

# Course Work 1

Module: CI6115 Programming III - Patterns and Algorithms

Module Leader: Mr. Nathindu Himansha

Assignment Type: Individual

Submission Date: 2024.11.09

Student ID: E112691 / K2434232

Student Name: Ranuga Disansa Belpa Gamage

Student Email: [e112691@esoft.academy](mailto:e112691@esoft.academy)

## Part 1 - To Remain with the Assignment after Marking

<b>Student ID: E112691 / K2434232</b>	<b>Student Name: Ranuga Disansa Gamage</b>
<b>Module Code: CI6115</b>	<b>Module Name: Programming III - Patterns and Algorithms</b>
<b>Assignment number: 01</b>	<b>ESoft Module Leader: Mr. Nathindu Himansha</b>
<b>Date set: 10/10/2024</b>	<b>Date due: 09/11/2024</b>

### Guidelines for the Submission of Coursework

1. Print this coversheet and securely attach both pages to your assignment. You can help us ensure work is marked more quickly by submitting at the specified location for your module. You are advised to keep a copy of every assignment.

2. Coursework deadlines are strictly enforced by the University.

3. You should not leave the handing in of work until the last minute. Once an assignment has been submitted it cannot be submitted again.

**Academic Misconduct:** **Plagiarism** and/or **collusion** constitute **academic misconduct** under the University's Academic Regulations. Examples of academic misconduct in coursework: making available your work to other students; presenting work produced in collaboration with other students as your own (unless an explicit assessment requirement); submitting work, taken from sources that are not properly referenced, as your own. By printing and submitting this coversheet with your coursework you are confirming that the work is your own.

ESoft Office Use Only:

Date stamp: work received

## Coursework Coversheet

### Part 2 – Student Feedback

<b>Student ID: E112691 / K2434232</b>	<b>Student Name: Ranuga Disansa Gamage</b>
<b>Module Code: CI6115</b>	<b>Module Name: Programming III - Patterns and Algorithms</b>
<b>Assignment number: 01</b>	<b>ESoft Module Leader: Mr. Nathindu Himansha</b>
<b>Date set: 10/10/2024</b>	<b>Date due: 09/11/2024</b>

Strengths (areas with well-developed answers)

Weaknesses (areas with room for improvement)

Additional Comments

**ESoft Module Lecturer:**

**Provisional mark as %:**

**ESoft Module Marker:**

**Date marked:**

## Acknowledgement

I would like to express my gratitude to those who have supported me in completing this coursework.

First, I would like to thank Mr. Nathindu Himansha, the Module Leader for CI6115 Programming III - Patterns and Algorithms. His guidance and expertise have been invaluable throughout this assignment, providing me with a deeper understanding of programming patterns and algorithms.

Additionally, I am grateful for the continuous encouragement from my family, whose support has been essential throughout the completion of this coursework.

Thank you.

## Table of Contents

Class Diagram.....	8
Layers in the Diagram .....	9
Presentation Layer .....	9
Service Layer.....	9
Repository Layer .....	9
Domain Layer .....	9
Relationships in the Diagram.....	10
Overall Structure of the Diagram .....	10
Test Cases .....	11
Test Plan.....	11
Test Scope .....	11
Core Components Under Test .....	11
Testing Approaches .....	11
Test Cases.....	12
Test Case 1: Appointment Scheduling.....	12
Test Case 2: View All Appointments .....	13
Test Case 3: Update Appointment .....	14
Test Case 4: Cancel Appointment.....	15
Test Case 5: Complete Appointment.....	16
Future Testing Plans.....	16
Implementation .....	17
Make Appointments.....	17
Update Appointment Details .....	18
View Appointment Details Filtered by Date.....	19
Search for an Appointment Using Either the Patient's Name or the Appointment ID .....	20
Accept Registration Fees When Placing Appointments .....	21
Calculate the Total Fee and Taxes for Treatments After the Appointment.....	22
Generate an Invoice for the Payment, Clearly Stating How the Total Was Calculated .....	23
Double Appointment Prevention Explanation (Using "Dermatologist") .....	24
Concepts: Architectural and Design Patterns.....	25
Architectural Patterns .....	25
Model-View-Controller (MVC) Pattern.....	25
Code Example (Controller):.....	25
Design Patterns.....	26
Singleton Pattern .....	26

Code Example (Singleton): .....	26
Service and Repository Layers.....	27
Service Layer.....	27
Code Example (Service Layer):.....	27
Repository Layer .....	28
Code Example (Repository Layer):.....	28
Error Handling & Validation .....	29
Concurrency Handling.....	29
Conclusion .....	30
Code Link .....	31
References.....	32

## Table of Figures

Figure 1.....	8
Figure 2.....	12
Figure 3.....	13
Figure 4.....	14
Figure 5.....	15
Figure 6.....	16
Figure 7.....	17
Figure 8.....	18
Figure 9.....	19
Figure 10.....	20
Figure 11.....	21
Figure 12.....	22
Figure 13.....	23
Figure 14.....	24
Figure 15.....	25
Figure 17.....	26
Figure 19.....	27
Figure 20.....	28
Figure 21.....	29
Figure 22.....	29

## Class Diagram

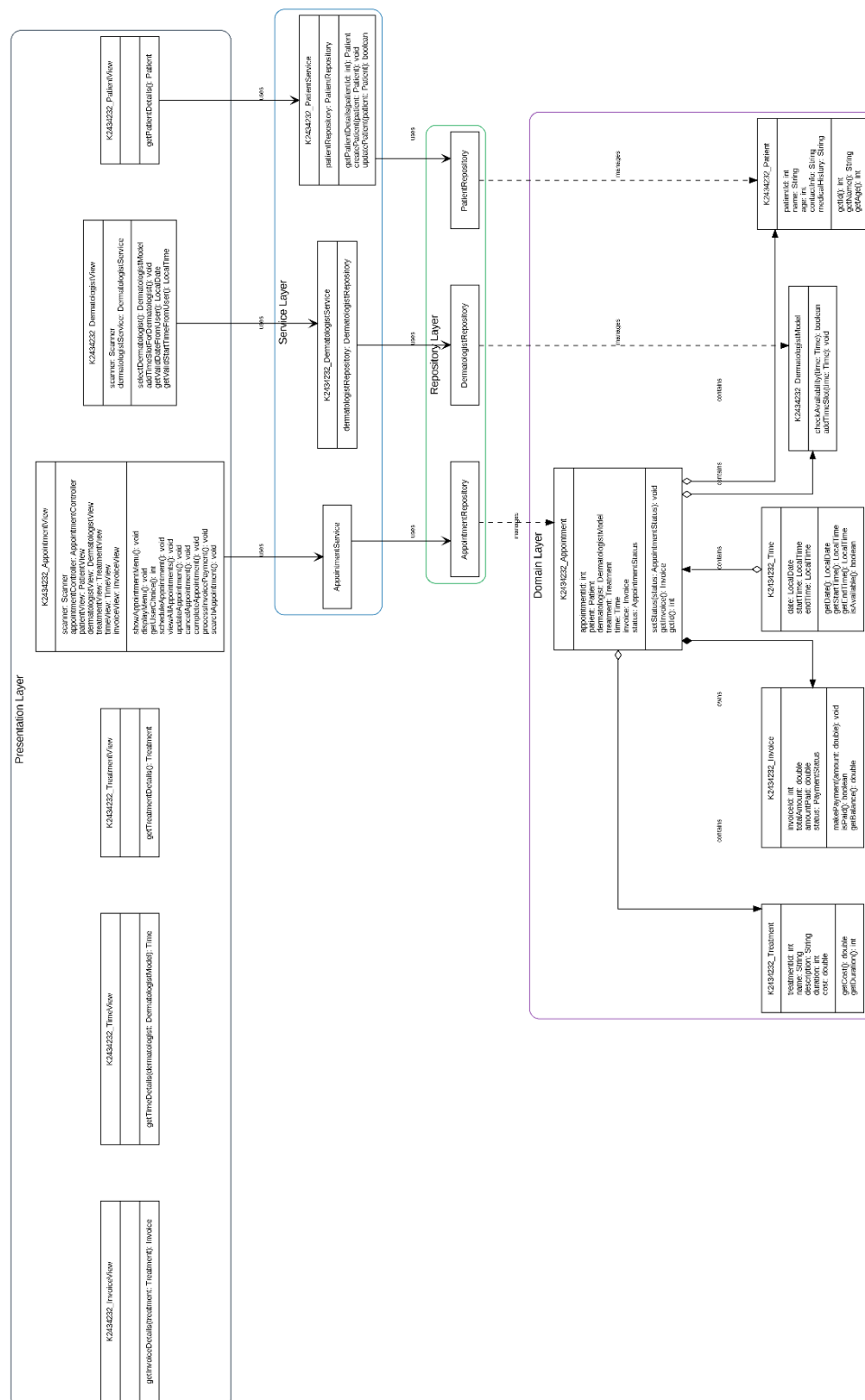


Figure 1

The UML Class Diagram represents the structure of a system designed for managing appointments in a healthcare environment. It consists of various components organized into four distinct layers: Presentation Layer, Service Layer, Repository Layer, and Domain Layer.



## Layers in the Diagram

### Presentation Layer

- This layer contains the classes responsible for displaying the user interface and interacting with the user. The primary view classes include:
  - **AppointmentView**: Manages the appointment scheduling and viewing process.
  - **PatientView**: Displays details about patients.
  - **DermatologistView**: Handles interactions related to dermatologists, such as selecting a dermatologist and adding time slots.
  - **TreatmentView**: Manages the display and retrieval of treatment details.
  - **TimeView**: Displays available time slots for a dermatologist.
  - **InvoiceView**: Manages the display of invoice details for treatments.

### Service Layer

- This layer defines the business logic and service operations that process data between the Presentation and Repository Layers. Key service classes include:
  - **AppointmentService**: Handles the creation, updating, and cancellation of appointments.
  - **DermatologistService**: Manages operations related to dermatologists, including time slot availability and selecting dermatologists.
  - **PatientService**: Manages patient data, including creating and updating patient records.

### Repository Layer

- This layer is responsible for interacting with the database and providing persistent storage for entities. The repository classes include:
  - **AppointmentRepository**: Manages appointment data and provides methods for saving, finding, updating, and deleting appointments.
  - **DermatologistRepository**: Handles the storage of dermatologist records and their time slots.
  - **PatientRepository**: Manages the persistent storage of patient records.

### Domain Layer

- The domain layer contains the core business entities or models that represent real-world concepts. This includes:
  - **Appointment**: Represents an appointment with properties like patient, dermatologist, treatment, time, and invoice.
  - **Patient**: Represents a patient, including their name, age, and medical history.
  - **DermatologistModel**: Represents a dermatologist, including their specialization and available time slots.
  - **Treatment**: Represents a treatment with details like name, description, duration, and cost.
  - **Invoice**: Represents an invoice for a treatment with payment details.
  - **Time**: Represents the time details for an appointment, including date, start, and end times.

## Relationships in the Diagram

- **View to Service Layer:** The view classes interact with the service layer to request data and perform actions like scheduling and viewing appointments.
- **Service to Repository Layer:** The service layer communicates with the repository layer to fetch or store data in the database.
- **Domain Layer Relationships:** The domain models (like Appointment, Patient, Dermatologist, etc.) contain references to each other, depicting the relationships such as "contains" (e.g., an appointment contains a patient, dermatologist, treatment, and time).
- **Repository to Domain Layer:** The repositories manage the domain models, providing methods to save, retrieve, update, and delete records in the persistent storage.

## Overall Structure of the Diagram

- The diagram emphasizes the separation of concerns, with each layer handling specific responsibilities (presentation, service, repository, and domain).
- Relationships between layers are established through **uses** and **manages** relationships, while associations between domain models are depicted using **contains** or **owns** relationships.
- The diagram helps visualize the flow of information and operations across the system, ensuring a well-structured design for managing appointments in a healthcare setting.

## Test Cases

### Test Plan

The test plan is designed to validate the core functionalities of the **Clinic Appointment Manager**. The system focuses on managing **appointments, invoices, dermatologists, and patients**. This plan will initially involve **manual user testing**, and in the future, automated testing will be implemented using tools like **Maven**.

### Test Scope

#### Core Components Under Test

1. **Appointment Management System**
  - Appointment creation and scheduling
  - Time slot validation
  - Double-booking prevention
  - Appointment updates and cancellations
  - Search functionality
2. **Financial Processing**
  - Registration fee handling
  - Treatment cost calculations
  - Invoice generation
  - Tax calculations
3. **User Management**
  - Patient record management
  - Dermatologist scheduling
  - Time slot availability

### Testing Approaches

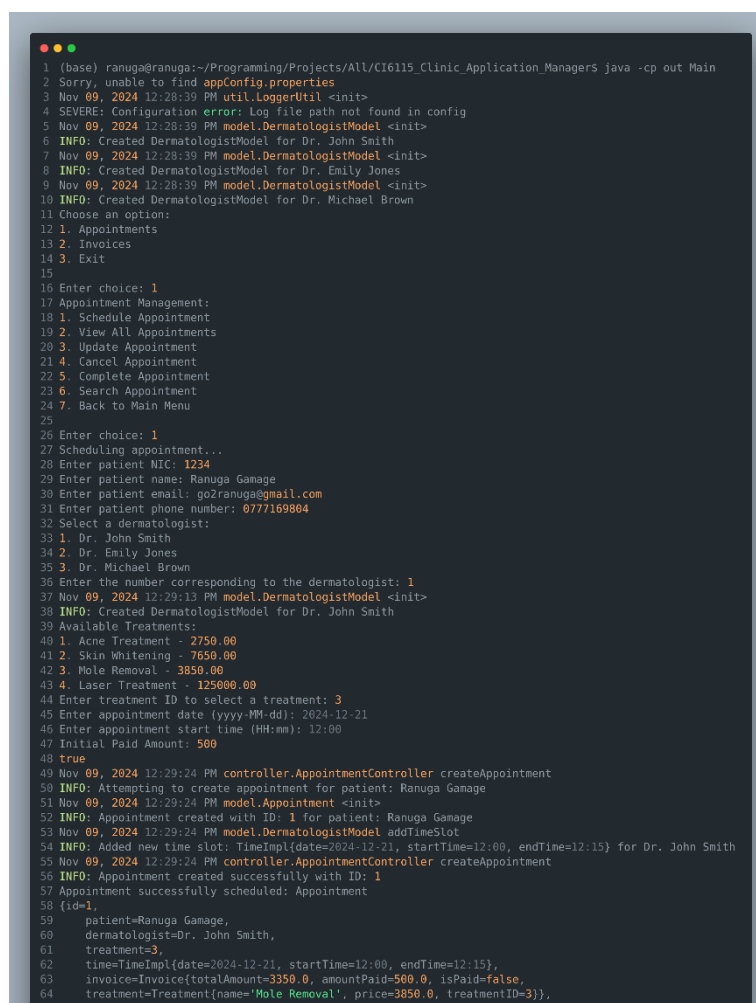
#### *Manual Testing Phase*

- **User Interface Testing:** Verification of all UI components and workflows
- **Integration Testing:** Ensuring different components work together correctly
- **Data Validation:** Verifying input handling and data consistency
- **Error Handling:** Checking system response to invalid inputs and edge cases

## Test Cases

### Test Case 1: Appointment Scheduling

- **Description:** Ensure the system can successfully schedule an appointment for a patient.
- **Steps:**
  1. From the main menu, select **1. Appointments**.
  2. Choose **1. Schedule Appointment** from the Appointment Management menu.
  3. Select a **Dermatologist** (e.g., Dr. John Smith).
  4. Choose a **Treatment Type** (e.g., Acne Treatment).
  5. Set a valid **appointment time**.
- **Expected Output:** A confirmation message with appointment details.
- **Validation:** The appointment is correctly scheduled and added to the system database.

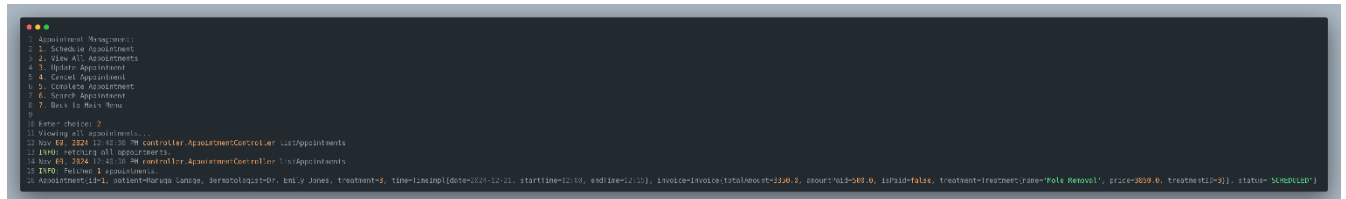


```
1 (base) ranuga@ranuga:~/Programming/Projects/All/CI6115_Clinic_Application_Managers java -cp out Main
2 Sorry, unable to find appConfig.properties
3 Nov 09, 2024 12:28:39 PM util.LoggerUtil <init>
4 SEVERE: Configuration error: Log file path not found in config
5 Nov 09, 2024 12:28:39 PM model.DermatologistModel <init>
6 INFO: Created DermatologistModel for Dr. John Smith
7 Nov 09, 2024 12:28:39 PM model.DermatologistModel <init>
8 INFO: Created DermatologistModel for Dr. Emily Jones
9 Nov 09, 2024 12:28:39 PM model.DermatologistModel <init>
10 INFO: Created DermatologistModel for Dr. Michael Brown
11 Choose an option:
12 1. Appointments
13 2. Invoices
14 3. Exit
15
16 Enter choice: 1
17 Appointment Management:
18 1. Schedule Appointment
19 2. View All Appointments
20 3. Update Appointment
21 4. Cancel Appointment
22 5. Complete Appointment
23 6. Search Appointment
24 7. Back to Main Menu
25
26 Enter choice: 1
27 Scheduling appointment...
28 Enter patient NIC: 1234
29 Enter patient name: Ranuga Ganage
30 Enter patient email: go2ranuga@gmail.com
31 Enter patient phone number: 0777169804
32 Select a dermatologist:
33 1. Dr. John Smith
34 2. Dr. Emily Jones
35 3. Dr. Michael Brown
36 Enter the number corresponding to the dermatologist: 1
37 Nov 09, 2024 12:29:13 PM model.DermatologistModel <init>
38 INFO: Created DermatologistModel for Dr. John Smith
39 Available Treatments:
40 1. Acne Treatment - 2750.00
41 2. Skin Whitening - 7650.00
42 3. Mole Removal - 3850.00
43 4. Laser Treatment - 125000.00
44 Enter treatment ID to select a treatment: 3
45 Enter appointment date (yyyy-MM-dd): 2024-12-21
46 Enter appointment start time (HH:mm): 12:00
47 Initial Paid Amount: 500
48 true
49 Nov 09, 2024 12:29:24 PM controller.AppointmentController createAppointment
50 INFO: Attempting to create appointment for patient: Ranuga Ganage
51 Nov 09, 2024 12:29:24 PM model.Appointment <init>
52 INFO: Appointment created with ID: 1 for patient: Ranuga Ganage
53 Nov 09, 2024 12:29:24 PM model.DermatologistModel addTimeSlot
54 INFO: Added new time slot: TimeImpl{date=2024-12-21, startTime=12:00, endTime=12:15} for Dr. John Smith
55 Nov 09, 2024 12:29:24 PM controller.AppointmentController createAppointment
56 INFO: Appointment created successfully with ID: 1
57 Appointment successfully scheduled: Appointment
58 {id=1,
59   patient=Ranuga Ganage,
60   dermatologist=Dr. John Smith,
61   treatment=3,
62   time=TimeImpl{date=2024-12-21, startTime=12:00, endTime=12:15},
63   invoice=Invoice{totalAmount=3350.0, amountPaid=500.0, isPaid=false,
64   treatment=Treatment{name='Mole Removal', price=3850.0, treatmentID=3}},
65   timeSlot=TimeSlot{date=2024-12-21, startTime=12:00, endTime=12:15}}
```

Figure 2

## Test Case 2: View All Appointments

- **Description:** Ensure the system displays all existing appointments correctly.
- **Steps:**
  1. From the main menu, select **1. Appointments**.
  2. Choose **2. View All Appointments**.
- **Expected Output:** A list of all appointments with details such as patient, dermatologist, treatment, and time.
- **Validation:** All existing appointments are displayed accurately in the system.



```
1 Appointment Management
2 1. Schedule Appointment
3 2. View All Appointments
4 3. Update Appointment
5 4. Cancel Appointment
6 5. Consistent Appointment
7 6. Search Appointment
8 7. Back to Main Menu
9
10 Enter choice: 2
11 Viewing all appointments:
12 Nov 19, 2024 12:10:38 PM controller.AppointmentController: ListAppointments
13 INFO: Fetching all appointments
14 Nov 19, 2024 12:10:38 PM controller.AppointmentController: ListAppointments
15 INFO: Fetching 1 appointments
16 Appointment[id=1, patient=Marwan Camgo, dermatologist=Dr. Emily Jones, treatment=9, time=TimeSlot(date=2024-12-22, startTime=10, endTime=12), invoiceInvoice.totalAmount=3300.0, amountPaid=0.0, isPaid=false, treatment=treatmentName=Folk Renewal, price=3000.0, treatmentID=9], status= SCHEDULED]
```

Figure 3

### Test Case 3: Update Appointment

- **Description:** Ensure that existing appointments can be updated (e.g., changing the time or treatment).
- **Steps:**
  1. From the main menu, select **1. Appointments**.
  2. Choose **3. Update Appointment**.
  3. Select an existing **appointment** to update.
  4. Modify the **appointment time** or **treatment**.
- **Expected Output:** A confirmation message that the appointment has been updated successfully.
- **Validation:** The system should update the appointment in the database with the new details.

```

1 Appointment Management
2 1. Create Appointment
3 2. View All Appointments
4 3. Update Appointment
5 4. Delete Appointment
6 5. Search Appointment
7 6. Add to Book Myra
8
9
10 Page 1/1
11
12 Loading an appointment...
13
14 Success! All appointments
15
16 Error: data is not in the controller AppointmentController: Appointment
17
18 Error: data is not in the controller AppointmentController: Appointment
19
20 Error: Invalid appointment
21
22 Error: The ID of the appointment you want to update does not exist
23
24 Error: data is not in the controller AppointmentController: Appointment
25
26 Error: data is not in the controller AppointmentController: Appointment
27
28 Error: data is not in the controller AppointmentController: Appointment
29
30 Error: data is not in the controller AppointmentController: Appointment
31
32 Error: data is not in the controller AppointmentController: Appointment
33
34 Error: data is not in the controller AppointmentController: Appointment
35
36 Error: data is not in the controller AppointmentController: Appointment
37
38 Error: data is not in the controller AppointmentController: Appointment
39
40 Error: data is not in the controller AppointmentController: Appointment
41
42 Error: data is not in the controller AppointmentController: Appointment
43
44 Error: data is not in the controller AppointmentController: Appointment
45
46 Error: data is not in the controller AppointmentController: Appointment
47
48 Error: data is not in the controller AppointmentController: Appointment
49
50 Error: data is not in the controller AppointmentController: Appointment
51
52 Error: data is not in the controller AppointmentController: Appointment
53
54 Error: data is not in the controller AppointmentController: Appointment
55
56 Error: data is not in the controller AppointmentController: Appointment
57
58 Error: data is not in the controller AppointmentController: Appointment
59
60 Error: data is not in the controller AppointmentController: Appointment
61
62 Error: data is not in the controller AppointmentController: Appointment
63
64 Error: data is not in the controller AppointmentController: Appointment
65
66 Error: data is not in the controller AppointmentController: Appointment
67
68 Error: data is not in the controller AppointmentController: Appointment
69
70 Error: data is not in the controller AppointmentController: Appointment
71
72 Error: data is not in the controller AppointmentController: Appointment
73
74 Error: data is not in the controller AppointmentController: Appointment
75
76 Error: data is not in the controller AppointmentController: Appointment
77
78 Error: data is not in the controller AppointmentController: Appointment
79
80 Error: data is not in the controller AppointmentController: Appointment
81
82 Error: data is not in the controller AppointmentController: Appointment
83
84 Error: data is not in the controller AppointmentController: Appointment
85
86 Error: data is not in the controller AppointmentController: Appointment
87
88 Error: data is not in the controller AppointmentController: Appointment
89
90 Error: data is not in the controller AppointmentController: Appointment
91
92 Error: data is not in the controller AppointmentController: Appointment
93
94 Error: data is not in the controller AppointmentController: Appointment
95
96 Error: data is not in the controller AppointmentController: Appointment
97
98 Error: data is not in the controller AppointmentController: Appointment
99
100 Error: data is not in the controller AppointmentController: Appointment

```

Figure 4

## Test Case 4: Cancel Appointment

- **Description:** Ensure that appointments can be canceled successfully.
- **Steps:**
  1. From the main menu, select **1. Appointments**.
  2. Choose **4. Cancel Appointment**.
  3. Select an existing **appointment** to cancel.
- **Expected Output:** A confirmation message indicating the appointment has been canceled.
- **Validation:** The canceled appointment is removed from the system and no longer appears in the list of active appointments.

[illegible]

Figure 5

## Test Case 5: Complete Appointment

- **Description:** Ensure that appointments can be marked as completed once the consultation is finished.
- **Steps:**
  1. From the main menu, select **1. Appointments**.
  2. Choose **5. Complete Appointment**.
  3. Select an existing **appointment** that has been completed.
- **Expected Output:** A confirmation message indicating the appointment status has been updated to "Completed."
- **Validation:** The system should reflect the updated appointment status as "Completed," and both the patient and dermatologist should be notified of the completion.



```
1 Appointment completed.
2 4. Complete appointment
3 2. View all appointments
4 3. Update Appointment
5 4. Create Appointment
6 5. Complete appointment
7 6. Search Appointment
8 7. Back to Main Menu
9
10 Enter choice: 5
11
12 Complete an appointment...
13 Get all appointments.
14 No appointments found. Try again.
15 Back to Main Menu
16
17 Enter choice: 5
18
19 Complete an appointment...
20 Get all appointments.
21 Appointment found. Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
22 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
23 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
24 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
25 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
26 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
27 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
28 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
29 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
30 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
31 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
32 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
33 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
34 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
35 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
36 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
37 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
38 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
39 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
40 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
41 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
42 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
43 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
44 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
45 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
46 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
47 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
48 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
49 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
50 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
51 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
52 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
53 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
54 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
55 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
56 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
57 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
58 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
59 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
60 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
61 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
62 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
63 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
64 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
65 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
66 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
67 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
68 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
69 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
70 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
71 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
72 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
73 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
74 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
75 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
76 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
77 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
78 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
79 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
80 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
81 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
82 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
83 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
84 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
85 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
86 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
87 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
88 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
89 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
90 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
91 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
92 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
93 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
94 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
95 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
96 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
97 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
98 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
99 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
100 Appointment ID: 1. Patient ID: 1. Doctor ID: 1. Status: 1.
```

Figure 6

## Future Testing Plans

In the future, we plan to automate the testing process using **Maven** or similar tools to run tests continuously. This will allow us to ensure the stability of the system after every update. For now, the testing is done manually by performing user interactions to confirm that all features are functioning as expected.

Additionally, we aim to integrate a broader range of test cases covering areas like system security (authentication/authorization), edge cases, and error handling. These will be expanded and automated over time to improve test coverage and reliability.

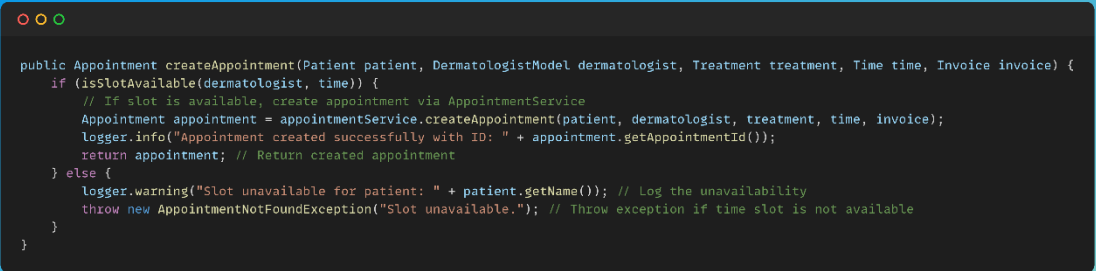


## Implementation

### Make Appointments

This section outlines the process of creating an appointment. It involves selecting a patient, a dermatologist, a treatment, and an available time slot. The key here is verifying if the time slot is available for the dermatologist before proceeding with appointment creation.

#### Code Explanation:



```
public Appointment createAppointment(Patient patient, DermatologistModel dermatologist, Treatment treatment, Time time, Invoice invoice) {
    if (isSlotAvailable(dermatologist, time)) {
        // If slot is available, create appointment via AppointmentService
        Appointment appointment = appointmentService.createAppointment(patient, dermatologist, treatment, time, invoice);
        logger.info("Appointment created successfully with ID: " + appointment.getAppointmentId());
        return appointment; // Return created appointment
    } else {
        logger.warning("Slot unavailable for patient: " + patient.getName()); // Log the unavailability
        throw new AppointmentNotFoundException("Slot unavailable."); // Throw exception if time slot is not available
    }
}
```

codesnap.dev

Figure 7

#### Expanded Explanation:

- The system first checks if the selected time is available using the isSlotAvailable function.
- If the time is available, the system proceeds to create the appointment using appointmentService.createAppointment.
- A logger tracks the process, providing transparency for any action (appointment creation in this case).
- If the time slot is unavailable, an exception is thrown, and the user is informed with a log and a custom exception (AppointmentNotFoundException).

## Update Appointment Details

This feature allows users to modify existing appointments. For example, they might want to change the time or the dermatologist. Before making any updates, the system ensures that the new time slot is available.

### Code Explanation:



```
public boolean updateAppointment(int appointmentId, Patient patient, DermatologistModel dermatologist, Treatment treatment, Time time) {
    Appointment appointment = getAppointmentById(appointmentId); // Retrieve the appointment by ID
    if (isSlotAvailable(dermatologist, time)) { // Check if new time is available
        boolean updated = appointmentService.updateAppointment(appointmentId, patient, dermatologist, treatment, time);
        if (updated) {
            logger.info("Successfully updated appointment with ID: " + appointmentId); // Log success
        } else {
            logger.warning("Failed to update appointment with ID: " + appointmentId); // Log failure
        }
        return updated; // Return update status
    } else {
        logger.warning("Cannot update, slot unavailable."); // Log error for slot unavailability
        throw new InvalidAppointmentTimeException("Slot unavailable."); // Throw exception if time slot is unavailable
    }
}
```

codesnap.dev

Figure 8

### Expanded Explanation:

- The updateAppointment method retrieves an appointment using getAppointmentById.
- It verifies whether the new time slot is available through isSlotAvailable.
- If the slot is available, it proceeds to update the appointment and logs success. If the update fails, it logs an error.
- In case of an unavailable slot, an exception (InvalidAppointmentTimeException) is thrown.

## View Appointment Details Filtered by Date

The ability to filter appointments by date provides a more organized way for users to manage appointments. This is particularly useful for clinic administrators to track daily or weekly appointments.

### Code Explanation:

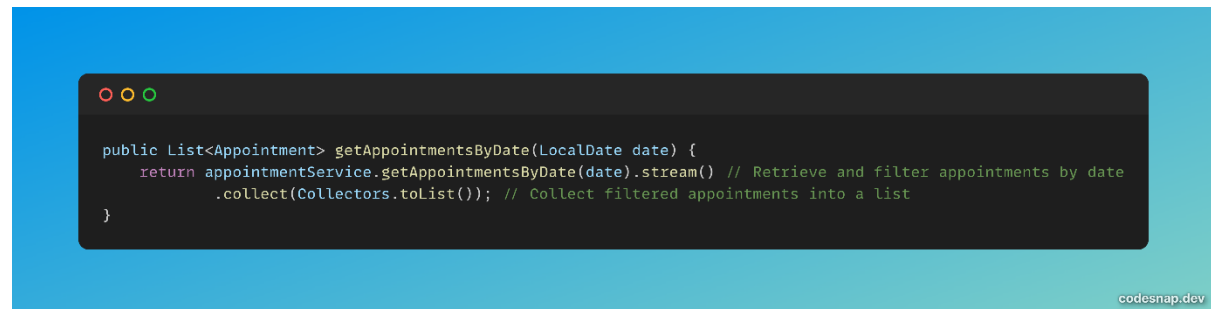


Figure 9

### Expanded Explanation:

- This method uses `appointmentService.getAppointmentsByDate(date)` to fetch all appointments on a specific date.
- It then filters the results using Java Streams (`.stream()`), making it easier to handle large lists of appointments efficiently.
- The results are collected into a `List` that is returned to the user.

## Search for an Appointment Using Either the Patient's Name or the Appointment ID

This feature improves user experience by allowing a flexible search for appointments, making it easier to locate specific ones, whether the user knows the patient's name or the appointment ID.

### Code Explanation:

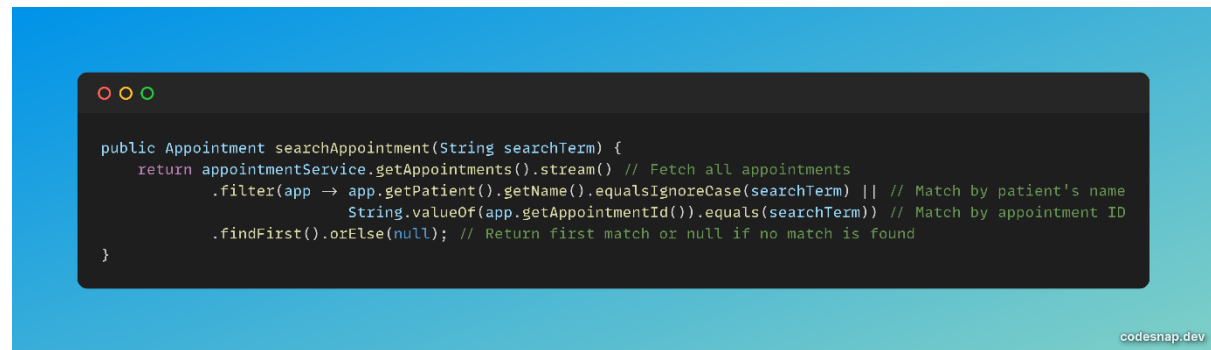


Figure 10

### Expanded Explanation:

- This search method allows users to search appointments using either the patient's name or the appointment ID.
- The method uses Java Streams to filter appointments based on the provided searchTerm.
  - The equalsIgnoreCase method compares patient names in a case-insensitive manner.
  - The String.valueOf(app.getAppointmentId()).equals(searchTerm) ensures the appointment ID is compared as a string.
- If a match is found, the first result is returned. If no match is found, it returns null.

## Accept Registration Fees When Placing Appointments

This feature ensures that a registration fee is applied when scheduling an appointment. It updates the patient's balance to reflect the fee deducted.

### Code Explanation:

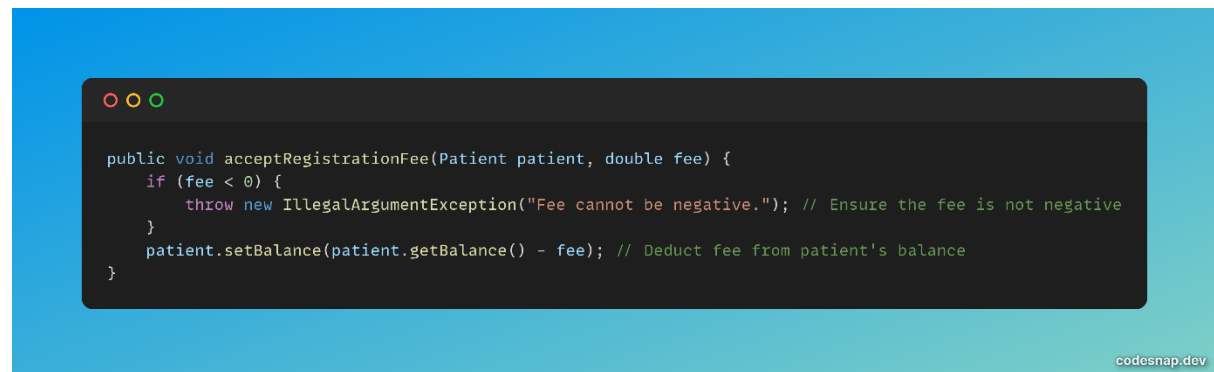


Figure 11

### Expanded Explanation:

- The acceptRegistrationFee method verifies that the fee is valid (i.e., not negative) before deducting it from the patient's balance.
- The setBalance method updates the patient's account, reducing the balance by the amount of the fee.

## Calculate the Total Fee and Taxes for Treatments After the Appointment

After the treatment is completed, the total fee is calculated, which includes the treatment cost and any applicable taxes.

### Code Explanation:

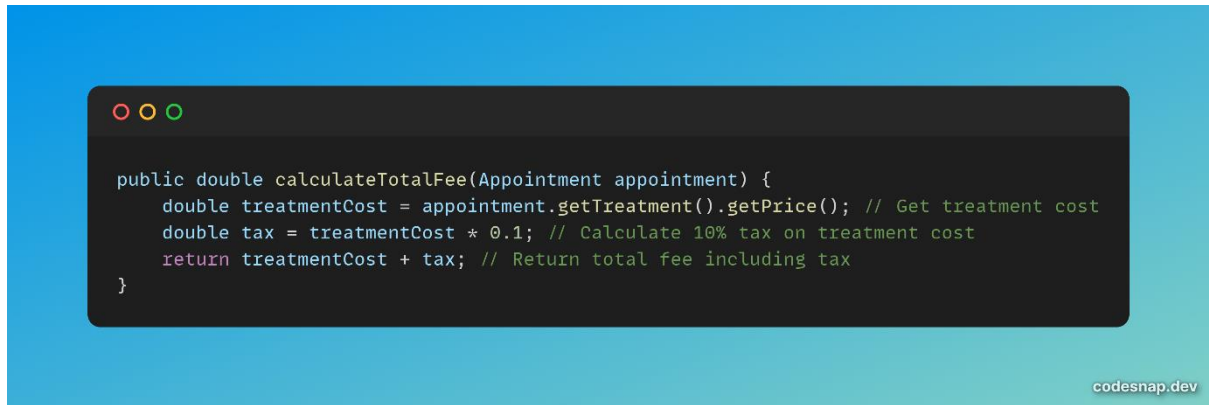


Figure 12

### Expanded Explanation:

- The calculateTotalFee method calculates the total fee by adding a 10% tax to the base treatment cost.
- This ensures the total fee reflects both the cost of the treatment and the tax, which is commonly required in healthcare billing.

## Generate an Invoice for the Payment, Clearly Stating How the Total Was Calculated

This feature generates an invoice, providing a breakdown of how the total payment is calculated, including treatment cost and taxes.

### Code Explanation:



Figure 13

### Expanded Explanation:

- The method uses the calculateTotalFee function to calculate the total amount due.
- The Invoice object is created, which includes the initial payment amount and details about the treatment provided.
- The invoice helps in generating a clear statement for both the patient and the dermatologist about the amount due, including taxes and treatments.

## Double Appointment Prevention Explanation (Using "Dermatologist")

This part ensures that no two patients can book an appointment with the same dermatologist at the same time. It checks for overlapping time slots before confirming an appointment.

### Code Explanation:

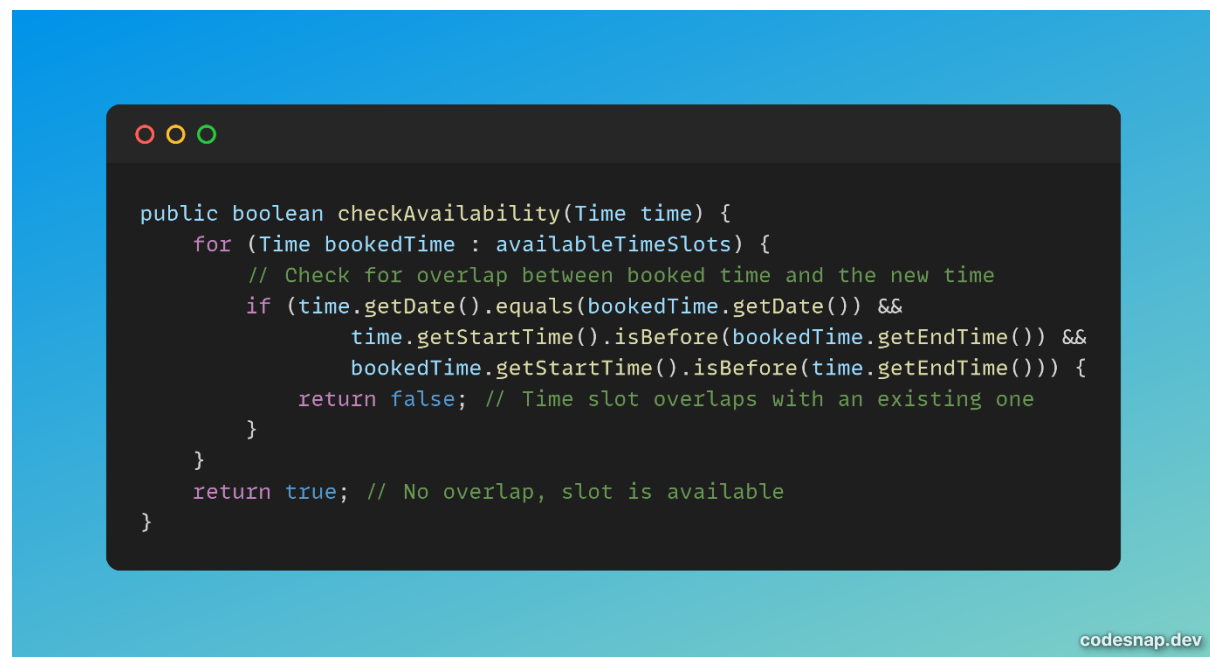


Figure 14

### Expanded Explanation:

- The checkAvailability function checks for time slot overlaps between an existing booking and the new request.
- The conditions check whether the appointment date matches and if the time intervals overlap:
  - If the new appointment's start time is before the existing appointment's end time.
  - If the existing appointment's start time is before the new appointment's end time.
- If any overlap is detected, it returns false, indicating the slot is not available.
- If no overlap is found, the method returns true, confirming the slot is available.

This ensures that there are no double bookings and that the system maintains the integrity of scheduling.



## Concepts: Architectural and Design Patterns

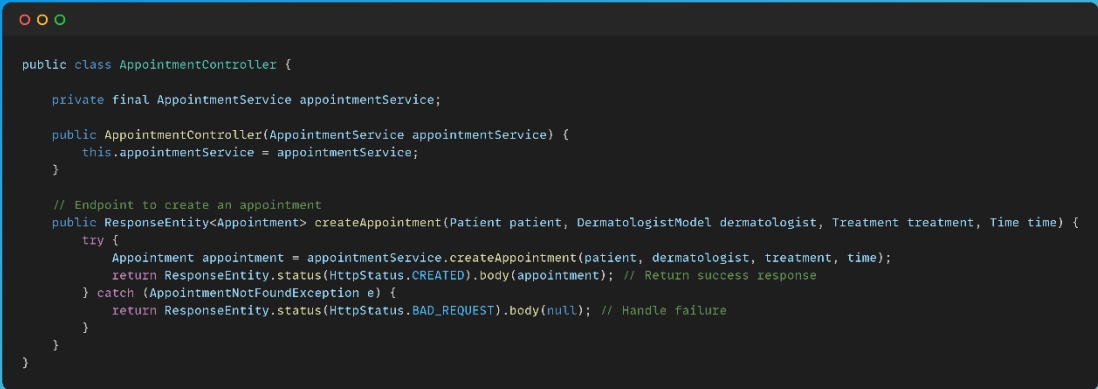
### Architectural Patterns

#### Model-View-Controller (MVC) Pattern

The **MVC** design pattern has been used to separate concerns in the application, ensuring a clear distinction between the data, the user interface, and the business logic. This structure helps in making the system more maintainable, modular, and scalable.

- **Model:** The model represents the core logic and data. For example, the Appointment, User, Invoice, and Treatment classes represent entities and the business logic that interacts with them. These models interact with the database through repositories.
- **View:** This is responsible for presenting data to the user, typically a UI. This part is outside the scope of the back-end code, but it would likely use technologies like React.js (as you mentioned) to display data retrieved from the server.
- **Controller:** The controller handles user inputs and actions (e.g., creating appointments, making payments). It interacts with services to process data and update the view accordingly.

#### Code Example (Controller):



```
public class AppointmentController {  
  
    private final AppointmentService appointmentService;  
  
    public AppointmentController(AppointmentService appointmentService) {  
        this.appointmentService = appointmentService;  
    }  
  
    // Endpoint to create an appointment  
    public ResponseEntity<Appointment> createAppointment(Patient patient, DermatologistModel dermatologist, Treatment treatment, Time time) {  
        try {  
            Appointment appointment = appointmentService.createAppointment(patient, dermatologist, treatment, time);  
            return ResponseEntity.status(HttpStatus.CREATED).body(appointment); // Return success response  
        } catch (AppointmentNotFoundException e) {  
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null); // Handle failure  
        }  
    }  
}
```

codesnap.dev

Figure 15

## Design Patterns

### Singleton Pattern

The **Singleton** pattern is applied to ensure that certain services (like AppointmentService or PaymentService) are instantiated only once and shared across the application. This ensures that the service remains consistent and does not waste resources by being instantiated multiple times.

Code Example (Singleton):

A code editor window with a dark background and light blue text. The window has three colored circles (red, yellow, green) in the top-left corner. The code is for a Java class named AppointmentService. It includes a private static instance, a private constructor, and a public static getInstance() method that uses a synchronized block to ensure only one instance is created.

```
public class AppointmentService {  
    private static AppointmentService instance;  
  
    private AppointmentService() {  
        // Private constructor to prevent instantiation  
    }  
  
    public static AppointmentService getInstance() {  
        if (instance == null) {  
            synchronized (AppointmentService.class) {  
                if (instance == null) {  
                    instance = new AppointmentService();  
                }  
            }  
        }  
        return instance;  
    }  
}
```

codesnap.dev


Figure 16

## Service and Repository Layers

### Service Layer

The **Service Layer** handles business logic, ensuring that the controllers remain thin and focused only on handling user input. The service layer coordinates interactions between the controller and repository. It's where validation, complex processing, and calculations happen.

#### Code Example (Service Layer):



```
public class AppointmentService {  
  
    private final AppointmentRepository appointmentRepository;  
    private final InvoiceService invoiceService;  
  
    public AppointmentService(AppointmentRepository appointmentRepository, InvoiceService invoiceService) {  
        this.appointmentRepository = appointmentRepository;  
        this.invoiceService = invoiceService;  
    }  
  
    public Appointment createAppointment(Patient patient, DermatologistModel dermatologist, Treatment treatment, Time time, Invoice invoice) {  
        if (!isSlotAvailable(dermatologist, time)) {  
            throw new AppointmentNotFoundException("Slot is unavailable.");  
        }  
        // Create and save appointment  
        Appointment appointment = new Appointment(patient, dermatologist, treatment, time, invoice);  
        appointmentRepository.save(appointment);  
  
        // Generate invoice for the patient  
        invoiceService.generateInvoice(appointment);  
        return appointment;  
    }  
  
    public boolean isSlotAvailable(DermatologistModel dermatologist, Time time) {  
        // Business logic for checking availability  
        return true;  
    }  
}
```

Figure 17

## Repository Layer

The **Repository Layer** abstracts the data access logic. It handles all CRUD operations and allows the service layer to interact with the data without worrying about the underlying database details.

Code Example (Repository Layer):

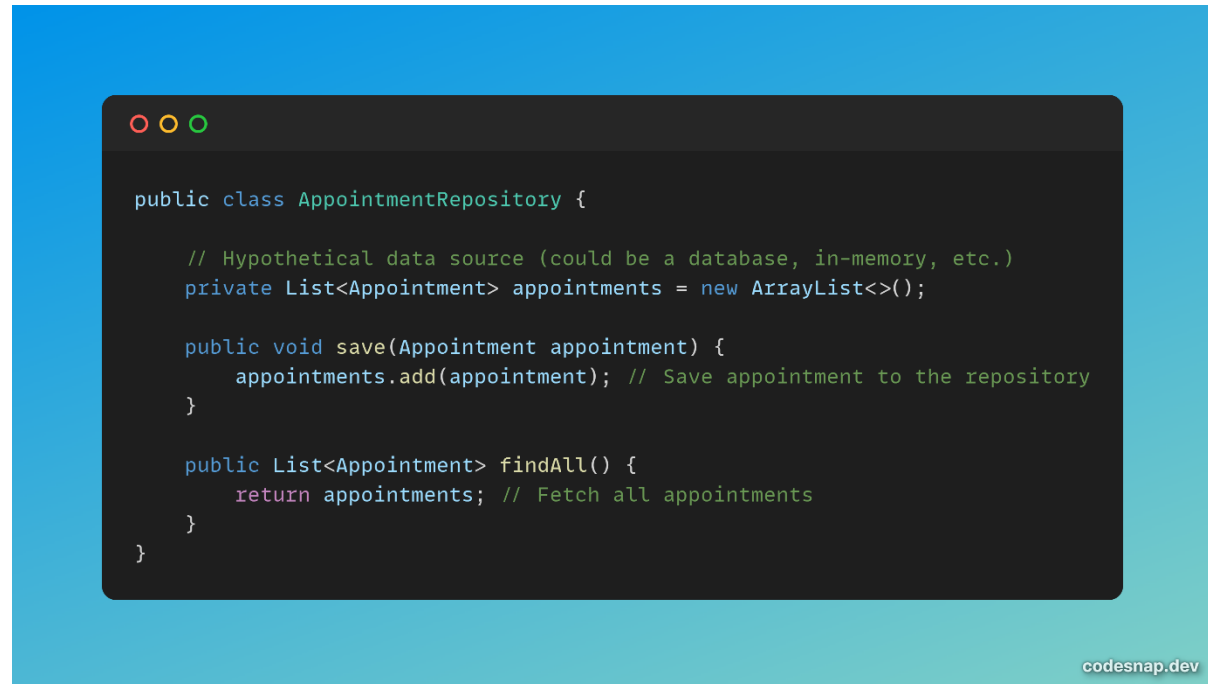


Figure 18

## Error Handling & Validation

- **Error Handling:** Custom exceptions are thrown for specific errors (e.g., `AppointmentNotFoundException`), allowing for meaningful error messages to be provided back to the user.

### Code Example:

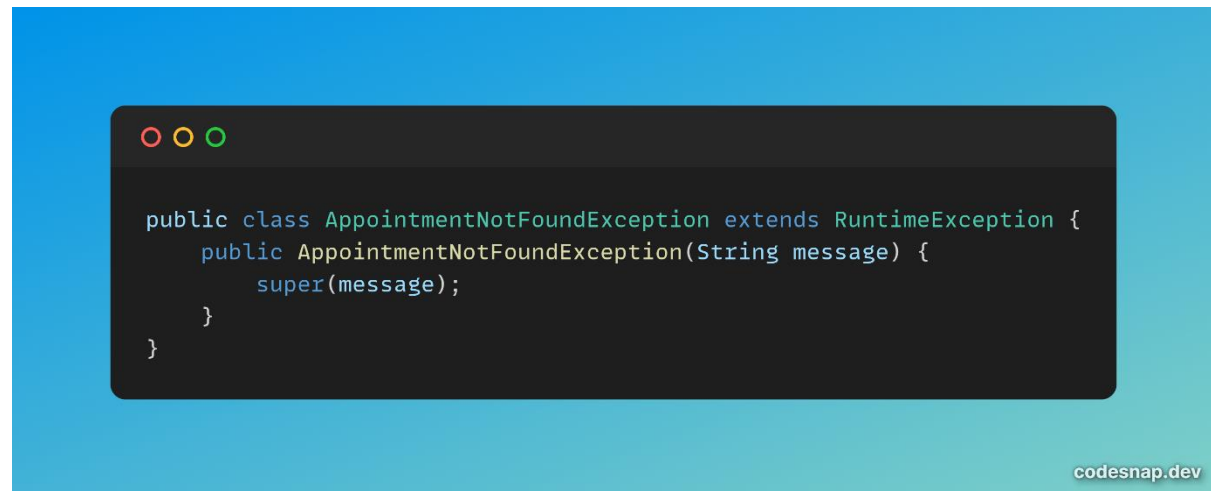


Figure 19

- **Validation:** Validation is performed at multiple levels (e.g., checking appointment availability and ensuring non-negative fees). This prevents invalid data from being saved or processed.

### Code Example:

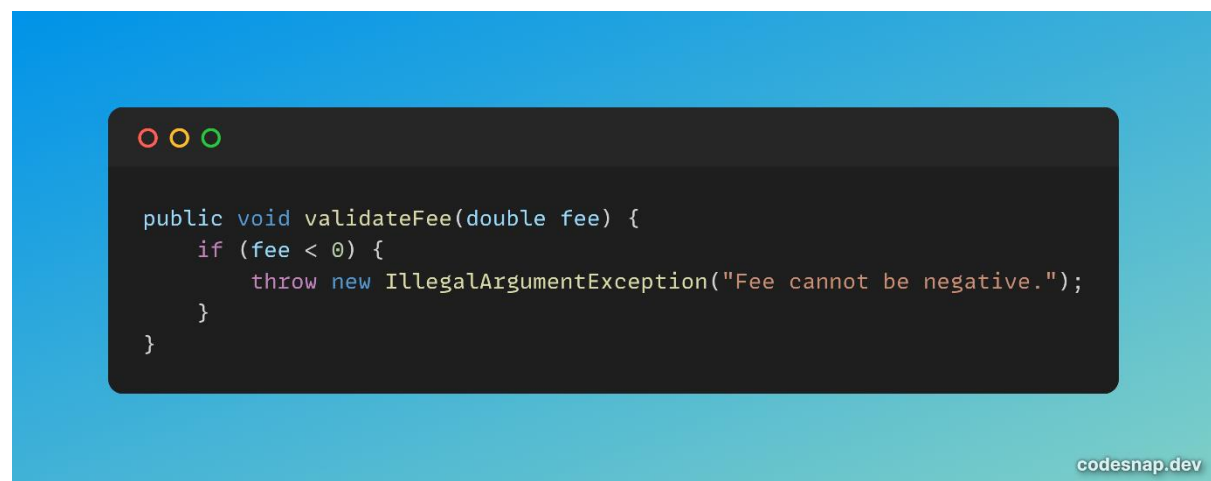


Figure 20

## Concurrency Handling

To handle multiple simultaneous requests (e.g., multiple users booking appointments at the same time), synchronization mechanisms like **Double-Checked Locking** are used to ensure that the system does not allow conflicting updates to the same appointment.

## Conclusion

The Aurora Skin Care Clinic Appointment Management System demonstrates the practical implementation of fundamental software engineering principles through a layered architectural approach. The system's core strength lies in its clean separation of concerns using the MVC pattern, with distinct presentation, service, repository, and domain layers. This architectural decision, combined with the implementation of key design patterns including Factory, Singleton, and Strategy patterns, resulted in a maintainable and well-structured codebase.

The technical implementation features comprehensive error handling through custom exceptions and thread-safe operations for concurrent appointment bookings. The service layer encapsulates complex business logic while the repository pattern provides clean data access abstraction. These architectural choices are supported by robust testing, with test cases covering core functionalities including appointment scheduling, time slot validation, and payment processing.

From a code organization perspective, the system maintains SOLID principles throughout its implementation. The Factory pattern creates appointments in a standardized way, the Singleton pattern ensures consistent service instances, and the Strategy pattern enables flexible appointment validation algorithms. This combination of design patterns and principles demonstrates practical application of software engineering concepts while maintaining clean code practices and ensuring system reliability through proper error handling and validation.

## Code Link

The source code can be accessed from the following link:

[https://github.com/Programmer-RD-AI/CI6115\\_Aurora\\_Skin\\_Care](https://github.com/Programmer-RD-AI/CI6115_Aurora_Skin_Care)

Higher resolution images can be accessed from the following link:

[https://drive.google.com/drive/folders/1WZb0vX5q9IQLC0B\\_zDV3BOZNs9jkZSzy?usp=sharing](https://drive.google.com/drive/folders/1WZb0vX5q9IQLC0B_zDV3BOZNs9jkZSzy?usp=sharing)

## References

GeeksforGeeks. “Appointment Scheduling System in Spring Boot.” *GeeksforGeeks*, 20 June 2024, [www.geeksforgeeks.org/appointment-scheduling-system-in-spring-boot/](http://www.geeksforgeeks.org/appointment-scheduling-system-in-spring-boot/). Accessed 9 Nov. 2024.

---. “Collections in Java - GeeksforGeeks.” *GeeksforGeeks*, 28 Apr. 2019, [www.geeksforgeeks.org/collections-in-java-2/](http://www.geeksforgeeks.org/collections-in-java-2/).

Newman, Andre. “Logging Exceptions in Java.” *Log Analysis / Log Monitoring by Loggly*, 13 Oct. 2015, [www.loggly.com/blog/logging-exceptions-in-java/](http://www.loggly.com/blog/logging-exceptions-in-java/). Accessed 9 Nov. 2024.

“Object Oriented Design Principles in Java.” *Stack Abuse*, 4 Nov. 2019, [stackabuse.com/object-oriented-design-principles-in-java/](http://stackabuse.com/object-oriented-design-principles-in-java/).

W3Schools. “Java Enums.” *W3schools.com*, 2019, [www.w3schools.com/java/java\\_enums.asp](http://www.w3schools.com/java/java_enums.asp).