CSE 2112: Object Oriented Programming
Project: Hospital Management Sytem

Submitted By:

Ahmed Nesar Tahsin Chowdhury (02)

Mehrajul Abadin Miraj (20)

Jannatin Tajri (42)

Submitted To:

Dr. Muhammed Ibrahim

Associate Professor

Department of Computer Science and Engineering

University of Dhaka

Md. Ashraful Islam

Assistant Professor

Department of Computer Science and Engineering

University of Dhaka

Requirement Analysis:

The Hospital Management System (HMS) is a comprehensive software application developed using JavaFX and SQLite database, aimed at streamlining and automating various administrative tasks and activities within a hospital environment. This project serves as a reliable and efficient solution to manage doctor-patient relationships, appointments, and overall hospital operations.

The primary objective of the Hospital Management System is to provide a user-friendly interface that simplifies the day-to-day operations of doctors and patients. By leveraging the power of JavaFX, a modern and intuitive graphical user interface (GUI) has been created to ensure ease of use and enhance productivity.

The system offers a wide range of functionalities, including the ability to add, delete, and modify doctor and patient information. Hospital staff can effortlessly schedule appointments between doctors and patients, allowing for efficient allocation of resources and minimizing waiting times. Additionally, the system allows for the rescheduling of appointments, ensuring flexibility in managing changes or emergencies.

A key feature of the HMS is its robust search functionality. Users can swiftly search for doctors, patients, and appointments based on various parameters such as name, specialization, medical history, or appointment date. This enables hospital staff to quickly locate the relevant information they need, facilitating better decision-making and providing optimal patient care.

The integration of SQLite database ensures data reliability. All the information related to doctors, patients, appointments, and other essential details are easily accessible. The database management system offers scalability, ensuring the system can handle a growing number of doctors, patients, and appointments without compromising performance.

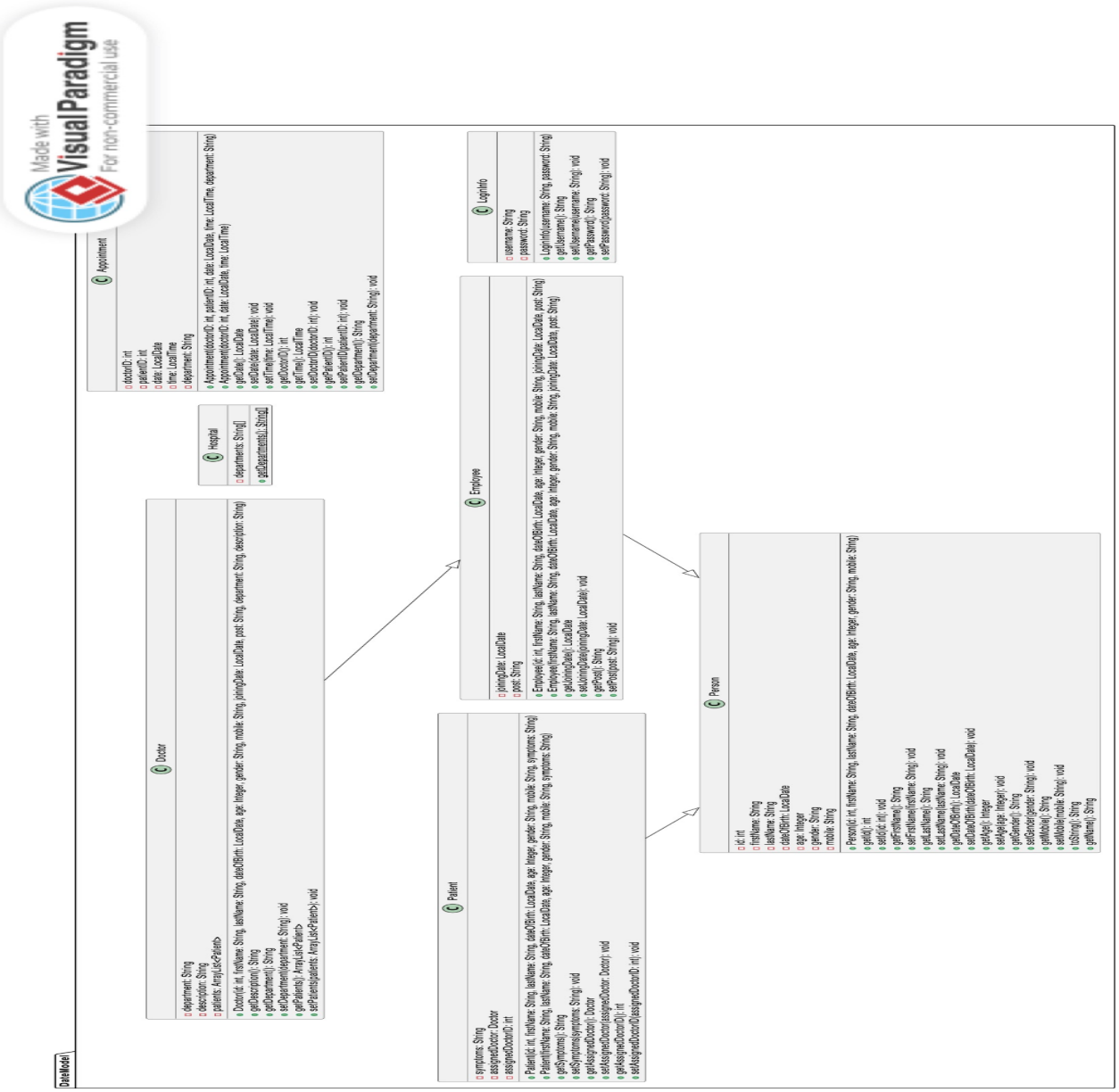
Overall, the Hospital Management System developed using JavaFX and SQLite database provides an efficient and user-friendly solution for hospitals to manage their day-to-day operations effectively. By automating various administrative tasks, the system reduces paperwork, streamlines processes, and improves the overall efficiency of the hospital.

System Design:

Our project is divided into four packages. They are: datamodel, database, utils and controllers. The packages are discussed in detail below:

datamodel:

The following class hierarchy is used to represent the data models in our project:



The package has two levels of inheritance. The detailed description of the classes are discussed here:

- **Person:**
  This is an abstract class made for holding a person's information like first name, last name, data of birth etc. Since both patients and employees share these information, constructing this superclass ensures efficiently written code. It is an abstract class because it doesn't need to be instantiated. There are some getter and setter methods in this class to use the information.

- **Patient:**
  Patient is a subclass of the Person class that stores information like symptoms, assigned doctor id etc. There are some getter and setter methods in this class to use the information.

- **Employee:**
  This class contains information related to employees like post, joining date etc. There are some getter and setter methods in this class to use the information.

- **Doctor:**
  This class contains information related to employees like department, job description etc. There are some getter and setter methods in this class to use the information.

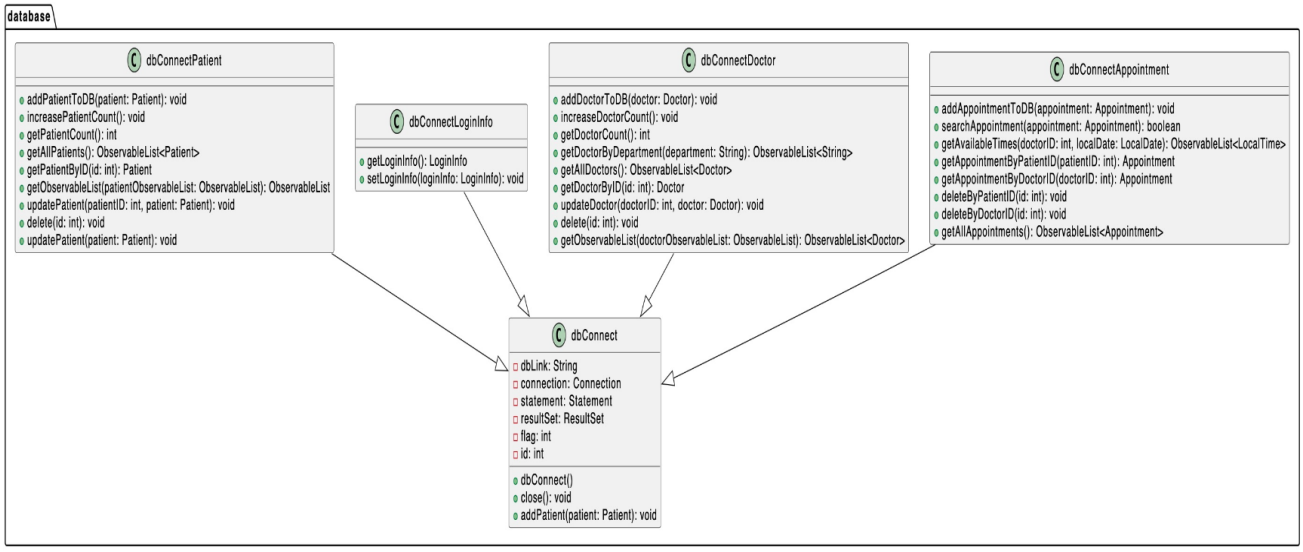
- **LoginInfo:**
  This class contains information related to login.

- **Appointment:**
  This class contains information related to an appointment, like the corresponding doctor id, patient id, date, time, and department. There are some getter and setter methods in this class to use the information.

- **Hospital:**
  This class contains the names of the departments available at the hospital.

database:

Inside the database package, there are some database related files which handle everything related to databases, like searching for some information, querying and so on. The class hierarchy is:



- **dbConnect:**
  This is the superclass to all the classes available in the database package.
  - The **constructor** of this java file forms the connection to the SQLite database of the project.
  - The **close()** method is used to close the connections, result set, and statement.

- **dbConnectPatient:**
  This module contains methods that are related to patient information in the database. The methods are discussed below:
  - **addPatientToDB()** takes a Patient object, and adds the information to the database.
  - **increasePatientCount()** increases the count of the number of patients in the hospital, and updates the database.
  - **getPatientCount()** returns the number of patients (in total) in the hospital.
  - **getAllPatients()** returns an Observable List of Patient type that contains all the patients in the hospital.
  - **getPatientById()** takes an id, and returns the corresponding patient.

- **updatePatient()** is used to update the information of a patient.
- **delete()** method takes an id, and deletes the corresponding patient information.

- **dbConnectDoctor:**
  This module contains methods that are related to doctor information in the database. The methods are discussed below:
  - **addDoctorToDB()** takes a Doctor object, and adds the information to the database.
  - **increasePatientCount()** increases the count of the number of doctors in the hospital, and updates the database.
  - **getDoctorCount()** returns the number of doctors (in total) in the hospital.
  - **getAllDoctors()** returns an Observable List of Doctor type that contains all the doctors in the hospital.
  - **getDoctorById()** takes an id, and returns the corresponding doctor.
  - **updateDoctor()** is used to update the information of a doctor.
  - **delete()** method takes an id, and deletes the corresponding doctor information.
  - **getDoctorByDepartment()** takes a department, and returns an ObservableList<Doctor> that contains all the doctors in that department.

- **dbConnectAppointment:**
  This module contains methods that are related to appointment information in the database. The methods are discussed below:
  - **addAppointmentToDB()** takes an Appointment object, and adds the information to the database.
  - **searchAppointment()** takes an appointment, and returns true if such an appointment is present in the database.
  - **getAvailableTimes()** takes a doctor id and a LocalDate object, and returns ObservableList<LocalTime> that contains all the free appointment times of that doctor.
  - **getAppointmentByPatientID()** takes a patient id, and returns an Appointment object that contains the corresponding appointment information.
  - **getAppointmentByDoctorID()** takes a doctor id, and returns an Appointment object that contains the corresponding appointment information.
  - **deleteByPatientID()** and **deleteByDoctorID()** are used to delete appointment records by patient id and doctor id respectively.

    - **getAllAppointments()** returns an Observable List<Appointment> that contains all the scheduled appointments.

- **dbConnectLoginInfo:**
  This module contains methods that are related to login information in the database. The methods are discussed below:
    - **getLoginInfo()** gets the login info required to login from the database. Returns a LoginInfo object.
    - **setLoginInfo()** takes a LoginInfo object, and updates the username-password on the database.

utils:

This package contains some utility methods.

- **AlertUtils:**
  This module contains some alert related methods.
    - **getConfirmation()** shows an alert box to the user and lets the user decide whether they want to go through with their command or not.
    - **showAlert()** takes a message as a String, and displays an alert with that message.

- **Authorizer:**
  This module is used to authorize login info.
    - **authorize()** takes a LoginInfo object and returns true if that login info contains the correct information.

- **DerivedValueCallback:**
  This is an utility class for Table Columns.
    - The **call()** method is used to show information in a column based on the property name.

controllers:

This package contains the controller classes.

- **Controller:**
  Superclass to all the controllers. The methods are:
    - **switchToMenu()** switches to menu page.
    - **switchToDoctor()** switches to doctor page.

    - **switchToPatient()** switches to patient page.

- **LoginPageController:**
  Controller class for the login page. The methods are:
    - **initialize()** method is used to initialize the text fields.
    - **login()** method is used to check whether the login info entered is correct or not, and switches to the menu if the information is correct. This method is called when the login button is pressed.

- **MenuController:**
  Controller class for the menu page. The methods are:
    - **initialize()** method for initializing.
    - **logout()** method is called when the logout button is pressed. It is used to log out of the application. Takes the software back to the login page.

- **UpdateLoginInfoController:**
  Controller class for the update login info page. The methods are:
    - **save()** to check if username and password is entered, and save the changes if they are done correctly.

- **DoctorController:**
  Controller class for the doctor page. The methods are:
    - **addDoctor()** method is used to load the Add Doctor Page (linked to the Add Doctor Button).
    - **searchDoctor()** method is used to load the page where doctors can be searched by their name (linked to the Search Doctor Button).

- **PatientController:**
  Controller class for the patient page. The methods are:
    - **addPatient()** method is used to load the Add Patient Page (linked to the Add Patient button).
    - **searchPatient()** method is used to load the page where patients can be searched by their name (linked to the Search Patient button).

- **SearchAppointment:**
  Controller for the Search Appointment page. The methods are:
    - **initialize()** method for initializing.
    - **switchToMenu()** for returning to the menu page.

- **AddDoctorController:**

Controller for the page where doctor information is to be given and added to the software, The methods are:

- **initialize():** Used for initializing.
- **submit()** method is called when the submit button is pressed. It checks whether information is entered correctly, shows an alert, and takes the user back to the doctor page if the information is successfully saved.

- **DoctorSearchController:**
  Controller for the doctor search page. The methods are:
  - **initialize()** method for initializing.
  - **switchToDisplayDoctor()** switches to the Display Doctor Page.
  - **returnToDoctorPage()** switches to the Doctor Page.

- **AddPatientController:**
  Controller for the page where patient information is to be given and added to the software, The methods are:
  - **createNewPatient()** is linked to the Create New Patient button. Checks whether the information entered is correct, sends the information to the database, and switches to the appointment page.
  - **goToAppointmentPage()** switches to the appointment page.
  - **returnToPatientPage()** switches to the patient page (linked to the return button on the top right).

- **PatientSearchController:**
  Controller for the page where patient information is searched, The methods are:
  - **initialize()** method is used for initializing.
  - **switchToDisplayPatient()** switches to the display patient page.
  - **switchToPatient()** switches to the patient page.

- **DisplayDoctorController:**
  Controller class for the page that displays information related to a doctor. The methods are:
  - **setDoctor()** method takes a Doctor and and sets the Doctor object in the class.
  - **init()** method initializes all the fields.
  - **modifyButtonClicked()** is linked to the Modify button. When the button is pressed, the function gets called. It checks whether all the information is entered correctly, and if so, the information is sent to the database.

- **deleteDoctorButtonClicked()** asks the user if the user is sure about his decision, and then calls the database to delete this doctor.

- **DisplayPatientController:**
  Controller class for the page that displays information related to a patient. The methods are:
  - **setPatient():** setter for the Patient object.
  - **initialize()** method initializes the labels.
  - **init()** method initializes the text fields containing information of the patient.
  - **modifiyPatientClicked():** checks if information is altered correctly, and sends it to the database.
  - **reschedulePatientClicked():** asks the database to detete the current appointment, and switches to the page where a new appointment can be created.
  - **deletePatientClicked():** gets confirmation from the user, and calls the database method to delete the patient.

- **DisplayAppointmentController:**
  Controller class for the page that displays information related to an appointment. The methods are:
  - **setAppointment():** setter for the appointment object.
  - **init()** method initializes the labels according to the information in the appointment object.

- **NewAppointment:**
  Controller class for the page that allows the user to add an appointment for a patient. The methods are:
  - **setPatient():** setter for the Patient object.
  - **initialize():** initializes different methods for the choice boxes.
  - **submit():** Linked to the Submit button. Checks whether all the choice boxes have been chosen correctly, and calls database methods to store the appointment information.

Discussion - Java Features Used:

1. **Lambda Expressions:** Lambda expressions in Java are a powerful feature introduced in Java 8 that allows the implementation of functional programming concepts. A lambda expression is a concise way to represent an anonymous function, which can be treated as a method argument or assigned to a functional interface. In the project, there are a lot of files in which lambda expressions were

used. For example, in PatientSearchController, lambda expressions are used to add listener to the Table View that displays all the patients.

2. **Inheritance:** Inheritance is a fundamental concept in object-oriented programming (OOP) and plays a vital role in Java programming language. It allows classes to inherit or acquire properties and behaviors from other classes, forming a hierarchy or relationship between them. In the project, we used inheritance in the several packages including datamodel, database etc. as can be seen from the class diagram in the System Design section.
3. **Polymorphism:** Polymorphism is a powerful feature of Java that promotes code reusability, flexibility, and modularity. In the project, we used method overriding while implementing the Initializable interface in our controller classes. For instance, in the PatientSearchController, initialize() method of the Initiaizable interface was overridden to add listeners to the table view, among other things. The Initializable interface was overridden in other files as well, thus the same method name has different functionalities in different files. This is polymorphism,
4. **Encapsulation:** Encapsulation is a fundamental concept in object-oriented programming (OOP) that aims to bundle data and methods together into a single unit called a class. It provides a way to organize code by hiding internal details and exposing a public interface for interacting with the class. In our project, an instance of using encapsulation is the use of model classes like Doctor, Patient etc.
5. **Collections Framework:** The Collections Framework in Java is a unified architecture that provides a set of classes and interfaces to handle and manipulate groups of objects, commonly referred to as collections. In the database related files, ArrayList was used.
6. **FXCollections:** FXCollections is a utility class provided by JavaFX that offers convenient methods for creating and manipulating observable collections. In database related files and some controller classes, ObservableList was used.
7. **Abstraction:** Abstract class (the Person class in datamodel package) was used in the project to ensure maximum code efficiency.
8. **Exception Handling:** Java allows exception handling through the use of try-catch block, among other ways. This was done in the project in several files. For instance, in the database related files, SQLException can occur in several methods, and they were handled using try-catch block.

Future Plan:

- Improve the graphical representation of the Software.
- Feature upgrade for making it more reliable.
- Using networking to support multiple users simultaneously.
- Allow other users like doctors, patients, nurses etc. to use the software.
- Take user feedback and improve features according to their opinion.
- Introduce new features like bill payment, keeping patient history, lab testing etc.

Conclusion:

In conclusion, developing the hospital management project using JavaFX has been instrumental in providing a practical understanding of Object-Oriented Programming (OOP) concepts. JavaFX's features and capabilities enabled us to apply encapsulation, inheritance, polymorphism, abstraction, and association concepts effectively. The project demonstrated the relevance and power of OOP principles in building robust and adaptable software solutions.