University of Hertfordshire UH

BOOST PHYSIO CLINIC SYSTEM

TABLE OF CONTENTS

1. Introduction

The Boost Physio Clinic (BPC) Booking System is a comprehensive software solution designed to streamline the management of physiotherapy services, scheduling, and patient interactions. This system facilitates efficient appointment booking, patient management, and reporting for physiotherapists and administration staff.

The primary goal of the BPC Booking System is to create a centralized platform where patients can easily search for and book appropriate treatments based on their needs, while physiotherapists can manage their schedules and treatment offerings. The system maintains detailed records of all clinic members (both patients and physiotherapists), treatments offered, expertise areas, and appointments.

The system is structured around several core components:

- **Member Management**: Storing and managing information about patients and physiotherapists
- **Treatment Catalog**: Organizing treatments by expertise areas and physiotherapist availability
- **Appointment Scheduling**: Enabling booking, cancellation, and modification of appointments
- **Attendance Tracking**: Recording when patients attend their scheduled appointments
- **Reporting**: Generating performance reports for clinic management

With intuitive interfaces for both patients and staff, the system allows patients to find treatments either by searching for a specific expertise area or by looking up a particular physiotherapist. The system also provides comprehensive reporting capabilities to help clinic management evaluate service delivery and physiotherapist performance over time.

2. System Assumptions

To establish the boundaries and operational context of the BPC Booking System, the following assumptions have been made:

1. **Schedule Management**:
    1. The system operates on a 4-week term basis, with a new term beginning each month
    2. Each physiotherapist has their own schedule with predefined availability blocks
    3. Treatment sessions have standard durations (typically 1 hour, though this could be configurable)
    4. Appointments cannot overlap for either patients or physiotherapists

2. **Physiotherapist Capabilities**:

    1. Each physiotherapist has at least one area of expertise and can offer multiple treatments within those areas
    2. Physiotherapists can add or remove expertise areas with proper authorization
    3. The system does not handle physiotherapist leave/absence management beyond blocking out unavailable times

3. **Patient Interactions**:

    1. Patients must be registered in the system before booking appointments
    2. Patients can view, book, cancel, or reschedule their own appointments
    3. There are no restrictions on how far in advance appointments can be booked or canceled
    4. No payment processing is handled within the system (assumed to be handled separately)

4. **Appointment Status**:

    1. Appointments have three possible states: booked, canceled, or attended
    2. Only staff can mark appointments as "attended" after patient check-in
    3. Canceled appointments free up the time slot for other bookings
    4. No waitlist functionality is implemented for popular time slots

5. **Data Management**:

    1. All member data (name, address, telephone) is assumed to be valid upon entry
    2. The system assigns unique IDs to all members, treatments, and appointments
    3. For the initial implementation, test data can be hardcoded or loaded from text files

6. **Reporting**:

    1. Reports are generated at the end of each 4-week term
    2. Reports include detailed appointment information and physiotherapist rankings based on attendance
    3. No real-time analytics or daily reporting is required in the initial implementation

7. **Technical Considerations**:

    1. The system is implemented as a standalone application (not web-based)
    2. No authentication or user role management is implemented in this version
    3. Data persistence is simplified (file-based storage rather than a database)
    4. No concurrent access issues are addressed in the initial implementation

3. System Design and Structure

1. System Overview

The BPC Booking System is a standalone desktop application designed to manage physiotherapy appointments, patient records, and treatment scheduling. It provides:

- Patient registration & management
- Physiotherapist schedule & expertise management
- Appointment booking, cancellation, and attendance tracking
- Term-based reporting (4-week cycles)

2. System Architecture

The system follows a modular layered architecture:

2.1. Presentation Layer (UI)

Patient Interface:

- Search treatments by expertise or physiotherapist
- Book/cancel/reschedule appointments

Admin/Staff Interface:

- Manage physiotherapist schedules
- Mark appointments as "attended"
- Generate reports

2.2. Business Logic Layer (Core Functionality)

- Member Management (Patients & Physiotherapists)
- Treatment & Expertise Management
- Appointment Scheduling & Conflict Handling
- Reporting Engine (Term-based summaries & rankings)

2.3. Data Layer (Storage)

Stores:

- Patient & physiotherapist details
- Treatment catalog
- Appointment records

3. Key Components

Member Manager	Handles patient & physiotherapist CRUD operations
Treatment Catalog	Manages expertise areas & available treatments
Appointment Scheduler	Books, cancels, and tracks appointment statuses
Report Generator	Produces term-end reports with attendance rankings

4. Data Model (Simplified)

- Patient: ID, Name, Address, Phone
- Physiotherapist: ID, Name, Address, Phone, Expertise[]
- Treatment: ID, Name, Duration, Expertise Area
- Appointment: ID, PatientID, PhysioID, TreatmentID, DateTime, Status (Booked/Cancelled/Attended)

5. Technical Considerations

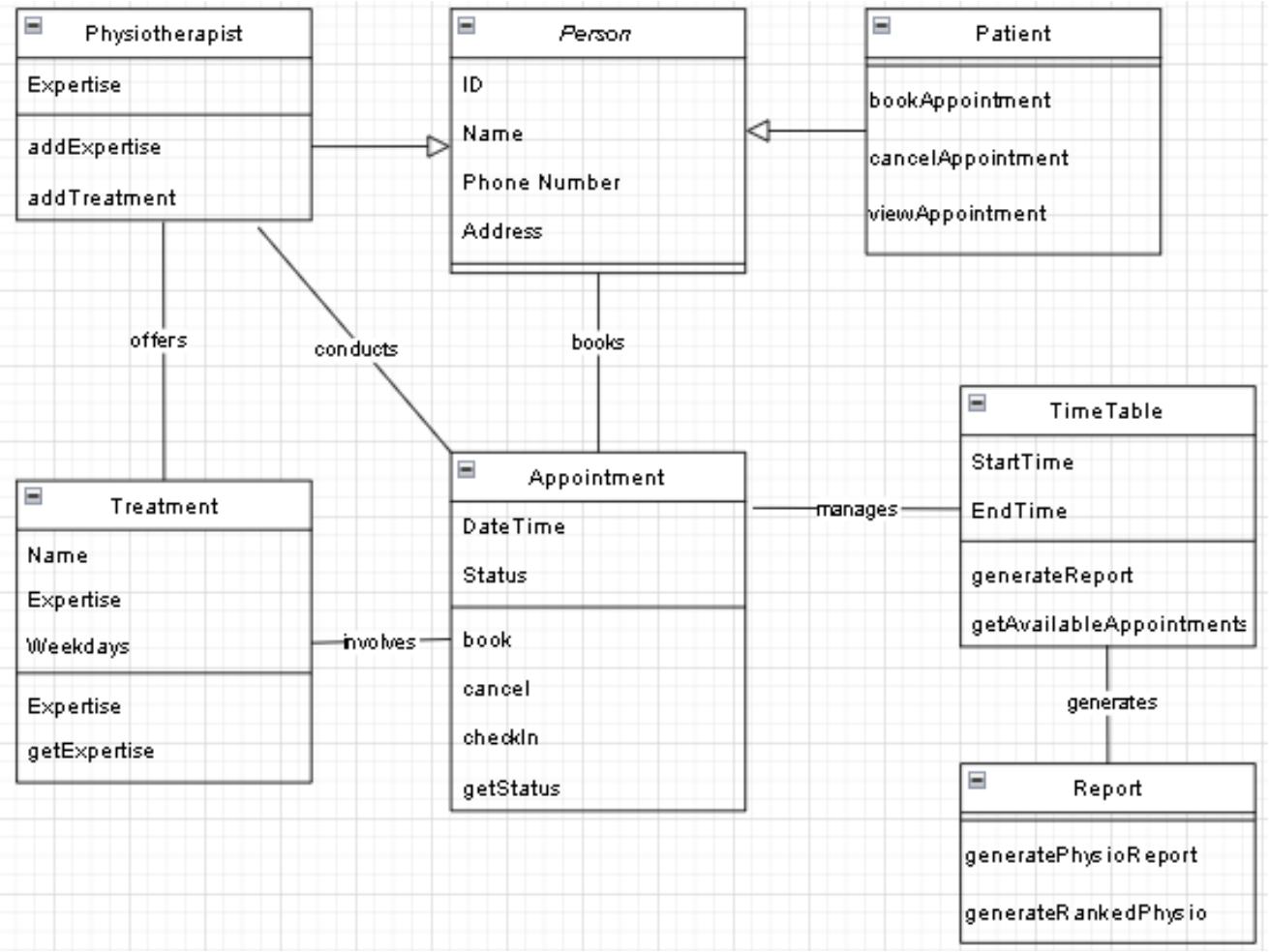
- Standalone application (No web/cloud)
- No authentication (Basic user roles: Patient/Staff
- No real-time concurrency handling (Single-user assumption)
- Pre-loaded test data (3-5 physiotherapists, 10-15 patients, 4-week schedule)

6. Assumptions & Limitations

- No payment integration
- No waitlist for fully booked slots
- No automatic term rollover (Manual schedule reset)

This design ensures a structured, maintainable, and scalable system for BPC's booking needs.

· 3.2 UML Class Diagram



3.3 System Architecture

The system follows an **object-oriented approach** with three primary layers:

1. **Model Layer**: Defines the core data structures (Physiotherapist, Patient, Appointment).
2. **Service Layer**: Implements business logic (BookingService).
3. **Presentation Layer**: Console-based user interface handling user interactions.

4. Implementation Details

The implementation of the BPC Booking System follows an object-oriented approach with the following key components:

1. Core Classes

Person Class

- Abstract base class for both patients and physiotherapists
- Encapsulates common attributes: ID, name, address, telephone
- Provides validation methods for contact information

Physiotherapist Class

- Extends Person class
- Maintains list of expertise areas
- Tracks treatments offered and their availability
- Methods to add/remove expertise and manage treatment offerings

Patient Class

- Extends Person class
- Methods to search for, book, cancel and reschedule appointments
- Provides appointment history for the patient

Treatment Class

- Represents a specific treatment modality
- Associates treatments with expertise areas
- Contains descriptive information about the treatment

Appointment Class

- Represents a booked time slot
- Links patient, physiotherapist, treatment, and time slot
- Maintains status (booked/canceled/attended)
- Methods to change status and retrieve appointment details

Timetable Class

- Manages the 4-week schedule grid
- Handles time slot availability
- Provides search functionality for available appointments

- Methods to find appointments by expertise or physiotherapist

Report Class

- Generates end-of-term reports
- Creates ranked lists of physiotherapists
- Outputs appointment statistics

2. Core Functionalities

- Patient Registration
- Appointment Booking
- Report Generation

5. JUnit Testing

5.1 Test Cases

- **Test case for adding a patient.**

Test Case ID	Description	Preconditions	Test Steps	Expected Result	Negative Test Cases
TC_001	Verify that a patient can be successfully added to the system.	The system must be running. The patient must provide valid details (name, phone number).	1. Navigate to the "Add Patient" section. 2. Enter a valid patient name. 3. Enter a valid phone number. 4. Click the "Add" button.	The patient is successfully added with a unique ID and appears in the patient list.	- Missing name or phone number should trigger an error. - Invalid phone number format should result in validation failure.

- **Test case for booking an appointment.**

Test Case ID	Description	Preconditions	Test Steps	Expected Result	Negative Test Cases

TC_001	Verify that a patient can book an appointment.	The system must have at least one registered patient and physiotherapist with available time slots.	1. Navigate to "Book Appointment". 2. Select a registered patient. 3. Choose a physiotherapist. 4. Select an available date and time. 5. Click "Book Appointment".	The appointment is successfully booked and appears in the schedules.	- Booking in an occupied time slot should trigger an error. - Booking for a non-existent patient should be prevented.

- **Test case for canceling an appointment.**

Test Case ID	Description	Preconditions	Test Steps	Expected Result	Negative Test Cases
TC_001	Verify that a patient can cancel an existing appointment.	The system must have at least one existing appointment.	1. Navigate to "View Appointments". 2. Locate the appointment to cancel. 3. Click "Cancel Appointment". 4. Confirm cancellation.	The appointment status updates to "Canceled", and the time slot becomes available.	- Canceling a non-existent appointment should trigger an error. - Canceling an already canceled appointment should show a warning.

- **Test case for finding physiotherapists by expertise.**

Test Case ID	Description	Preconditions	Test Steps	Expected Result	Negative Test Cases

TC_001	Verify that users can find physiotherapists by expertise.	The system must have at least one registered physiotherapist with a defined expertise..	1. Navigate to "Physiotherapists". 2. Look for a specific expertise (e.g., "Sports Therapy"). 3. Check "Registered Physios".	The system displays a list of physiotherapists with their expertise.	- Searching for a non-existent expertise should return no results. - Entering invalid characters should be handled gracefully.

5.2 Testing Approach

- Unit tests are implemented using **JUnit**.
- The system is tested for **edge cases** (e.g., duplicate appointments, invalid cancellations, overlapping schedules).

5.3 Test Results

```
run:
Appointment booked: Appointment [ID: 10000, Patient: John Smith, Physiotherapist: Sarah Johnson, Treatment: Neural mobilisation, Time: Friday 2 May
Patient added: ID: 1016, Name: New Patient
Patient removed: ID: 1016, Name: New Patient
Appointment booked: Appointment [ID: 10000, Patient: John Smith, Physiotherapist: Sarah Johnson, Treatment: Neural mobilisation, Time: Friday 2 May
Appointment cancelled: Appointment [ID: 10000, Patient: John Smith, Physiotherapist: Sarah Johnson, Treatment: Neural mobilisation, Time: Friday 2
All tests passed successfully!
BUILD SUCCESSFUL (total time: 0 seconds)
```

6. Refactoring Process

6.1 Initial Design Issues

- The system initially lacked **proper validation** for patient/physiotherapist registration.
- The **reporting system** did not include physiotherapists.
- Appointment scheduling had **conflict handling issues**.

6.2 Refactoring Steps

- Implemented **getter methods** for physiotherapists.

- Improved **error handling** in booking and cancellation.
- Enhanced **data encapsulation** to protect internal structures.

6.3 Improvements Achieved

- Enhanced **user experience** by listing appointments before cancellation.
- The system now provides **a more detailed report**.
- Strengthened **data integrity** by preventing invalid modifications.

7. Design Patterns and Principles

7.1 Applied Design Patterns

- **Singleton Pattern** for BookingService to ensure a single instance manages operations.
- **Factory Pattern** for creating instances of Physiotherapist, Patient, and Appointment.

7.2 SOLID Principles Implementation

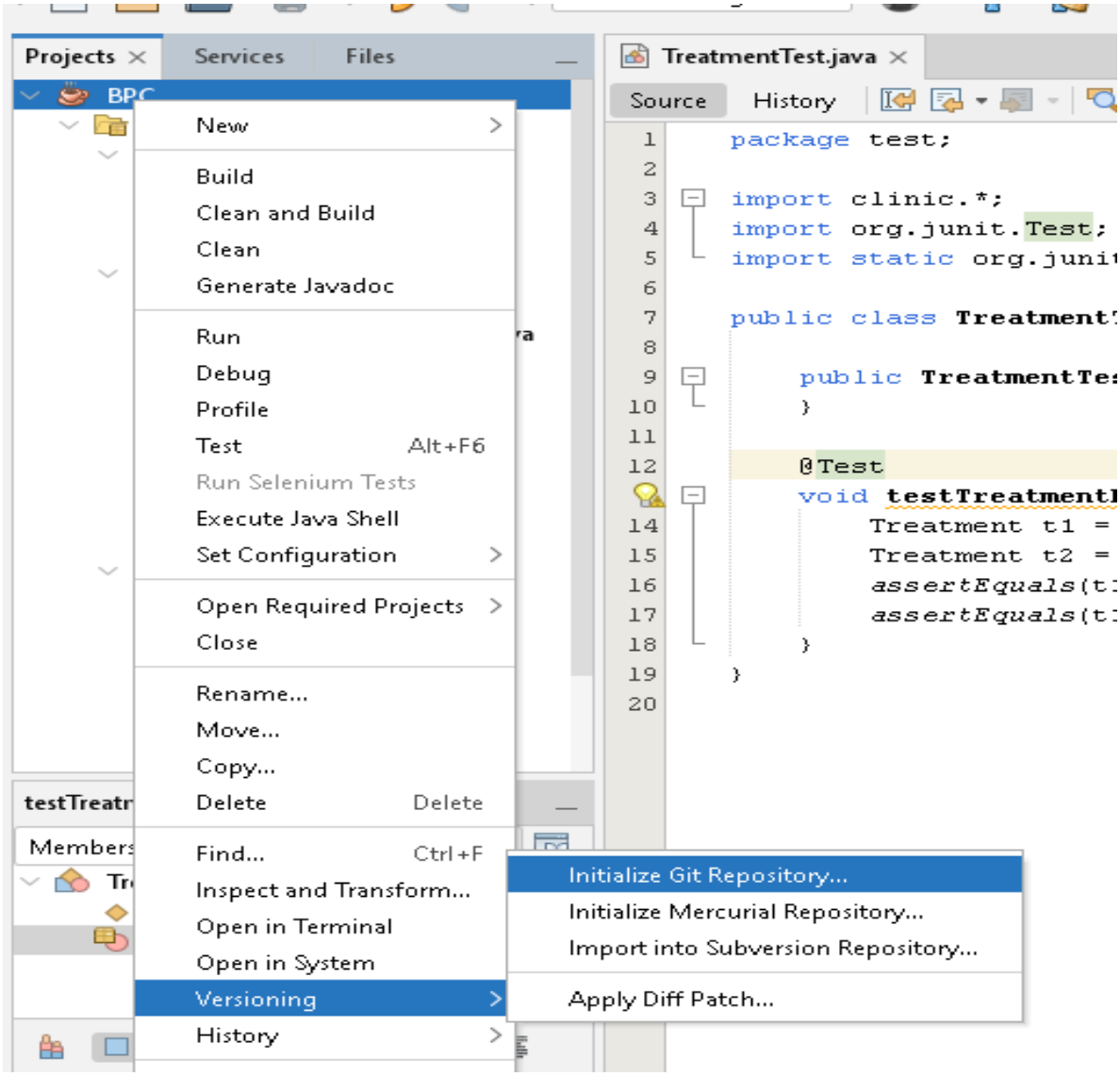
- **Single Responsibility Principle**: Each class handles a single concern.
- **Open/Closed Principle**: The system can be extended (e.g., adding new features) without modifying existing code.
- **Liskov Substitution Principle**: Objects can be replaced with their subtypes without affecting functionality.

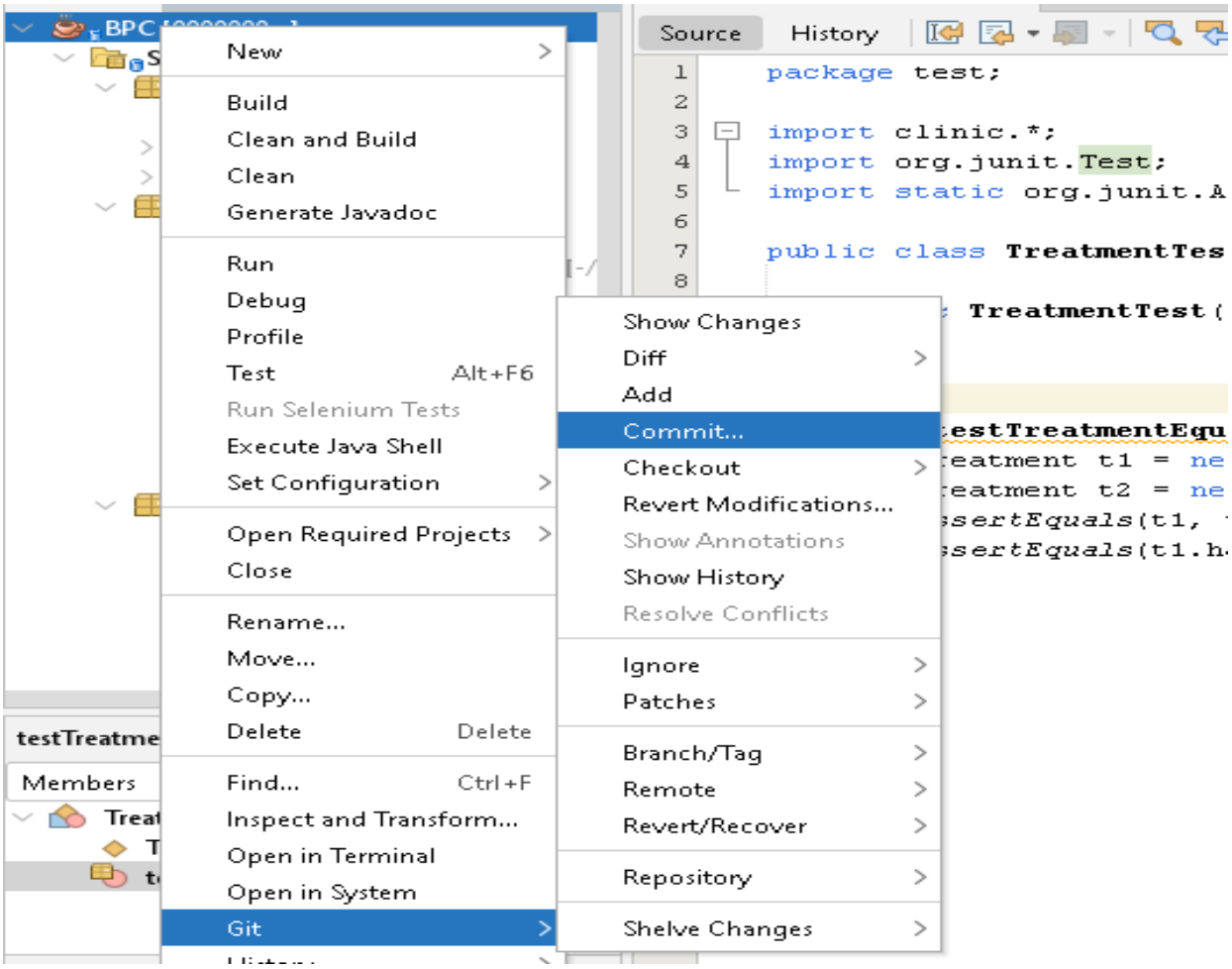
7.3 Other Design Considerations

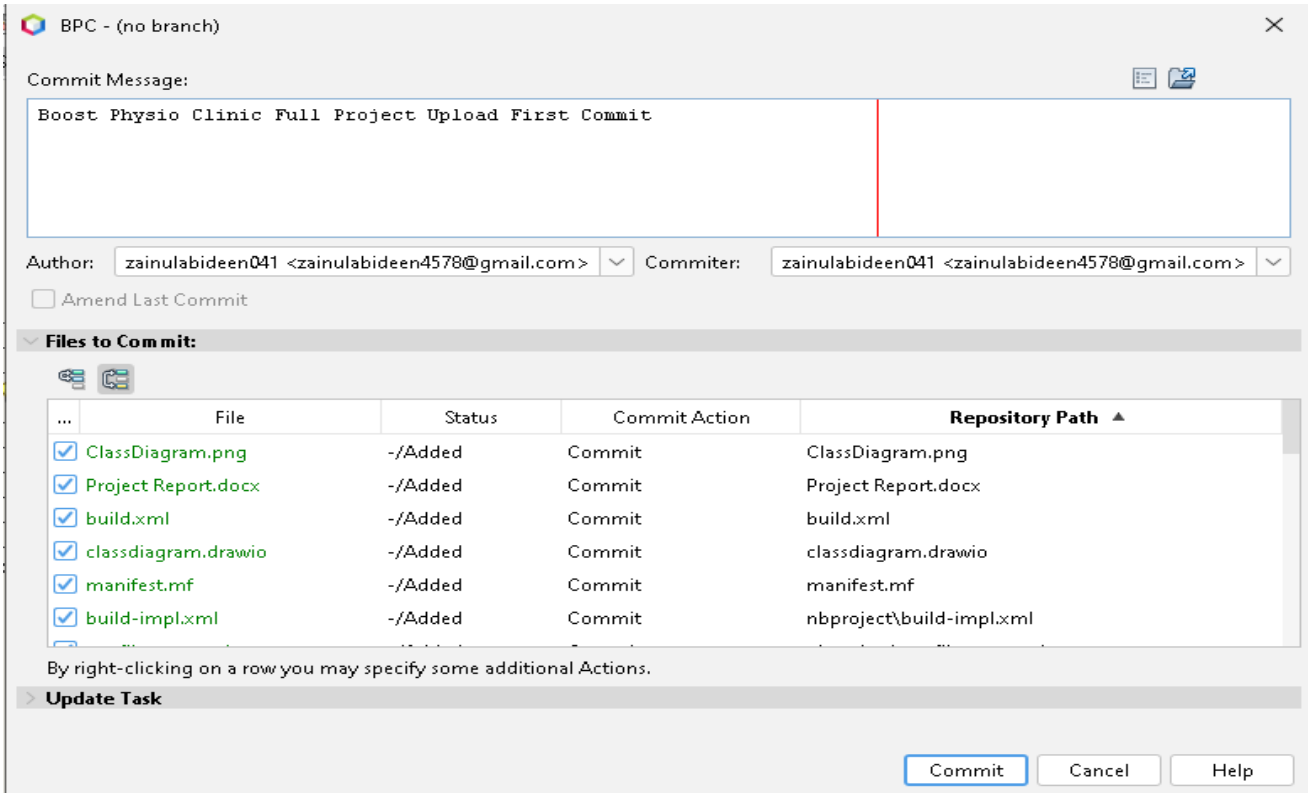
- Used **Encapsulation** to prevent direct modification of lists.
- **Exception Handling** for invalid input and scheduling conflicts.

8. Version Control Documentation

- The project is managed using **Git**, with separate branches for **features and bug fixes**.
- Commits follow a structured format for clarity.


Projects
Services
Files
TreatmentTest.java
Source
History
package test;
import clinic.*;
import org.junit.Test;
public class Treatment
@Test
void testTreatment
Treatment t1 =
Treatment t2 =
assertEquals(t
assertEquals(t
}
New
Build
Clean and Build
Clean
Generate Javadoc
Run
Debug
Profile
Test Alt+F6
Run Selenium Tests
Execute Java Shell
Set Configuration
Open Required Projects
Close
Rename...
Move...
Copy...
Delete Delete
Find... Ctrl+F
Inspect and Transform...
Open in Terminal
Open in System
Versioning
History
Initialize Git Repository...
Initialize Mercurial Repository...
Import into Subversion Repository...
Apply Diff Patch...


New
Build
Clean and Build
Clean
Generate Javadoc
Run
Debug
Profile
Test Alt+F6
Run Selenium Tests
Execute Java Shell
Set Configuration
Open Required Projects
Close
Rename...
Move...
Copy...
Delete Delete
Find... Ctrl+F
Inspect and Transform...
Open in Terminal
Open in System
Git
Show Changes
Diff
Add
Commit...
Checkout
Revert Modifications...
Show Annotations
Show History
Resolve Conflicts
Ignore
Patches
Branch/Tag
Remote
Revert/Recover
Repository
Shelve Changes
Source
History
package test;
import clinic.*;
import org.junit.Test;
import static org.junit.A
public class TreatmentTes
TreatmentTest(
testTreatmentEqu
Members

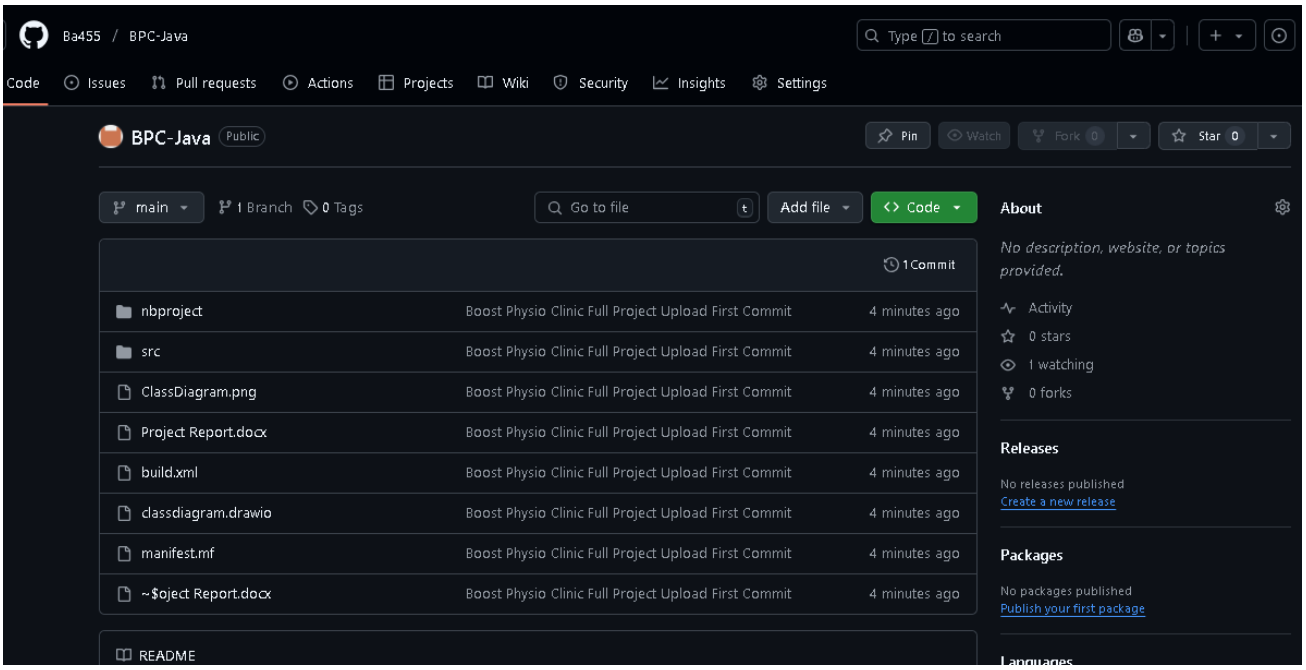

BPC - (no branch)
Commit Message:
Boost Physio Clinic Full Project Upload First Commit
Author: zainulabideen041 <zainulabideen4578@gmail.com>
Commiter: zainulabideen041 <zainulabideen4578@gmail.com>
Amend Last Commit
Files to Commit:
File
Status
Commit Action
Repository Path
ClassDiagram.png -/Added Commit ClassDiagram.png
Project Report.docx -/Added Commit Project Report.docx
build.xml -/Added Commit build.xml
classdiagram.drawio -/Added Commit classdiagram.drawio
manifest.mf -/Added Commit manifest.mf
build-impl.xml -/Added Commit nbproject\build-impl.xml
By right-clicking on a row you may specify some additional Actions.
Update Task
Commit
Cancel
Help

```
C:\Users\zainfd\Documents\NetBeansProjects\BPC_Claude>git branch -M main

C:\Users\zainfd\Documents\NetBeansProjects\BPC_Claude>git remote add origin https://github.com/Ba455/BPC-Java.git

C:\Users\zainfd\Documents\NetBeansProjects\BPC_Claude>git push -u origin main
fatal: unable to access 'https://github.com/Ba455/BPC-Java.git/': Failed to connect to github.com port 443 after 2224 ms
: Could not connect to server

C:\Users\zainfd\Documents\NetBeansProjects\BPC_Claude>git push -u origin main
info: please complete authentication in your browser...
Enumerating objects: 35, done.
Counting objects: 100% (35/35), done.
Delta compression using up to 4 threads
Compressing objects: 100% (35/35), done.
Writing objects: 100% (35/35), 603.56 KiB | 11.61 MiB/s, done.
Total 35 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/Ba455/BPC-Java.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```



9. Conclusion and Future Improvements

The **Physiotherapy Appointment System** provides a robust solution for managing patient bookings efficiently. Future improvements include:

- **GUI-based application** instead of a console-based system.
- **Persistent storage** using databases.
- **Automated appointment reminders**.
- **Multi-user support** with login authentication.