Bring ideas to life
VIA University College

Clinical Management System

Taha Mohamed Alzain	(2669055)
Muhammad Nadeem	(266704)
Fadi Atia Dasus	(266265)
Oskars Arajs	(266534)
Alexandru Vieru	(267013)

Supervisors

Ole Ilsgaard Hougaard
Martin Zmija

ICT Engineering, VIA University College, Horsens

2nd Semester

Abstract

The purpose of this project is to create a client/server system that would be able to handle daily responsibilities related to that of a clinic. The system will have to handle patients, employees, appointments etc.

This system will increase the efficiency of a clinic by storing and retrieving patient, employee, appointment details/information from a database that holds all the clinics related information.

There are benefits of using this system as information is stored inside of a database instead of using paper storage locations.

There are multiple Clinical Management system software’s on the market with different purposes. This system deals with three actors, Manager, Doctor and Secretary. Manager can fire and hire employees, secretary deals with patient information, doctor deals with medicine and patient’s medical cases.

Table of content

List of Figures

1.Introduction

Denmark's healthcare system has been regarded as excellent in terms of world standards. It is easy to see how of a complex system they are using for management. Denmark's health care IT system is rated as one of the most efficient systems in the world. Other countries that are seeking to increase the efficiency of their healthcare system can learn from the experience of Danish healthcare (Kirkegard.P., 2013).

Currently, developing countries are still storing information on paper or cards. Nurses are still writing patient information manually and organize it in racks. This, outdated system, is expensive, inefficient and not secured.

In Recent Decades, the use of Information Technology application is more frequent in practice, so medical practitioners consider the necessity of using a similar software applications in their respective healthcare centers.

The proposed system (Clinical Management System) is a computerized system that saves patient records. It will reduce the burden related to the daily responsibilities of Doctors and Nurses. The system has integrated functionality for retrieving staff and patient records, appointments, prescription, medical cases and medicine.

The reason behind the project was to make a System that would be implemented in a Client/ Server type of program. For this reason, RMI (Remote Method Invocation) was used. All the information passed through the application should be saved on a database using a server, and all of the passed information could be read and wrote by any user (in this project: manager, doctor, nurses).

The first stage of the project was to realize what will be the user stories and requirements i.e. what are the functionalities that the system should be able to operate. Once the requirements are completed the analysis part takes place. where use cases, use case descriptions, and conceptual diagram will be discussed.

The next stage is the design part where Activity Diagrams, Class Diagrams and Sequence Diagrams are being discussed. After the design section the next one is the implementation section, in which the code is being discussed and in detail the parts related to RMI and Client/Server. The final stage in the project report is Testing the system. JUnit testing was used on some parts of the system, like "**addEmployeeController**" to test if the methods are working as they should.

2 User Stories and Requirements

In this chapter the purpose will be to establish the requirements based on the user stories from the customer. The requirements are one of the major tasks in designing systems development (Alsaleh.S and Haroon.H., 2016). It is very critical to determine the right requirements for the system otherwise the system won't have all the functionality needed. The requirements can be made and separated into two groups: functional and non-functional requirements.

2.1 User Stories

In this subchapter the user stories from the customer will be presented since the requirements for this report will be made based on them.

1. As a manager I want to be able to add an employee to the system so that they can handle their tasks.
2. As a manager I want to search for employees so that I can keep track of them.
3. As a manager I want to be able to remove employee so that I can delete the employee no longer exist.
4. As a manager I want to be able to edit existing employee so that I can keep new informations of employee.
5. As a secretary I want to store patient personal information so that I can book a appointment for them.
6. As a secretary I want to be able to search for patient information so that I can obtain relevant information about a certain patient.
7. As a secretary I want to be able to manage an appointment so that I can organize patient meetings.
8. As a secretary I want to be able to view patient's medical records so that their condition can be monitored.
9. As a secretary I want to view patients medicine so that I can send request for renew.
10. As a doctor I want to add medicine to the system so that I can add medicine to the prescription of the patient.
11. As a doctor I want to edit Medicine if name or quantity is changed so that patient will get right medicine in right dose.
12. As a doctor I want to remove medicine from the system so that medicine should be deleted from the system.
13. As a doctor I want to add a patient's medical condition so that I can form a medical case.
14. As a doctor I want to prescribe medicine that can be stored in the patient's personal record so that I can handle the data
15. As a doctor I want to renew medicine prescription that can be stored in the patient's personal record so that I can satisfy his/her requests.
16. As a doctor I want to approve medicine prescription request that comes from the secretary so that patients can get what they requested.

2.2 Functional Requirements

Based on the information above the following requirements have been created.

1. The system must allow the manager to be able to add new employee based on type.
2. The system must allow the manager to be able to remove employee.
3. The system must allow the manager to be able to edit existing employee.
4. The system must allow the manager to be able to search for certain employee.
5. The system must allow the secretary to be able to add new patient information.
6. The system must allow the secretary to be able to remove patient information.
7. The system must allow the secretary to be able to edit existing patient information.
8. The system must allow the secretary to be able to search for patient information.
9. The system must allow the secretary to be able to add an appointment.
10. The system must allow the secretary to be able to remove an appointment.
11. The system must allow the secretary to be able to edit an appointment.
12. The system must allow the secretary to be able to get a list of appointments.
13. The system must allow the secretary to be able to send a request in order to get the medicine renewed by the doctor.
14. The system must allow the doctor to be able to add a patient’s medical case.
15. The system must allow the doctor to be able to add a medical prescription to the patient’s medical record.
16. The system must allow the doctor to be able to renew medicine prescription for the patient.
17. The system must allow the doctor to be able to approve the requested medicine that comes from the secretary.
18. The system must allow the doctor to be able to add new medicine information.
19. The system must allow the doctor to be able to remove medicine.
20. The system must allow the doctor to be able to edit existing medicine.
21. The system must allow the doctor to be able to search for medicine.

2.2 Non-Functional Requirements

1. The System must follow Client/Server architecture (RMI).
2. The system must be developed in java.
3. The usability of the system must be tested by end users.
4. The system must store information in a database.
5. The system must handle multiple requests at the same time.
6. The system must be up 24/7.

3 Analysis

The **use case diagram** is the depiction of what a system can do for a user. Use case diagram is based on the scenarios so these two are connected to each other. A **scenario** is the representation of what is going to happen when someone uses the system. Furthermore, **use case descriptions** have been made for each use case of the actors which participate in this flow.

3.1 Scenarios

Here only one scenario is presented and that is "add an appointment" scenario of Clinical Management system

Add an appointment
Patient call the clinic to book an appointment;
Secretary search patient.;
Secretary write reason for appointment;
System returns available dates;
Secretary pick desire date;
System save the information.

<u>Classes</u>

- Appointment;
- Date;
- Database.

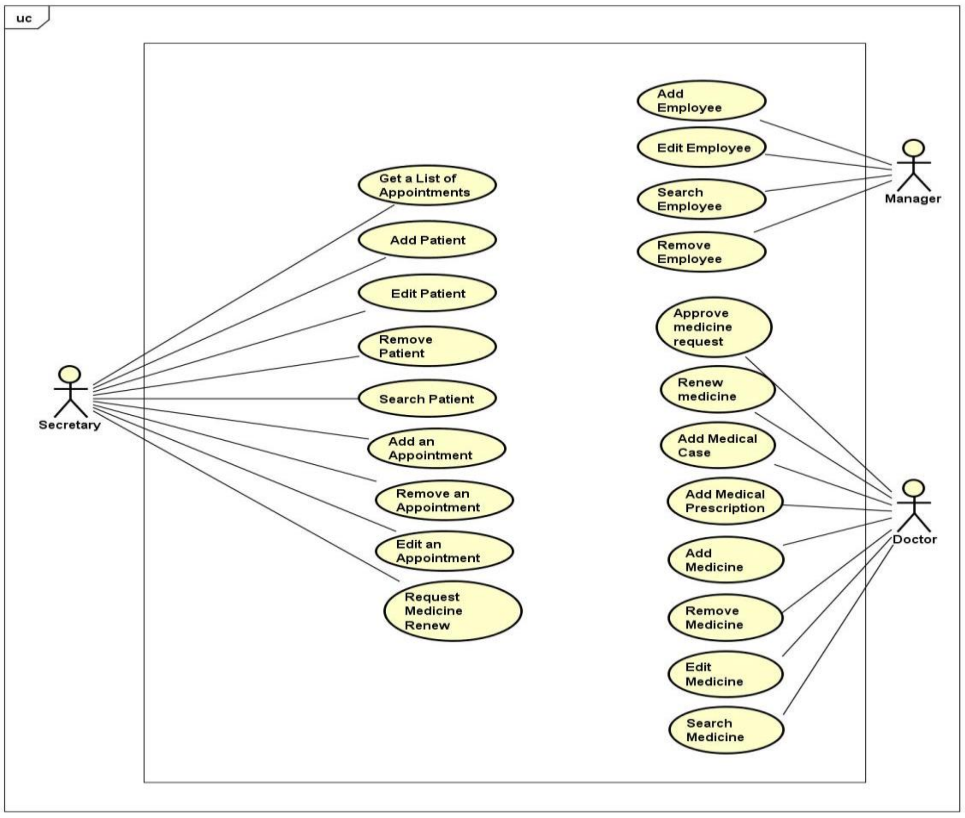
<u>Methods</u>

- Search;
- Save.

3.2 Use case diagram

The use case diagram is a graphical representation of the users that are interacting with the system while performing certain tasks. Based on the scenarios it's possible to create as many use cases as the system needs and these cases are presented in ***Figure 3.1***.

Figure 3.1 Use case diagram



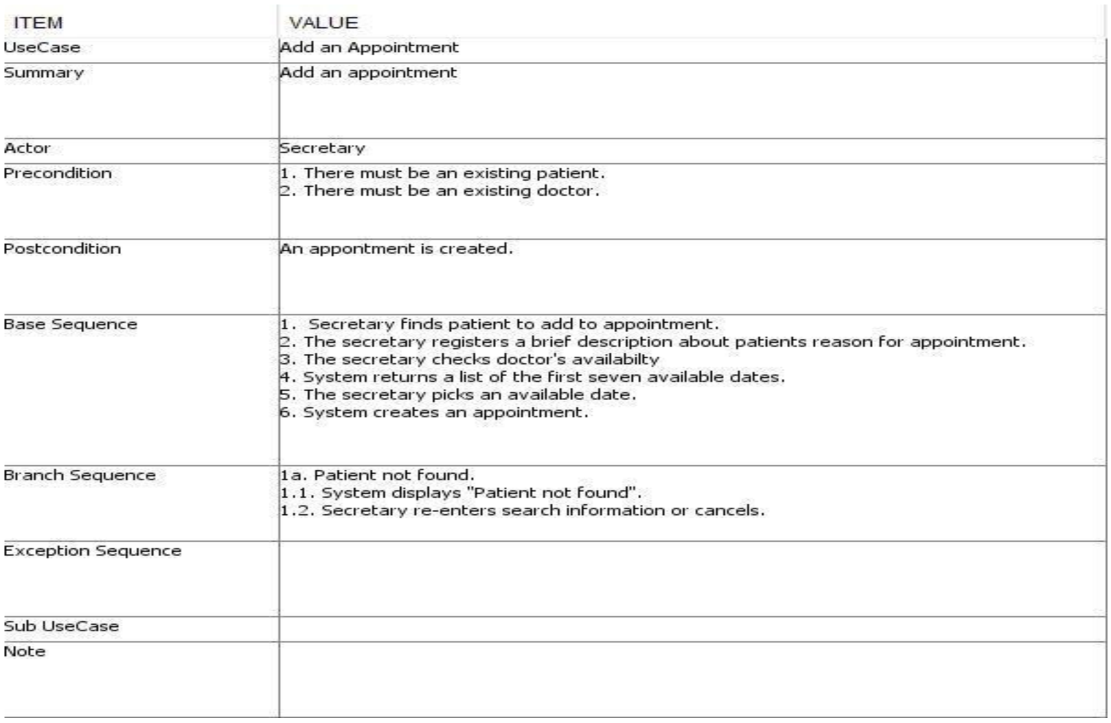
In ***Figure 3.1,*** there are 3 actors involved in the process and each of them have their own individual use cases as shown. These use cases are the actions that each actor needs to perform as part of their daily activities.

3.3 Use case description

Based on the use case diagram shown above, several use case descriptions were made for this subchapter. Only one-use case description will be shown as an example while the rest can be seen in ***Appendix 2.***

The use case description is a collection of values, preconditions, postconditions and base sequences which form a detailed view of the actor’s actions while interacting with the system.

Figure 3 2 Add an appointment use case description



ITEM	VALUE
UseCase	Add an Appointment
Summary	Add an appointment
Actor	Secretary
Precondition	1. There must be an existing patient. 2. There must be an existing doctor.
Postcondition	An apportment is created.
Base Sequence	1. Secretary finds patient to add to appointment. 2. The secretary registers a brief description about patients reason for appointment. 3. The secretary checks doctor's availabilty 4. System returns a list of the first seven available dates. 5. The secretary picks an available date. 6. System creates an appointment.
Branch Sequence	1a. Patient not found. 1.1. System displays "Patient not found". 1.2. Secretary re-enters search information or cancels.
Exception Sequence	
Sub UseCase	
Note	

Figure 3.2 shows, what steps the secretary will take for the system to successfully complete the requested action and what steps does the system make.

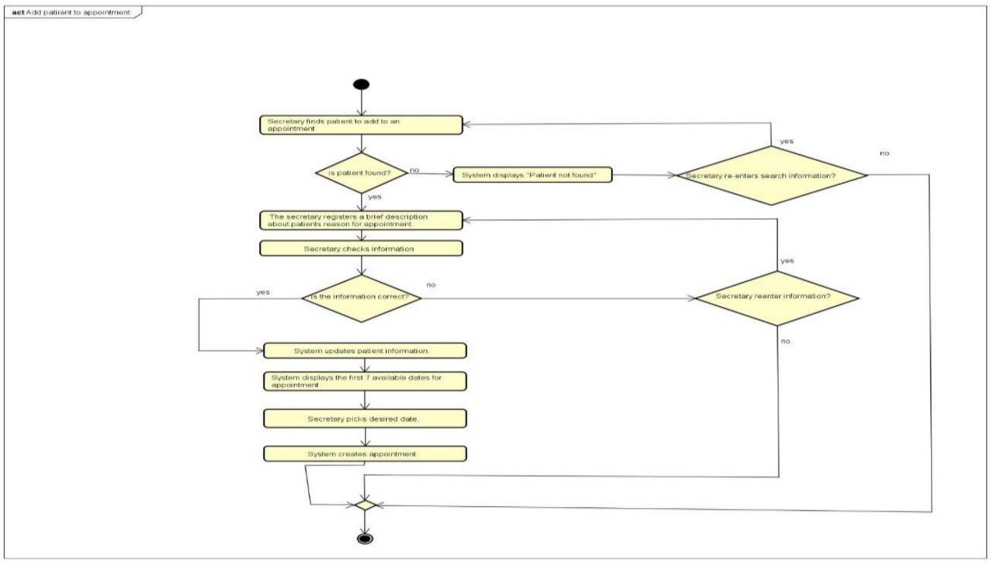
The precondition of a use case means, that some steps have to be achieved for the use case to be able to reach the end of the branch sequence. In this case the doctor and patient have to exist to continue with the use case. The actor (secretary) searches the patient by name. If the entered information is not valid the system will display that the patient is not found and will not go further. The actor will have the option to cancel the operation or re-enter the search information.

After completing the search, the actor will enter a brief description regarding the reasons for the requested appointment and will search for a list of available appointments. The system returns a list of appointments and then the secretary will than pick an available date and create the appointment. The system will save the created appointment in the database.

3.4 Activity diagram

The activity diagram describes the flow between the action of adding an appointment and what consequences some interactions will produced if executed by the actor.

Figure 3.3 Secretary add appointment activity diagram


act Add patiient to appointment
Secretary finds patient to add to an appointment
is patient found?
no
System displays "Patient not found"
Secretary re-enters search information?
yes
no
yes
The secretary registers a brief description about patients reason for appointment.
Secretary checks information
yes
Is the information correct?
no
Secretary reenter information?
yes
no
System updates patient information.
System displays the first 7 available dates for appointment
Secretary picks desired date.
System creates appointment

Secretary is tasked with add an appointment using the new system and is required to take certain steps before the request is completed as shown in ***Figure 3.3.***

Since the activity diagram is dependent on the use case description, so the steps taken are the same as in use case description “add appointment”.

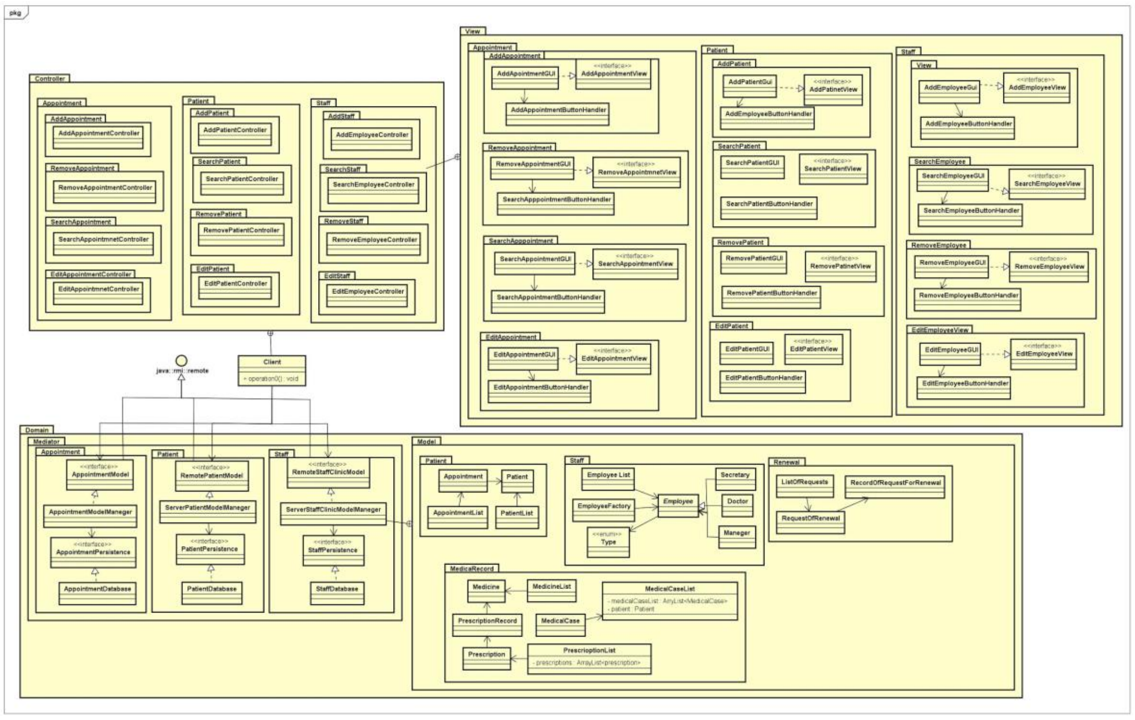
For more examples of activity diagrams, see ***Appendix 4.***

3.5 Conceptual diagram

Conceptual Diagram demonstrates how all of the classes in their packages interact with each other. This is important since it provides a graphical representation on how the system will be created and how each component will interact with another to make it functional.

This is represented as a conceptual diagram as shown in ***Figure 3.4*** bellow.

Figure 3.4 Conceptual diagram



Following the SOLID principles, the conceptual diagram is divided in three main packages Domain, Controller and View. In addition, all main packages have their sub packages. Domain carries Mediator and Model.

In Model there is Staff, Patient, Renewal and Medical Records, and most of the classes with their lists. In Mediator package there are model managers and databases for Staff, Patient and Appointment. Controller and View have Add, Search, Remove and Edit for Staff, Patient and Appointment.

3.6 Database

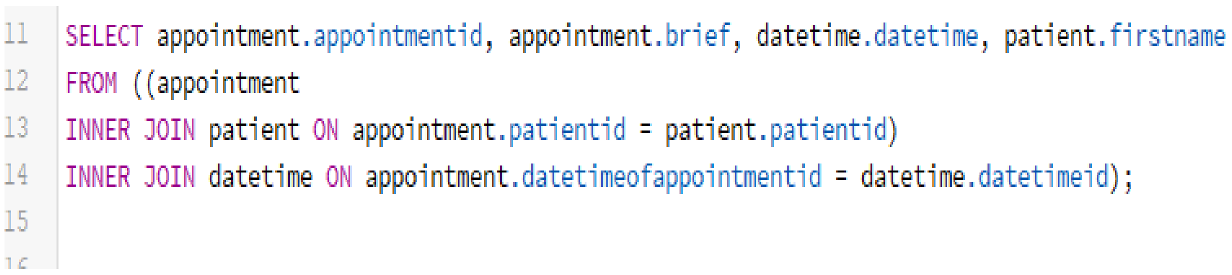
For this project one of the requirements was to store the needed information. For this reason, the database system has been chosen to fulfil this need.
The database will act as a data storage unit, and the software will provide a way to insert and retrieve data from it.

The database created for this project contains 10 tables and the ER model approach has been chosen for its design.

The entities act as storage units for the data and they are related to each other by means of primary keys and foreign keys. The primary keys are unique identifiers for certain fields in one table and they are represented as foreign keys in others, making a connection between two or more tables.

As an example, the following query is used in remove appointment, which is retrieving data from three different tables having relation together.

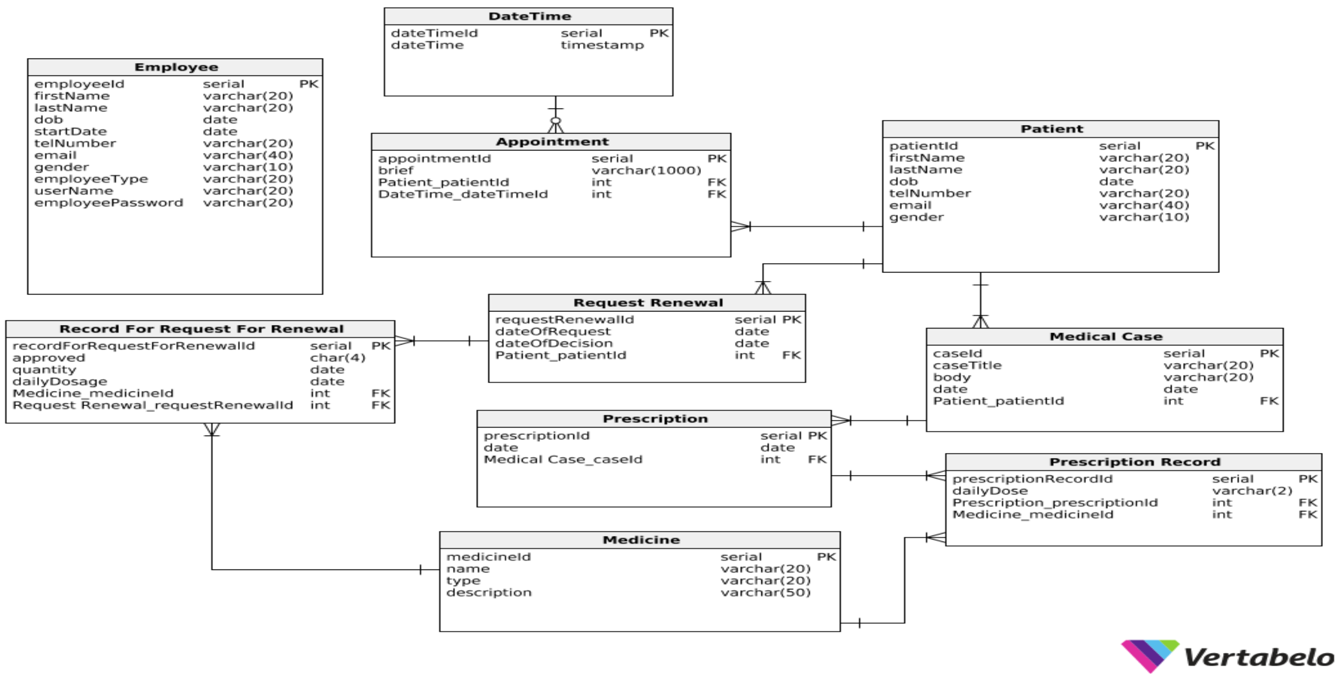
Figure 3.5 Sql Query



```sql
SELECT appointment.appointmentid, appointment.brief, datetime.datetime, patient.firstname
FROM ((appointment
INNER JOIN patient ON appointment.patientid = patient.patientid)
INNER JOIN datetime ON appointment.datetimeofappointmentid = datetime.datetimeid);
```

For the current database there are only one to many relations between the entities as shown in ***Figure 3.6.***

Figure 3.6 ER Diagram for Clinical Management System



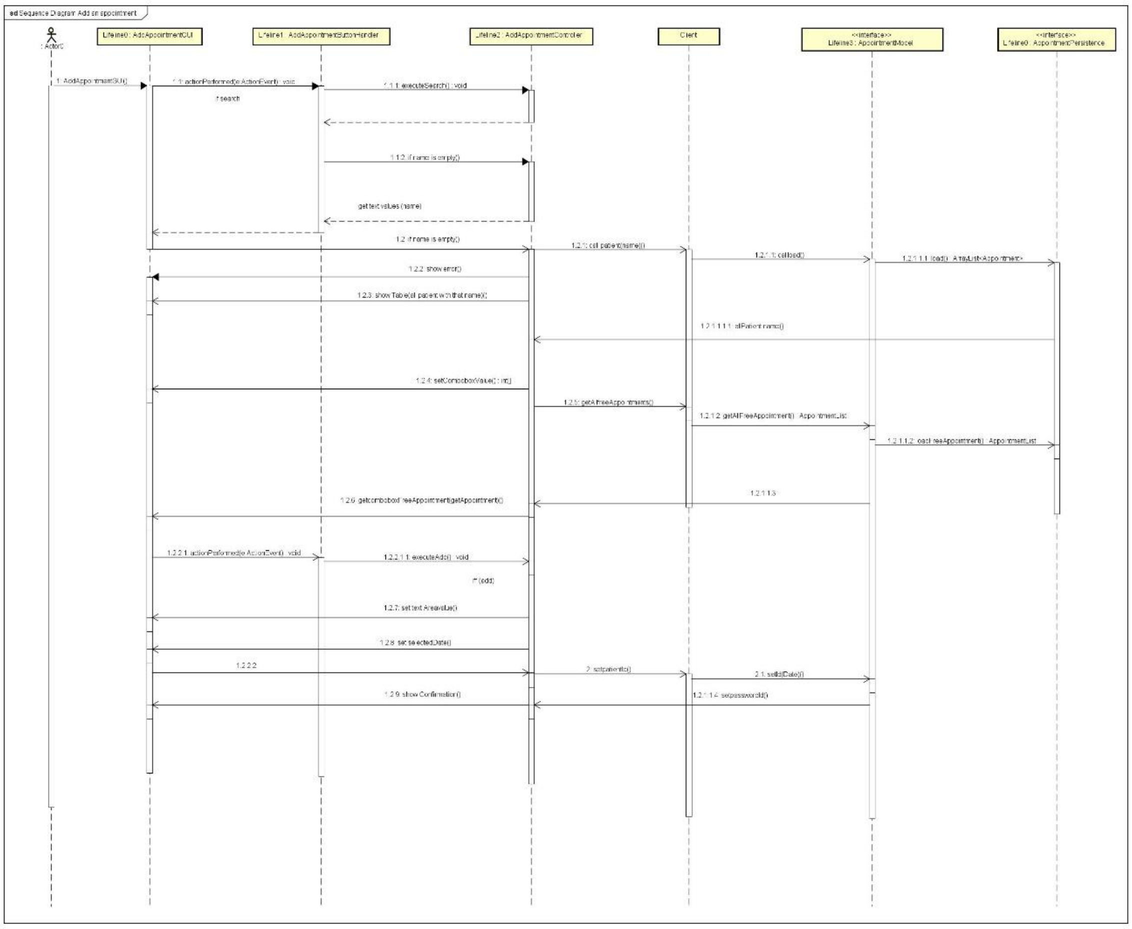
4 Design

4.1 Sequence Diagram

To get a better understanding on how the system behaves while certain commands are being given a sequence diagram is created. He sequences diagram will provide a visual representation on what steps the system will take and which methods will be executed to fulfil a certain task.

The following figure will present an example of a sequence diagram from the report. Add an appointment diagram has been selected for this example.

Figure 4.1 Add an appointment sequence diagram


sd Sequence Diagram Add an appointment
: Actor0
Lifeline0 : AddAppointmentGUI
Lifeline1 : AddAppointmentButtonHandler
Lifeline2 : AddAppointmentController
Client
<<interface>> Lifeline3 : AppointmentModel
<<interface>> Lifeline0 : AppointmentPersistence
1: AddAppointmentGUI()
1.1: actionPerformed(e:ActionEvent) : void
if search
1.1.1: executeSearch() : void
1.1.2: if name is empty()
get text values (name)
1.2: if name is empty()
1.2.1: call patients(name())
1.2.1.1: callload()
1.2.2: show error()
1.2.3: showTable(all patient with that name)()
1.2.4: setComboboxValue() : int()
1.2.6: getcomboboxFreeAppointment(getAppointment)()
1.2.2.1: actionPerformed(e:ActionEvent) : void
1.2.2.1.1: executeAdd() : void
if (add)
1.2.7: settext AreavisLe()
1.2.8: set selectedDate()
1.2.2.2
1.2.9: showConfirmation()

For this example, there are 6 life cycles consisting of the **AddAppointmentGUI**, the button handler, the controller, **AddAppointmentModel** interface, Client and **AddAppointmentPersistence** interface.

The first step is when the actor will interact with the GUI by searching for the patient he wants to add an appointment to. An action performed will be sent to the button handler which will then execute the search method towards the controller. The controller will

use the get patient information method to retrieve it from the model. If the name is empty than it will return an error to the GUI.
If the field is not empty than it will execute the **callPatient** method followed by **callLoad** towards the **appointmentModel.** The **appointmentModel** will load the request from the **persistence** and the request will be returned to the GUI for the user to see.

By using the **showTable** method it will return a table with the patient's information.

For the second step of the sequence, the user will perform another action through the GUI, however this time the performed action will be to add an appointment for the selected patient. First the GUI will send an action performed request to the **buttonHandler** and then the handler will send an **executeAdd** command to the controller. An if statement if (add) will return a **get text areaValue()** method with the patient's fields such as id and a date for the appointment.

Than an **addAppointment** method will be executed and the appointment will be added for the selected patient.

A show confirmation message will be returned to the GUI.

4.2 Class Diagram and Design Patterns

Class Diagram demonstrates what methods are needed to make the software function.
If consider the conceptual diagram , class diagram should be too big, if we have all methods with add, remove, edit, search for staff, patient and appointment. So, diagram is divided in 2 diagrams and make it one for Staff, and other for Patient and Appointment. Here only staff class diagram is considering and interpreting. For others see the attachment ***Appendix 4 Diagrams.***

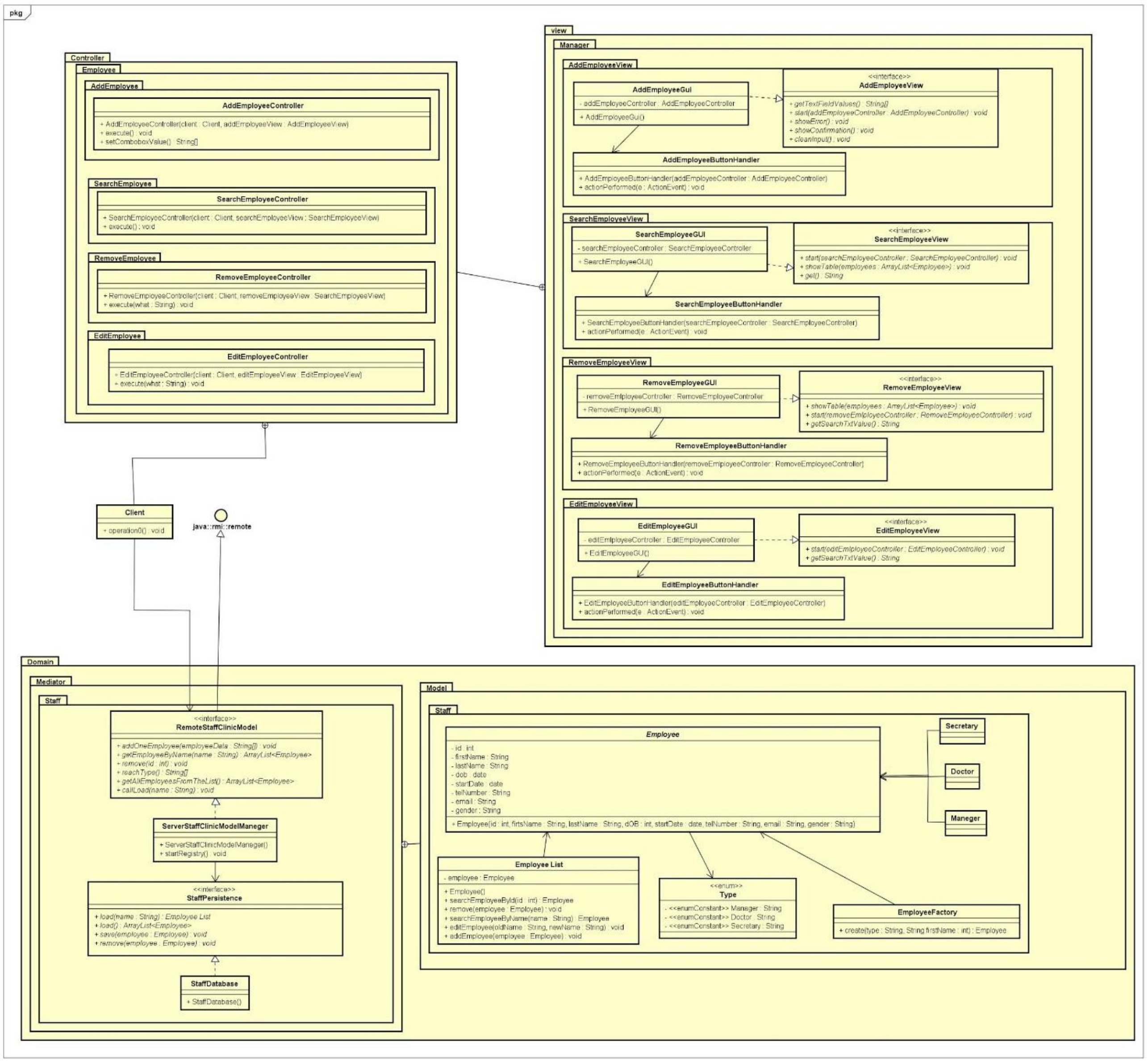
Design Pattern are used to solve the common problems faced in software design. By using design pattern, codes become flexible and reusable.

Five design patterns are used in the designing phase of the software i.e. MVC, Factory, Facade, and Adapter.

4.2.1 MVC (Model View Controller)

MVC is the most commonly used design pattern. The application consists of data model, presentation (view) and control for this information. Basically, in this project MVC is the representation of class diagram. So, ***Figure 4.2*** shows the class Diagram and MVC.

Figure 4.2 MVC and Class Diagram



4.2.2 Factory Design Pattern

The goal of the factory design pattern is to separate the process of creating concrete objects from the client that uses this object to reduce the independence of the client on the concrete implementation.

For this implementation, the design needs three things, first employee factory, the products that the factory makes which are employees objects, and the client that use the factory which is the server model manager.

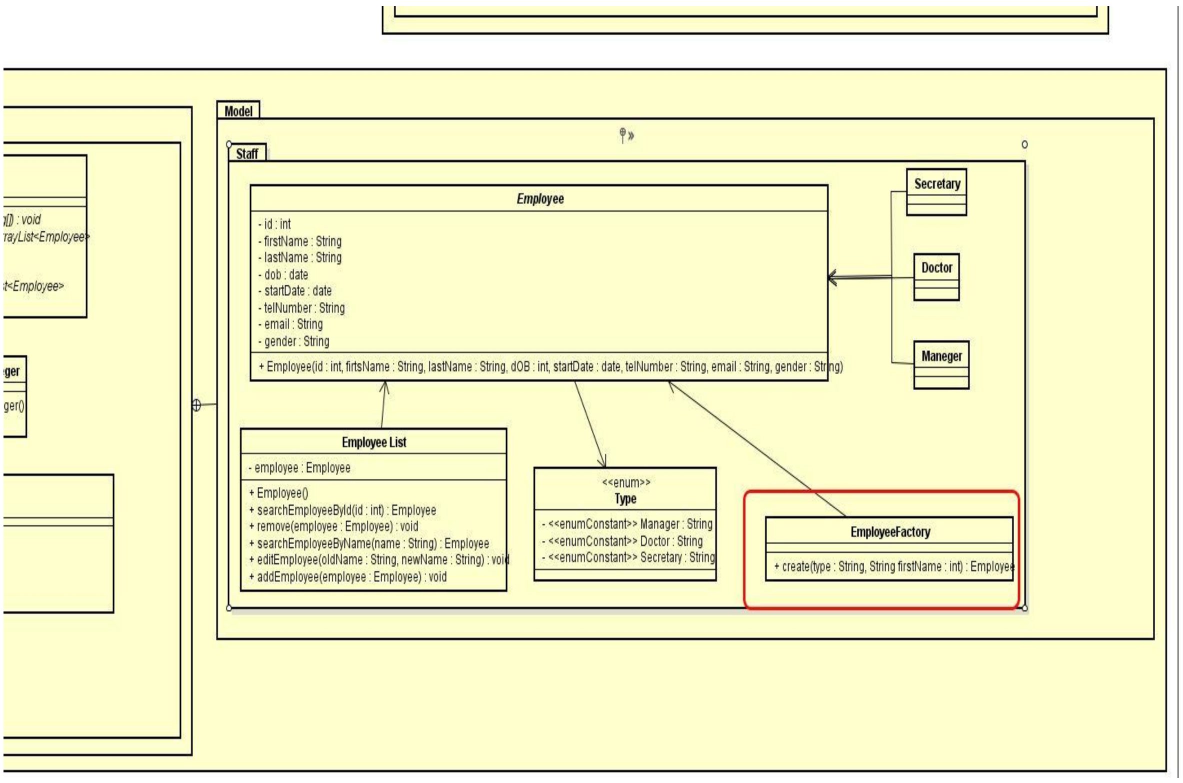
employee class is an abstract class, which acts as an interface to force all employee types to inherit the same attributes. Moreover, it makes the code open for extending as many types as the user wants in the future.

The program relying on the abstract class not on the implementation itself, because of all the concrete objects following the roles of the employee class. Thus, the factory does not care about which class it will deal with to create the employee object all it needs to know is the type, which makes the code open for adding more types without changing any existing code.

Advantages of the factory design pattern, and why it has been used in this concrete implementation?

The biggest advantage for this design pattern is to encapsulate the creation of the employees, which can be possible with the factory object whose sole responsibility is creating an employee based on its type. Furthermore, this design pattern served the project in maintaining the open-close solid principle and helped to keep the code more maintainable, flexible and open.

Figure 4.3 Factory design pattern

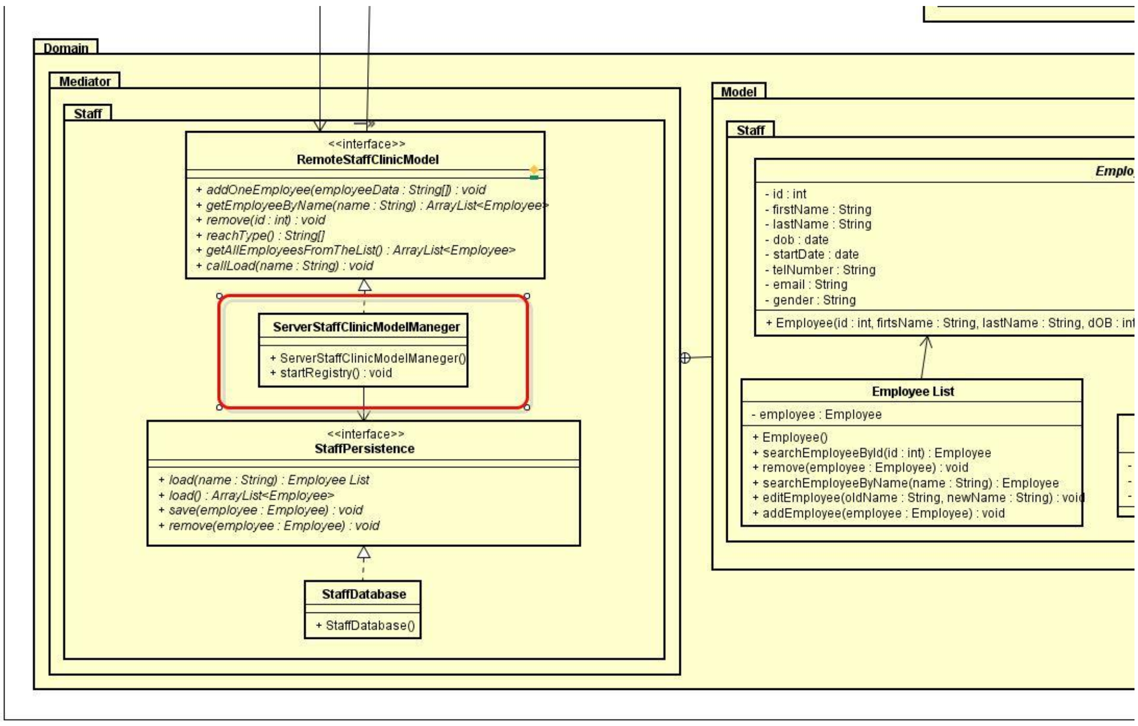

Model
Staff
Employee
- id : int
- firstName : String
- lastName : String
- dob : date
- startDate : date
- telNumber : String
- email : String
- gender : String
+ Employee(id : int, firtsName : String, lastName : String, dOB : int, startDate : date, telNumber : String, email : String, gender : String)
Secretary
Doctor
Maneger
Employee List
- employee : Employee
+ Employee()
+ searchEmployeeById(id : int) : Employee
+ remove(employee : Employee) : void
+ searchEmployeeByName(name : String) : Employee
+ editEmployee(oldName : String, newName : String) : void
+ addEmployee(employee : Employee) : void
<<enum>>
Type
- <<enumConstant>> Manager : String
- <<enumConstant>> Doctor : String
- <<enumConstant>> Secretary : String
EmployeeFactory
+ create(type : String, String firstName : int) : Employee

4.2.3 Facade Design Pattern

Facade Design pattern is a facade class that makes complex interface easier to use. Server Staff Clinic Model Manager (**ServerStaffClinicModelManager**) class is a facade.

In ***Figure 4.4*** **ServerStaffClinicModelManager** is connected to **EmployeeList** class and **StaffPersistence** Class. As it also has the **interface** with **RemoteStaffClinicModel,** so all the methods are implemented in ServerStaffClinicModelManager.

Figure 4.4 Facade


Domain
Mediator
Staff
<<interface>>
RemoteStaffClinicModel
+ addOneEmployee(employeeData : String[]) : void
+ getEmployeeByName(name : String) : ArrayList<Employee>
+ remove(id : int) : void
+ reachType() : String[]
+ getAllEmployeesFromTheList() : ArrayList<Employee>
+ callLoad(name : String) : void
ServerStaffClinicModelManeger
+ ServerStaffClinicModelManeger()
+ startRegistry() : void
<<interface>>
StaffPersistence
+ load(name : String) : Employee List
+ load() : ArrayList<Employee>
+ save(employee : Employee) : void
+ remove(employee : Employee) : void
StaffDatabase
+ StaffDatabase()
Model
Staff
Emplo
- id : int
- firstName : String
- lastName : String
- dob : date
- startDate : date
- telNumber : String
- email : String
- gender : String
+ Employee(id : int, firtsName : String, lastName : String, dOB : int
Employee List
- employee : Employee
+ Employee()
+ searchEmployeeById(id : int) : Employee
+ remove(employee : Employee) : void
+ searchEmployeeByName(name : String) : Employee
+ editEmployee(oldName : String, newName : String) : void
+ addEmployee(employee : Employee) : void

4.2.4 Adapter Design Pattern

Adapter is the final design pattern in this project. Adapter helps to connect two different interfaces that cannot be connected directly. In ***figure 8,*** Client cannot connect directly with the database. To connect the client with the database, adapter is needed.

Figure 4.5 Adapter Design Pattern

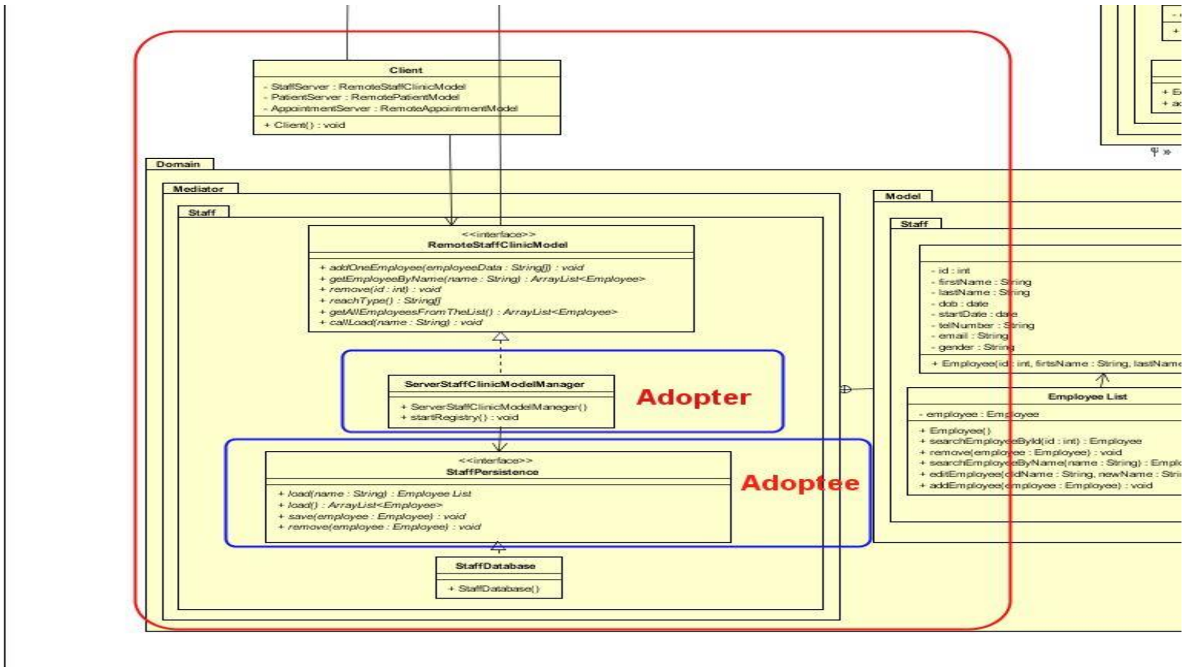

Client
- StaffServer : RemoteStaffClinicModel
- PatientServer : RemotePatientModel
- AppointmentServer : RemoteAppointmentModel
+ Client() : void
Domain
Mediator
Staff
<<interface>>
RemoteStaffClinicModel
+ addOneEmployee(employeeData : String[]) : void
+ getEmployeeByName(name : String) : ArrayList<Employee>
+ remove(id : int) : void
+ reachType() : String[]
+ getAllEmployeesFromTheList() : ArrayList<Employee>
+ callLoad(name : String) : void
ServerStaffClinicModelManager
+ ServerStaffClinicModelManager()
+ startRegistry() : void
Adopter
<<interface>>
StaffPersistence
+ load(name : String) : Employee List
+ load() : ArrayList<Employee>
+ save(employee : Employee) : void
+ remove(employee : Employee) : void
Adoptee
StaffDatabase
+ StaffDatabase()
Model
Staff
- id : int
- firstName : String
- lastName : String
- dob : date
- startDate : date
- telNumber : String
- email : String
- gender : String
+ Employee(id : int, firtsName : String, lastName
Employee List
- employee : Employee
+ Employee()
+ searchEmployeeById(id : int) : Employee
+ remove(employee : Employee) : void
+ searchEmployeeByName(name : String) : Emplo
+ editEmployee(oldName : String, newName : Stri
+ addEmployee(employee : Employee) : void

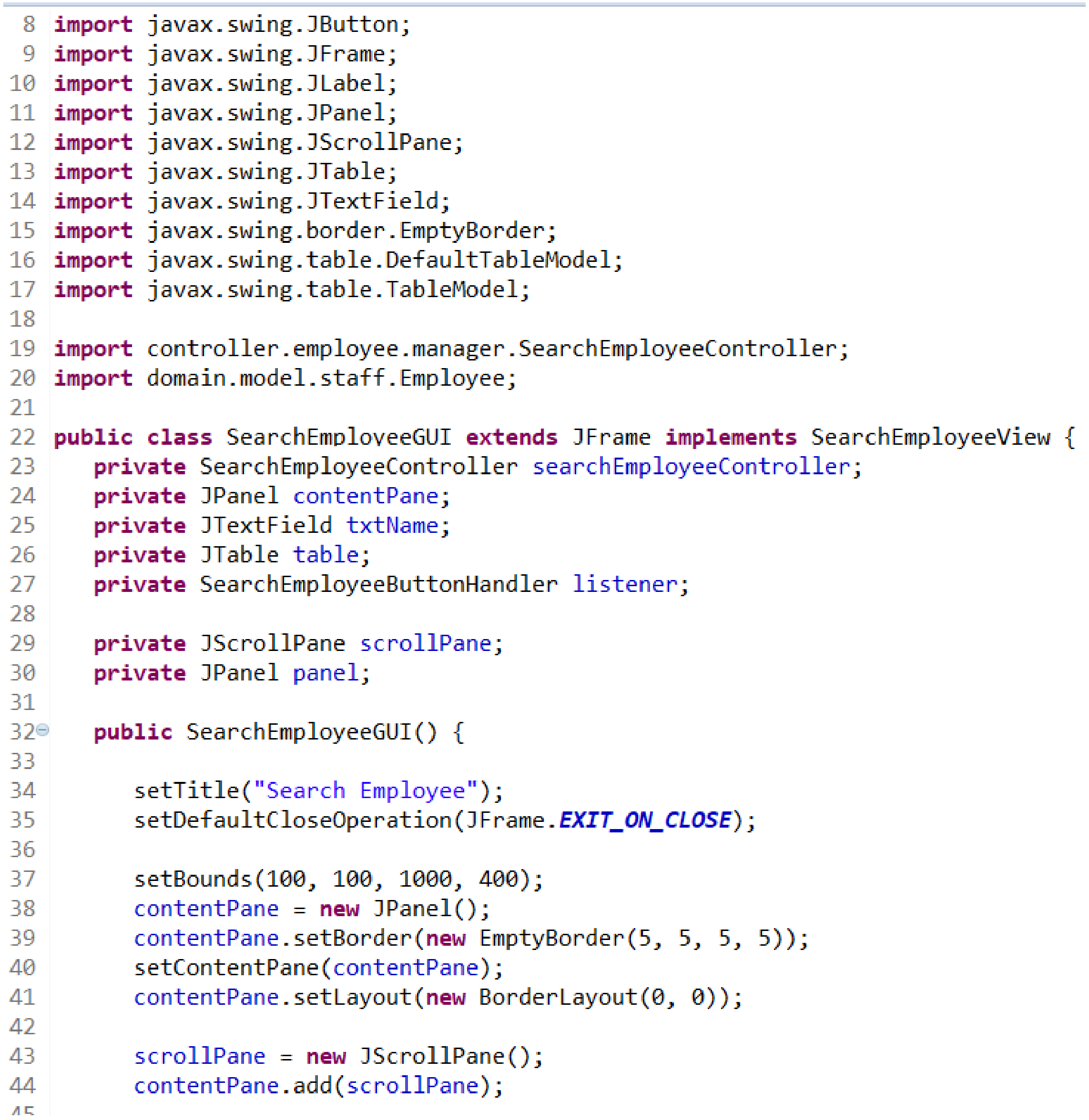
Figure 4.5 is the good interpretation of adapter design pattern. ServerStaffClinicModelManager is Adopter and StaffPersistence is Adoptee. So, there is dependency between client and the Database.

4.3 GUI

In this subchapter the discussion will be about GUI. This is important since it will be the main tool that the customer will use to perform his daily activities.

The method chosen to create this is swing as shown in the example below.

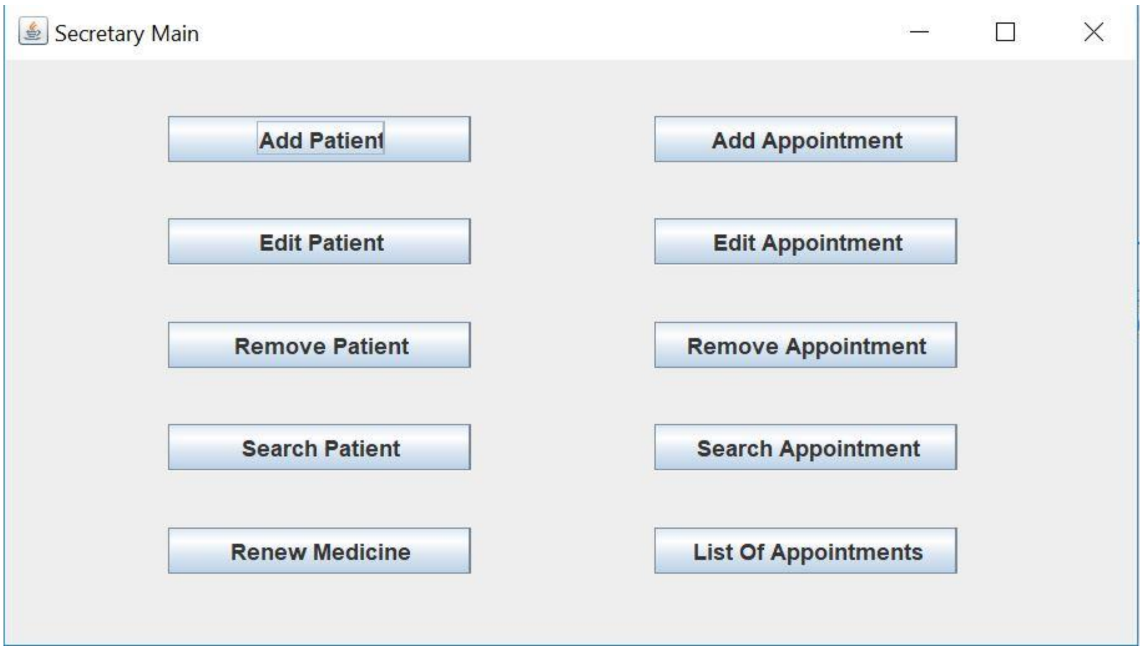
Figure 4.6 Swing



```java
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;

import controller.employee.manager.SearchEmployeeController;
import domain.model.staff.Employee;

public class SearchEmployeeGUI extends JFrame implements SearchEmployeeView {
   private SearchEmployeeController searchEmployeeController;
   private JPanel contentPane;
   private JTextField txtName;
   private JTable table;
   private SearchEmployeeButtonHandler listener;

   private JScrollPane scrollPane;
   private JPanel panel;

   public SearchEmployeeGUI() {

      setTitle("Search Employee");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      setBounds(100, 100, 1000, 400);
      contentPane = new JPanel();
      contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
      setContentPane(contentPane);
      contentPane.setLayout(new BorderLayout(0, 0));

      scrollPane = new JScrollPane();
      contentPane.add(scrollPane);
```

In this document, only one example will be shown regarding the GUI, for the rest see ***Appendix 3 - User Guide.***

Figure 4.7 Secretary main GUI



The above picture shows the layout of the main GUI window for the user of type secretary.

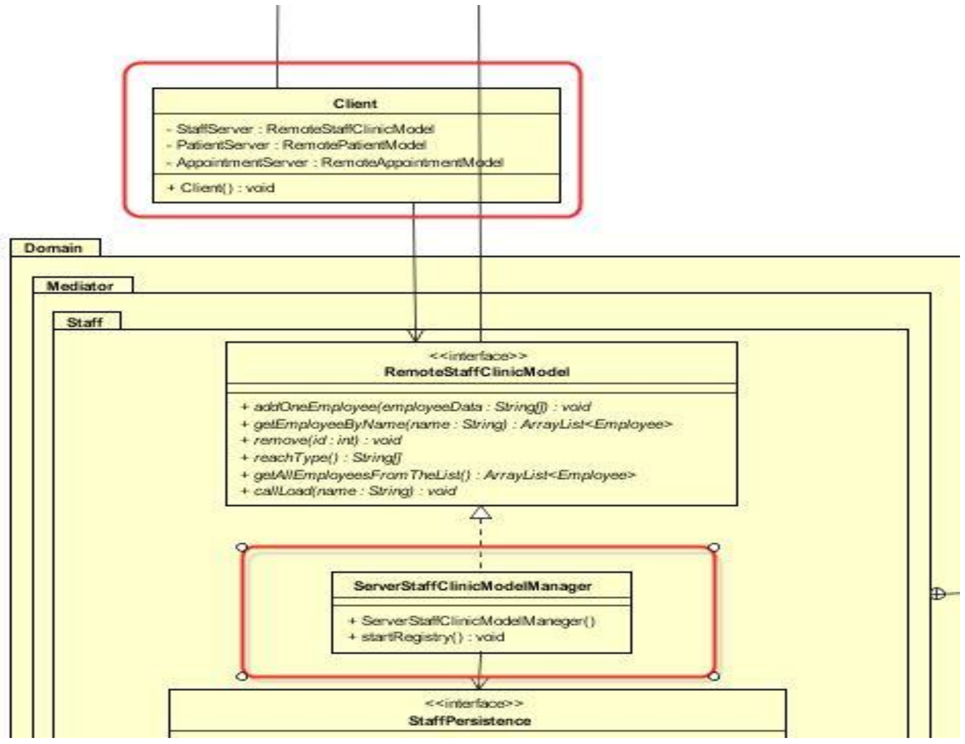
This example has been chosen since it is the largest and most complex in this project. Once the login will be performed, this frame will appear only for secretary and not for other types of employees.

As it can be seen, the secretary has multiple options when handling data. When an option is clicked, this will direct the user to the GUI that is assigned to further the selected action.

4.4 Client/ Server (RMI)

Program must be implement for Client / Server system. RMI is chosen for client and server system because client/server system can pass data back and forth.

Figure 4.8 RMI


Client
- StaffServer : RemoteStaffClinicModel
- PatientServer : RemotePatientModel
- AppointmentServer : RemoteAppointmentModel
+ Client() : void
Domain
Mediator
Staff
<<interface>>
RemoteStaffClinicModel
+ addOneEmployee(employeeData : String[]) : void
+ getEmployeeByName(name : String) : ArrayList<Employee>
+ remove(id : int) : void
+ reachType() : String[]
+ getAllEmployeesFromTheList() : ArrayList<Employee>
+ callLoad(name : String) : void
ServerStaffClinicModelManager
+ ServerStaffClinicModelManeger()
+ startRegistry() : void
<<interface>>
StaffPersistence

In ***Figure 4.8*** **ServerStaffClinicModelManager** class is Server and **Client** class is client. Server create the registry at the port 1099 and rebind with the name **StaffServer**. Client will only look up the registry with the same name and same port.

Thus, process starts and as many client can reach the server as they are. In projected system, three server ports are created for three packages (staff, patient, appointment) and client will awake the methods from all three server packages.

5 Implementation

The picture below shows one of the most interesting parts of the code since there is more than one package involved in this class.

addAppointmentController represents the power of the MVC design pattern and the flexibility of interacting with more than one package simultaneously. This controller is responsible one major thing, which is making the communication between the view interface and the client. Taking into consideration the solid principles, this controller does not break the single responsibility principle, since its only responsibility is making the connection between the mediator and the view.
The aim of using this controller is to Search for a specific patient from the database and return the result to the user, even if there were two patients sharing the same name they will be returned to the user as a list.

Managing the search function required the following steps:
The process will start with taking the value that has been entered from the user getSearchTxtValue().

. sidebar, to avoid searching for an empty value, the system will warn the user if there are no entries in the text field.

The controller will pass the name to the database using this method client.callLoadPateint (name) which will make its way through the client and from it to the server. Server will redirect the command to retrieve the data from the database using load method. Searching inside the database is so organized since the system applies some roles when writing and when reading.

name=name.substring(0,1).toUpperCase()+
getSearchTxtValue().substring(1).toLowerCase.
After retrieving patient from the database, all patients will be saved in an arraylist of type Patient and the controller will ask the model to get the arraylist.

addAppointmentGUI.showTable(client.getAllPatientsFromTheList()); showtable will be responsible for showing the result using a table.

Next step is to call these two methods to fill the comboBoxes with the available appointment from the database, and setPatientIdComboBoxValue which fills the second comboBox with the founded patients from the list.

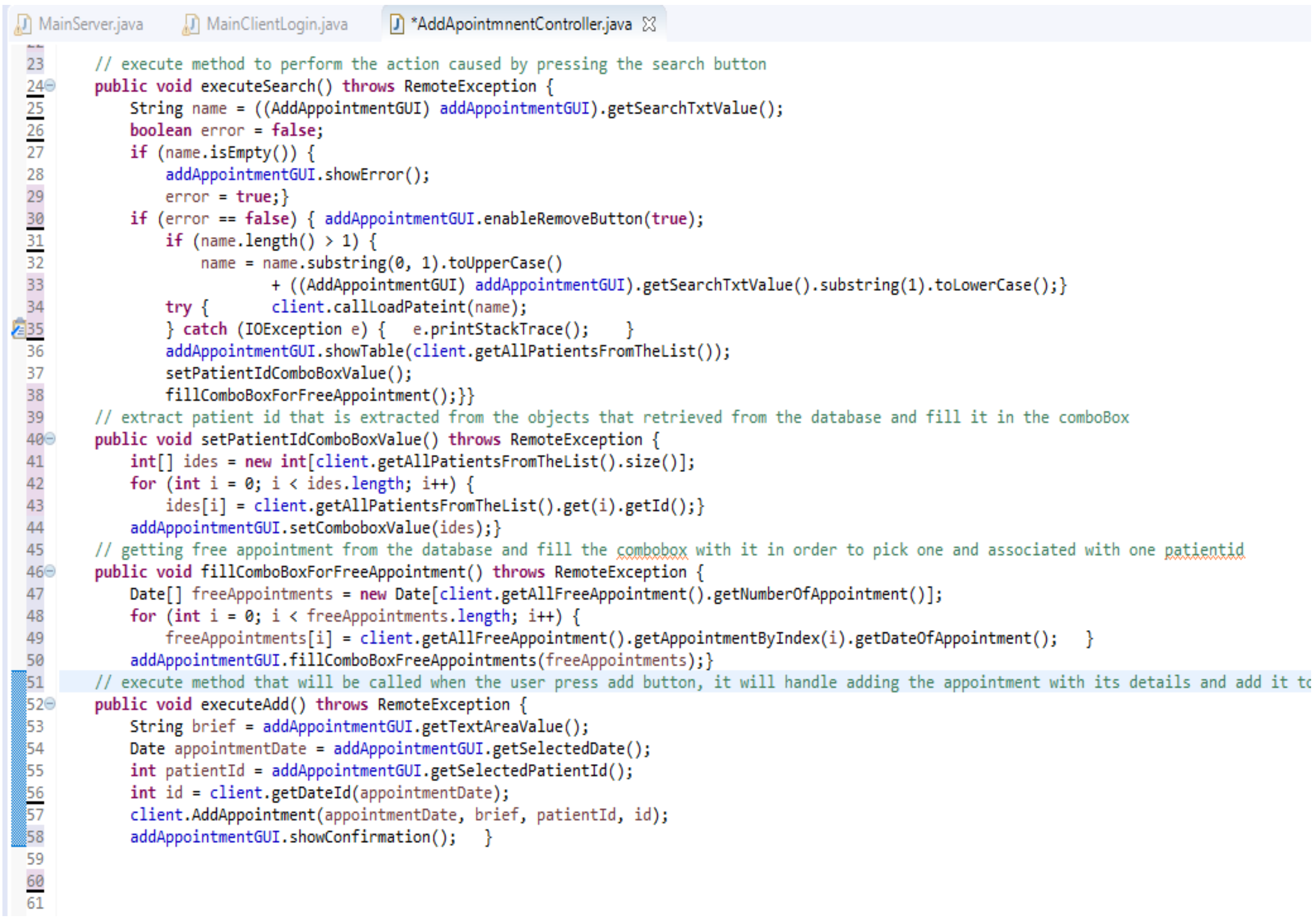
From this point on, the user is ready to select an appointment from the list and assign a patient id to it from the combobox.
by pressing add button executeAdd() method will run and collect all the information using these methods

```
  .getTextAreaValue();
 getSelectedDate();
 getSelectedPatientId();
```

The last step is to get the id for the date to create an appointment object and write it to the database. client.getDateId(appointmentDate); last and not least adding the appointment to the database using this method
client.AddAppointment(appointmentDate, brief, patientId, id);

Finally, a confirmation window will pop up to the user to confirm the success of the entire process. addAppointmentGUI.showConfirmation();

Figure 5.1 Add employee controller



```java
    // execute method to perform the action caused by pressing the search button
    public void executeSearch() throws RemoteException {
        String name = ((AddAppointmentGUI) addAppointmentGUI).getSearchTxtValue();
        boolean error = false;
        if (name.isEmpty()) {
            addAppointmentGUI.showError();
            error = true;}
        if (error == false) { addAppointmentGUI.enableRemoveButton(true);
            if (name.length() > 1) {
                name = name.substring(0, 1).toUpperCase()
                        + ((AddAppointmentGUI) addAppointmentGUI).getSearchTxtValue().substring(1).toLowerCase();}
            try {       client.callLoadPateint(name);
            } catch (IOException e) {    e.printStackTrace();     }
            addAppointmentGUI.showTable(client.getAllPatientsFromTheList());
            setPatientIdComboBoxValue();
            fillComboBoxForFreeAppointment();}}
    // extract patient id that is extracted from the objects that retrieved from the database and fill it in the comboBox
    public void setPatientIdComboBoxValue() throws RemoteException {
        int[] ides = new int[client.getAllPatientsFromTheList().size()];
        for (int i = 0; i < ides.length; i++) {
            ides[i] = client.getAllPatientsFromTheList().get(i).getId();}
        addAppointmentGUI.setComboboxValue(ides);}
    // getting free appointment from the database and fill the combobox with it in order to pick one and associated with one patientid
    public void fillComboBoxForFreeAppointment() throws RemoteException {
        Date[] freeAppointments = new Date[client.getAllFreeAppointment().getNumberOfAppointment()];
        for (int i = 0; i < freeAppointments.length; i++) {
            freeAppointments[i] = client.getAllFreeAppointment().getAppointmentByIndex(i).getDateOfAppointment();    }
        addAppointmentGUI.fillComboBoxFreeAppointments(freeAppointments);}
    // execute method that will be called when the user press add button, it will handle adding the appointment with its details and add it to
    public void executeAdd() throws RemoteException {
        String brief = addAppointmentGUI.getTextAreaValue();
        Date appointmentDate = addAppointmentGUI.getSelectedDate();
        int patientId = addAppointmentGUI.getSelectedPatientId();
        int id = client.getDateId(appointmentDate);
        client.AddAppointment(appointmentDate, brief, patientId, id);
        addAppointmentGUI.showConfirmation();    }
```

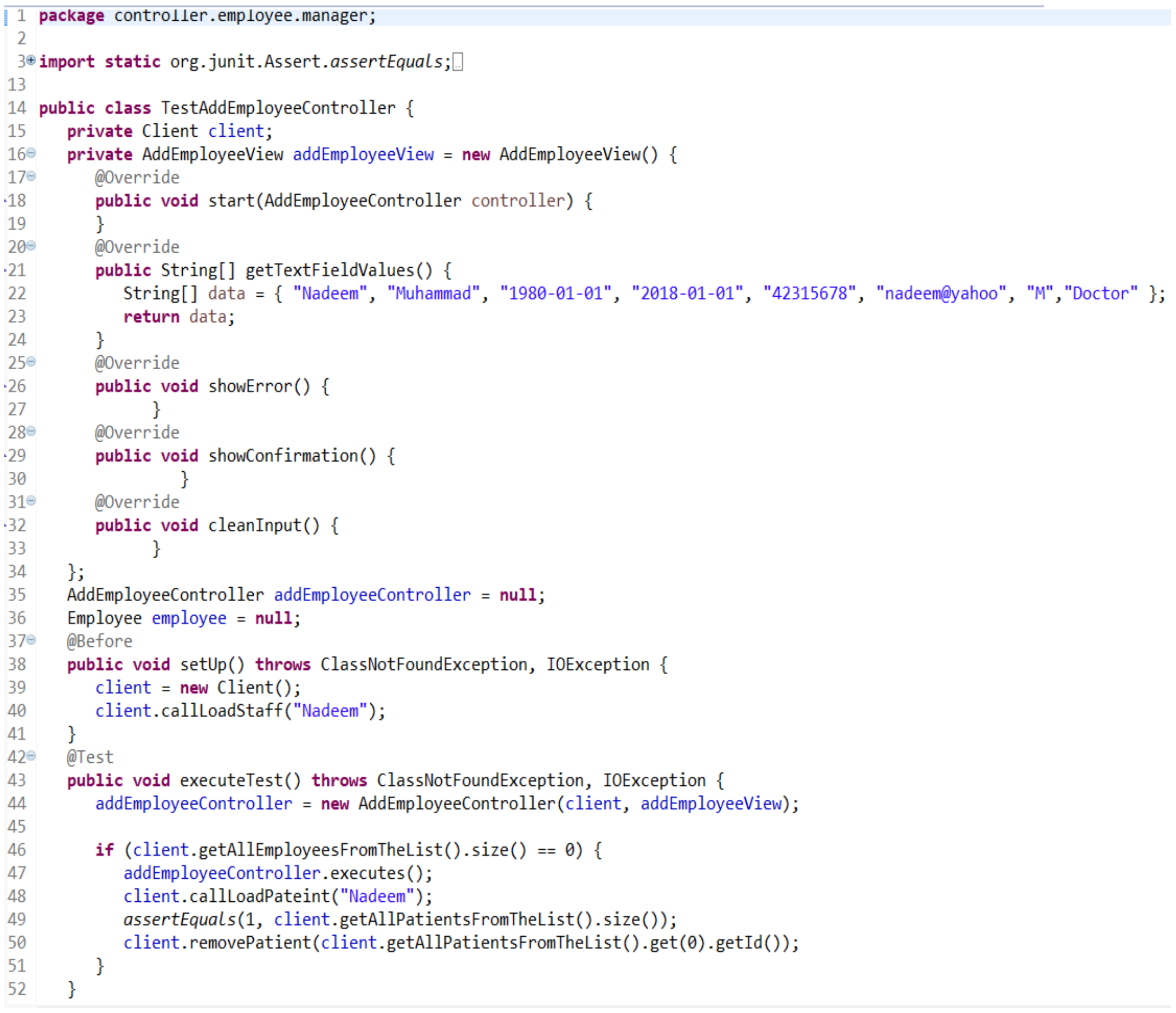
6 Testing

When the program is finished it is time to test it.

The testing of the program consists of all methods being tested to assure that each and every method is working as it should and no exceptions are being thrown. To achieve the desired result Junit testing was used. Unit testing has been chosen since it offers the possibility to test methods one by one.

Here is one class that got all its methods tested i.e. addEmployeeController
To see the rest of the tests, see ***Appendix 6 Testing***

Figure 6.1 Add employee controller test



```java
package controller.employee.manager;

import static org.junit.Assert.assertEquals;

public class TestAddEmployeeController {
   private Client client;
   private AddEmployeeView addEmployeeView = new AddEmployeeView() {
      @Override
      public void start(AddEmployeeController controller) {
      }
      @Override
      public String[] getTextFieldValues() {
         String[] data = { "Nadeem", "Muhammad", "1980-01-01", "2018-01-01", "42315678", "nadeem@yahoo", "M","Doctor" };
         return data;
      }
      @Override
      public void showError() {
            }
      @Override
      public void showConfirmation() {
               }
      @Override
      public void cleanInput() {
            }
   };
   AddEmployeeController addEmployeeController = null;
   Employee employee = null;
   @Before
   public void setUp() throws ClassNotFoundException, IOException {
      client = new Client();
      client.callLoadStaff("Nadeem");
   }
   @Test
   public void executeTest() throws ClassNotFoundException, IOException {
      addEmployeeController = new AddEmployeeController(client, addEmployeeView);

      if (client.getAllEmployeesFromTheList().size() == 0) {
         addEmployeeController.executes();
         client.callLoadPateint("Nadeem");
         assertEquals(1, client.getAllPatientsFromTheList().size());
         client.removePatient(client.getAllPatientsFromTheList().get(0).getId());
      }
   }
```

In this class, execute method has been chosen to be tested. Firstly, the addEmployeeController has been initialized inside the **executeTest()** and it contains the client and the view as argument.

Conditions are made to check that client side has the list of employees. Employee is searched from the database, then check with **assertEqual()** to see if the employee has been added. And in remove the employee from the List. The test gave us the green sign, so all the methods are functioning.

7 Conclusion

The project “Clinical management system” has been completed, but due to lack of time there are few delimitations.

It is required by the university that the project should follow the Client Server and it also should have multiple design patterns. To follow these requirements, many user stories where drawn and it took a long time to make a correct class diagram that would be relevant to the user’s needs. Designing and Coding was challenging and took a large portion of time. Since the class diagram is quite big and it is divided into two parts, meaning that coding took a lot of time out of the project.

In conclusion, the project is fully functioning and most of the requirements have been met. The design of the project is good, and it follows the SOLID principles, so any amendments can be possible for future development.

These are the delimitations:

8 References

Guidelines Development and Summary, IT-SDJ2X-S18 Sesion Material,[Last accessed 05/04/2018] via link: https://studienet.via.dk/Class/IT-SDJ2X-S18/Session%20Material/SDJ2-S18-19GuidelinesDeploymentAndSummary.pdf

Ken Schwaber, Jeff Sutherland, 2011; The Scrum Guide, [Last accessed 10/04/2018] via link: https://studienet.via.dk/Class/IT-SWE1X-S18/Session%20Material/Scrum_Guide.pdf

Project report, 2017 (Appendix 3) VIA Engineering Guidelines [Last accessed 10/04/2018] via link: https://studienet.via.dk/projects/Engineering__project_methodology/General/Guidelines/2017%20Project%20Report%20(Appendix%203)%20-%20VIA%20Engineering%20Guidelines.pdf

Project description, 2017 (Appendix 1) VIA Engineering Guidelines [Last accessed 27/02/2018] via link: https://studienet.via.dk/projects/Engineering__project_methodology/General/Guidelines/2017%20Project%20Description%20(Appendix%201)%20-%20VIA%20Engineering%20Guidelines.pdf

Process report,2017 (Appendix 2) VIA Engineering Guidelines, [Last accessed 05/04/2018] via link: https://studienet.via.dk/projects/Engineering__project_methodology/General/Guidelines/2017%20Process%20Report%20(Appendix%202)%20-%20VIA%20Engineering%20Guidelines.pdf

Rohit Joshi, 2015; Java Design Patterns, [Last accessed 14/03/2018] via link; http://enos.itcollege.ee/~jpoial/java/naited/Java-Design-Patterns.pdf

Thomas Connolly, Carolyn Begg, 2015; Database Systems A Practical Approach to Design,Implementation and Management, [Last accessed 18/05/2018] via link; http://people.stfx.ca/x2011/x2011asx/5th%20Year/Database/Database%20Management%20Textbook.pdf

Videos:

Programming Knowledge Learning, 2014,July,Java Eclipse GUI Tutorial 1 Creating First GUI Project in Eclipse, [Video file]. Retrieved from https://www.youtube.com/watch?v=r8Qiz9Bn1Ag

9 Appendices

9.1 Appendix 1 Project Description

9.2 Appendix 2 Use case description

9.3 Appendix 3 USER GUIDE

9.4 Appendix 4 Diagrams

9.5 Appendix 5 Scenarios

9.6 Appendix 6 Testing