Medical Management System Code Presentation

By

Louni mohammed said

Sala7 7oudaifa

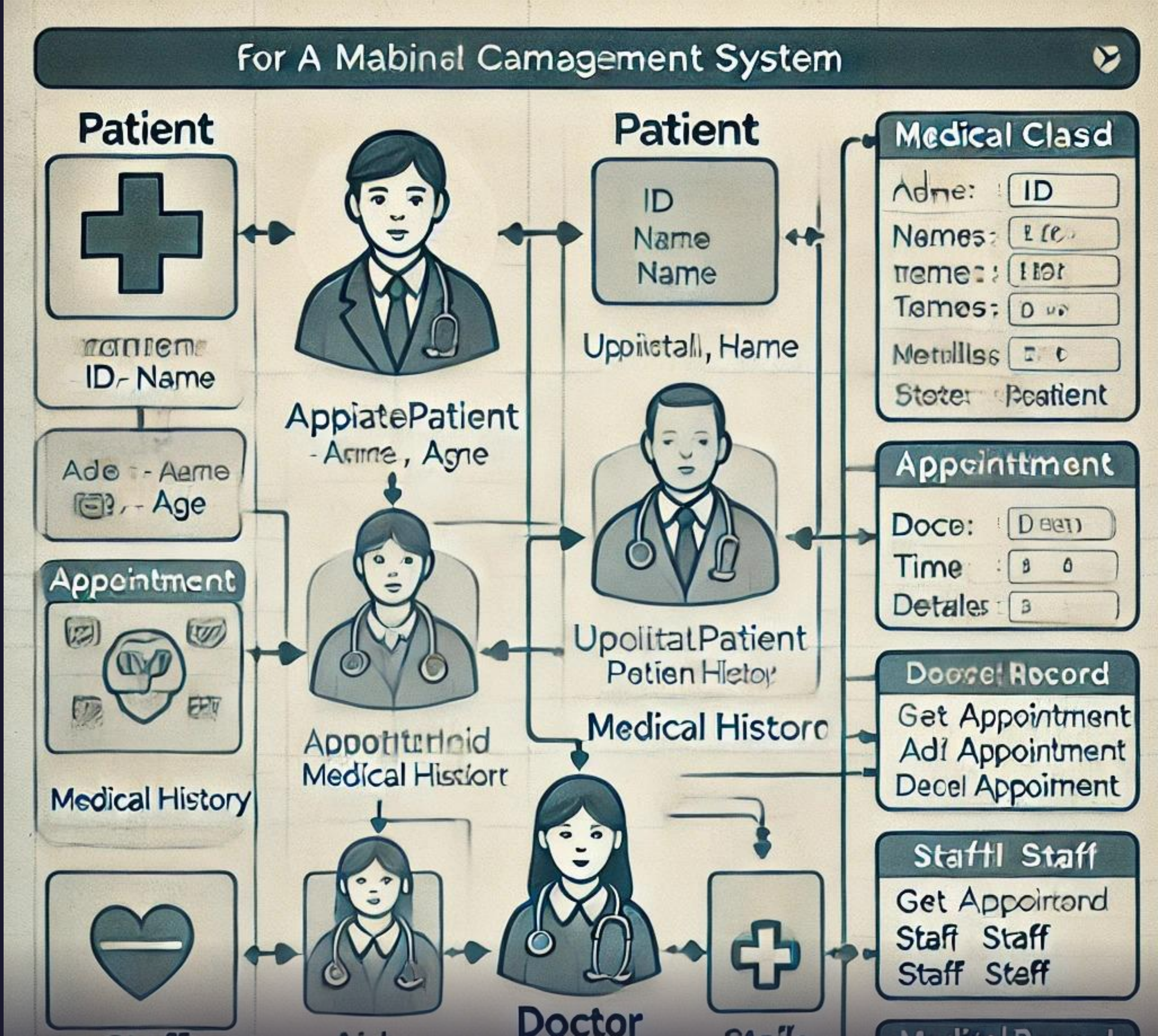
Sehab hamzaabderrahmane

Introduction :

• Objective: • A comprehensive application for managing medical records, appointments, and patient files with role-specific functionalities. • Why This System? o Centralized data management. o Automation of repetitive tasks (e.g., document generation). o Role-specific workflows for doctors, secretaries, and patients. • Key Features: o Medical Record Management. o Appointment Scheduling. o Patient File Handling.

Architectural Overview :

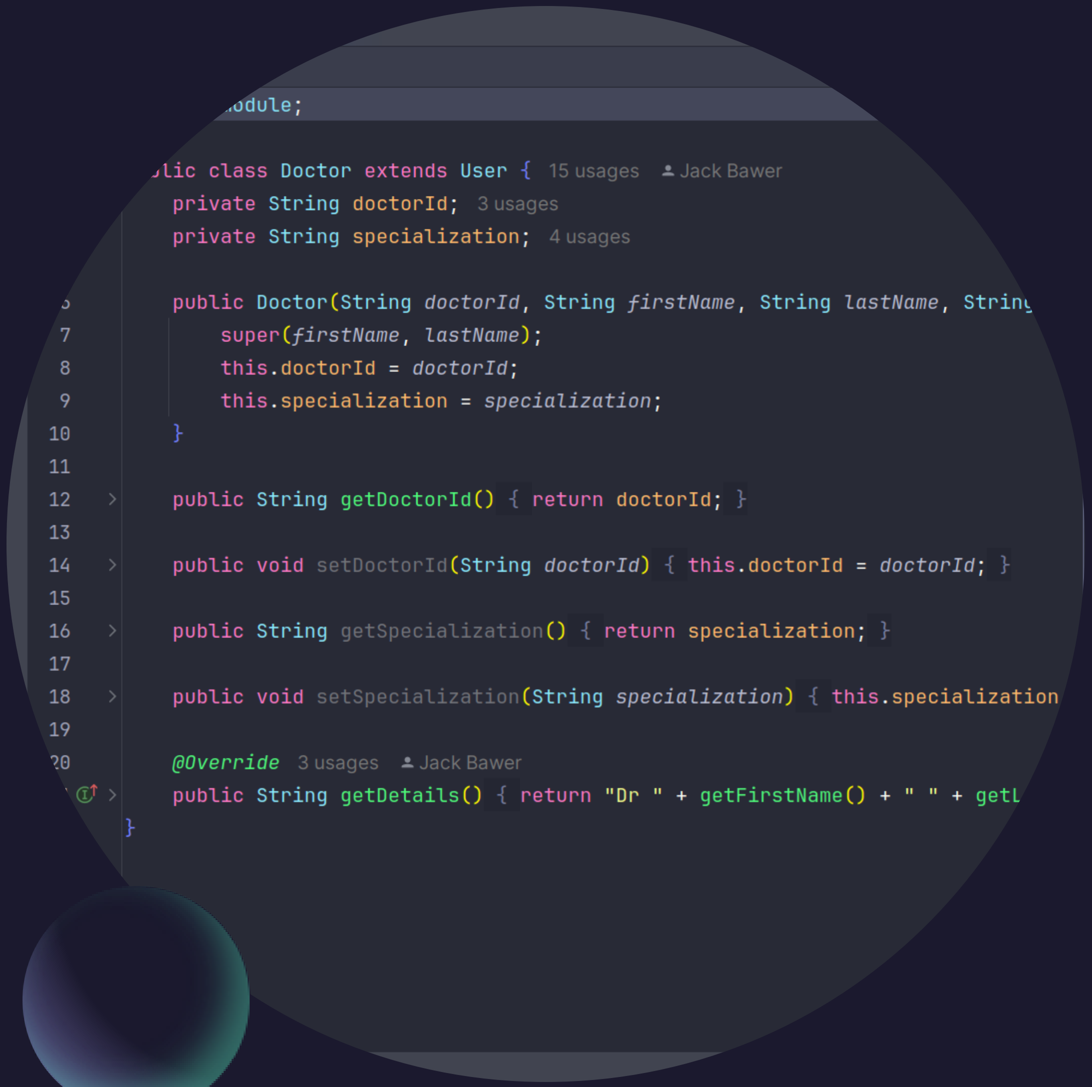
• • Core Layers: o Modules: Encapsulates the data and entity logic (e.g., Patient, Doctor). o Services: Provides operations on modules (e.g., AppointmentService). o User Interface: Bridges the user interaction with back-end logic. • Object-Oriented Principles Used: o Encapsulation: Data protection via private attributes and accessors. o Inheritance: Common functionality in the User class shared by Doctor, Patient, and Secretary. o Polymorphism: Abstract methods in User enable dynamic behavior in subclasses. o Abstraction: Services hide implementation details from the UI.



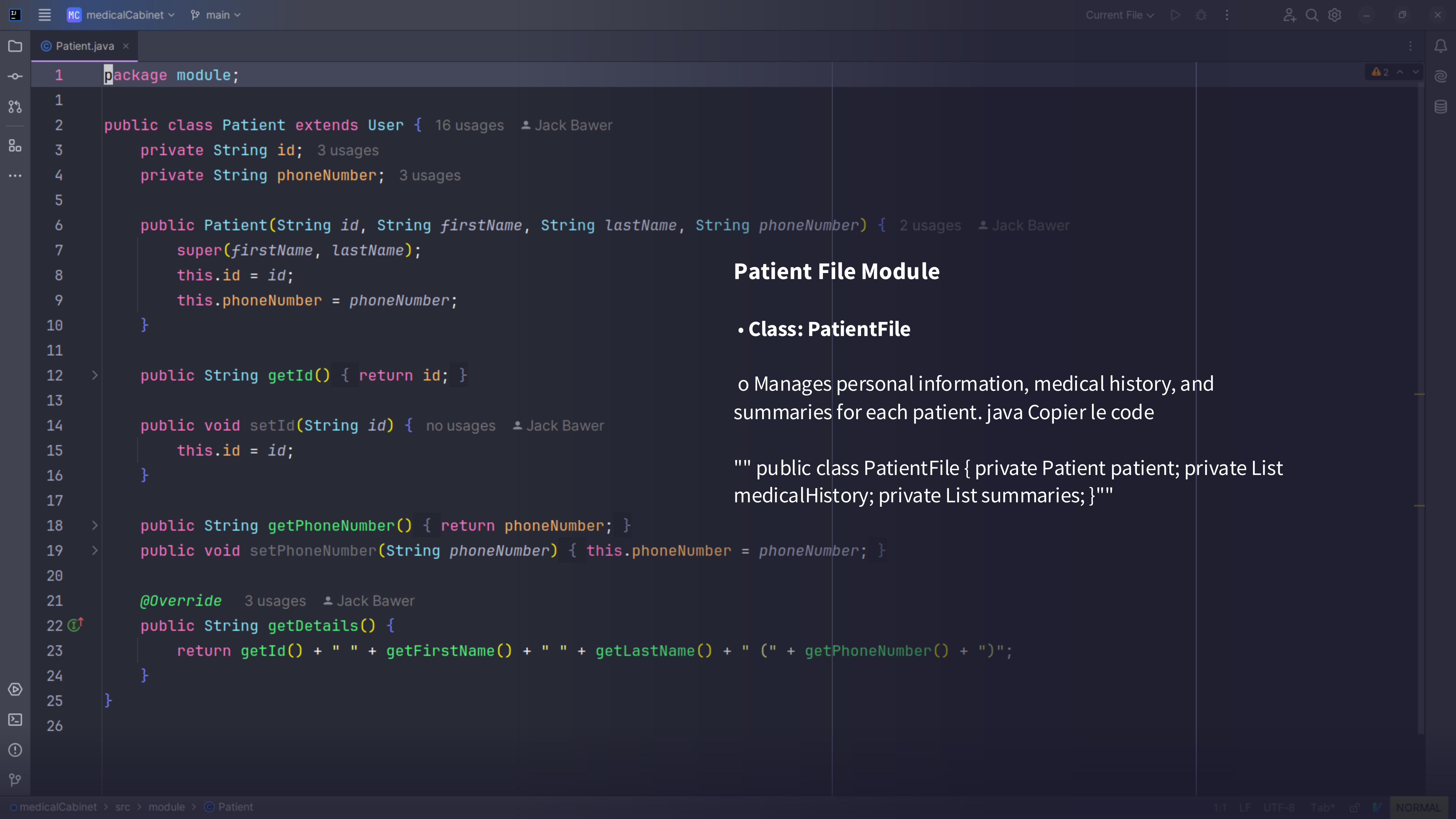
Key Entities in the System User Module:

• Abstract Class:
User
o Purpose:
Shared base class for all user types. java
Copier le code
public abstract class User { private String firstName; private String lastName; public abstract String getDetails(); }

• Derived Classes:
o Doctor:
Adds specialization and doctorId. o
Patient: Includes id and phoneNumber. o
Secretary:
Represents administrative staff. Record Module
• Abstract Class: Record
o Encapsulates shared fields like dateTime and patient.
o Extended by specific records:
▪ Consultation: Stores observations, prescriptions, and summaries.
▪ Appointment: Includes doctor, secretary, and status details.



```
odule;

lic class Doctor extends User {  15 usages  Jack Bawer
    private String doctorId;  3 usages
    private String specialization;  4 usages

    public Doctor(String doctorId, String firstName, String lastName, Strin
        super(firstName, lastName);
        this.doctorId = doctorId;
        this.specialization = specialization;
    }

    public String getDoctorId() { return doctorId; }

    public void setDoctorId(String doctorId) { this.doctorId = doctorId; }

    public String getSpecialization() { return specialization; }

    public void setSpecialization(String specialization) { this.specialization

    @Override  3 usages  Jack Bawer
    public String getDetails() { return "Dr " + getFirstName() + " " + getL
}
```



```java
package module;

public class Patient extends User {  16 usages  Jack Bawer
    private String id;  3 usages
    private String phoneNumber;  3 usages

    public Patient(String id, String firstName, String lastName, String phoneNumber) {  2 usages  Jack Bawer
        super(firstName, lastName);
        this.id = id;
        this.phoneNumber = phoneNumber;
    }

    public String getId() { return id; }

    public void setId(String id) {  no usages  Jack Bawer
        this.id = id;
    }

    public String getPhoneNumber() { return phoneNumber; }
    public void setPhoneNumber(String phoneNumber) { this.phoneNumber = phoneNumber; }

    @Override  3 usages  Jack Bawer
    public String getDetails() {
        return getId() + " " + getFirstName() + " " + getLastName() + " (" + getPhoneNumber() + ")";
    }
}
```

Patient File Module

• **Class: PatientFile**

o Manages personal information, medical history, and summaries for each patient. java Copier le code

"" public class PatientFile { private Patient patient; private List medicalHistory; private List summaries; }""

Service Layer AppointmentService

• Purpose:

Manages all appointment-related operations.

• Key Methods:

o addAppointment(Appointment appointment)

o getAppointmentsByDoctor(String doctorId)

o getAppointmentsByPatient(String patientId)

o cancelAppointment(LocalDateTime dateTime, String patientId) MedicalRecordService

• Purpose:

Handles CRUD operations for medical records.

• Key Methods:

o addMedicalRecord(MedicalRecord record)

o getMedicalRecord(String patientId)

o updateRecord(MedicalRecord record) o deleteRecord(String patientId) PatientFileService

• Purpose:

Centralizes and updates patient-specific data.

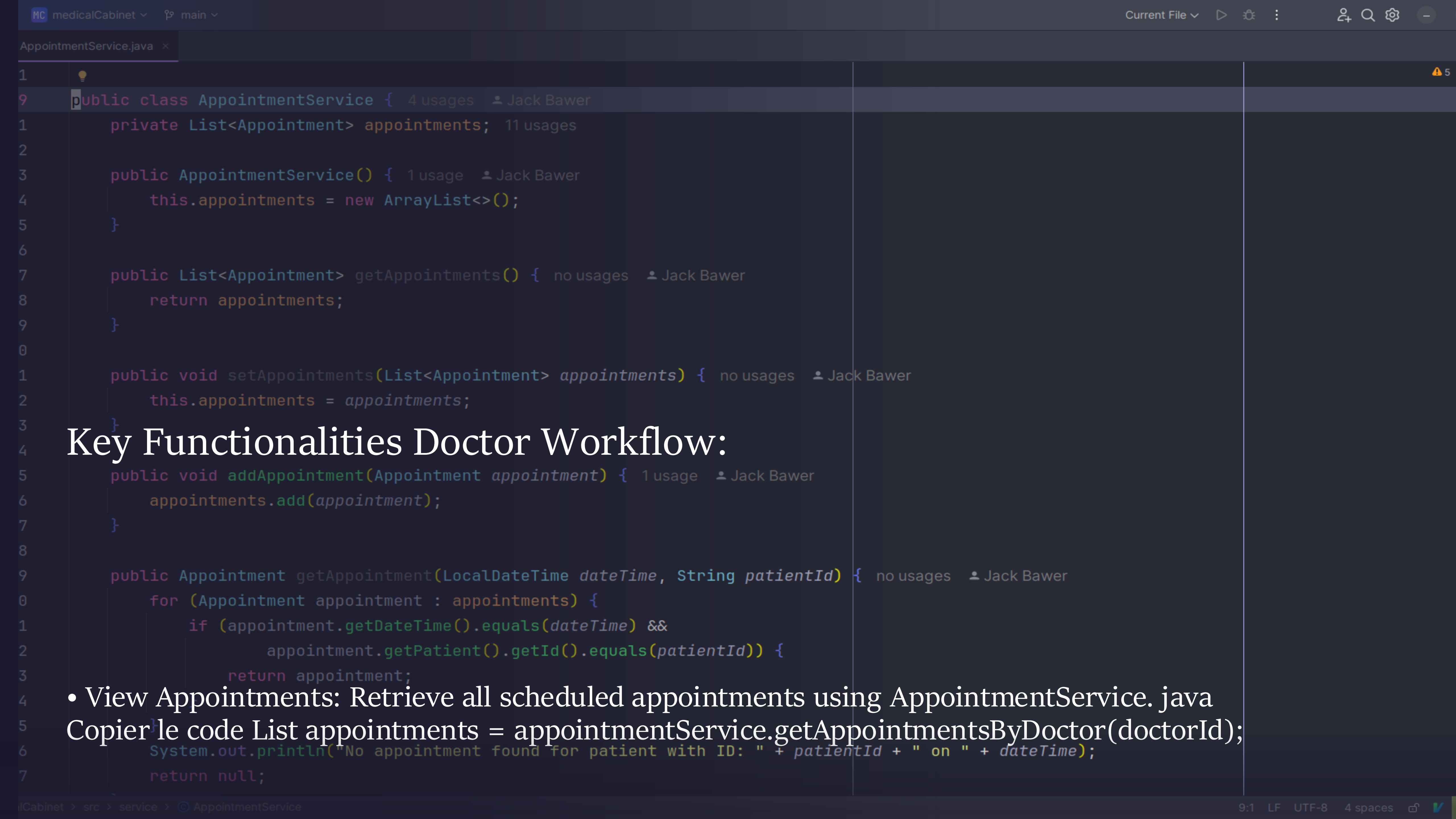
• Key Methods:

o addPatientFile(PatientFile file)

o getPatientFile(String patientId) o updatePatientFile(PatientFile file)

Key Functionalities Doctor Workflow:

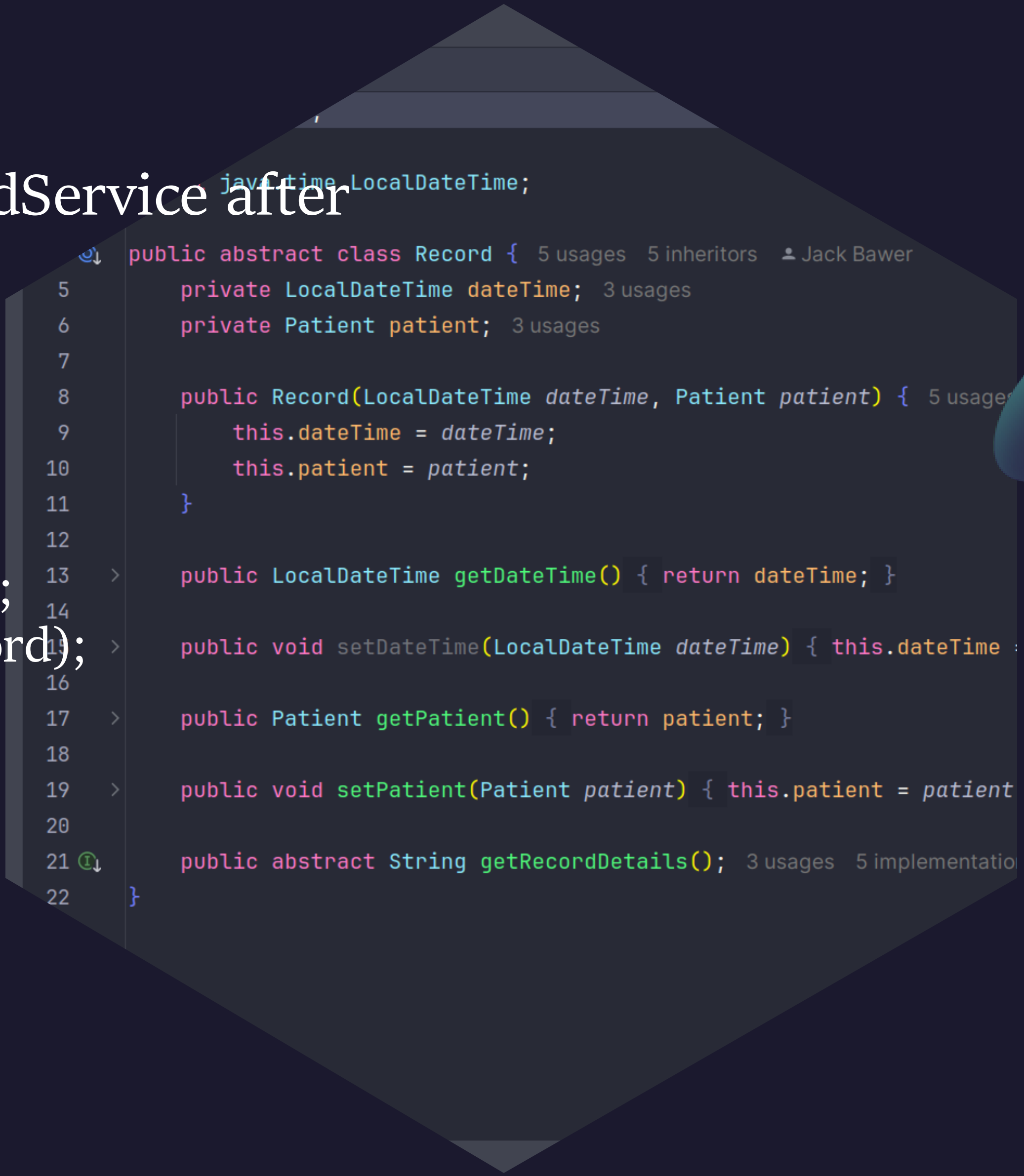
• View Appointments: Retrieve all scheduled appointments using AppointmentService. java Copier le code List appointments = appointmentService.getAppointmentsByDoctor(doctorId);



```
public class AppointmentService {  4 usages  Jack Bawer
    private List<Appointment> appointments;  11 usages

    public AppointmentService() {  1 usage  Jack Bawer
        this.appointments = new ArrayList<>();
    }

    public List<Appointment> getAppointments() {  no usages  Jack Bawer
        return appointments;
    }

    public void setAppointments(List<Appointment> appointments) {  no usages  Jack Bawer
        this.appointments = appointments;
    }

    public void addAppointment(Appointment appointment) {  1 usage  Jack Bawer
        appointments.add(appointment);
    }

    public Appointment getAppointment(LocalDateTime dateTime, String patientId) {  no usages  Jack Bawer
        for (Appointment appointment : appointments) {
            if (appointment.getDateTime().equals(dateTime) &&
                    appointment.getPatient().getId().equals(patientId)) {
                return appointment;
        System.out.println("No appointment found for patient with ID: " + patientId + " on " + dateTime);
        return null;
```

- Record Consultation:
 Create a consultation with MedicalRecordService after gathering observation and prescription

 details. java
Copier le code
Consultation consultation = new Consultation(...);
medicalRecordService.updateRecord(medicalRecord);



```
java.time.LocalDateTime;

public abstract class Record {  5 usages  5 inheritors  Jack Bawer
    private LocalDateTime dateTime;  3 usages
    private Patient patient;  3 usages

    public Record(LocalDateTime dateTime, Patient patient) {  5 usage
        this.dateTime = dateTime;
        this.patient = patient;
    }

    public LocalDateTime getDateTime() { return dateTime; }

    public void setDateTime(LocalDateTime dateTime) { this.dateTime

    public Patient getPatient() { return patient; }

    public void setPatient(Patient patient) { this.patient = patient

    public abstract String getRecordDetails();  3 usages  5 implementatio
}
```

Secretary Workflow:

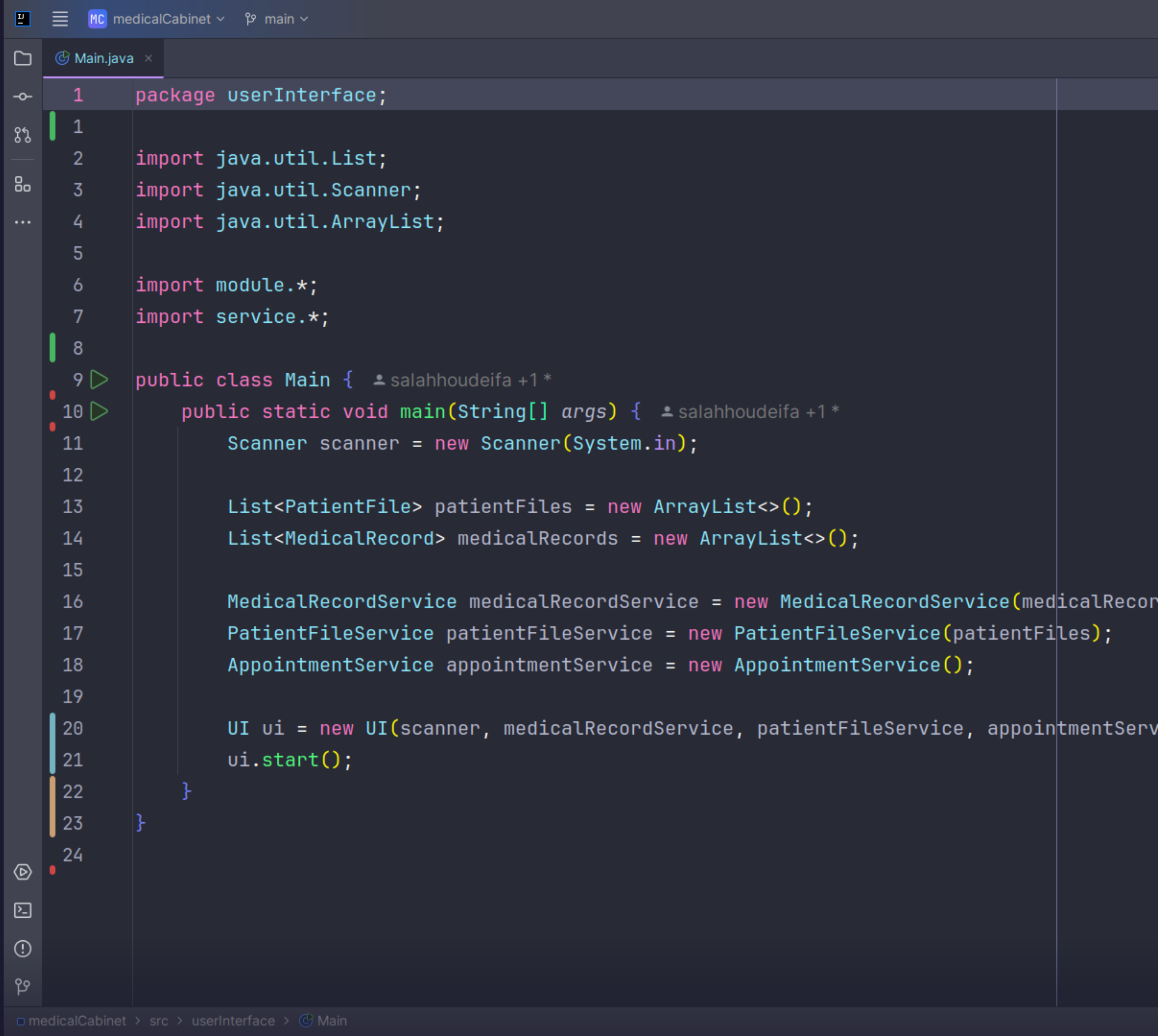
- Schedule Appointments:Add appointments through

AppointmentService. java
Copier le code
```
 appointmentService.addAppointment(new Appointment(...));
```

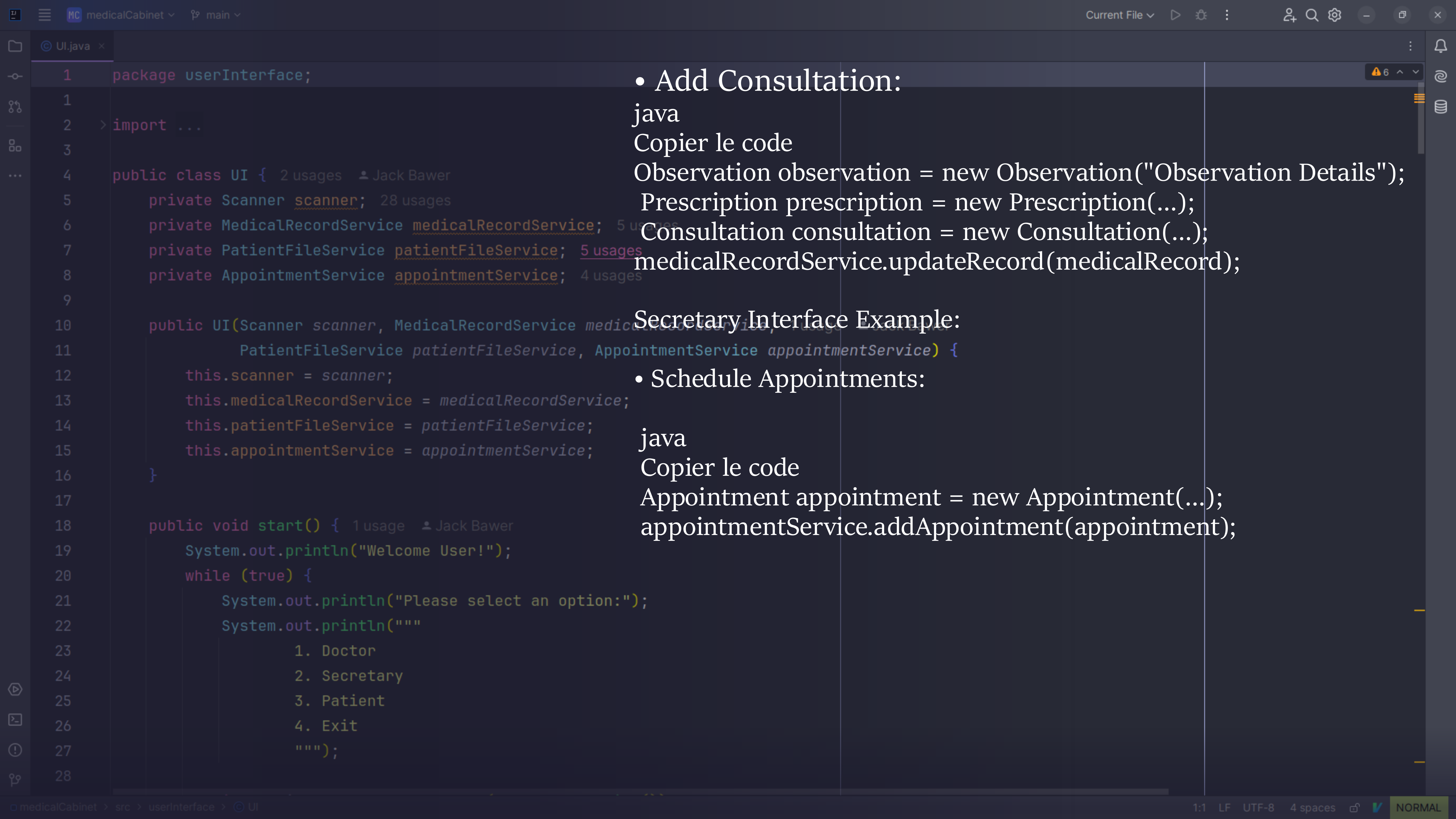
 • Manage Patient Files:
Create and update patient records in
PatientFileService. java
Copier le code
```
patientFileService.addPatientFile(new PatientFile(...));
```

User Interface Layer UI Class

• Centralized interaction point for Doctor, Secretary, and Patient.
 • Example: Doctor options in the interface.
 java
 Copier le code
 case 1 -> handleDoctor();
case 2 -> handleSecretary();
 case 3 -> handlePatient();
 Doctor Interface Example:
• View Appointments:
java
 Copier le code
List <appointments> = appointmentService.getAppointmentsByDoctor(doctorId);
 appointments.forEach(a -> System.out.println(a.getRecordDetails()));



```java
package userInterface;

import java.util.List;
import java.util.Scanner;
import java.util.ArrayList;

import module.*;
import service.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        List<PatientFile> patientFiles = new ArrayList<>();
        List<MedicalRecord> medicalRecords = new ArrayList<>();

        MedicalRecordService medicalRecordService = new MedicalRecordService(medicalRecor
        PatientFileService patientFileService = new PatientFileService(patientFiles);
        AppointmentService appointmentService = new AppointmentService();

        UI ui = new UI(scanner, medicalRecordService, patientFileService, appointmentServ
        ui.start();
    }
}
```



- Add Consultation:

java
Copier le code
```java
Observation observation = new Observation("Observation Details");
 Prescription prescription = new Prescription(...);
 Consultation consultation = new Consultation(...);
medicalRecordService.updateRecord(medicalRecord);
```

Secretary Interface Example:

- Schedule Appointments:

java
Copier le code
```java
Appointment appointment = new Appointment(...);
appointmentService.addAppointment(appointment);
```

```java
package userInterface;

import ...

public class UI {
    private Scanner scanner;
    private MedicalRecordService medicalRecordService;
    private PatientFileService patientFileService;
    private AppointmentService appointmentService;

    public UI(Scanner scanner, MedicalRecordService medicalRecordService,
              PatientFileService patientFileService, AppointmentService appointmentService) {
        this.scanner = scanner;
        this.medicalRecordService = medicalRecordService;
        this.patientFileService = patientFileService;
        this.appointmentService = appointmentService;
    }

    public void start() {
        System.out.println("Welcome User!");
        while (true) {
            System.out.println("Please select an option:");
            System.out.println("""
                    1. Doctor
                    2. Secretary
                    3. Patient
                    4. Exit
                    """);
```

Example of Integration Consultation Class

- Example of a Record subclass with specific attributes for consultations.

java

Copier le code

```
 public class Consultation extends Record {
private Doctor doctor;
private Observation observation;
private Prescription prescription; }
```

Appointment Class

- Handles scheduling with references to Doctor and Secretary.

java

```
 Copier le code public class Appointment extends Record {
 private Secretary secretary;
private Doctor doctor;
private String status;
 }
```

Summary

•Strengths:

o Strong modularity through OOP principles.

o Simplified workflows for each user type.

o Scalable service layer for future enhancements.

• Future Improvements:

o Add a graphical user interface (GUI).

o Integrate notifications for appointment reminders.

o Deploy on web and mobile platforms.