departments.php>]

Appendix

CRC Cards

The various classes in the system and their tasks and responsibilities

User	
Responsibilities	Collaborators
Name	
ID (unique, final)	
email	
Phone Number	Patient
Department	Department
Register Patient	
getters	
setters	

Doctor *extends User*	
Responsibilities	Collaborators
Name	
ID (unique, final)	
Address	
Phone Number	
Department	
Speciality	Department
	Patient
Admit Patient	Bed
Discharge Patient	
Assign Bed	
Get Patients medical data	
Edit patient medical data	
Move Department	
Getters	

Department *Abstract*	
Responsibilities	Collaborators
Name	
ID (unique, final)	
Bed List	
User List	
Patient List	User
	Patient
Add User	
Remove User	
Add Patient	
Remove Patient	
Getters	

Login Manager	
Responsibilities	Collaborators
Users	
	Users
Login Manager	
Check ID	

Nurse *extends User*	
Responsibilities	Collaborators
Name	
ID (unique, final)	
Address	
Phone Number	
Department	
	Department
Admit Patient	Patient
Discharge Patient	Bed
Assign Bed	
Get Patients medical data	
Edit patient medical data	
Move Department	
Getters	

Admin *extends User*	
Responsibilities	Collaborators
Name	
ID (unique, final)	
Address	
Phone Number	
Department	
	Department
Admit Patient	Patient
Discharge Patient	Bed
Assign Bed	User
Add User	Nurse
Remove User	Doctor
Get Patients medical data	
Edit patient medical data	
Move departments	
Getters	

Inpatient *extends Department*	
Responsibilities	Collaborators
Name	
ID (unique, final)	
User List	
Patient List	
Bed List	
	User
Add User	Patient
Remove User	Bed
Add Patient	
Remove Patient	
Getters	
Setters	

Management *extends Department*	
Responsibilities	Collaborators
Name	
ID (unique, final)	
User List	
Patient List	
Bed List	
Single Instance	User
	Patient
Add User	
Remove User	
Add Patient	
Remove Patient	
Getters	
Setters	

Emergency *extends Department*	
Responsibilities	Collaborators
Name	
ID (unique, final)	
User List	
Patient List	
Bed List	
Single Instance	User
	Patient
Add User	
Remove User	
Add Patient	
Remove Patient	
Getters	
Setters	

Patient	
Responsibilities	Collaborators
Name	
ID (unique, final)	
Phone Number	
Address	
Deceased	
Health Record	
Department	Department
Birthday	
Bed	
Update Department	
Update Bed	
Getters	
Setters	

Bed	
Responsibilities	Collaborators
Bed Id	
Patient	
Department	
Id counter	Patient
	Department
Add Patient	
Get Patient	
Remove Patient	
Getters	

HMS	
Responsibilities	Collaborators
getDepartment	

Persistence Layer	
Responsibilities	Collaborators
Departments	
subfolders	
save	Department
load department	
delete	
Load Object	

Query	
Responsibilities	Collaborators
In size	
Out size	
Em Size	
Patient Size	
max patients	
min patients	
out users	
in users	
em users	
user sizes	Department
max users	
min users	
departments	
most patients	
least patients	
most users	
least users	
total patients	
total users	

Outpatient *extends Department*	
Responsibilities	Collaborators
Name	
ID (unique, final)	
User List	
Patient List	
Bed List	
Single Instance	User
	Patient
Add User	
Remove User	
Add Patient	
Remove Patient	
Getters	
Setters	

Work Distribution

Group Member	**Java**	**Report**
Asger Conradsen	Persistence Layer, Login Manager, unit tests, & revision/bug fixing	Persistence Layer, Advanced Query Mechanism, Unit Tests, UML & Object IDs
Anna Hogan	Cucumber tests	User Stories, CRC Cards, Introduction & Conclusion
Pieter O'Hearn	User, Patient, Cucumber tests, unit tests & GUI	CRC Cards, GUI & UML Diagram
Karoline Østergaard	Departments, Query Mechanism & Cucumber test files	Project management & Software Architecture
Jack Rodman	User/Patient & GUI	User manual, Screencast
Kun Zhu	Departments & User Stories	Project management, Introduction & Conclusion

Table 3: *The main responsibilities of each member. The classes include both the coding and writing the unit tests.*