Vanier College

Deliverable 4

Medical Management System

Jacques-Conrad Calagui-Painchaud

420-SF2-RE

Yi Wang

2025-05-11

Contents

Project Description

The Online Medical Management System is designed for digital health clinics and hospitals to streamline common medical services. The application allows patients to book appointments with doctors, receive e-prescriptions, and view their medical records and lab reports. Doctors can offer healthcare suggestions and consult patient records. The system also allows users to connect with blood and eye donors.

Program Features and Screenshots

Runtime-Polymorphism

```java
/**
 * Prescribes a medical order to a patient
 * @param patient The patient in need of a medical order
 * @param medicalOrder The order to be prescribed
 */
public void prescribeMedicalOrder(Patient patient, MedicalOrder medicalOrder) {
    if (patient == null) {
        throw new NullPointerException("Patient cannot be null");
    }

    if (medicalOrder == null) {
        throw new NullPointerException("Medical order cannot be null");
    }

    if (medicalOrder instanceof Prescription prescription) {
        patient.getPrescriptions().add(prescription);
    } else if (medicalOrder instanceof LabTest labTest) {
        patient.getLabTests().add(labTest);
    } else if (medicalOrder instanceof DonorRequest donorRequest) {
        patient.getDonorRequests().add(donorRequest);
    }
}
```

```java
public static void main(String[] args) {
    Patient patient = new Patient( name: "Nelson", User.Sex.MALE);
    Doctor doctor = new Doctor( name: "Hippocrates", User.Sex.MALE);

    Prescription prescription = new Prescription(doctor, patient, medication: "a", dosage: "b", frequency: "c", LocalDateTime.now());
    LabTest labTest = new LabTest(doctor, patient, labName: "L");

    doctor.prescribeMedicalOrder(patient, prescription);
    doctor.prescribeMedicalOrder(patient, labTest);

    patient.viewPrescriptions();
    patient.viewLabTests();
}
```

```
Medication: a
Dosage: b
Frequency: c
Expiry Date: 2025-05-12T00:23:31.812134900
Status: IN_PROGRESS

Lab Name: L
Collection Date: null
Result Date: null
Summary: null
Status: IN_PROGRESS
```

I/O writing

```java
/**
 * Exports donor information to a separate csv file
 */
public static void exportDonors() {
    String filePath = "src/main/resources/donors.csv";

    try (FileWriter fileWriter = new FileWriter(filePath)) {
        for (Donor donor : donors) {
            int id = donor.getId();
            String name = donor.getName();
            User.Sex sex = donor.getSex();
            DonorType donorType = donor.getDonorType();
            BloodType bloodType = donor.getBloodType();
            boolean isAvailable = donor.isAvailable();

            fileWriter.write( str: id + ","
                    + name + ","
                    + sex + ","
                    + donorType + ","
                    + bloodType + ","
                    + isAvailable + "\n");
        }
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

```java
public static void main(String[] args) {
    Donor donor = new Donor( name: "donorguy", User.Sex.MALE, DonorType.BLOOD, BloodType.A_NEGATIVE,  isAvailable: true);
    Donor donor2 = new Donor( name: "donorlady", User.Sex.FEMALE, DonorType.EYE,  isAvailable: false);

    HospitalManagementSystem.exportDonors();
}
```

```
0,donorguy,MALE,BLOOD,A_NEGATIVE,true
1,donorlady,FEMALE,EYE,null,false
```

Implementation of Comparable<Appointment> in the Appointment class

```java
@Override
public int compareTo(Appointment o) {
    return date.compareTo(o.date);
}
```

```java
public static void main(String[] args) {
    Patient patient = new Patient( name: "Nelson", User.Sex.MALE);
    Doctor doctor = new Doctor( name: "Hippocrates", User.Sex.MALE);

    LocalDateTime now = LocalDateTime.now();
    LocalDateTime later = LocalDateTime.of( year: 2027, month: 1, dayOfMonth: 1, hour: 1, minute: 1);
    LocalDateTime muchLater = LocalDateTime.of( year: 3999, month: 1, dayOfMonth: 1, hour: 1, minute: 1);

    List<Appointment> appointments = new ArrayList<>();

    Appointment appointment1 = new Appointment(patient, doctor, muchLater);
    Appointment appointment2 = new Appointment(patient, doctor, now);
    Appointment appointment3 = new Appointment(patient, doctor, later);

    appointments.add(appointment1);
    appointments.add(appointment2);
    appointments.add(appointment3);

    Collections.sort(appointments);
```

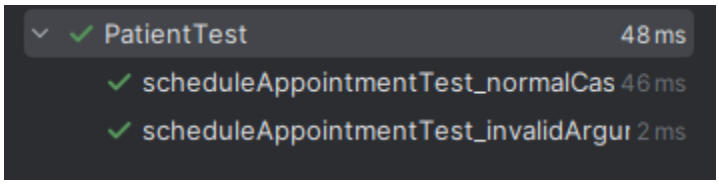
```
Patient: Nelson
Doctor: Hippocrates
Date: 2025-05-12T00:19:20.274714800
Status: REQUESTED

Patient: Nelson
Doctor: Hippocrates
Date: 2027-01-01T01:01
Status: REQUESTED

Patient: Nelson
Doctor: Hippocrates
Date: 3999-01-01T01:01
Status: REQUESTED
```

Implementation of Comparator<Prescription> in the Prescription class

```java
public class PrescriptionComparator implements Comparator<Prescription> {
    String type;

    public PrescriptionComparator(String type) {
        this.type = type;
    }

    @Override
    public int compare(Prescription o1, Prescription o2) {
        return switch (type) {
            case "expiry" -> expiryDate.compareTo(o1.expiryDate);
            case "status" -> o1.getStatus().compareTo(o2.getStatus());
            default -> o1.medication.compareTo(o2.medication);
        };
    }
}
```

Unit testing example for scheduleAppointment method in Patient class

```java
@Test
public void scheduleAppointmentTest_normalCase() {
    Patient patient = new Patient( name: "Nelson", User.Sex.MALE);
    Doctor doctor = new Doctor( name: "Hippocrates", User.Sex.MALE);
    LocalDateTime time  = LocalDateTime.of( year: 2025,  month: 12,  dayOfMonth: 16,  hour: 23,  minute: 56);

    patient.scheduleAppointment(doctor, time);

    Assertions.assertEquals(patient.getAppointments().getLast(), doctor.getAppointments().getLast());
}

@Test
public void scheduleAppointmentTest_invalidArgumentsCase() {
    Patient patient = new Patient( name: "Nelson", User.Sex.MALE);
    Doctor doctor = new Doctor( name: "Hippocrates", User.Sex.MALE);
    LocalDateTime time  = LocalDateTime.of( year: 460,  month: 1,  dayOfMonth: 1,  hour: 1,  minute: 1);

    Assertions.assertThrows(IllegalArgumentException.class,
            () -> patient.scheduleAppointment(doctor, time));

    Assertions.assertThrows(NullPointerException.class,
            () -> patient.scheduleAppointment( doctor: null, time));
}
```



Challenges

Planning out the project was by far the most difficult part. It took me 10x as long to plan than to code anything. It seemed like no matter what I did, something wasn't right, and when I started coding I noticed that more and more issues would come up due to poor planning. One such example of poor planning would be Patient and Appointment each having variables referencing each other. This sometimes lead to stack overflow issues, like in Patient's toString() method which calls Appointments toString() method, which calls Patient's toString(), which calls… It was a simple change to edit it to 'patient.getName()' instead of 'patient,' but this would almost certainly lead to more issues down the line if I were to continue on the project.

I didn't end up implementing a couple things like importing, because I couldn't figure out any good way of importing appointments. To import and appointment, I would need to somehow store a Patient or Doctor in a single element (this is one of the issues that arose from Patient and Appointment referencing each other)

Learning Outcomes

Planning is **extremely** important and takes **much** longer than any actual coding does. Poor planning leads to frustration and often led me to consider restarting (and I essentially did). Additionally, having a robust plan before starting to code saves you a lot of headache from having to try to understand how every component is going to interact with each other element whilst your coding, and keeps you focused on one thing at a time.