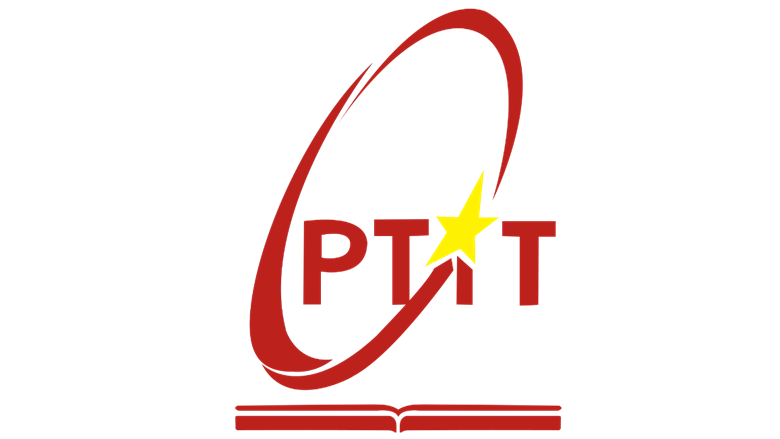
A white rectangular frame with a black border

AI-generated content may be incorrect.

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN 1**

****

**BÁO CÁO BÀI TẬP LỚN**

**LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

**ĐỀ TÀI: PHÁT TRIỂN HỆ THỐNG ỨNG DỤNG**

**QUẢN LÝ KHÁM BỆNH VÀ PHÂN KHOA**

**Nhóm học phần:** 16

**Nhóm bài tập lớn:** 15

**Giảng viên:** Nguyễn Mạnh Sơn

**Thành Viên:** Mai Thị Dịu -

Vũ Thành Nguyên -

Trần Thu Thiên - B23DCCN785

**Hà Nội 2025**

**LỜI CẢM ƠN**

Chúng em xin chân thành cảm ơn Khoa Công Nghệ Thông Tin 1 thuộc Học viện Công Nghệ Bưu Chính Viễn Thông đã tạo điều kiện và đưa môn Lập trình Hướng Đối Tượng vào chương trình học, trang bị cho chúng em những kiến thức nền tảng và tư duy lập trình hiện đại, thiết yếu cho việc phát triển các ứng dụng thực tế.

Đặc biệt, chúng em xin bày tỏ lòng tri ân sâu sắc nhất tới Thầy Nguyễn Mạnh Sơn, người đã tận tình hướng dẫn, truyền đạt kiến thức chuyên môn vững vàng, và cung cấp những góp ý quý báu. Sự chỉ bảo của Thầy là động lực lớn nhất giúp chúng em có thể hoàn thành đề tài "Phát triển Hệ thống Ứng dụng Quản lý Khám bệnh và Phân khoa Tự động" một cách tốt nhất.

Chúng em hiểu rằng, do giới hạn về thời gian và kinh nghiệm, báo cáo không thể tránh khỏi những thiếu sót. Kính mong nhận được những nhận xét, đánh giá và góp ý từ Thầy/Cô để bài làm của chúng em được hoàn thiện hơn.

Chúng em xin chân thành cảm ơn!

**MỤC LỤC**

[I. GIỚI THIỆU TỔNG QUAN 5](#_heading=h.u3l11uhcvo5g)

[1. Đặt vấn đề & Mục tiêu 5](#_heading=h.plx5lt6rsyad)

[1.1. Đặt vấn đề 5](#_heading=h.qbato5p9au2l)

[1.2. Mục tiêu 5](#_heading=h.y76srnde99i5)

[2. Phạm vi ứng dụng 6](#_heading=h.lbcnkelmzkku)

[2.1. Chức năng Tập trung 6](#_heading=h.gbria3tx9e0m)

[2.2. Công nghệ sử dụng 6](#_heading=h.5sldyb4coj6k)

[2.3. Giới hạn 6](#_heading=h.rd61807wftwa)

[**II. PHÂN TÍCH & THIẾT KẾ HỆ THỐNG 7**](#_heading=h.cqu4ufcqmtwg)

[1. Yêu cầu chức năng & phi chức năng 7](#_heading=h.2bax8yw3w5mt)

[1.1. Yêu cầu chức năng 7](#_heading=h.jihgkz4habl)

[1.2. Yêu cầu phi chức năng 7](#_heading=h.1aho2337cbc9)

[2. Kiến trúc hệ thống 8](#_heading=h.5yd1hci3yyw1)

[3. Thiết kế hướng đối tượng 8](#_heading=h.urvc8xx331hr)

[3.1. Tính đóng gói 8](#_heading=h.w5hdx4c64xr4)

[3.2. Tính trừu tượng 9](#_heading=h.okpdgtblmmnp)

[3.3 Mối quan hệ giữa các lớp 10](#_heading=h.yfjomuupclbs)

[**III. THIẾT KẾ CƠ SỞ DỮ LIỆU 10**](#_heading=h.g8wphffsecmj)

[1. Mục tiêu thiết kế CSDL 11](#_heading=h.fjffovmmk5h1)

[2. Sơ đồ thực thể liên kết 11](#_heading=h.266z6zfpphbw)

[3. Sơ đồ quan hệ dữ liệu 12](#_heading=h.ynp3nlb3pxda)

[4. Quy tắc nghiệp vụ cốt lõi 12](#_heading=h.8o8xazv3c00e)

[4.1. Seed dữ liệu mẫu 12](#_heading=h.wmmluojsc1yy)

[4.2. Cơ chế cấp số thứ tự 14](#_heading=h.veusoiqzbonw)

[4.3. Quy tắc phân khoa 16](#_heading=h.7ess5r58ox25)

[4.4. Bảo mật tài khoản 17](#_heading=h.agip3wvefiqj)

[**IV. TRIỂN KHAI & CHI TIẾT KỸ THUẬT 17**](#_heading=h.ksjmylocmsa4)

[1. Cấu trúc dự án & Khởi tạo 17](#_heading=h.snf8ma2dawvh)

[1.1. Cấu trúc thư mục cốt lõi 17](#_heading=h.9ucfd7xsfsgd)

[1.2. Module Cấu hình & Khởi tạo 18](#_heading=h.e286cn6obbzy)

[2. Lớp dịch vụ & Logic nghiệp vụ 19](#_heading=h.ci4wftrw3go2)

[2.1. Quy trình Xác thực Người dùng (AuthService) 20](#_heading=h.olcxbfemog8)

[2.2. Xử lý Phân khoa và Hàng đợi (TriageService & QueueNumberService) 21](#_heading=h.doc3aga81g53)

[2.3. Quản lý Hàng đợi và Khám bệnh (DoctorService): 24](#_heading=h.p94wtgh6cwvn)

[3. Lớp truy cập dữ liệu 25](#_heading=h.p40ckadnb4xq)

[3.1. Các lớp quản lý CSDL 26](#_heading=h.wyrrsehn1nyw)

[3.2. Các lớp truy cập dữ liệu 31](#_heading=h.uumrdg12615l)

[3.3. Sự phối hợp với lớp Service 38](#_heading=h.ezvpnhicaf6h)

[4. Lớp giao diện người dùng 40](#_heading=h.b2wg2q7n5gj2)

[4.1. Lớp PatientApp 40](#_heading=h.relu8kycjuki)

[4.2. Lớp DoctorLoginFrame 41](#_heading=h.1rkt1gkc0xhq)

[4.3. Lớp DoctorDashboardFrame 41](#_heading=h.ct80gis9g53e)

[4.4. Lớp PrescriptionDialog 42](#_heading=h.s06fxazdd372)

[4.5 Lớp ExamResultDialog 42](#_heading=h.tweap3hqwppg)

[5. Các lớp tiện ích 42](#_heading=h.f8sy9vkp5u92)

[5.1 Lớp PasswordHasher 42](#_heading=h.5qu9ommr7br7)

[5.2 Lớp UiUtils 43](#_heading=h.d92rplizml1)

[**V. LUỒNG XỬ LÝ CHÍNH 43**](#_heading=h.i7q36kdbj966)

[1. Luồng xử lý bệnh nhân 43](#_heading=h.vkhkb8lr9l4o)

[2. Luồng xử lý bác sĩ 44](#_heading=h.kjjh539g428h)

[**VI. KẾT QUẢ & ĐÁNH GIÁ 44**](#_heading=h.5hokjz2of2eo)

[1. Hướng dẫn chạy thử & Kiểm thử 44](#_heading=h.es8xgx2zlbim)

[1.1. Hướng dẫn chạy thử 44](#_heading=h.dkeflap3ucwv)

[1.2. Kịch bản kiểm thử 45](#_heading=h.i20pq3r1olql)

[**VII. KẾT LUẬN & HƯỚNG PHÁT TRIỂN 47**](#_heading=h.3nus5law3wvt)

[**PHỤ LỤC 47**](#_heading=h.c289ym5fuqhx)

# GIỚI THIỆU TỔNG QUAN

## Đặt vấn đề & Mục tiêu

### 1.1. Đặt vấn đề

Trong lĩnh vực quản lý y tế hiện nay, hiệu quả của quy trình tiếp nhận ban đầu đóng vai trò then chốt trong việc quyết định chất lượng dịch vụ và khả năng xử lý các ca bệnh. Các phòng khám thường gặp khó khăn trong việc quản lý lượng hồ sơ, thông tin bác sĩ, và kiểm soát số thứ tự một cách có hệ thống. Quy trình phân khoa thường được thực hiện thủ công bởi nhân viên tiếp nhận, dựa trên kinh nghiệm và các quy tắc nội bộ.

Điều này tiềm ẩn các rủi ro như:

* Sai sót do ghi chép và xử lý thông tin thủ công.
* Thiếu tính minh bạch và khó khăn trong việc theo dõi số thứ tự.
* Hiệu suất làm việc thấp khi lượng bệnh nhân đông.

Để giải quyết vấn đề này, việc xây dựng một hệ thống phần mềm nhằm số hóa toàn bộ quy trình quản lý và cung cấp công cụ hỗ trợ cho nhân viên trong việc phân khoa là cần thiết. Dự án **“Phát triển Hệ thống Ứng dụng Quản lý và Phân khoa”** tập trung vào việc xây dựng một giải pháp quản lý mạnh mẽ và có cấu trúc.

### 1.2. Mục tiêu

Mục tiêu chính của đề tài là xây dựng một hệ thống ứng dụng quản lý khám bệnh đạt được các tiêu chí sau:

* Ứng dụng nguyên lý Lập trình hướng đối tượng:
  + Áp dụng **Kiến trúc nhiều lớp (Layered Architecture)** để tách biệt rõ ràng trách nhiệm giữa các phần: **Dữ liệu (DAO)**, **Logic Nghiệp vụ (Service)** và **Giao diện người dùng (UI).**
  + **Đóng gói (Encapsulation):** Thiết kế các lớp **Model** với thuộc tính private và truy cập qua getters/setters.
  + **Trừu tượng (Abstraction):** Áp dụng mẫu thiết kế **DAO** để che giấu các chi tiết thao tác SQL và quản lý kết nối CSDL khỏi các lớp nghiệp vụ.
  + Đảm bảo mã nguồn dễ bảo trì, mở rộng và tuân thủ các chuẩn lập trình hướng đối tượng.
* Hỗ trợ Quản lý và Quy trình phân khoa:
  + Cung cấp giao diện trực quan cho nhân viên/y tá để **nhập thông tin và triệu chứng** bệnh nhân, sau đó **gán khoa khám** theo quyết định thủ công (hỗ trợ bán tự động).
  + Xây dựng module **QueueNumberService** để cấp phát số thứ tự khám bệnh, giúp quản lý thứ tự khám bệnh hiệu quả và minh bạch.
  + Giảm thiểu sai sót do ghi chép thủ công và đảm bảo thông tin bệnh nhân được chuyển giao đồng bộ giữa khu vực tiếp nhận và phòng khám.
* Quản lý thông tin phòng khám:
  + Xây dựng các lớp **DAO** **(PatientDao, DoctorDao, DepartmentDao)** để quản lý đầy đủ thông tin về bệnh nhân, bác sĩ, và khoa khám.
  + Hỗ trợ lưu trữ lịch sử khám bệnh, kết quả khám và đơn thuốc
  + Cung cấp cơ sở dữ liệu đồng bộ (SQLite), dễ dàng tra cứu, hỗ trợ cho việc báo cáo và thống kê về hoạt động của phòng khám.
* Bảo mật cơ bản:
  + Áp dụng module **PasswordHasher** trong **AuthService** để mã hóa mật khẩu tài khoản nhân viên/bác sĩ trước khi lưu trữ.
  + Hạn chế truy cập trái phép, bảo vệ thông tin cá nhân của bệnh nhân và các dữ liệu nhạy cảm của phòng khám.

## Phạm vi ứng dụng

Phạm vi của hệ thống này được xác định rõ ràng, tập trung vào việc số hóa quy trình quản lý và hỗ trợ phân khoa trong phạm vi một phòng khám vừa và nhỏ.

### 2.1. Chức năng Tập trung

* **Quản lý Thông tin Cốt lõi:** Quản lý các đối tượng nghiệp vụ chính (**Patient, Doctor, Department, Prescription**) thông qua các lớp **DAO** tương ứng.
* **Hỗ trợ Phân khoa:** Cung cấp giao diện để nhân viên tiếp nhận thực hiện việc **gán khoa** (**triage\_department\_id**) và quản lý hàng đợi (**QueueNumberService**).
* **Xử lý Nghiệp vụ Bác sĩ:** Hỗ trợ quy trình khám và kê đơn.

### 2.2. Công nghệ sử dụng

* **Ngôn ngữ Lập trình:** Java.
* **Cơ sở Dữ liệu:** SQLite - Hệ thống hoạt động Standalone (độc lập), không yêu cầu server CSDL bên ngoài.
* **Giao diện Người dùng:** Sử dụng thư viện giao diện cơ bản của Java (Swing), thể hiện qua các lớp UI.

### 2.3. Giới hạn

Hệ thống này không bao gồm các chức năng phức tạp như thanh toán, bảo hiểm y tế, thống kê chuyên sâu (Business Intelligence), hoặc tích hợp công nghệ phân loại tự động (AI/Machine Learning).

# PHÂN TÍCH & THIẾT KẾ HỆ THỐNG

## Yêu cầu chức năng & phi chức năng

### 1.1. Yêu cầu chức năng

Các yêu cầu chức năng của hệ thống được xác định nhằm hỗ trợ nhân viên phòng khám thực hiện các nghiệp vụ quản lý và phân khoa thủ công một cách hiệu quả, tuân theo kiến trúc đã thiết kế.

| Lĩnh vực | Yêu cầu | Lớp liên quan |
| --- | --- | --- |
| Quản lý dữ liệu | CRUD thông tin về **Bác sĩ**, **Khoa** và **Luật triệu chứng** (dùng để tham khảo) | **\*Dao.java** |
| Tiếp nhận & Phân khoa | Đăng ký bệnh nhân: Nhận thông tin và triệu chứng khai báo từ bệnh nhân | **TriageService** |
| Gán khoa: Nhân viên tiếp nhận chọn thủ công khoa khám cho bệnh nhân, dựa trên triệu chứng khai báo |
| Quản lý hàng đợi | Cấp phát số thứ tự tiếp theo một cách tuần tự cho bệnh nhân sau khi khoa đã được gán | **QueueNumberService** |
| Quản lý và cập nhật trạng thái của bệnh nhân: WAITING (chờ khám), COMPLETED (đã khám) | **PatientDao** |
| Nghiệp vụ bác sĩ | Đăng nhập & Xác thực | **AuthService** (sử dụng **PasswordHasher**) |
| Dashboard Bác sĩ: Hiển thị danh sách bệnh nhân đang chờ theo khoa và thứ tự ưu tiên | **DoctorService** |
| Ghi nhân kết quả: Cho phép bác sĩ nhập chẩn đoán, kết quả khám và cập nhật trạng thái bệnh nhân | **DoctorService** |
| Kê đơn thuốc: Cho phép tạo và lưu đơn thuốc mới vào CSDL | **PrescriptionDao** |
| Cấu hình | Khởi tạo hệ thống: Đảm bảo hệ thống khởi động đúng, đọc cấu hình và khởi tạo CSDL từ **schema.sql** nếu chưa tồn tại | **AppConfig** |

### 1.2. Yêu cầu phi chức năng

* **Kiến trúc hệ thống:** Phải tuân thủ **Kiến trúc nhiều lớp** để đảm bảo **Phân tách trách nhiệm (SoC)**.
* **Khả năng bảo trì:** Mã nguồn phải có tính mô đun hóa cao, dễ dàng thay đổi một lớp mà không ảnh hưởng đến lớp khác.
* **Bảo mật:** Mã hóa mật khẩu (password\_hash) bằng **PasswordHasher** trước khi lưu trữ, hạn chế truy cập trực tiếp và CSDL.
* **Nền tảng:** Hoạt động **Standalone** (độc lập) trên môi trường Java Runtime.
* **Hiệu suất:** Tốc độ xử lý các truy vấn cơ bản (CRUD) và logic nghiệp vụ phải nhanh (do sử dụng CSDL cục bộ).

## Kiến trúc hệ thống

Để đảm bảo hệ thống dễ bảo trì, dễ mở rộng và tuân theo các nguyên lý của Lập trình Hướng Đối Tượng, ứng dụng được thiết kế theo mô hình **Kiến trúc nhiều lớp (Layered Architecture)**. Mỗi lớp đảm nhận một nhóm trách nhiệm riêng biệt, thực hiện nguyên lý **Phân tách trách nhiệm (Separation of Concerns - SoC)**, giúp giảm thiểu sự phụ thuộc lẫn nhau.

| **Lớp** | **Trách nhiệm** | **Luồng giao tiếp** |
| --- | --- | --- |
| **UI** | Trình bày giao diện và nhận tương tác | Gửi yêu cầu xuống lớp Service |
| **Service** | Xử lý Logic nghiệp vụ | Nhận yêu cầu từ UI và gọi các DAO |
| **DAO** | Truy cập và quản lý dữ liệu | Lấy kết nối từ DB Manager và trả về Model cho Service. |
| **Model** | Biểu diễn và đóng gói dữ liệu nghiệp vụ | Được sử dụng bởi tất cả các lớp trên |

Một yêu cầu của người dùng luôn đi theo luồng một chiều: **UI - Service - DAO - DB**. Phản hồi đi theo chiều ngược lại.

## Thiết kế hướng đối tượng

Thiết kế hệ thống tuân thủ nghiêm ngặt các nguyên lý Lập trình Hướng Đối Tượng, đảm bảo tính mô đun hóa, khả năng mở rộng và dễ bảo trì cho mã nguồn.

### 3.1. Tính đóng gói

Các thực thể nghiệp vụ cốt lõi của hệ thống được mô hình hóa thành các lớp **POJO** (Plain Old Java Object) trong gói **com.clinic.triage.model/**. Các lớp này đại diện trực tiếp cho các bảng trong cơ sở dữ liệu và là thành phần cơ bản của hệ thống.

Các lớp Model chính:

* **Patient, Doctor, Department:** Đại diện cho các đối tượng quản lý thông tin chính (Bệnh nhân, Bác sĩ, Khoa).
* **Prescription:** Đối tượng đại diện cho đơn thuốc được tạo ra sau khi khám.
* **SymptomRule:** Đối tượng lưu trữ các quy tắc nghiệp vụ (dùng để tham khảo/hỗ trợ phân khoa).
* **PatientStatus:** Lớp enum định nghĩa trạng thái cố định của bệnh nhân (WAITING, COMPLETED).

Tính đóng gói là nguyên lý nền tảng được áp dụng cho tất cả các lớp Model, nhằm bảo vệ tính toàn vẹn của dữ liệu và kiểm soát việc truy cập vào trạng thái bên trong của đối tượng.

* **Bảo vệ Dữ liệu:** Tất cả các thuộc tính trong các lớp Model đều được khai báo là **private**. Điều này ngăn chặn việc truy cập hoặc thay đổi dữ liệu một cách tùy tiện từ bên ngoài lớp.
* **Kiểm soát Truy cập:** Việc truy cập hoặc thay đổi dữ liệu chỉ được thực hiện thông qua các phương thức **public** là **getters** (để lấy giá trị) và **setters** (để thiết lập giá trị).
* **Lợi ích:** Tính đóng gói giúp mã nguồn dễ bảo trì hơn, vì logic kiểm tra ràng buộc dữ liệu có thể được đặt ngay trong các phương thức setter (mặc dù trong POJO đơn giản, logic này thường tối thiểu).

### 3.2. Tính trừu tượng

Mẫu DAO: Các lớp trong gói **com.clinic.triage.dao/** (ví dụ: **DoctorDao, PatientDao, PrescriptionDao,...**) được thiết kế theo mẫu DAO. Mẫu thiết kế này là một kỹ thuật tiêu chuẩn trong kiến trúc phần mềm hướng đối tượng để tách biệt logic nghiệp vụ khỏi logic duy trì dữ liệu.

Tính trừu tượng là nguyên lý cốt lõi được áp dụng trong thiết kế lớp DAO. Nó giúp ẩn đi sự phức tạp và chi tiết kỹ thuật không cần thiết khỏi các lớp nghiệp vụ.

* **Che giấu chi tiết kỹ thuật:** Các lớp DAO **che giấu** hoàn toàn chi tiết về cách dữ liệu được lưu trữ và truy xuất (như câu lệnh **SQL**, cơ chế kết nối **JDBC**, và việc xử lý ResultSet).
* **Giao diện đơn giản:** Lớp **Service** chỉ tương tác với các phương thức trừu tượng và đơn giản của DAO như:
  + patientDao.create(patient)
  + doctorDao.findByUsername(username)
  + prescriptionDao.findAllByPatientId(id)
* **Lợi ích:**
  + **Giảm sự phụ thuộc:** Lớp Service chỉ phụ thuộc vào giao diện phương thức mà không phụ thuộc vào công nghệ CSDL.
  + **Dễ dàng mở rộng/thay đổi:** Nếu cần chuyển đổi cơ sở dữ liệu từ SQLite sang MySQL, chỉ cần viết lại nội dung bên trong các lớp DAO mà không cần thay đổi bất kỳ logic nào trong lớp Service hoặc UI. Điều này minh họa rõ ràng tính linh hoạt và khả năng bảo trì của hệ thống.

### 3.3 Mối quan hệ giữa các lớp

Mối quan hệ giữa các lớp trong hệ thống chủ yếu được xác định qua nguyên lý **Phân tách trách nhiệm (SoC)**, với sự tương tác được quản lý thông qua **Phối hợp** và **Ủy quyền**.

Mối quan hệ chính trong kiến trúc nhiều lớp (UI - Service - DAO - Model):

* **Service Phối hợp DAO:** Lớp **Service** không tự viết SQL, mà tạo mối quan hệ với một hoặc nhiều đối tượng **DAO** để hoàn thành nghiệp vụ.
* Lớp TriageService ủy quyền logic tính toán số thứ tự cho **QueueNumberService** và logic phân loại cho **SymptomClassifier**. Sự ủy quyền này đảm bảo mỗi lớp chỉ tập trung vào trách nhiệm duy nhất của mình.

Trong dự án, nguyên lý Kế thừa và Đa hình được áp dụng ở mức đô cơ bản hoặc tiềm năng để đảm bảo tính mở rộng trong tương lai:

* **Kế thừa Tĩnh:** Các lớp Model sử dụng các kiểu dữ liệu enum như **PatientStatus** để định nghĩa các trạng thái, nơi các giá trị thừa kế thuộc tính từ kiểu enum cơ sở.
* **Đa hình thông qua Interface:** Các lớp **DAO** về mặt thiết kế đều có thể được coi là các triển khai cụ thể của một Interface chung. Điều này cho phép áp dụng **Đa hình** bằng cách tham chiếu chúng thông qua kiểu Interface chung, giúp thay thế triển khai mà không cần thay đổi logic trong Service.
* **Đa hình trong Xử lý Ngoại lệ:** Các lớp DAO và Service thường bắt các loại ngoại lệ kỹ thuật (SQLException) và ném ra các ngoại lệ nghiệp vụ (IllegalStateException hoặc IllegalArgumentException) đã được trừu tượng hóa. Đây là một dạng **Đa hình** trong xử lý lỗi, cho phép tầng trên (UI) xử lý lỗi một cách thống nhất.

# THIẾT KẾ CƠ SỞ DỮ LIỆU

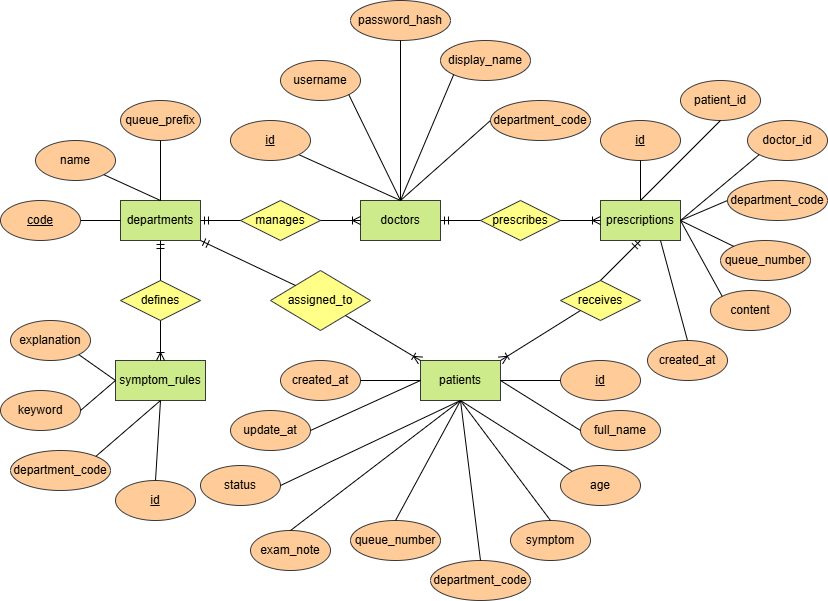
Hệ thống sử dụng **SQLite** (**clinic-triage.db**) làm hệ quản trị cơ sở dữ liệu. CSDL được thiết kế theo **mô hình quan hệ (Relational Database)** nhằm lưu trữ các thực thể nghiệp vụ, hỗ trợ quy trình quản lý thông tin khám bệnh và phân khoa thủ công.

## Mục tiêu thiết kế CSDL

* Lưu trữ và quản lý thông tin bệnh nhân, bác sĩ, và khoa khám.
* Hỗ trợ quy trình **phân khoa thủ công** và **lưu vết** lịch sử khám bệnh.
* Đảm bảo tính toàn vẹn và nhất quán dữ liệu thông qua các ràng buộc khóa chính/khóa ngoại.

## Sơ đồ thực thể liên kết

Thiết kế CSDL tuân thủ các nguyên tắc chuẩn hóa dữ liệu, đảm bảo tính toàn vẹn và nhất quán. Mô hình dữ liệu bao gồm năm bảng chính, được minh họa trực quan bằng **Sơ đồ Thực thể Liên kết (ERD)**.

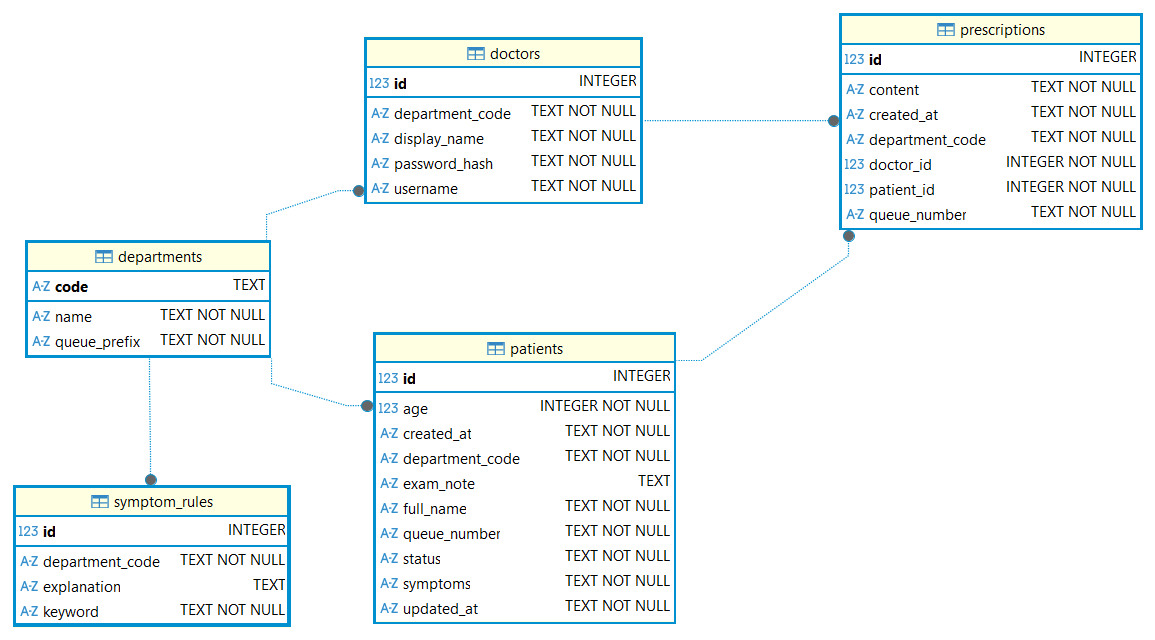


Các mối quan hệ chính:

| **Mối quan hệ** | **Loại** | **Thực thể** | **Mô tả** |
| --- | --- | --- | --- |
| manages | 1 : N | **departments - doctors** | Một khoa quản lý nhiều bác sĩ, mỗi bác sĩ chỉ thuộc một khoa |
| defines | 1 : N | **departments - symptom\_rules** | Một khoa có nhiều quy tắc triệu chứng, mỗi quy tắc chỉ thuộc một khoa |
| assigned\_to | N : 1 | **patients - departments** | Mỗi bệnh nhân được phân vào một khoa, một khoa có thể tiếp nhận nhiều bệnh nhân |
| prescribes | 1 : N | **doctors - prescriptions** | Một bác sĩ có thể kê nhiều đơn thuốc, mỗi đơn thuốc do một bác sĩ lập |
| receives | 1 : N | **patients - prescriptions** | Một bệnh nhân có thể nhận nhiều đơn thuốc, mỗi đơn thuốc thuộc về một bệnh nhân |

## Sơ đồ quan hệ dữ liệu

Trình bày cấu trúc chi tiết của các bảng, làm rõ Khóa chính (**PK**) và Khóa ngoại (**FK**) theo mô hình dữ liệu của dự án.



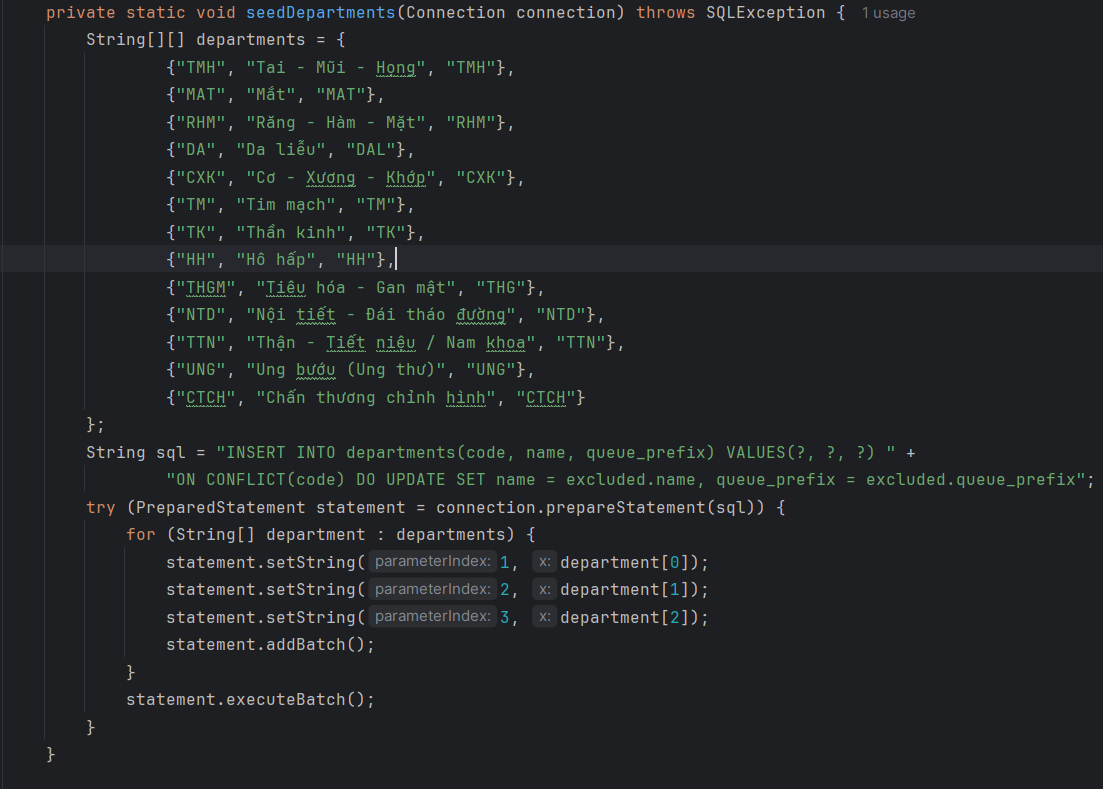
Mô hình dữ liệu này tuân thủ chuẩn hóa cơ bản, đảm bảo tính toàn vẹn dữ liệu bằng cách sử dụng khóa ngoại (department\_code, doctor\_id, patient\_id, …). Thiết kế hỗ trợ tối ưu cho mô hình **Model–DAO–Service** của kiến trúc ứng dụng, cho phép lớp **DAO** thực hiện các truy vấn phức tạp để quản lý hàng đợi (queue\_number) và lưu vết lịch sử khám (prescriptions).

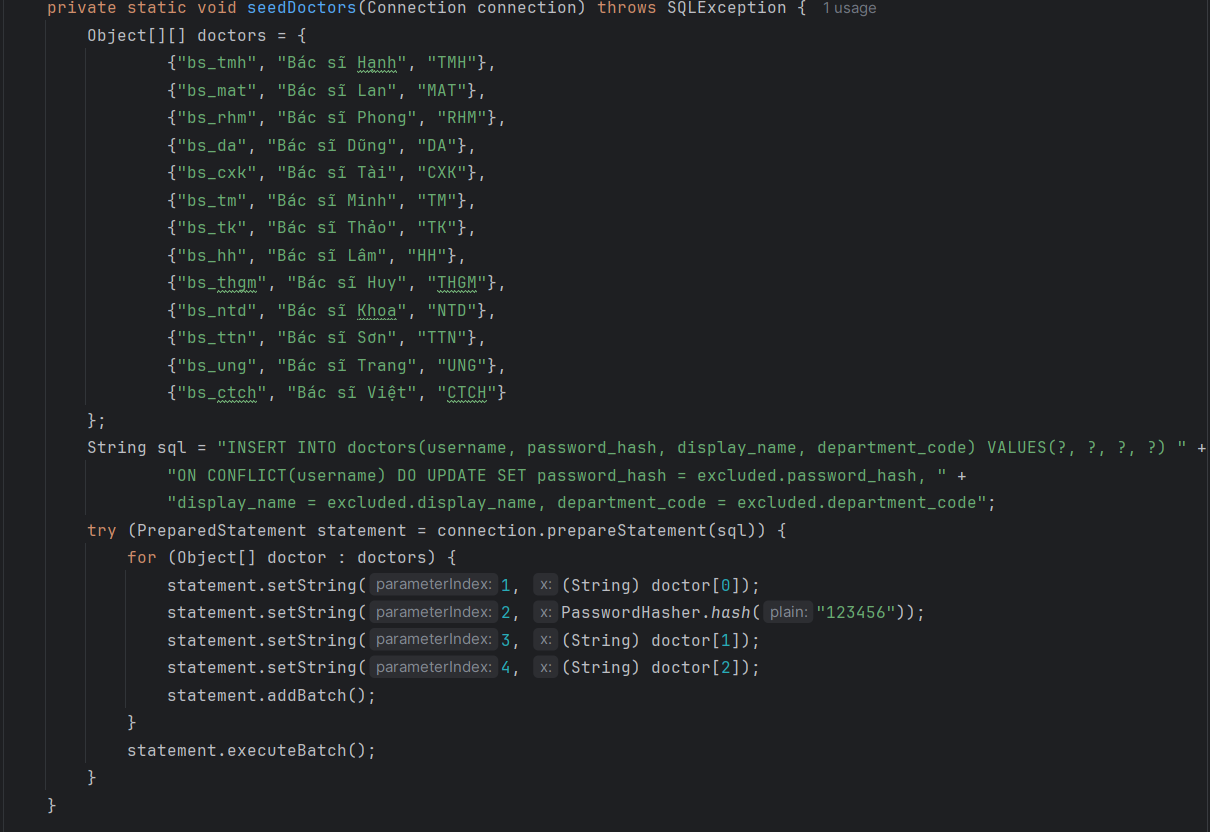
## Quy tắc nghiệp vụ cốt lõi

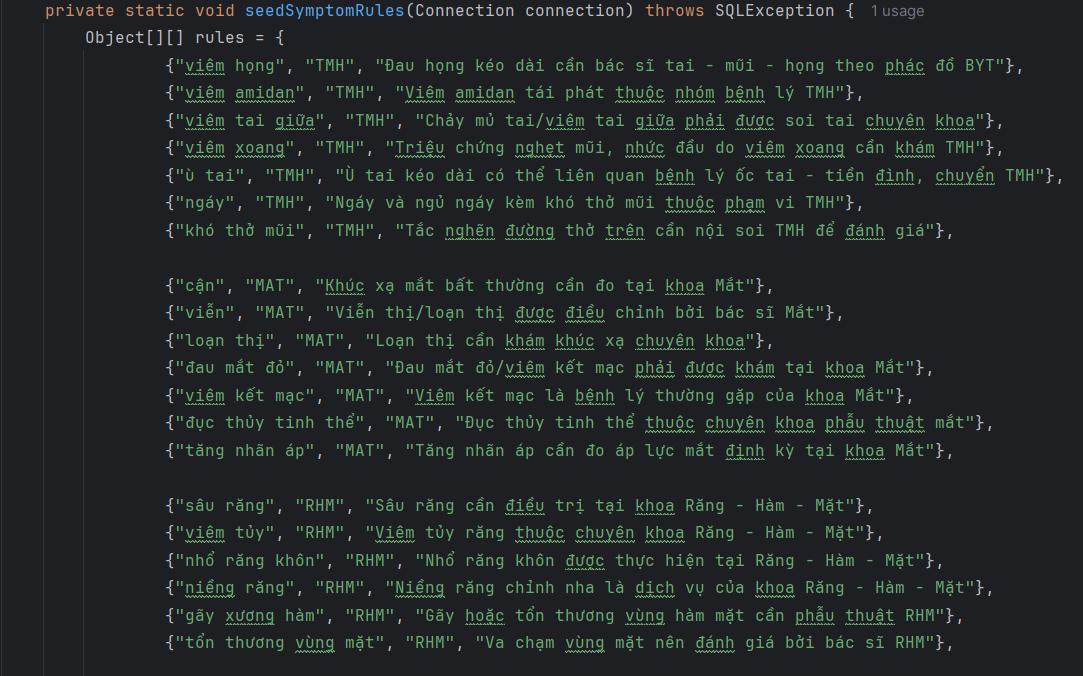
### 4.1. Seed dữ liệu mẫu

Quá trình khởi tạo CSDL: **DatabaseInitializer** sẽ thực hiện chèn **dữ liệu mẫu** cho các bảng departments, doctors, và symptom\_rules.

* **Tính toàn vẹn Dữ liệu:** Quá trình chèn sử dụng **JDBC Batch Insertion** và được bao bọc trong một **Giao dịch (Transaction)** để đảm bảo tất cả dữ liệu mẫu được chèn thành công hoặc không chèn gì cả, duy trì tính nhất quán của CSDL.
* **Dữ liệu Tham khảo:** Các bảng departments, doctors và symptom\_rules được seed với các dữ liệu cố định, đóng vai trò là dữ liệu tham khảo cho toàn bộ hệ thống.



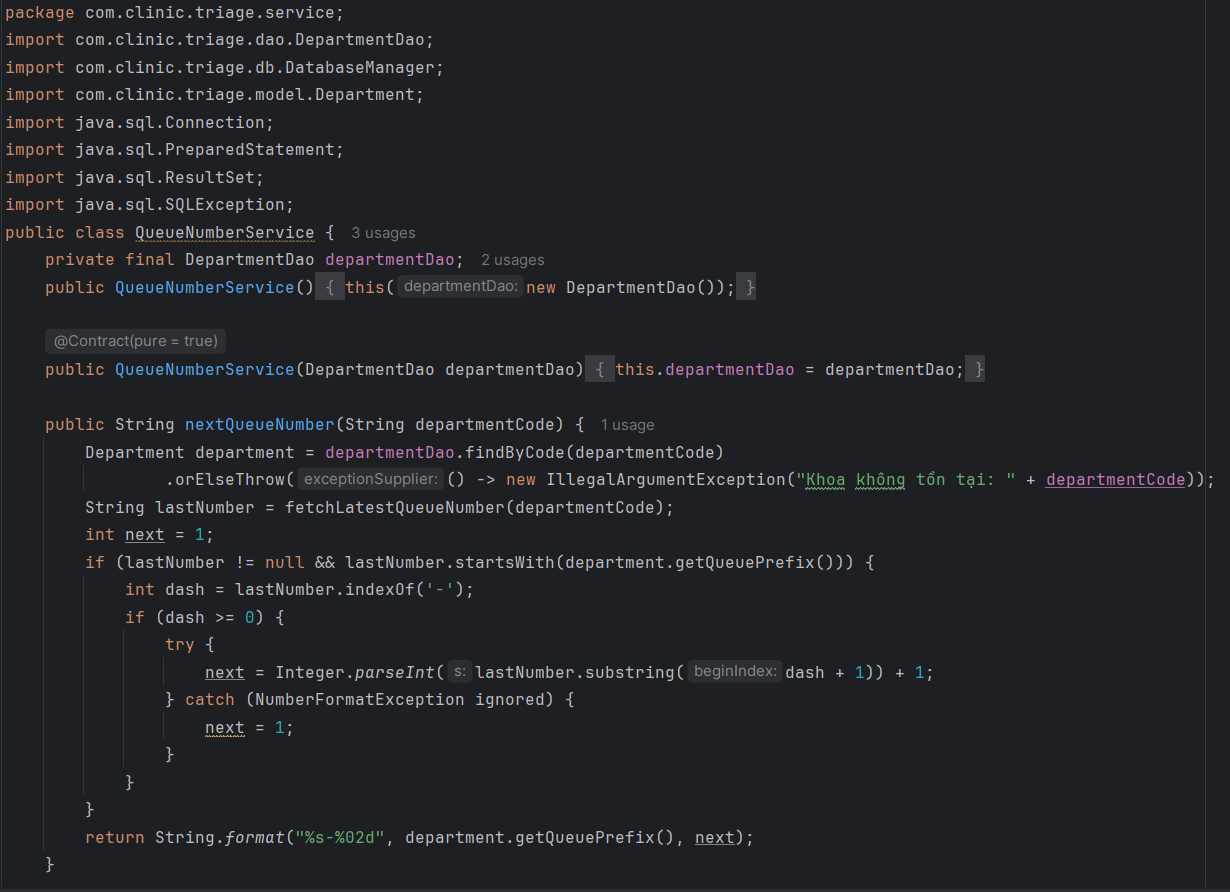


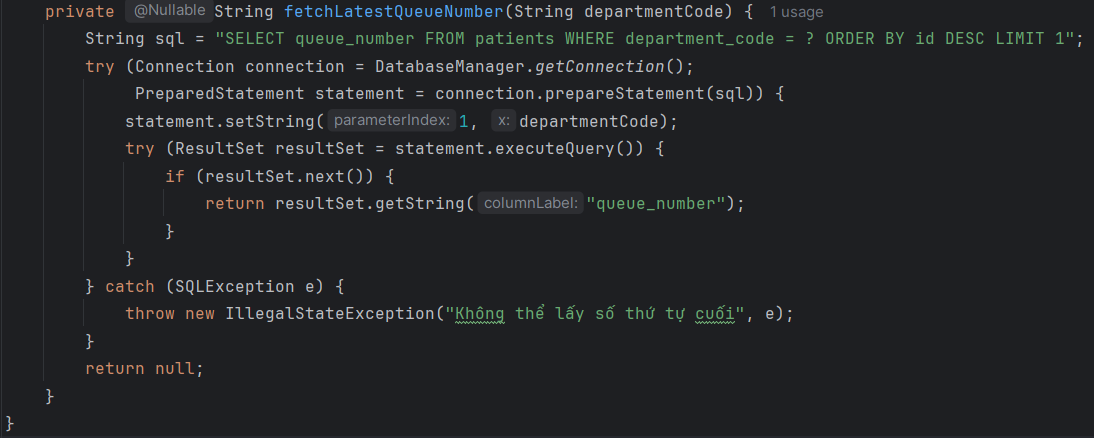


### 4.2. Cơ chế cấp số thứ tự

Dữ liệu trong bảng departments được sử dụng trực tiếp bởi lớp **QueueNumberService** để cấp số thứ tự:

* **Dữ liệu đầu vào:**
  + Tiền tố: Xác định tiền tố cho số thứ tự được cấp.
  + Số thứ tự cuối cùng: Số thứ tự lớn nhất đã được cấp cho khoa tương ứng.
* **Logic:**
  + **QueueNumberService** gọi **DepartmentDao.findByCode()** để lấy thông tin khoa, đảm bảo khoa tồn tại và lấy được queue\_prefix.
  + **QueueNumberService** phải truy vấn **PatientDao** để tìm số thứ tự cao nhất hiện tại của khoa đó, sau đó tăng số đếm lên 1, đảm bảo tính tuần tự và minh bạch của hàng chờ khám.

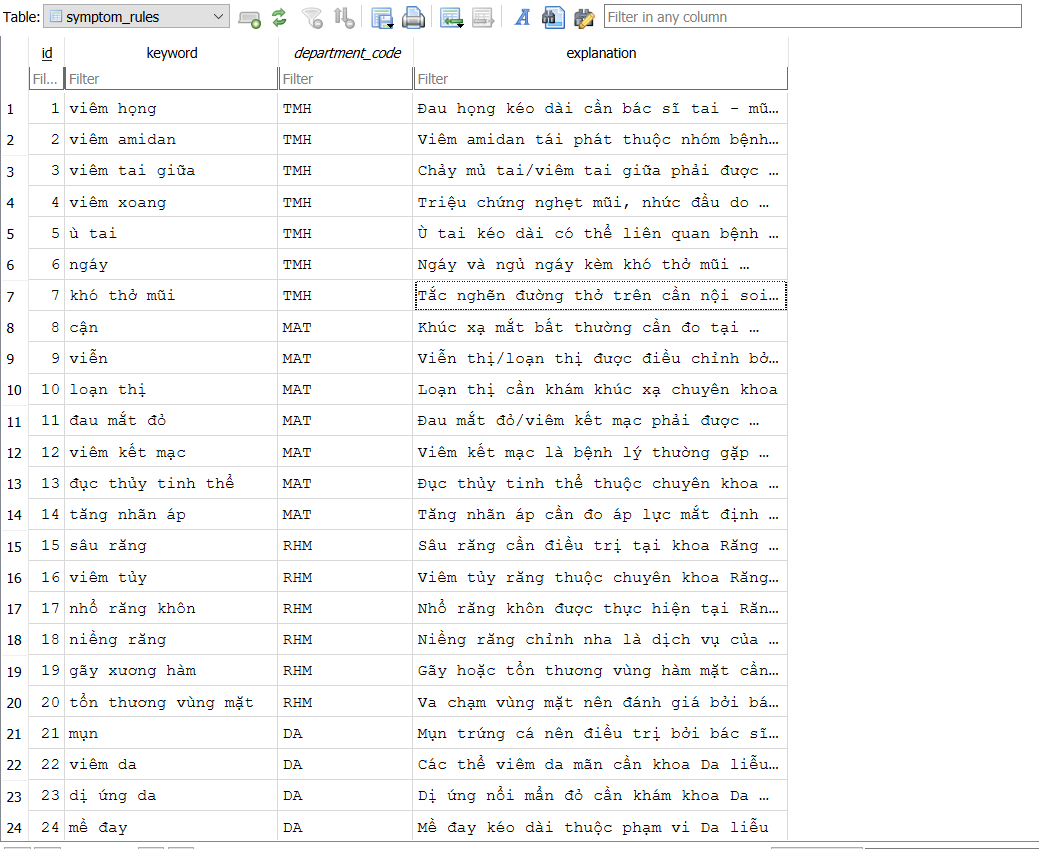




### 4.3. Quy tắc phân khoa

Chức năng **Phân khoa (Triage)** sử dụng dữ liệu được định nghĩa trước trong bảng **symptom\_rules**.

* **Cấu trúc:** Mỗi bản ghi lưu trữ một **keyword** triệu chứng và liên kết nó với một **department\_code** cụ thể với **explanation** để mô tả chi tiết về quy tắc.
* **Logic:** 
  + Lớp **SymptomClassifier** gọi **SymptomRuleDao.findAll()** để tải toàn bộ bộ quy tắc này vào bộ nhớ khi ứng dụng khởi động.
  + Khi nhân viên nhập triệu chứng của bệnh nhân và tham khảo bảng **symptom\_rules** để đưa ra quyết định gán khoa.



### 4.4. Bảo mật tài khoản

Cột **password\_hash** trong bảng doctors lưu trữ giá trị băm của mật khẩu (được tạo bởi PasswordHasher), đảm bảo mật khẩu thô không được lưu.

# TRIỂN KHAI & CHI TIẾT KỸ THUẬT

## Cấu trúc dự án & Khởi tạo

### 1.1. Cấu trúc thư mục cốt lõi

Cấu trúc thư mục logic của dự án được tổ chức theo tiêu chuẩn của Java và mô hình **Kiến trúc nhiều lớp (Layered Architecture)**, đảm bảo sự phân tách trách nhiệm rõ ràng.

* **OOP\_BTL-main/**: Thư mục gốc của dự án.
  + **.idea/**: Thư mục cấu hình IDE (IntelliJ IDEA).
  + **lib/**: Chứa các thư viện ngoài (**sqlite-jdbc**, slf4j) cần thiết cho việc kết nối CSDL và ghi log.
  + **p1/database/**: Nơi chứa file CSDL **clinic-triage.db** và file tạo schema **schema.sql**.
  + **src/main/java/com.clinic.triage/**: Thư mục mã nguồn chính, nơi các gói (packages) được tổ chức theo lớp logic:

| **Gói** | **Lớp logic tương ứng** | **Vai trò chính** |
| --- | --- | --- |
| **model/** | Model (Domain) | Chứa các đối tượng dữ liệu (POJO) như Patient, Doctor,... |
| **dao/** | Data Access Object (DAO) | Chứa các lớp truy cập dữ liệu, tách biệt *data access layer* (truy cập dữ liệu) và *business logic layer* (nghiệp vụ). |
| **service/** | Business Logic | Chứa các lớp xử lý nghiệp vụ phức tạp (TriageService, AuthService) |
| **db/** | Database Management | Chứa các lớp quản lý CSDL (DatabaseManager, DatabaseInitializer) |
| **config/** | Configuration | Chứa các lớp cấu hình hệ thống (Appconfig) |
| **util/** | Utility | Chứa các lớp tiện ích dùng chung (PasswordHasher, UiUtils) |
| **doctor.ui/**  **patient.ui/** | Presentation (UI) | Chứa các form giao diện người dùng (\*Frame, \*Dialog) |

### 1.2. Module Cấu hình & Khởi tạo

Các module này thuộc gói config/ và db/, đảm bảo hệ thống có thể đọc các thiết lập cần thiết và chuẩn bị cơ sở dữ liệu trước khi chạy logic nghiệp vụ.

| **Module** | **Vị trí** | **Trách nhiệm** |
| --- | --- | --- |
| **AppConfig.java** | config/ | Đọc các tham số cấu hình tĩnh từ file **app.properties** và cung cấp các cấu hình này cho toàn bộ ứng dụng |
| **DatabaseManager.java** | db/ | Quản lý kết nối CSDL **SQLite**. Lớp này được triển khai theo mẫu **Singleton** để đảm bảo kiểm soát duy nhất một đối tượng kết nối CSDL trong suốt thời gian chạy |
| **DatabaseInitializer.java** | db/ | Đảm bảo tính toàn vẹn và khả năng khởi động lần đầu. Lớp này kiểm tra sự tồn tại của file **clinic-triage.db** và thực thi **schema.sql** để tạo các bảng nếu chưa có |

Quy trình Khởi tạo hệ thống: Khi ứng dụng được chạy từ điểm chính (Main.java hoặc DoctorLoginFrame), trình tự khởi tạo diễn ra như sau:

* **Cấu hình:** Đọc các thiết lập ban đầu qua **AppConfig**.
* **Khởi tạo DB:** Lớp **DatabaseInitializer** được gọi để kiểm tra và tạo CSDL (nếu cần) thông qua **DatabaseManager**.
* **Sẵn sàng:** Sau khi CSDL sẵn sàng, các lớp **DAO** có thể sử dụng **DatabaseManager** để lấy đối tượng kết nối và bắt đầu thao tác dữ liệu

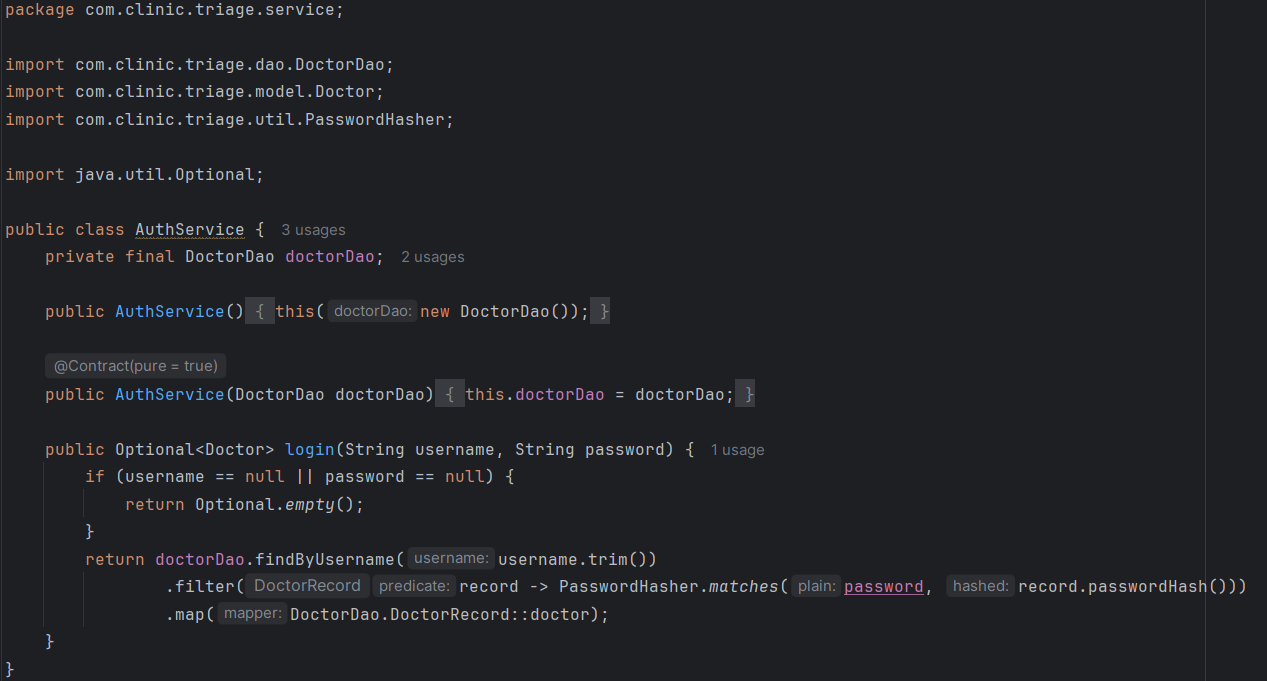
## Lớp dịch vụ & Logic nghiệp vụ

Lớp **Service** là trung tâm điều khiển của hệ thống. Các lớp trong gói service/ chịu trách nhiệm xử lý các quy trình phức tạp, kiểm tra ràng buộc nghiệp vụ, và điều phối các thao tác giữa giao diện người dùng (UI) và lớp truy cập dữ liệu (DAO).

| Lớp | Nhiệm vụ chính | Nguyên lý OOP & Liên kết |
| --- | --- | --- |
| AuthService.java | **Xử lý đăng nhập và xác thực** tài khoản | Gọi **DoctorDao** để lấy thông tin. Sử dụng **PasswordHasher** để so sánh mật khẩu đã mã hóa |
| TriageService.java | **Quản lý quy trình Phân khoa và Đăng ký** bệnh nhân | Phối hợp nhận department\_id được gán thủ công, gọi **QueueNumberService**, và lưu hồ sơ qua **PatientDao** |
| QueueNumberService.java | **Cấp số thứ tự** và quản lý hàng đợi theo khoa | Đảm bảo tính tuần tự và nhất quán khi cấp số mới (dựa trên department\_id và ngày hiện tại) |
| DoctorService.java | **Hỗ trợ nghiệp vụ chuyên môn** cho bác sĩ | Xử lý việc lấy danh sách bệnh nhân chờ khám, và logic lưu kết quả khám/chẩn đoán |
| SymptomClassifier | **Xác định bệnh nhân nên được phân vào khoa nào** dựa trên mô tả triệu chứng (description) | Liên kết với các DAO và model (**Department, SymptomRule**) để thực hiện nhiệm vụ phân loại triệu chứng. |

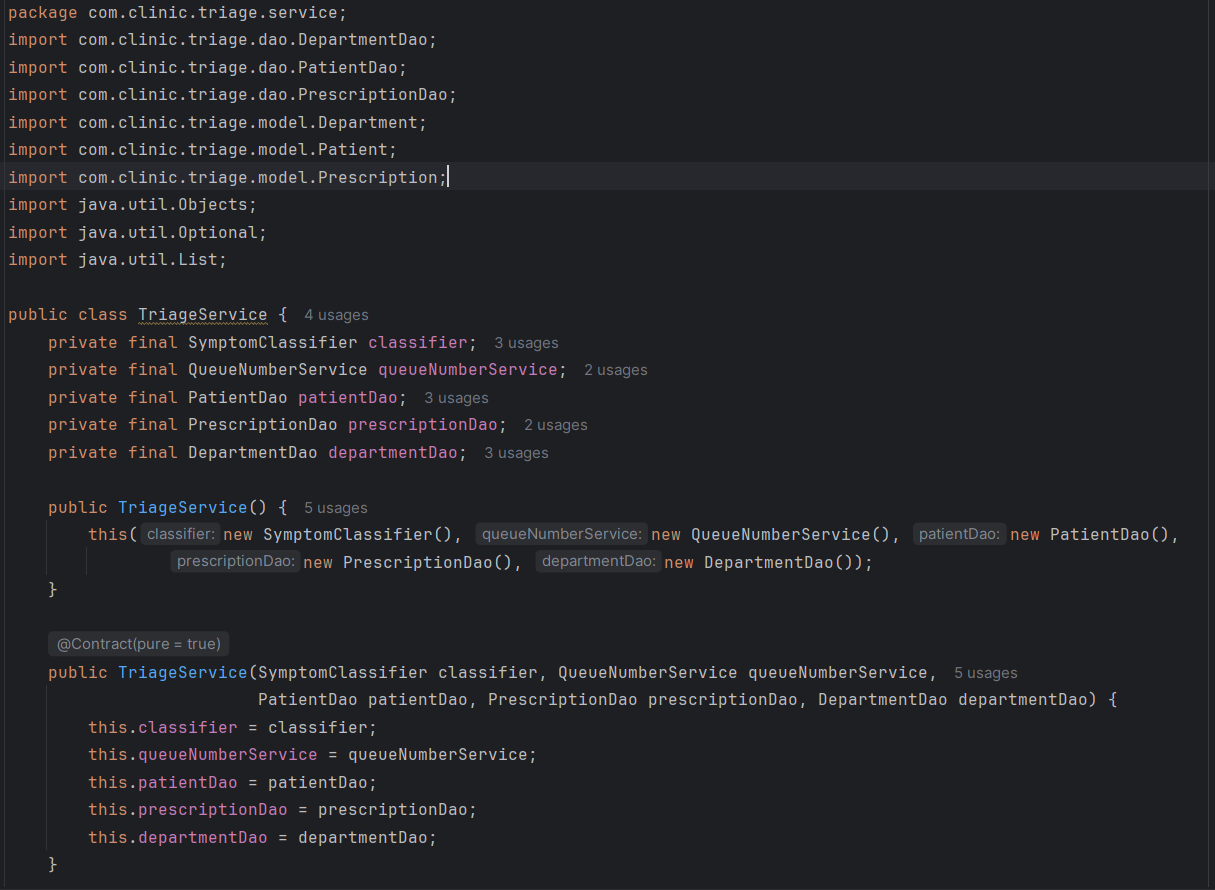
### 2.1. Quy trình Xác thực Người dùng (AuthService)

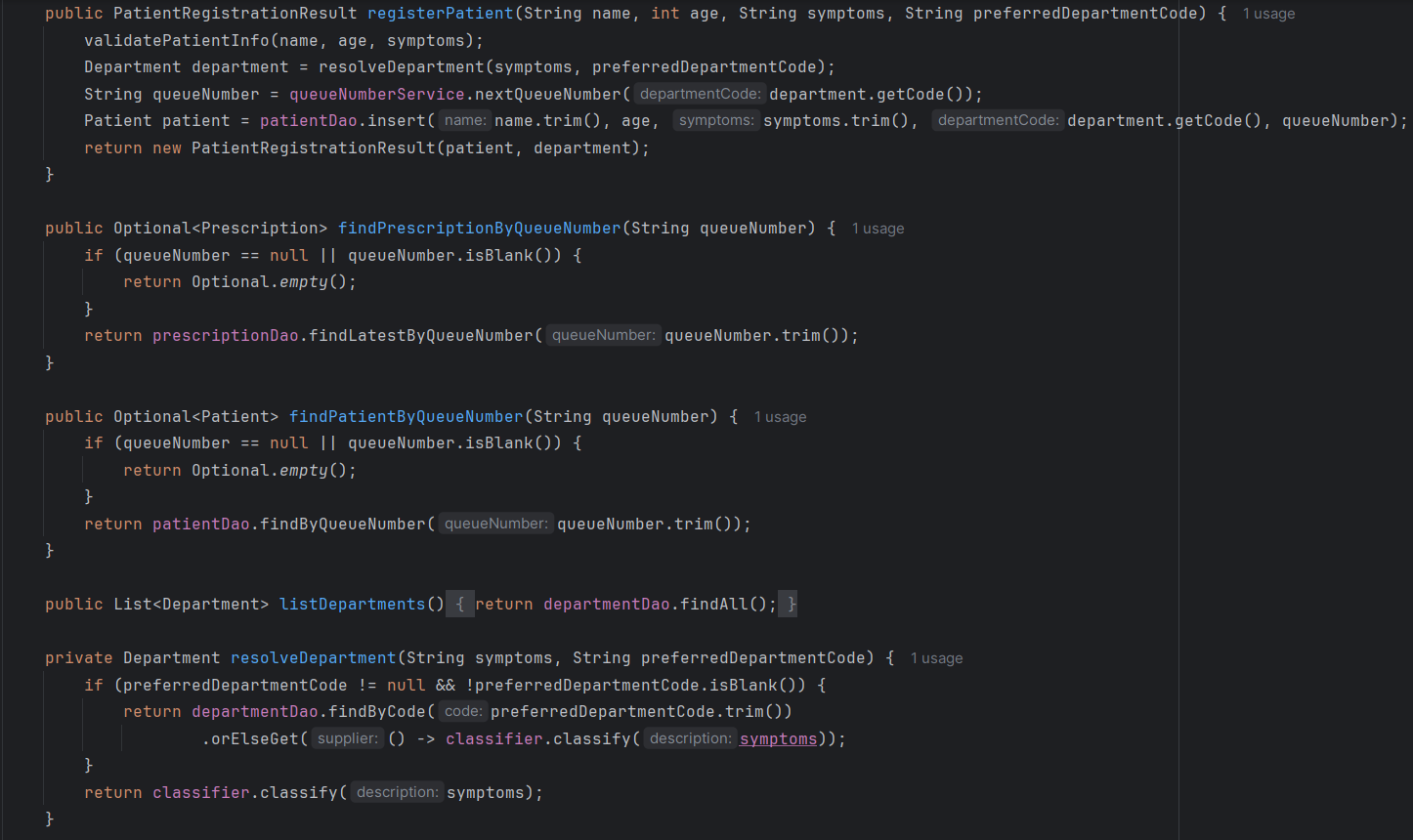
* **Logic:** Xử lý yêu cầu đăng nhập bằng cách nhận **username** và **password** thô từ giao diện (DoctorLoginFrame).
* **Mục tiêu:** Đảm bảo **bảo mật cơ bản** bằng cách không lưu trữ và không xử lý mật khẩu thô trong quá trình xác thực, đồng thời xác định vai trò người dùng (Doctor) trước khi cấp quyền truy cập vào dashboard.
* **Thao tác chi tiết:** 
  + Thu thập Input (UI): Giao diện DoctorLoginFrame thu thập username và password thô.
  + Truy vấn Hồ sơ (Service - DAO):
    - AuthService gọi phương thức trừu tượng: DoctorDao.findByUsername(username).
    - DoctorDao truy vấn CSDL để lấy bản ghi bác sĩ, bao gồm cả cột **password\_hash** đã được mã hóa.
  + Xác thực mật khẩu (Service - Utility):
    - AuthService sử dụng lớp tiện ích: PasswordHasher.matches(password, record.passwordHash()).
    - Hàm này thực hiện logic so sánh giá trị băm (hash) đã lưu với giá trị băm của mật khẩu thô vừa nhập, **mà không bao giờ giải mã** mật khẩu.
  + Kết quả & Truyền dữ liệu:
    - Nếu mật khẩu khớp: AuthService trả về đối tượng Doctor (dưới dạng Optional<Doctor>) cho lớp UI.
    - Nếu sai: Trả về Optional.empty() (hoặc thông báo lỗi nếu không tìm thấy username).

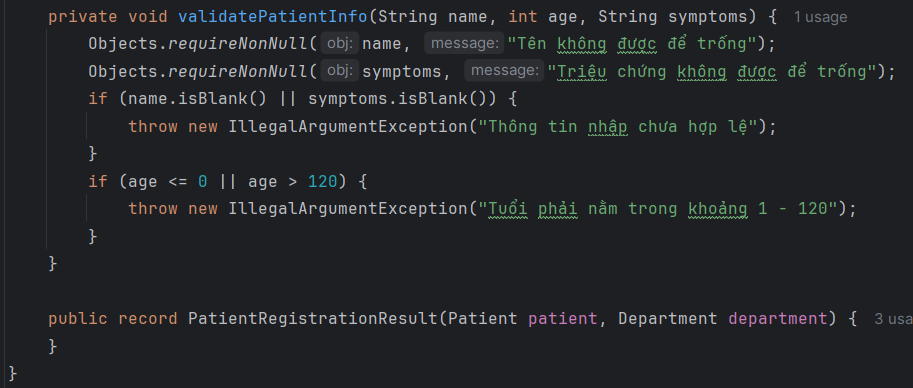


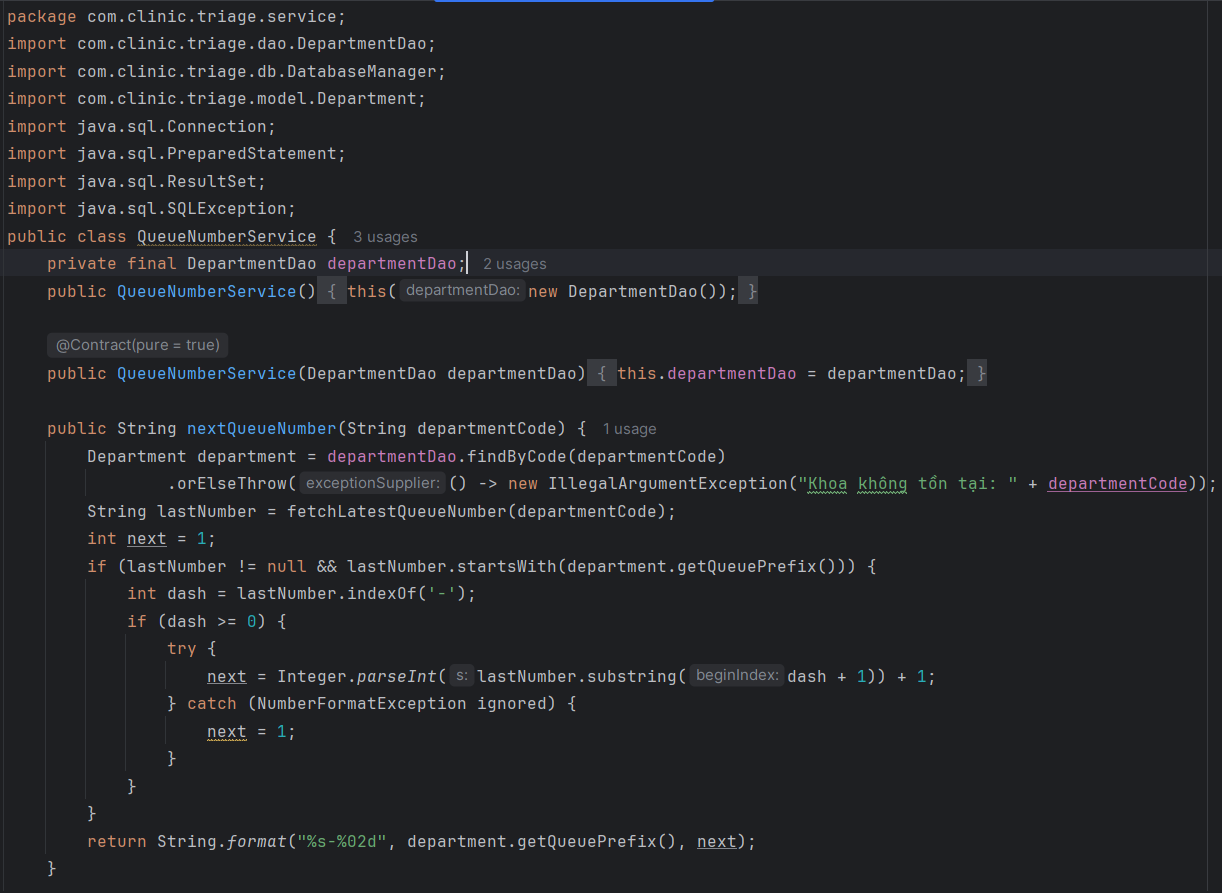
### 2.2. Xử lý Phân khoa và Hàng đợi (TriageService & QueueNumberService)

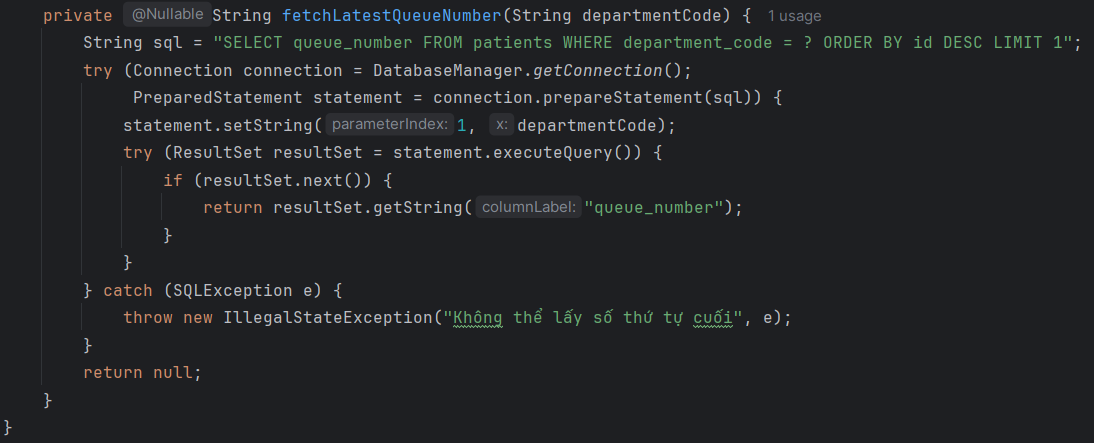
* **Logic:** Đăng ký hồ sơ bệnh nhân mới, xác định khoa khám và cấp số thứ tự tương ứng.
* **Mục tiêu:** Chuẩn hóa dữ liệu đầu vào, đảm bảo tính nhất quán của số thứ tự, và đưa bệnh nhân vào trạng thái chờ khám (WAITING) trong hệ thống.
* Quy trình này thể hiện sự phối hợp chặt chẽ giữa ba lớp: **TriageService** (Điều phối), **QueueNumberService** (Tính toán), và **PatientDao** (Lưu trữ).
* **Thao tác chi tiết:**
  + Nhận dữ liệu & Kiểm tra: **TriageService** nhận dữ liệu từ UI. Sử dụng phương thức **validatePatientInfo** để kiểm tra tính hợp lệ của dữ liệu (ví dụ: tuổi phải lớn hơn 0).
  + Xác định khoa: Xác định khoa khám. Hệ thống ưu tiên mã khoa (preferredDepartmentCode) do nhân viên tiếp nhận chọn. Nếu không có, nó gọi **SymptomClassifier** để sử dụng kết quả gợi ý.
  + Cấp số thứ tự: **TriageService** ủy quyền việc tính toán số thứ tự cho **QueueNumberService**. Lớp này tính toán số tiếp theo, đảm bảo số mới là tuần tự và duy nhất cho khoa đó.
  + Lưu trữ hồ sơ: **TriageService** thiết lập trạng thái status = WAITING cho hồ sơ. Sau đó, nó gọi **PatientDao** để lưu hồ sơ bệnh nhân đã hoàn chỉnh vào CSDL, cùng với mã khoa và số thứ tự đã xác định.





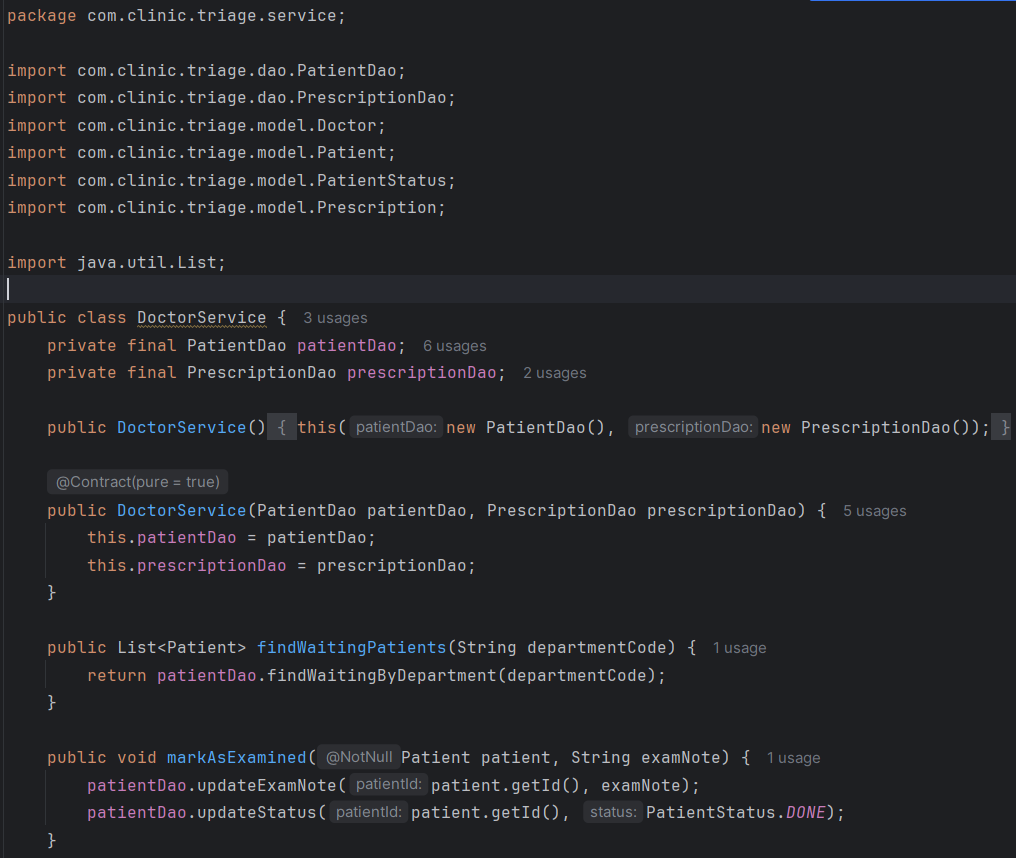


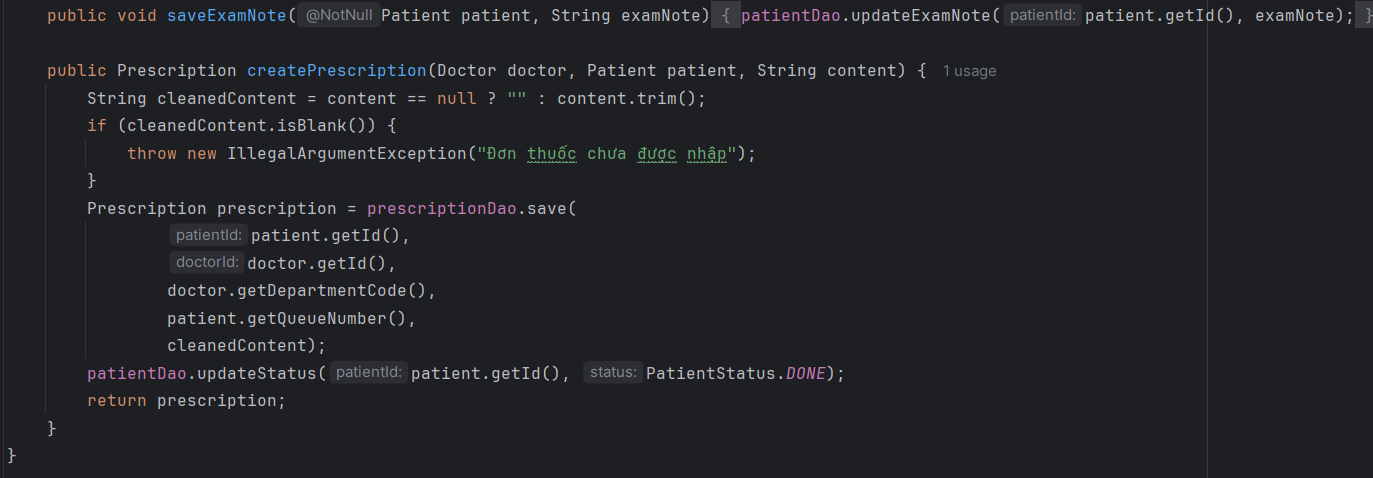




### 2.3. Quản lý Hàng đợi và Khám bệnh (DoctorService):

* **Logic:** Hỗ trợ bác sĩ theo dõi bệnh nhân và ghi nhận kết quả khám.
* **Mục tiêu:** **Chuẩn hóa quy trình làm việc** của bác sĩ trên hệ thống, đảm bảo dữ liệu khám bệnh (**ghi chú, đơn thuốc**) được lưu trữ đầy đủ và **cập nhật trạng thái** bệnh nhân một cách chính xác, giải phóng hàng đợi.
* **Thao tác chi tiết:** 
  + Xem hàng đợi bệnh nhân:
    - Xác định Khoa: Phương thức **findWaitingPatients(departmentCode)** nhận mã khoa làm việc của bác sĩ (thường được lấy từ đối tượng Doctor đã đăng nhập).
    - Truy vấn Hàng đợi: **DoctorService** ủy quyền truy vấn cho PatientDao. Truy vấn tìm danh sách bệnh nhân có **department\_id** khớp với khoa của bác sĩ và có trạng thái **status = WAITING**. Danh sách trả về được sắp xếp theo **queue\_number** để đảm bảo đúng thứ tự khám.
  + Kết thúc khám và kê đơn: Khi bác sĩ hoàn thành chẩn đoán, **DoctorService** điều phối hai thao tác quan trọng:
    - Lưu Đơn thuốc:Phương thức **createPrescription(...)** nhận thông tin đơn thuốc. **DoctorService** gọi **PrescriptionDao.save(prescription)** để lưu đơn thuốc vào CSDL.
    - Giải phóng Hàng đợi:Sau khi đơn thuốc được lưu thành công, **DoctorService** gọi **PatientDao.updateStatus(patientId, COMPLETED)** (hoặc DONE). Việc cập nhật trạng thái này chính thức giải phóng bệnh nhân khỏi hàng đợi của khoa, hoàn tất quy trình khám bệnh.





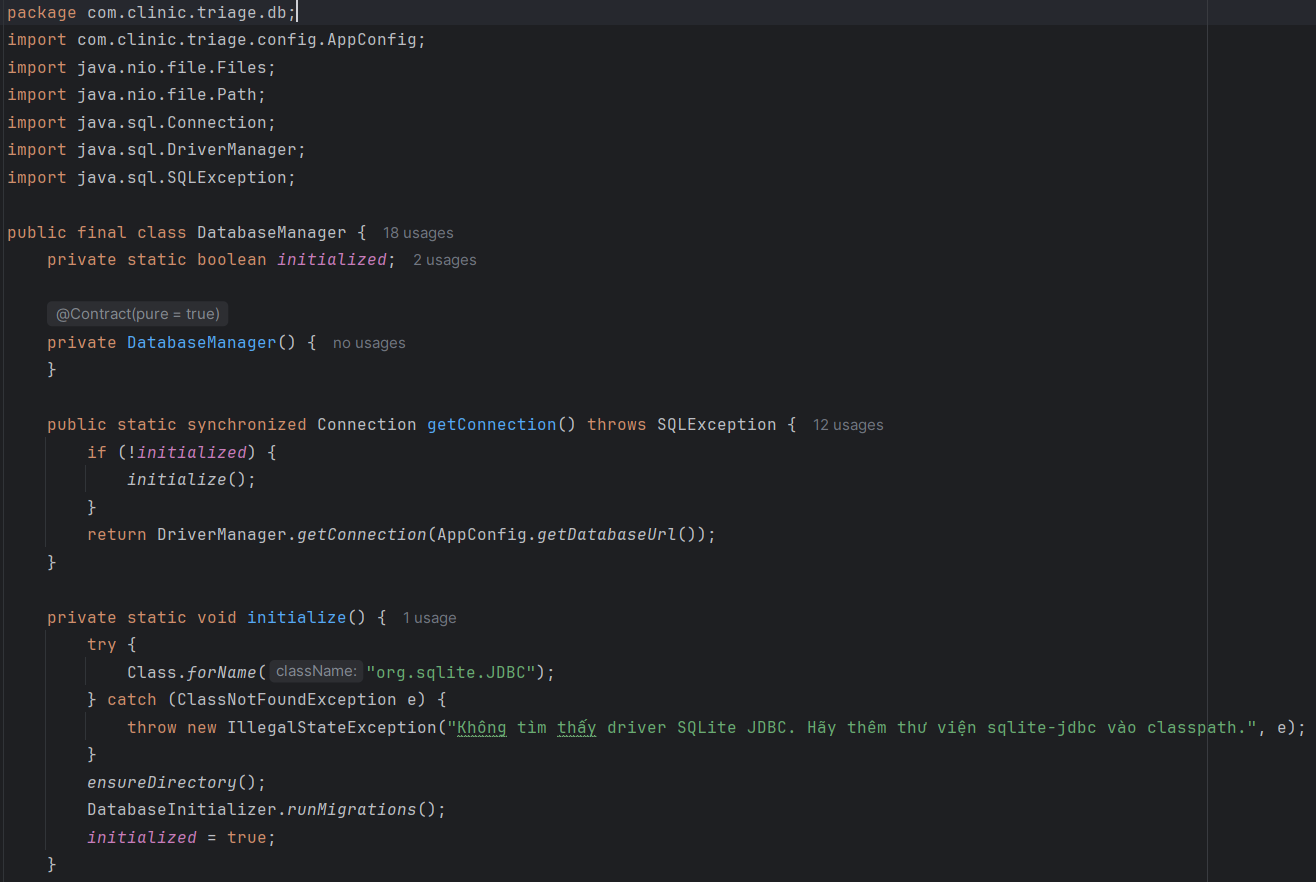
## Lớp truy cập dữ liệu

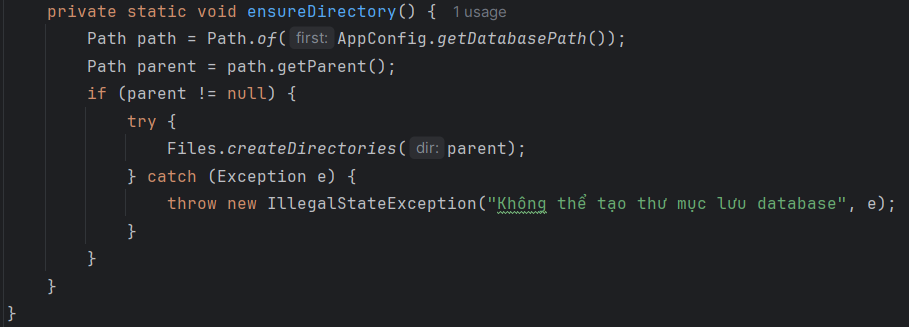
Lớp Truy cập dữ liệu là nền tảng của hệ thống, chịu trách nhiệm kết nối trực tiếp với **SQLite** và quản lý các giao dịch dữ liệu. Lớp này thể hiện nguyên lý **Tính trừu tượng** bằng cách che giấu các chi tiết kỹ thuật của CSDL.

### 3.1. Các lớp quản lý CSDL

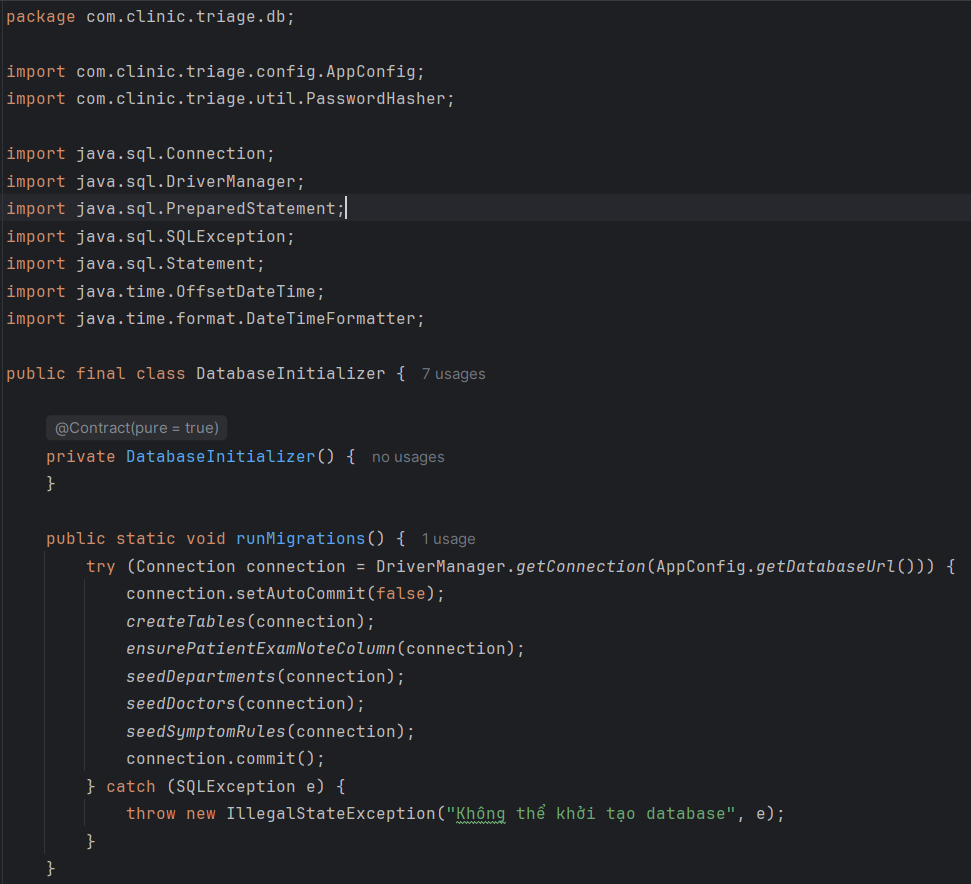
Các lớp trong gói này đóng vai trò nền tảng, đảm bảo môi trường CSDL được thiết lập chính xác và quản lý kết nối hiệu quả cho các lớp **DAO** và **Service**.

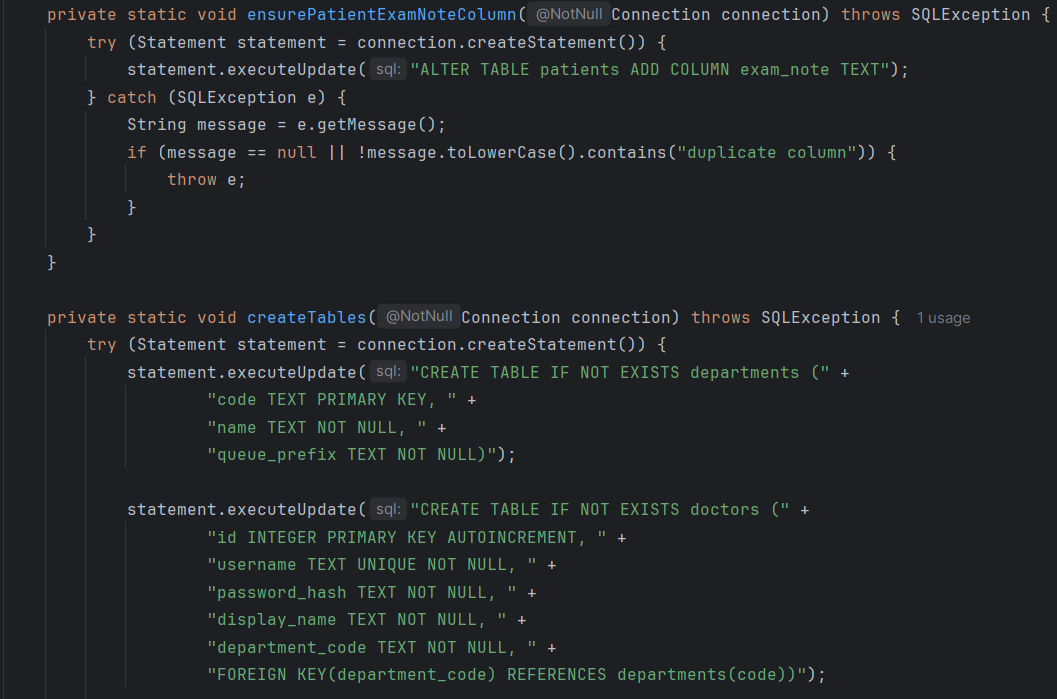
* DatabaseManager.java: **Quản lý kết nối -** Chịu trách nhiệm khởi tạo và quản lý đối tượng kết nối JDBC tới file **clinic-triage.db**.
  + Thực hiện việc tải driver **SQLite JDBC** và kiểm tra/tạo thư mục lưu trữ CSDL.
  + Cung cấp phương thức tĩnh **getConnection()** để mở một kết nối mới tới file DB. Phương thức này được đồng bộ hóa (synchronized) và kiểm tra cờ khởi tạo (initialized) để đảm bảo quá trình thiết lập ban đầu chỉ chạy một lần duy nhất.
  + Ủy quyền (delegate) việc tạo bảng và chèn dữ liệu mẫu cho **DatabaseInitializer** trong quá trình khởi tạo.

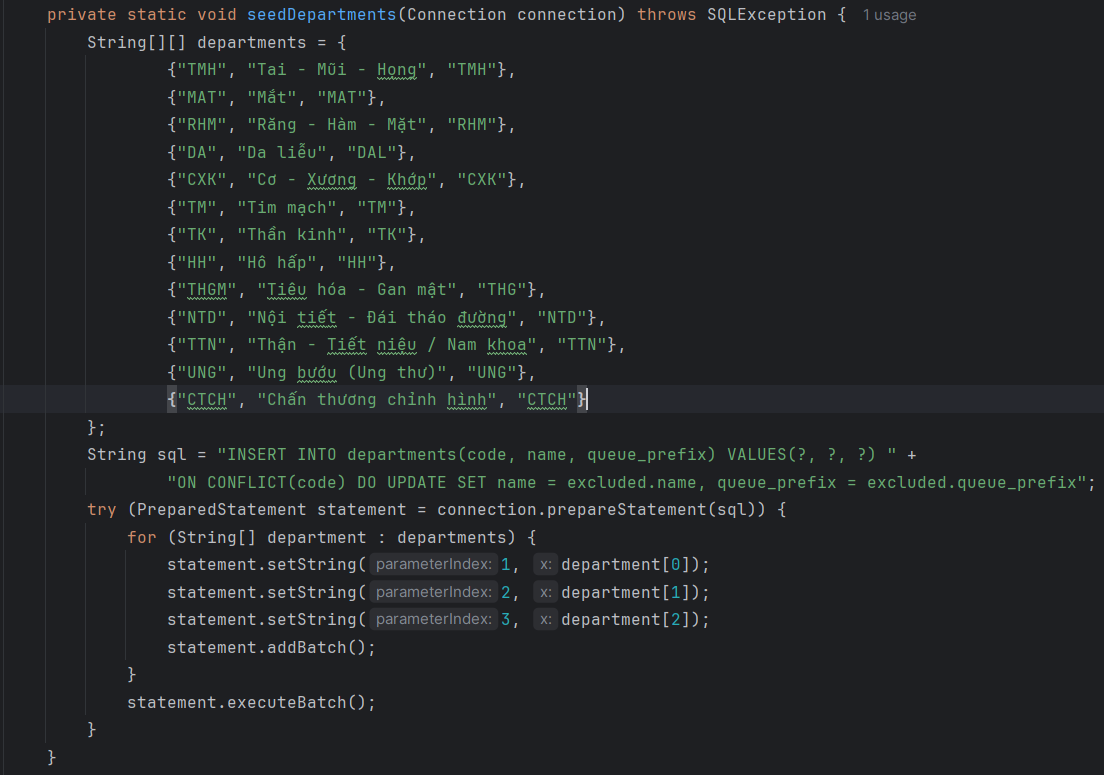


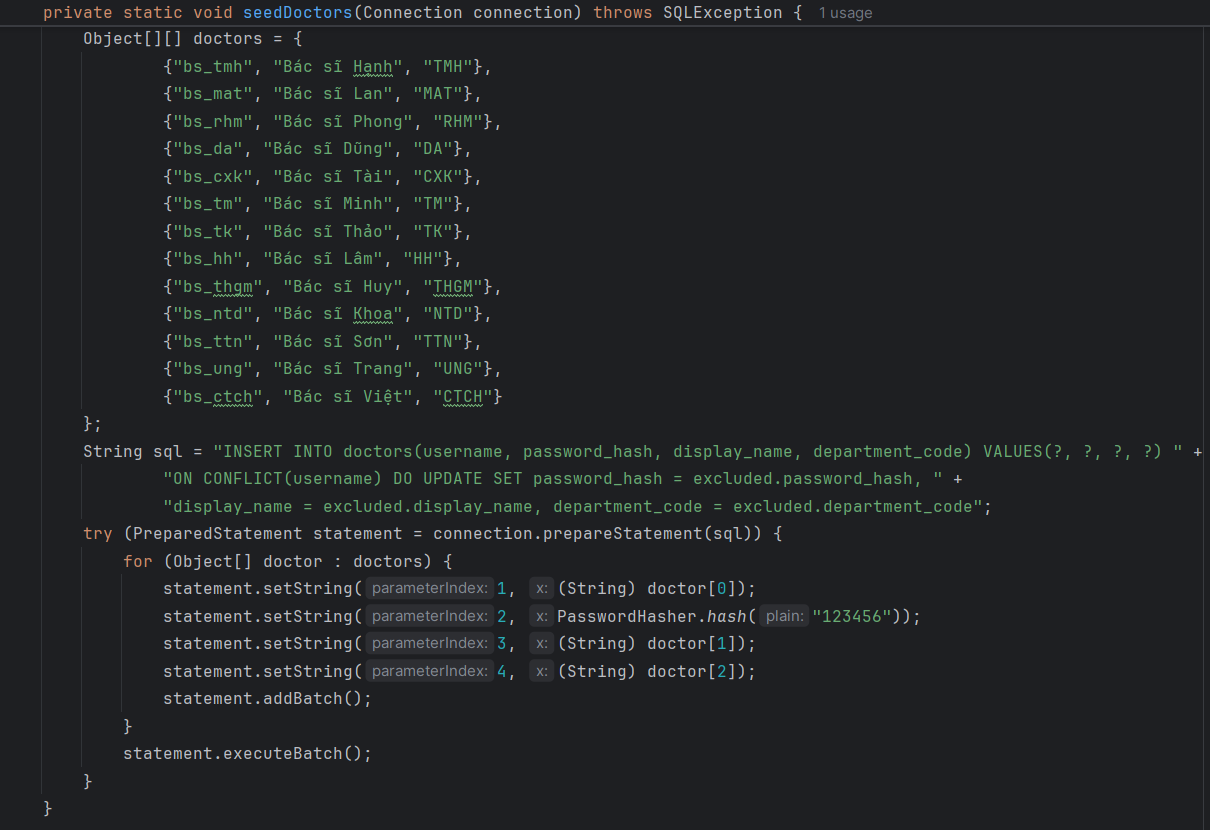


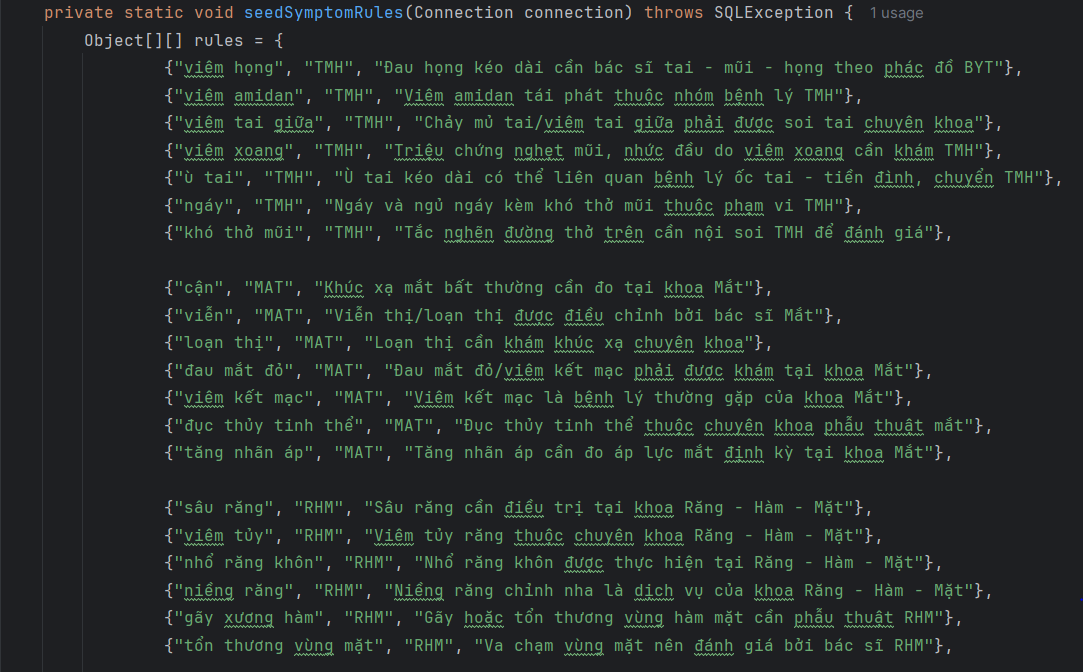
* DatabaseInitializer.java: **Khởi tạo CSDL -** Đảm bảo tính toàn vẹn và khả năng khởi động lần đầu (First-run capability).
  + Chứa phương thức **runMigrations()** để thực thi giao dịch tạo cấu trúc bảng (createTables()) và chèn dữ liệu mẫu (seedDoctors(), seedDepartments()).
  + Đảm bảo toàn bộ quá trình tạo DB diễn ra trong một **giao dịch (transaction)** duy nhất (sử dụng connection.setAutoCommit(false) và connection.commit()), giữ cho CSDL ở trạng thái nhất quán.











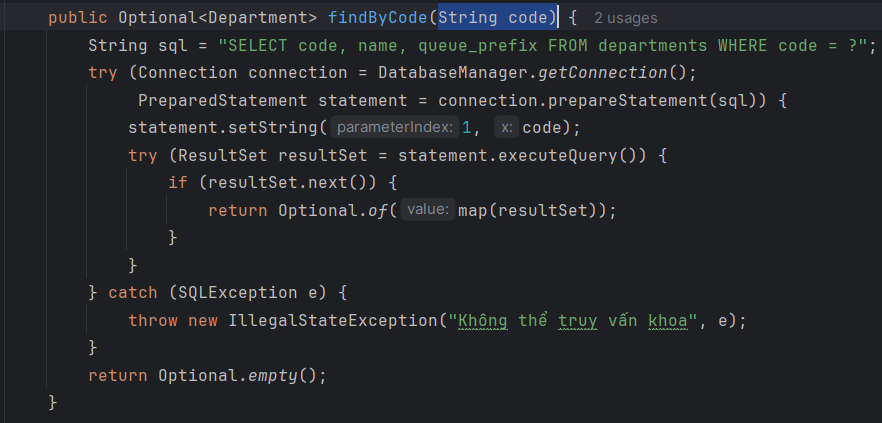
* Các lớp này hoạt động ở tầng dưới cùng, cung cấp các dịch vụ nền tảng cho lớp **DAO**. Mọi lớp DAO đều gọi **DatabaseManager.getConnection()** để thiết lập giao tiếp với CSDL, nhưng không cần quan tâm đến logic khởi tạo hay cách thức quản lý kết nối. Sự tách biệt này là một ví dụ rõ ràng của việc áp dụng nguyên lý **Phân tách trách nhiệm (SoC)**.

### 3.2. Các lớp truy cập dữ liệu

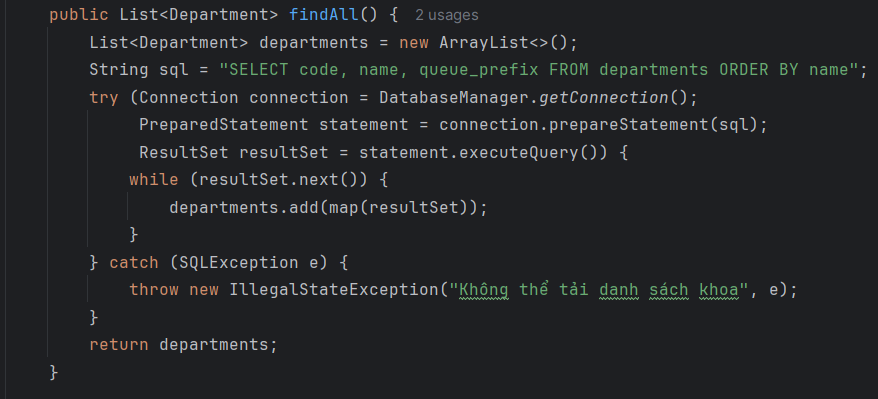
Các lớp trong gói dao/ là nơi hiện thực hóa nguyên lý **Tính trừu tượng**, chịu trách nhiệm truy vấn, lưu trữ và ánh xạ dữ liệu giữa **SQLite** và các đối tượng Model. Các lớp này đều sử dụng kết nối an toàn từ **DatabaseManager**.

| Lớp DAO | Thực thể Model | Chức năng cốt lõi |
| --- | --- | --- |
| PatientDao.java | Patient | **CRUD** hồ sơ bệnh nhân, cập nhật trạng thái (updateStatus), lưu ghi chú khám (updateExamNote). Hỗ trợ truy vấn danh sách bệnh nhân chờ (findWaitingByDepartment) |
| DoctorDao.java | Doctor | Truy vấn bác sĩ theo tên đăng nhập (findByUsername). Cung cấp password\_hash cho AuthService để xác thực |
| PrescriptionDao.java | Prescription | Lưu đơn thuốc mới (save) và truy vấn đơn thuốc gần nhất theo số thứ tự (findLatestByQueueNumber) |
| DepartmentDao.java | Department | Truy vấn theo mã khoa (findByCode) và lấy toàn bộ danh sách khoa (findAll) |
| SymptomRuleDao.java | SymptomRule | Truy vấn tất cả các quy tắc nghiệp vụ (findAll) để phục vụ logic gợi ý phân khoa (nếu có) hoặc tham khảo |

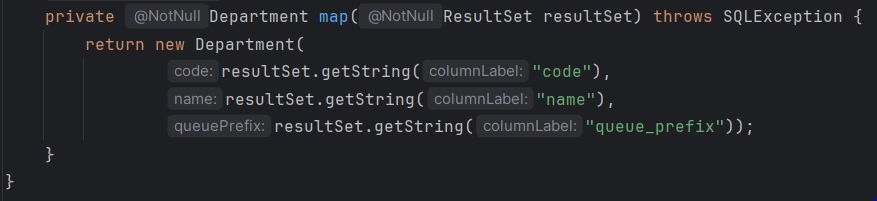
* Lớp **DepartmentDao:** Chịu trách nhiệm duy nhất là ánh xạ (map) dữ liệu từ bảng departments sang đối tượng Department và ngược lại.
  + Phương thức **findByCode(String code):** Tìm theo mã khoa- Truy vấn khoa dựa trên code. Trả về **Optional<Department>** để xử lý trường hợp không tìm thấy dữ liệu.



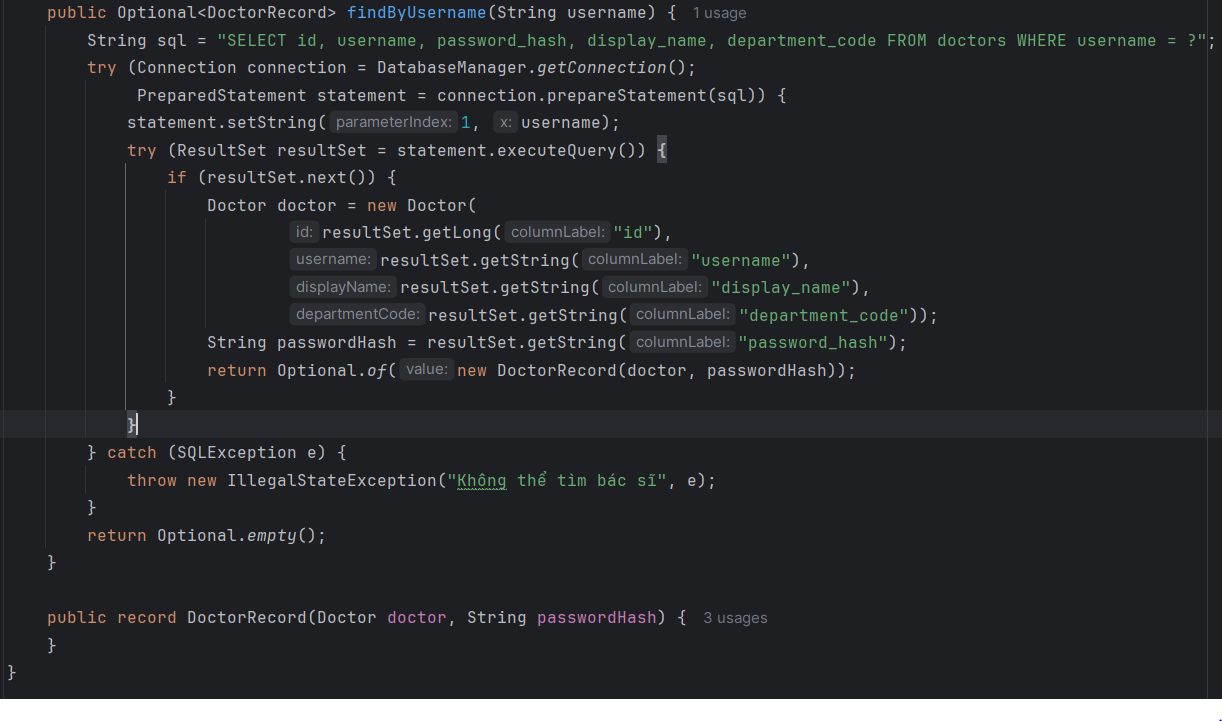
* + Phương thức **findAll():** Truy vấn toàn bộ danh sách khoa, thường được dùng để hiển thị trên giao diện hoặc trong các dropdown chọn khoa.



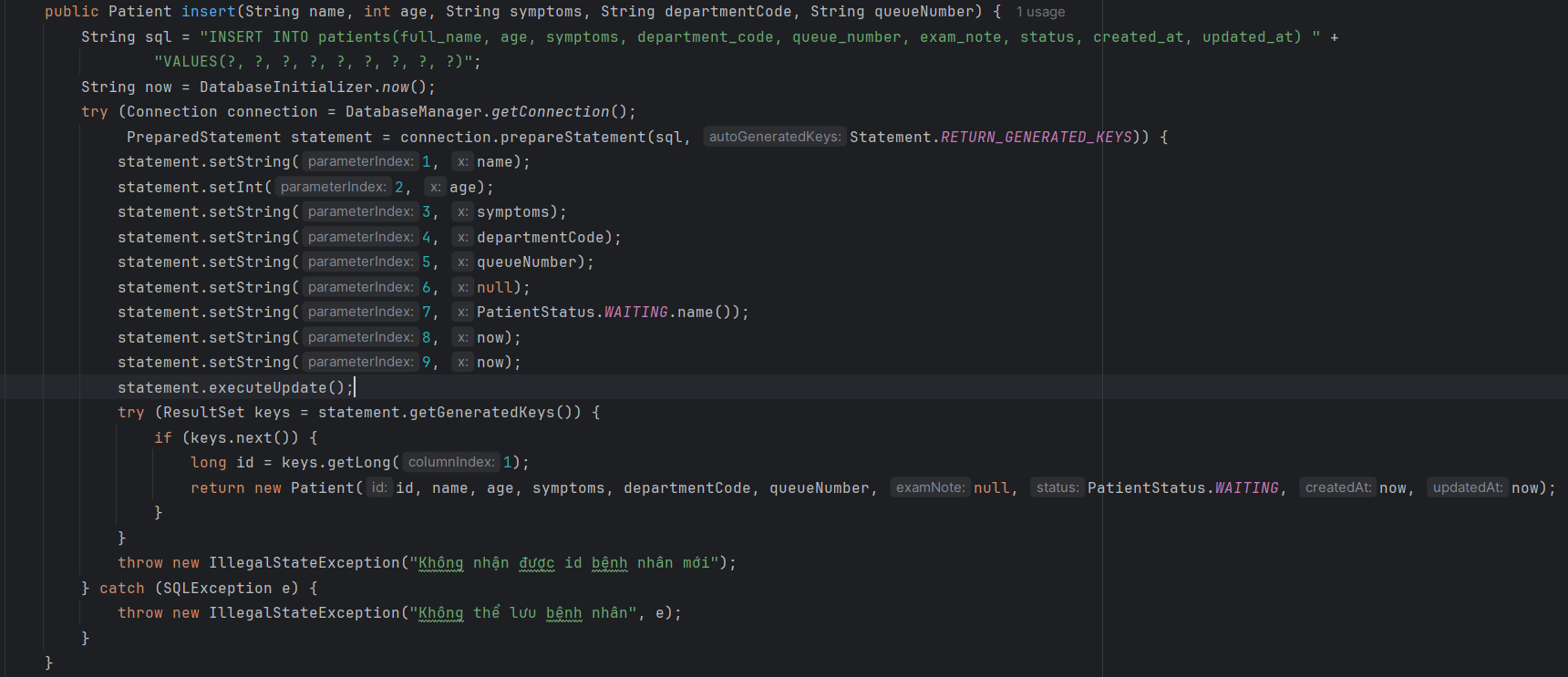
* + Phương thức **private Department map(ResultSet resultSet)** chuyển đổi các trường dữ liệu từ đối tượng ResultSet sang đối tượng **Department** trong tầng Model. Việc này đảm bảo lớp DAO trả về dữ liệu dưới dạng đối tượng đã được Đóng gói (Encapsulated) cho tầng Service.



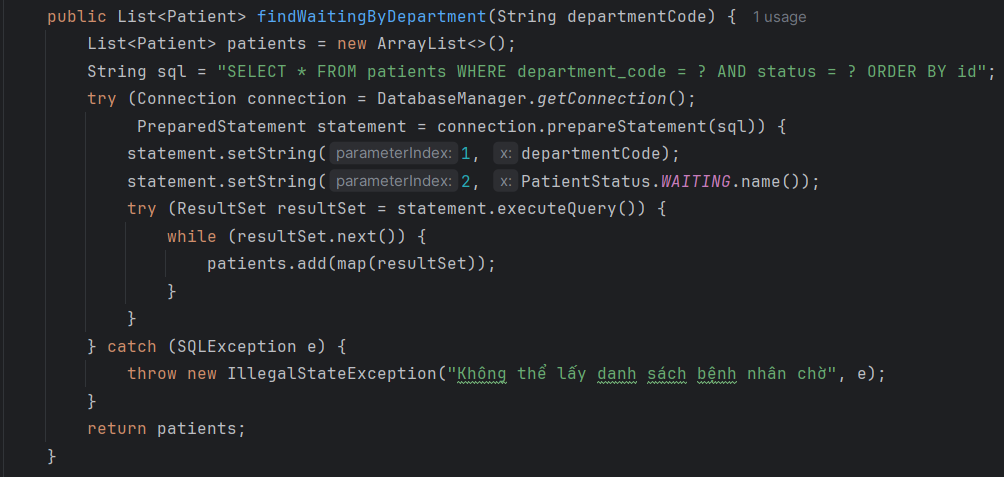
* Lớp **DoctorDao:** Chịu trách nhiệm duy nhất là ánh xạ dữ liệu từ bảng doctors sang đối tượng Doctor và thực hiện các truy vấn tìm kiếm cơ bản.
  + Phương thức **findByUsername(String username):** Tìm theo tên - Truy vấn bác sĩ dựa trên username. Trả về **Optional<DoctorRecord>** để xử lý trường hợp không tìm thấy dữ liệu.



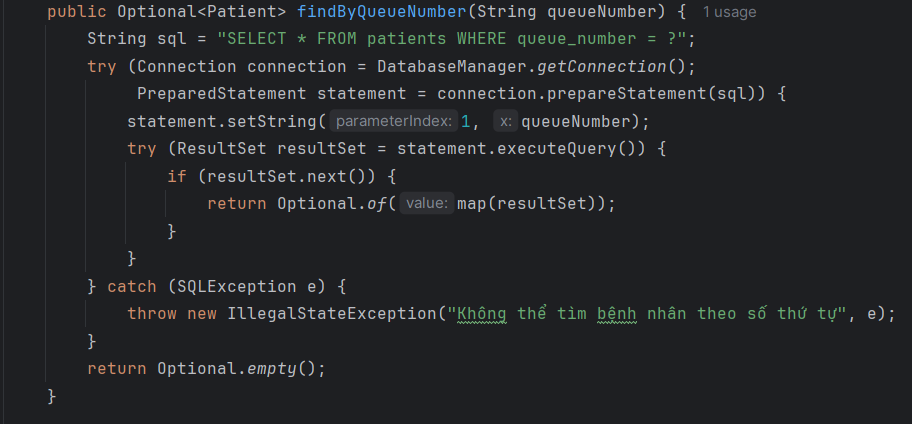
* Lớp **PatientDAO**: Quản lý các thao tác **CRUD**(Creat, Read, Update, Delete) cho bảng patients. Đặc biệt, nó xử lý các cập nhật liên quan đến trạng thái khám và ghi chú của bác sĩ.
  + Phương thức **insert(String name, int age, String symptoms, String departmentCode, String queueNumber):** Thêm hồ sơ bệnh nhân - Thực hiện INSERT dữ liệu nhận từ TriageService. Sử dụng **Statement.RETURN\_GENERATED\_KEYS** để lấy lại id mới tạo của bệnh nhân và trả về đối tượng Patient hoàn chỉnh.



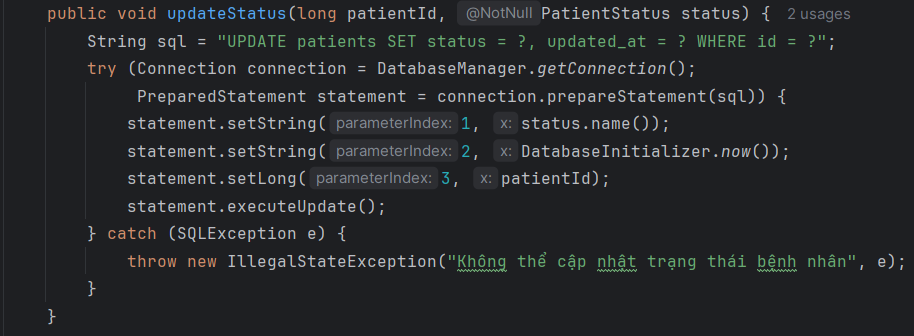
* + Phương thức **findWaitingByDepartment(code):** Truy vấn hàng đợi - Chức năng cốt lõi cho DoctorService. Lấy danh sách bệnh nhân có status = 'WAITING' VÀ department\_code khớp với mã khoa truyền vào.



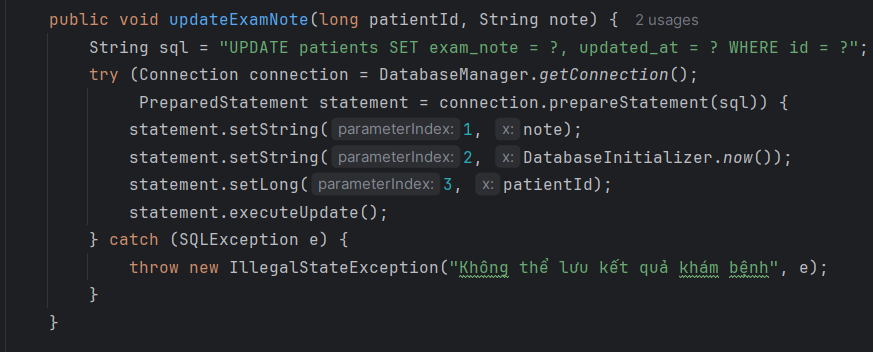
* + Phương thức **findByQueueNumber(queueNumber):** Tìm theo số thứ tự - Truy vấn bệnh nhân dựa trên queue\_number (dùng cho các chức năng tra cứu). Trả về **Optional<Patient>**.



* + Phương thức **updateStatus(id, status):** Cập nhật trạng thái - Chức năng quan trọng cho TriageService (gán WAITING) và DoctorService (gán DONE/COMPLETED). Chỉ cập nhật cột status và updated\_at.



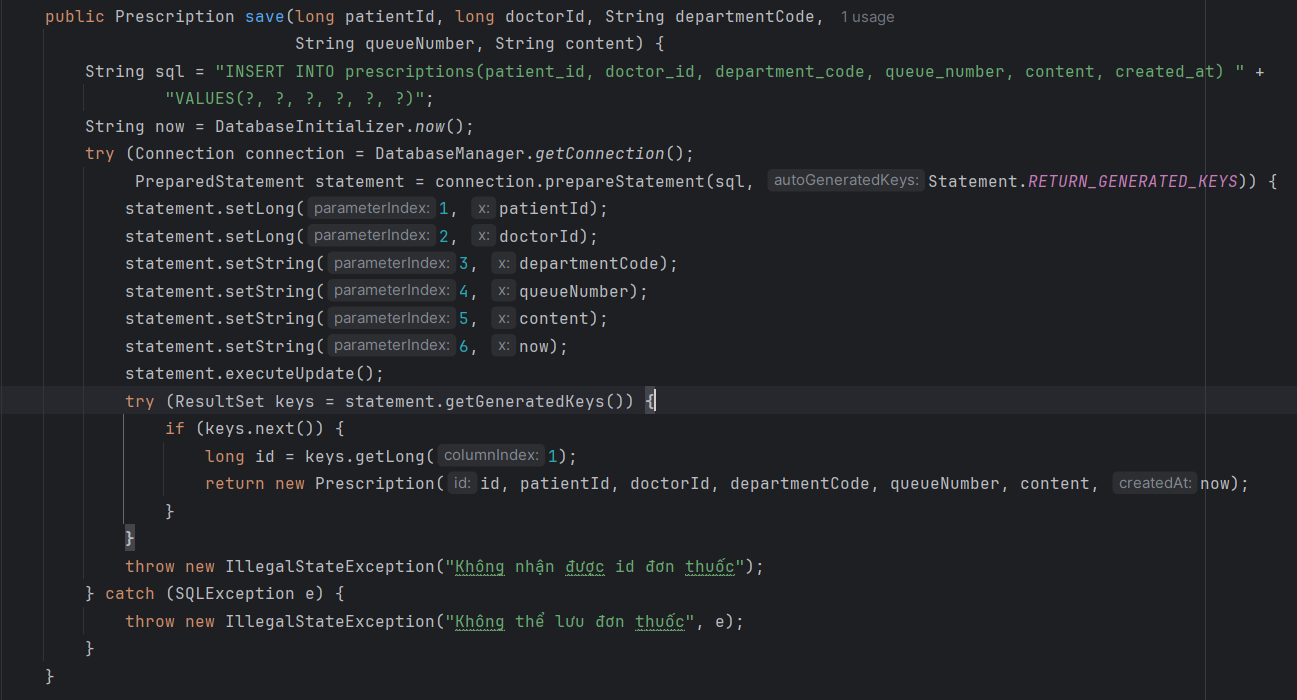
* + Phương thức **updateExamNote(id, note):** Lưu ghi chú khám - Lưu ghi chú/kết quả khám của bác sĩ vào cột exam\_note.



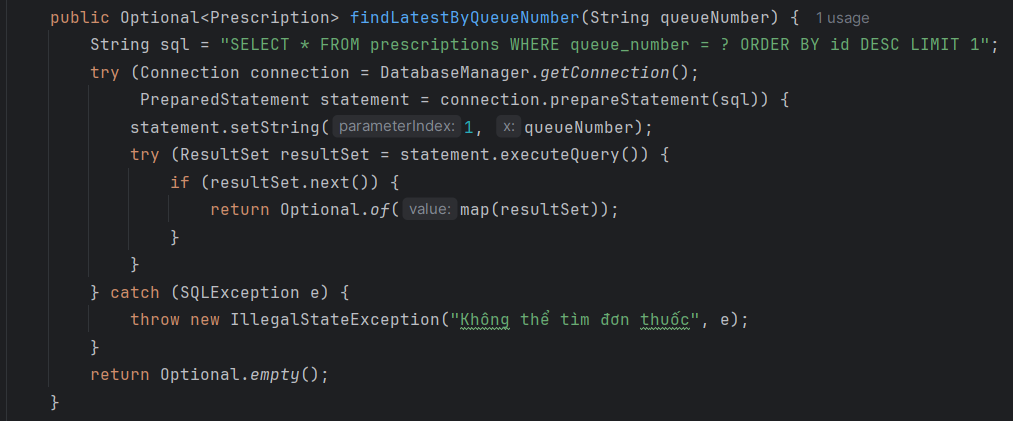
* + Phương thức **private Patient map(ResultSet)** chịu trách nhiệm chuyển đổi một bản ghi từ ResultSet thành một đối tượng **Patient** hoàn chỉnh. Đặc biệt, nó chuyển đổi chuỗi trạng thái từ DB thành giá trị enum **PatientStatus** (PatientStatus.valueOf(resultSet.getString("status"))), đảm bảo tính nhất quán của Model.



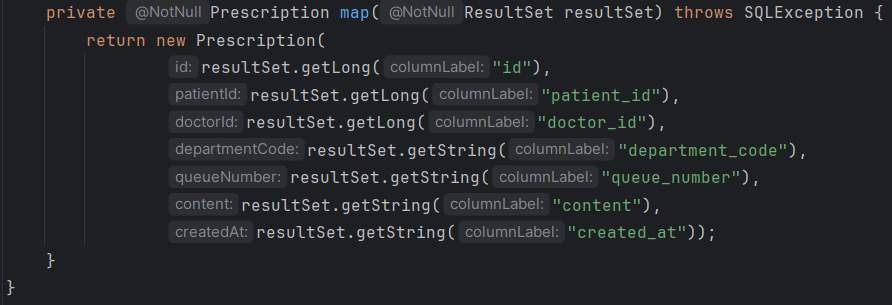
* Lớp **PrescriptionDao:** Chịu trách nhiệm duy nhất là ánh xạ dữ liệu từ bảng prescriptions sang đối tượng Prescription và thực hiện các thao tác lưu trữ/truy vấn.
  + Phương thức **save(long patientId, long doctorId, String departmentCode, String queueNumber, String content):** Lưu đơn thuốc mới - Chèn (INSERT) đơn thuốc mới vào bảng prescriptions. Nó lưu trữ các khóa ngoại (patient\_id, doctor\_id) cùng với thông tin quản lý quy trình (department\_code, queue\_number). Sử dụng Statement.RETURN\_GENERATED\_KEYS để lấy ID tự động tăng mới nhất và trả về đối tượng Prescription hoàn chỉnh.



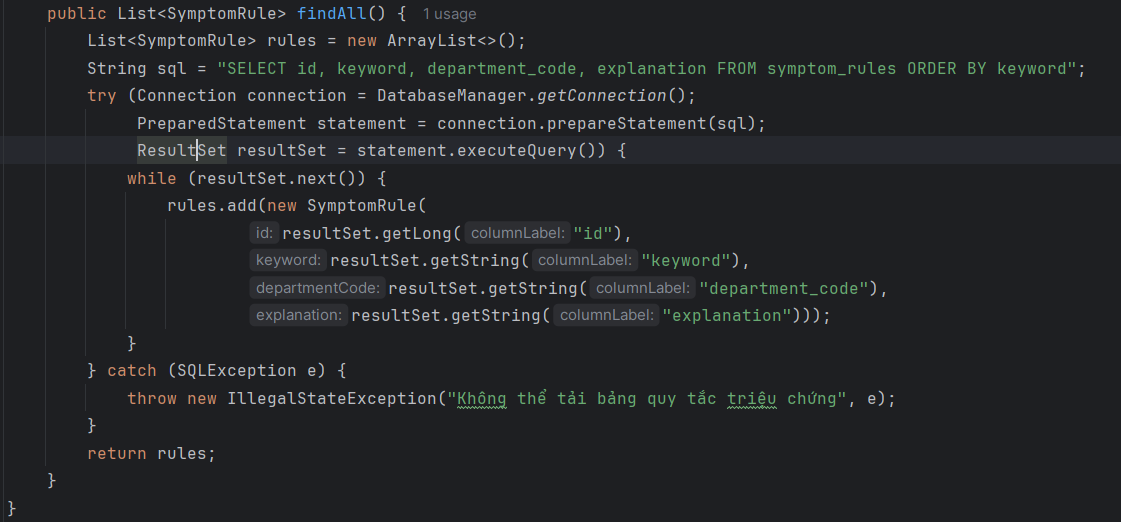
* + Phương thức **findLatestByQueueNumber(queueNumber):** Truy vấn đơn thuốc gần nhất - Tìm đơn thuốc dựa trên số thứ tự (queue\_number) của bệnh nhân. Sử dụng ORDER BY id DESC LIMIT 1 để lấy bản ghi được tạo gần nhất.



* + Phương thức **private Prescription map(ResultSet)** đảm bảo chuyển đổi dữ liệu thô từ CSDL thành đối tượng **Prescription** đã được **đóng gói** cho tầng Service.



* Lớp **SymptomRuleDao:** dùng để truy xuất dữ liệu từ bảng symptom\_rules trong cơ sở dữ liệu và ánh xạ chúng thành danh sách các đối tượng symptomRule
  + Phương thức **findAll( )** dùng để tải toàn bộ quy tắc triệu chứng từ DB. Dùng try-with-resources để mở Connection, PreparedStatement, ResultSet → tự động đóng tài nguyên.



### 3.3. Sự phối hợp với lớp Service

* Lớp **AuthService:** Đây là lớp dịch vụ trong ứng dụng, chịu trách nhiệm **xử lý logic liên quan đến xác thực bác sĩ (Doctor) khi đăng nhập.** AuthService sử dụng **DoctorDao** để truy xuất dữ liệu bác sĩ từ cơ sở dữ liệu.
  + Có 2 hàm khởi tạo Constructor mặc định và có tham số hữu ích khi viết unit test hoặc thay đổi cách truy xuất dữ liệu.
  + Phương thức **login (String username, String password)** giúp đảm bảo không cho phép đăng nhập với dữ liệu null, kiểm tra mật khẩu an toàn bằng cách dùng PasswordHasher. Trả về kết quả dưới dạng Optional → tránh lỗi NullPointerException.
* Lớp **DoctorService:** Đóng vai trò trung gian giữa tầng DAO (truy xuất dữ liệu) và logic nghiệp vụ của bác sĩ. Nó **quản lí bệnh nhân, ghi chú khám và đơn thuốc.**
  + Phương thức **findWaitingPatients( String departmentCode):** Lấy danh sách bệnh nhân đang chờ khám theo mã khoa (departmentCode).
  + Phương thức **markAsExamined(Patient patient, String examNote):** để cập nhật ghi chú khám( examNote) cho bệnh nhân, đổi trạng thái bệnh nhân sang DONE( đã khám xong).
  + Phương thức **saveExamNote(Patient patient, String examNote):** lưu ghi chú khám, không thay đổi trạng thái bệnh nhân.
  + Phương thức **createPrescription(Doctor doctor, Patient patient,String content):** tạo đơn thuốc cho bệnh nhân, gọi prescriptionDao.save(...) để lưu đơn thuốc vào DB.
* Lớp **QueueNumberService:** dùng để **sinh ra số thứ tự khám** (queue number) cho bệnh nhân theo từng khoa.
  + Phương thức **nextQueueNumber(String departmentCode):**  dùng để tạo số thứ tự khám mới cho bệnh nhân trong một khoa. Nó kiểm tra khoa có tồn tại không. Lấy số thứ tự cuối cùng của khoa đó từ bảng bệnh nhân, tính số tiếp theo (nếu chưa có thì bắt đầu từ 01).
  + Phương thức **fetchLatestQueueNumber(String departmentCode):** dùng để lấy số thứ tự khám gần nhất (queue\_number) của bệnh nhân trong một khoa cụ thể từ cơ sở dữ liệu để làm cơ sở sinh số tiếp theo.
* Lớp **SymptomClassifier:** dùng để **phân loại triệu chứng** của bệnh nhân vào đúng khoa (Department) dựa trên các quy tắc triệu chứng (SymptomRule).
  + Phương thức **classify(String description):** Xác định bệnh nhân nên vào khoa nào dựa trên mô tả triệu chứng. Chuẩn hóa mô tả triệu chứng (normalize) và duyệt qua tất cả quy tắc (cachedRules) :
    - Nếu từ khóa trong quy tắc xuất hiện trong mô tả → lấy khoa tương ứng từ departmentCache. Nếu tìm thấy → trả về khoa đó.
    - Nếu không khớp quy tắc nào → dùng khoa mặc định "NOI" (Nội tổng quát).
    - Nếu "NOI" chưa có → lấy khoa đầu tiên trong hệ thống.
  + Phương thức **getCachedRules( ):** Trả về danh sách quy tắc triệu chứng đã cache. Nếu cache chưa có → gọi ensureCache() để tải từ DB.
  + Phương thức **ensureCache( ):** Nếu departmentCache chưa khởi tạo → tải tất cả khoa từ DB và lưu vào map, nếu cachedRules chưa khởi tạo → tải tất cả quy tắc triệu chứng từ DB giúp tránh việc truy vấn DB nhiều lần.
  + Phương thức **normalize(String text):** dùng để chuẩn hóa chuỗi mô tả triệu chứng.
    - Nếu null → trả về chuỗi rỗng.
    - Ngược lại → chuyển về chữ thường và loại bỏ khoảng trắng thừa.
* Lớp **TriageService**: là lớp **dịch vụ trung tâm** trong hệ thống phân luồng bệnh nhân (triage). Nó kết hợp nhiều thành phần khác (SymptomClassifier, QueueNumberService, PatientDao, PrescriptionDao, DepartmentDao) để thực hiện các nghiệp vụ chính: đăng ký bệnh nhân, tìm bệnh nhân/đơn thuốc, và quản lý danh sách khoa.
  + Phương thức **registerPatient(String name, int age, String symptoms, String preferredDepartmentCode):** dùng để đăng kí bệnh nhân mới. Sử dụng validatePatientInfo(...) để kiểm tra dữ liệu nhập, resolveDepartment(...) để xác định khoa (ưu tiên khoa được chọn, nếu không thì phân loại theo triệu chứng), thêm bệnh nhân vào DB(patientDao.insert(...) ), queueNumberService.nextQueueNumber(...) để sinh số thứ tự.
  + Phươngthức**findPrescriptionByQueueNumber(StringqueueNumber):** dùng để tìm đơn thuốc mới nhất theo số thứ tự khám.
  + Phương thức **findPatientByQueueNumber(String queueNumber)**: để tìm bệnh nhân theo số thứ tự khám
  + Phương thức **listDepartments( ):** trả về danh sách tất cả khoa trong hệ thống (departmentDao.findAll())
  + Phương thức **resolveDepartment(String symptoms, String preferredDepartmentCode):** Nếu có mã khoa ưu tiên (preferredDepartmentCode) → tìm khoa đó. Nếu không có hoặc không tồn tại → phân loại bằng SymptomClassifier.
  + Phương thức **validatePatientInfo(String name, int age, String symptoms):** dùng để kiểm tra dữ liệu bệnh nhân

## Lớp giao diện người dùng

### 4.1. Lớp PatientApp

**PatientApp** lớp giao diện người dùng (UI) viết bằng Java Swing, đóng vai trò là cổng bệnh nhân trong hệ thống phân loại khám bệnh. Nó cho phép bệnh nhân đăng ký khám và tra cứu đơn thuốc.

* Chức năng đăng ký khám
  + Phương thức **buildRegistrationPanel( ):** dùng để tạo form nhập họ tên, tuổi, triệu chứng, nút **“Đăng ký và lấy số thứ tự”**, và vùng hiển thị kết quả.
  + Phương thức **registerPatient( ):** dùng để tiếp nhận thông tin bệnh nhân từ giao diện, gửi sang dịch vụ để đăng ký, nhận số thứ tự khám, và gọi triageService.registerPatient( name, age, symptoms, departmentCode) thông báo kết quả bằng Tiếng Việt, hiển thị khoa+ số thứ tự hoặc lỗi cho người dùng.
* Chức năng tra cứu đơn thuốc
  + Phương thức  **buildPrescriptionLookupPanel( ):** dùng để tạo form nhập mã bệnh nhân/số thứ tự, nút “Xem đơn thuốc”, và vùng hiển thị nội dung đơn thuốc.
  + Phương thức **lookupPrescription( ):** dùng để lấy số thứ tự từ ô nhập →gọi triageService.findPrescriptionByQueueNumber và triageService.findPatientByQueueNumberd để hiển thị thông tin đơn thuốc( số thứ tự, tên bệnh nhân, ngày kê, nội dung).

### 4.2. Lớp DoctorLoginFrame

**DoctorLogicFrame** là lớp giao diện Swing dùng cho **bác sĩ đăng nhập vào hệ thống quản lý khám bệnh.** Nó tạo một cửa sổ đăng nhập với form nhập tài khoản và mật khẩu, xử lý xác thực qua AuthService, và nếu thành công thì mở bảng điều khiển bác sĩ (DoctorDashboardFrame).

* Chức năng đăng nhập vào hệ thống quản lí khám bệnh
  + Phương thức **buildUi( ):** để tạo bố cục giao diện
  + Phương thức **doLogin( ):** dùng để xử lí đăng nhập xác thực qua AuthService và nếu hợp lệ thì mở bảng điều khiển bác sĩ (DoctorDashboardFrame).
  + Phương thức **setControlsEnabled(boolean enabled):** Bật/tắt trạng thái của nút đăng nhập để tránh bấm nhiều lần khi đang xử lý.

### 4.3. Lớp DoctorDashboardFrame

**DoctorDashboardFrame** là lớp giao diện **bảng điều khiển dành cho bác sĩ** trong hệ thống phân loại khám bệnh. Nó được dùng để bác sĩ quản lý hàng chờ bệnh nhân, hiển thị chi tiết bệnh nhân, và cho phép bác sĩ làm mới danh sách, đánh dấu bệnh nhân đã khám hoặc lưu ghi chú, kê đơn thuốc cho bệnh nhân.

* Chức năng quản lý hàng chờ bệnh nhân
  + Phương thức **buildUi( ):** tạo bố cục giao diện
    - Màn hình chính hiển thị thông tin bác sĩ và khoa
    - JList hiển thị danh sách bệnh nhân đang chờ trong khoa của bác sĩ
    - Gọi **refreshQueue( )** để tải lại danh sách bệnh nhân từ DoctorService cập nhật vào waitingModel. Nếu có bệnh nhân → tự động chọn bệnh nhân đầu tiên.
    - Có Timer refresh 10 giây/lần để cập nhật hàng chờ tự động khi nhiều máy cùng sử dụng
* Chức năng xem chi tiết bệnh nhân: Phương thức **showPatientDetail( )** hiển thị:số thứ tự, tên, tuổi, trạng thái, triệu chứng, ghi chú khám gần nhất (nếu có) khi chọn một bệnh nhân trong danh sách.
* Chức năng đánh dấu đã khám hoặc lưu ghi chú: Phương thức **markExamined():** dùng để đánh dấu trạng thái hoặc lưu ghi chú.
* Chức năng kê đơn thuốc cho bệnh nhân: Phương thức **prescribe( ):** dùng để nhập nội dung đơn thuốc và tạo bản ghi đơn thuốc gắn bác sĩ và bệnh nhân.

### 4.4. Lớp PrescriptionDialog

**PrescriptionDialog** là hộp thoại (dialog) trong giao diện bác sĩ để nhập và lưu nội dung đơn thuốc cho bệnh nhân.

* Chức năng lưu nội dung đơn thuốc
  + Kế thừa JDialog → tạo một cửa sổ nhỏ (modal dialog) hiển thị trên giao diện chính.
  + Khi đóng với nút lưu thì trả về chuỗi prescription cho DoctorDashboardFrame.

### 4.5 Lớp ExamResultDialog

**ExamResultDialog** là hộp thoại (modal dialog) trong giao diện bác sĩ để nhập kết quả khám hoặc ghi chú cho bệnh nhân.

* Chức năng lưu kết quả khám hoặc ghi chú
  + Kế thừa JDialog → tạo một cửa sổ nhỏ hiển thị trên giao diện chính, dùng để bác sĩ nhập ghi chú khám và chọn hành động (hoàn tất khám hoặc yêu cầu xét nghiệm).
  + Có ba nút hành động: Hủy, Yêu cầu xét nghiệm, Đã khám xong.

## Các lớp tiện ích

Lớp tiện ích chứa các phương thức hỗ trợ dạng static, không cho phép tạo đối tượng, và được dùng để

### 5.1 Lớp PasswordHasher

Lớp PasswordHasher: Hash/match mật khẩu bằng SHA-256 và salt tĩnh để tăng độ an toàn, ví dụ đơn giản vẫn bảo vệ ở mức cơ bản.

* Phương thức **hash( ):** chuyển null thành chuỗi rỗng để tránh NPE.
* Phương thức **matches( ):** kiểm tra đầu vào nếu giá trị đã băm hashed là null thì trả về false. Băm lại mật khẩu người dùng nhập bằng hash(...) rồi so chuỗi với hashed. Nếu trùng nhau, mật khẩu đúng.
* Ghép muối: nối SALT + plan rồi băm, kết quả là mảng byte.
* Chuyển đổi: gọi toHex(...) để chuyển mảng byte sang chuỗi hex dễ lưu trữ.

### 5.2 Lớp UiUtils

Lớp UiUtils – một lớp tiện ích (utility class) trong Java Swing, chuyên cung cấp các phương thức hỗ trợ cho giao diện người dùng.

* Phương thức **configureLookAndFeel( ):** đặt giao diện Look & Feel theo hệ điều hành(SystemLookAndFeel). Thay đổi font mặc định của tất cả component thành Segoe UI, 14pt. Biến lookAndFeelConfigured đảm bảo chỉ cấu hình một lần.
* Phương thức **center(Window window):** giúp cửa sổ hiển thị cân đối thân thiện.
* Phương thức **createTextArea( ):** tạo JTextArea có cấu hình line-wrap giúp văn bản dễ đọc, không bị tràn dòng.
* Phương thức **wrap(JComponent component):** giúp hiển thị nội dung dài có thể cuộn.
* Phương thức **showInfo(Component parent, String message)** và **showError( Component parent, String message):** hiển thị dialog thông báo / lỗi và căn giữa cửa sổ.
* Phương thức **runOnUiThread(Runnable runnable)**: Tránh lỗi khi cập nhật giao diện từ luồng khác.

# LUỒNG XỬ LÝ CHÍNH

## Luồng xử lý bệnh nhân

**PatientApp** bao gồm hai tab giao diện, tập trung vào quy trình tiếp nhận và tra cứu thông tin.

* Đăng ký Khám:
  + **Giao diện:** Nhân viên nhập **Tên, Tuổi, Triệu chứng** và **Chọn Khoa ưu tiên**.
  + **Logic Phân khoa:** Giao diện gọi **service/TriageService.registerPatient(...)**.
    - **TriageService** kiểm tra tính hợp lệ của dữ liệu.
    - **Xác định Khoa:** Ưu tiên khoa được chọn thủ công. Nếu không, gọi **SymptomClassifier** để so khớp từ khóa triệu chứng với bảng **symptom\_rules** và đưa ra quyết định/gợi ý phân khoa.
  + **Cấp số Thứ tự:** TriageService ủy quyền cho **QueueNumberService** để đọc số thứ tự lớn nhất đã cấp cho khoa đó và **tăng lên 1**.
  + **Lưu trữ:** TriageService gọi **PatientDao.insert** để lưu hồ sơ vào bảng patients với trạng thái **WAITING**.
  + **Phản hồi:** Ứng dụng hiển thị thông tin **Khoa** và **Số thứ tự** đã được cấp.
* Tra cứu đơn thuốc:
  + **Giao diện:** Người dùng nhập **Số thứ tự** (ví dụ: TMH-01).
  + **Truy vấn:** Gọi **TriageService.findPrescriptionByQueueNumber**.
  + **Phối hợp DAO:** Service gọi **PrescriptionDao** để lấy đơn thuốc mới nhất và gọi **PatientDao** để lấy thông tin bệnh nhân kèm theo.
  + **Hiển thị:** Hiển thị chi tiết nội dung đơn thuốc và kết quả khám (nếu có).

## Luồng xử lý bác sĩ

* Đăng nhập và xác thực:
  + **Giao diện:** **DoctorLoginFrame** nhận **Username/Password**.
  + **Xác thực:** Gọi **AuthService.login**. Service này:
    - Lấy password\_hash từ **DoctorDao**.
    - Sử dụng **util/PasswordHasher** (sử dụng SHA-256) để băm mật khẩu người dùng nhập và **so sánh** với password\_hash đã lưu.
  + **Mở Dashboard:** Nếu đúng, mở **DoctorDashboardFrame**.
* Quản lý hàng chờ và kê đơn:
  + **Hàng chờ:** **DoctorDashboardFrame** tự động **refresh mỗi 10 giây** (sử dụng Timer).
  + **Truy vấn:** Lưới giao diện gọi **DoctorService.findWaitingPatients** để lấy danh sách bệnh nhân có trạng thái **WAITING** và khớp với khoa của bác sĩ.
  + **Xử lý:** Bác sĩ chọn bệnh nhân để xem chi tiết triệu chứng/ghi chú.
  + **Hoàn tất Khám:** Nút **“Đã khám”** mở ExamResultDialog. Nếu chọn hoàn tất, **DoctorService.markAsExamined** được gọi để lưu ghi chú và chuyển trạng thái bệnh nhân sang **DONE**.
  + **Kê Đơn:** Nút **“Kê đơn thuốc”** mở PrescriptionDialog.
    - Gọi **DoctorService.createPrescription**.
    - Service lưu nội dung đơn thuốc vào bảng **prescriptions**.
    - Sau đó, tự động chuyển trạng thái bệnh nhân sang **DONE**, giải phóng bệnh nhân khỏi hàng chờ.

# KẾT QUẢ & ĐÁNH GIÁ

## Hướng dẫn chạy thử & Kiểm thử

### 1.1. Hướng dẫn chạy thử

Yêu cầu hệ thống và tài khoản mẫu:

* **Yêu cầu JDK:** Hệ thống cần có **JDK 17** hoặc mới hơn
* **Thư viện:** Hệ thống cần file JAR của **sqlite-jdbc**
* **CSDL: clinic-triage.db** sẽ được tạo tự động
* **Tài khoản mẫu:** Username: bs\_tmh / Password: 123456

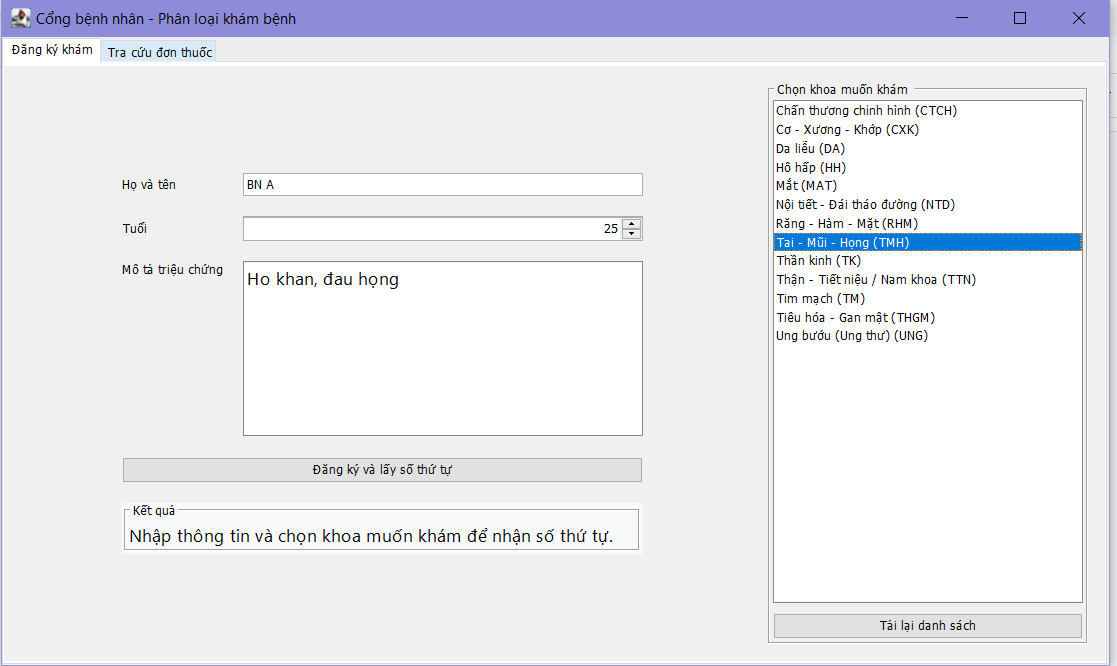
Quy trình khởi động:

* **Khởi tạo CSDL:** Khi ứng dụng được chạy lần đầu, lớp **DatabaseInitializer.java** sẽ tự động thực thi **schema.sql** và chèn dữ liệu mẫu (các khoa, bác sĩ, và quy tắc triệu chứng).
* **Chạy ứng dụng:**
  + Patient App: Chạy lớp **patient.ui.PatientApp** để mở cổng đăng ký và tra cứu.
  + Doctor App: Chạy lớp **doctor.ui.DoctorLoginFrame** để truy cập hệ thống.

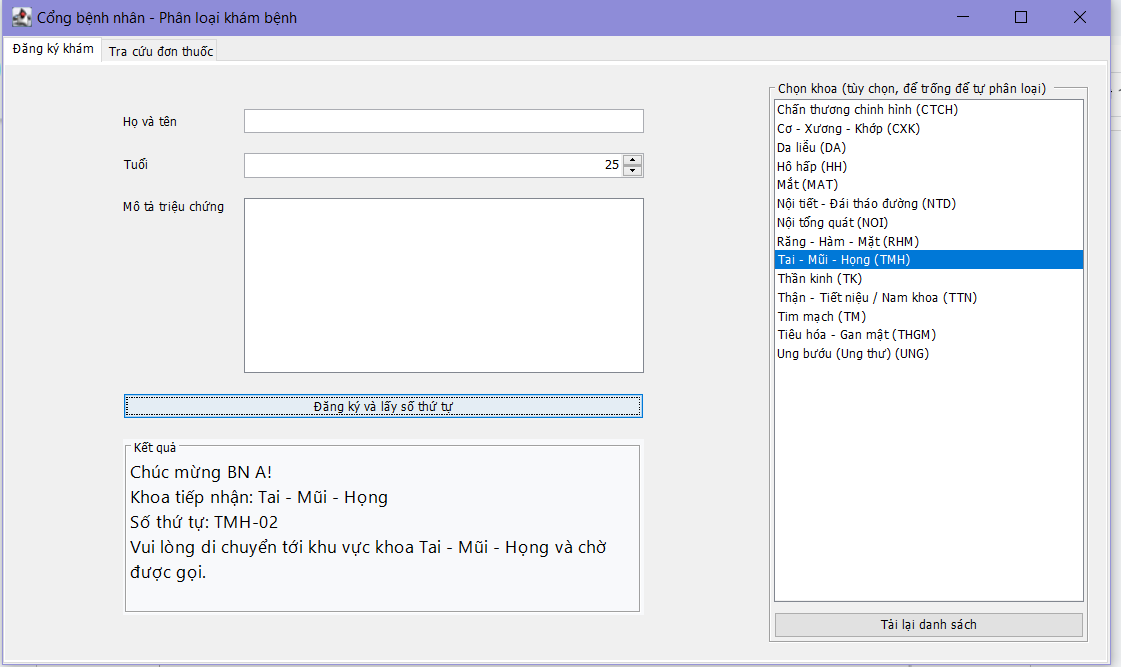
### 1.2. Kịch bản kiểm thử

Test case 1: Đăng ký và phân khoa

* Trên **PatientApp** nhập thông tin bệnh nhân A (BN A) với triệu chứng “Ho khan, đau họng” và chọn khoa Tai - Mũi - Họng (TMH).

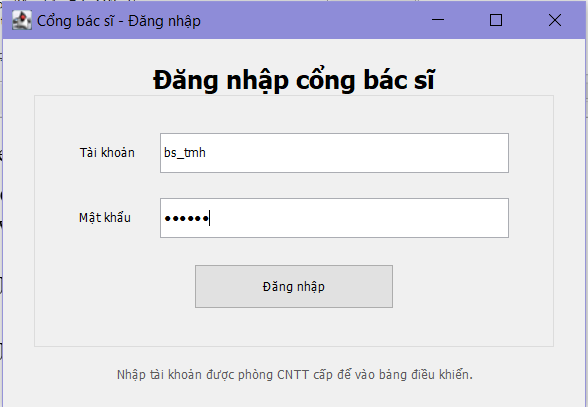


* Sau khi bấm Đăng ký và lấy số thứ tự, hệ thống xác nhận đăng ký thành công và bệnh nhân đã được tiếp nhận bởi khoa Tai - Mũi - Họng với Số thứ tự là TMH-02.

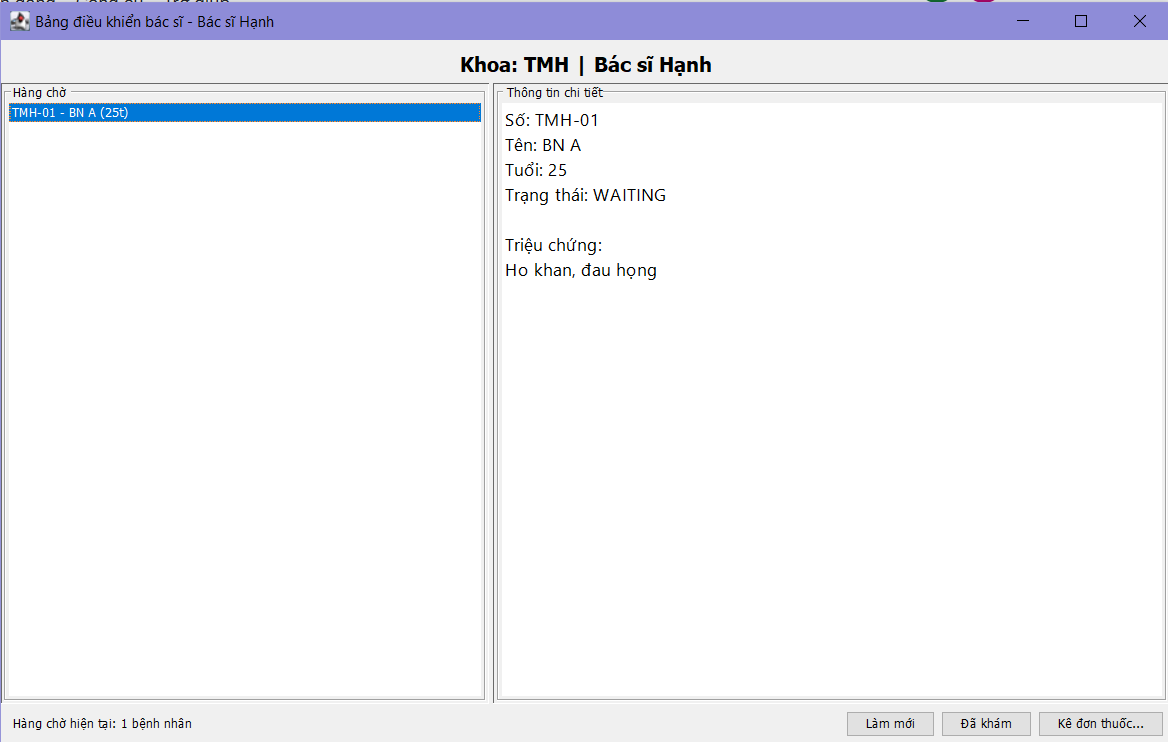


Test case 2: Xác thực & Hàng đợi:

* Trên **Cổng đăng nhập - Bác sĩ** đăng nhập bằng Username: bs\_tmh và Password: 123456.

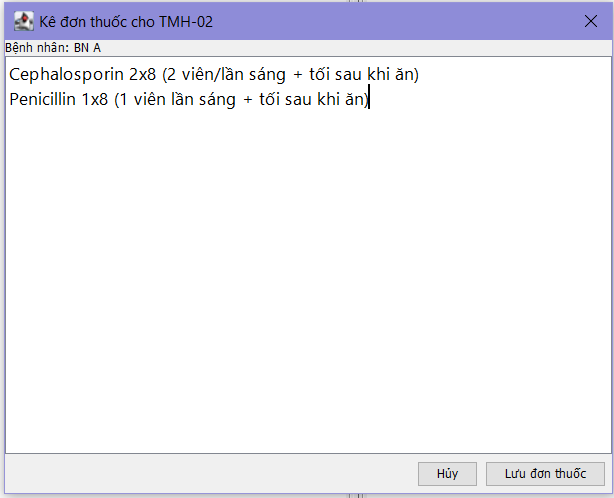


* Đăng nhập thành công qua **AuthService**. Dashboard chỉ hiển thị **BN A** (TMH-01).

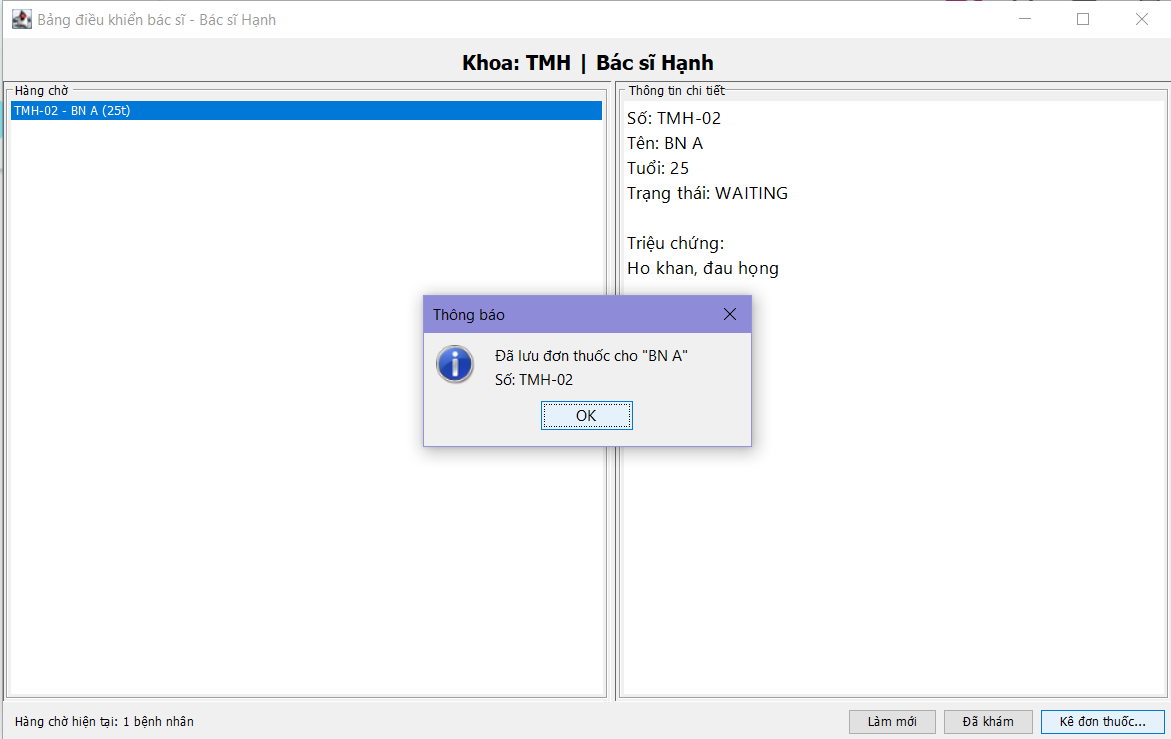


Test case 3: Kê đơn thuốc:

* Bác sĩ kê đơn cho BN A, bấm kê đơn thuốc để gọi **createPrescription.**

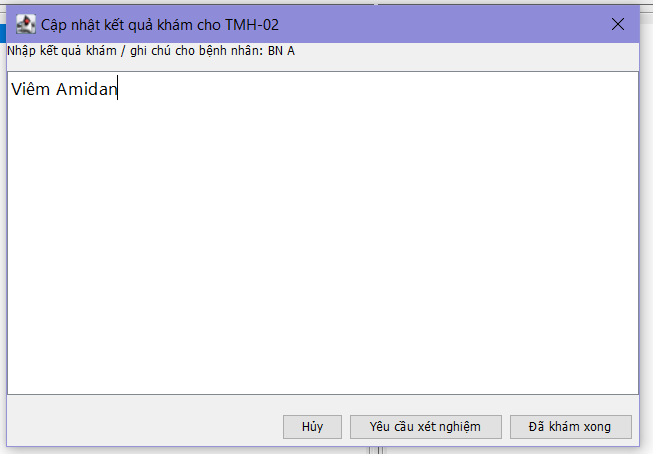


* Sau khi kê đơn xong bấm lưu đơn thuốc, Đơn thuốc được lưu vào **prescriptions** nhưng trạng thái BN A **vẫn là WAITING.**

****

Test case 4: Lưu ghi chú & Giải phóng

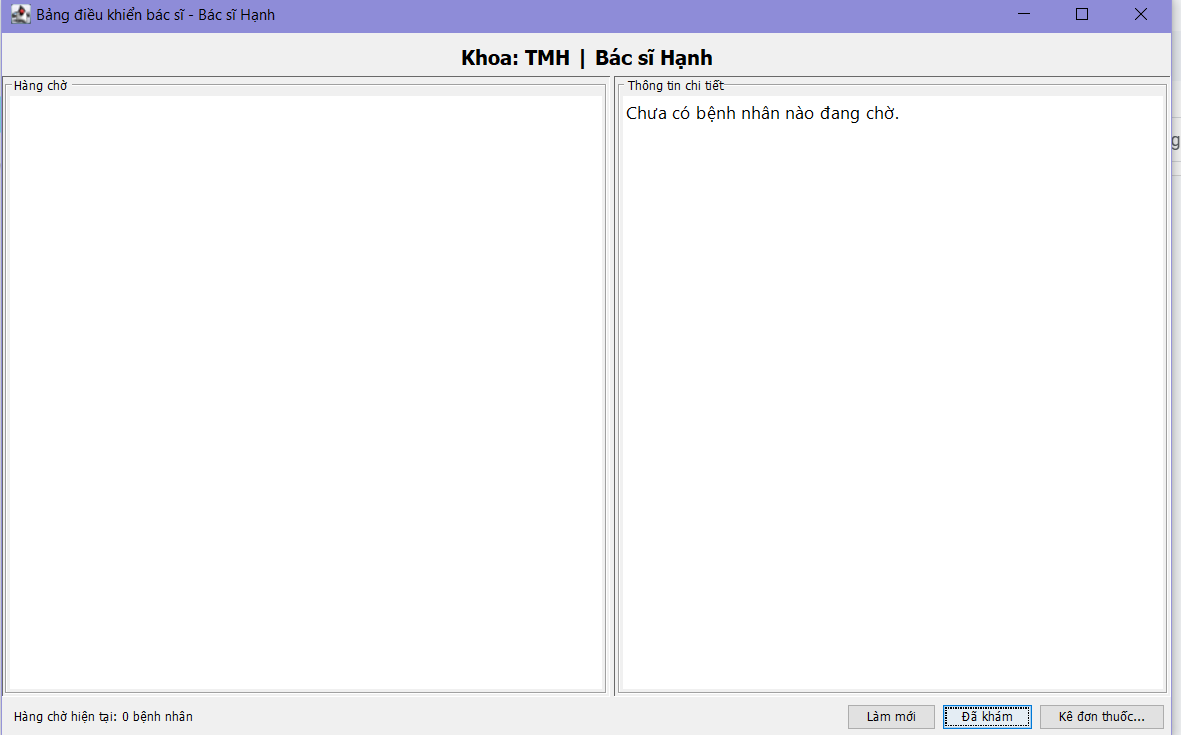
* Bác sĩ bs\_tmh chọn BN A, bấm nút **Đã khám** để nhập ghi chú.



* **DoctorService.saveExamNote** lưu ghi chú khám.

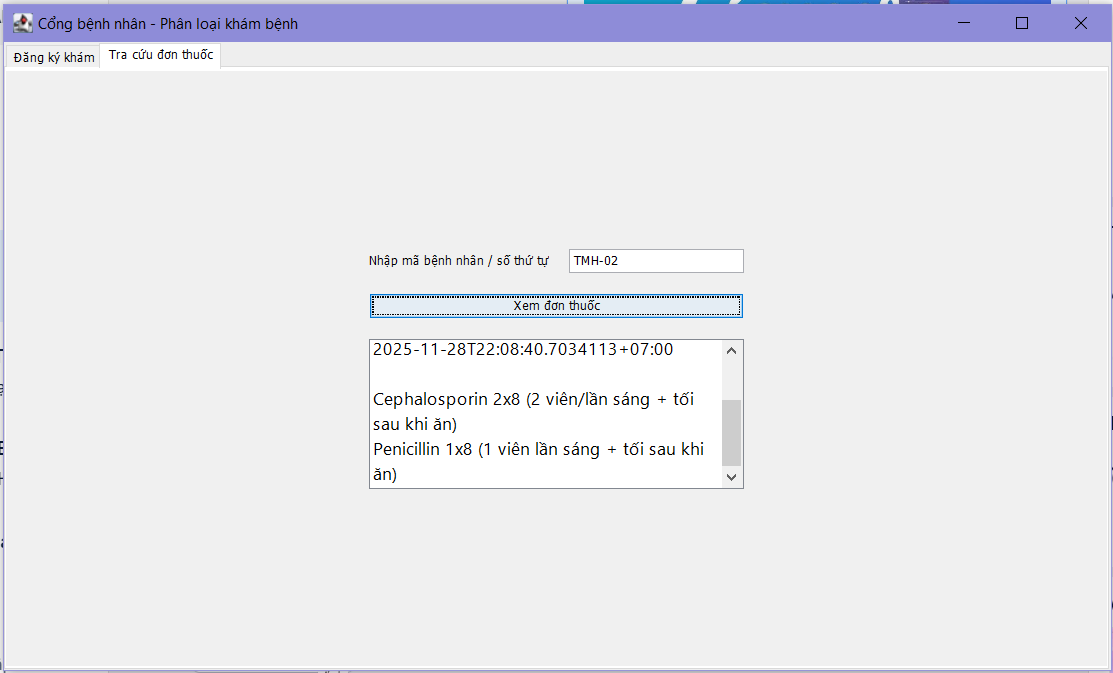


* Trạng thái BN A chuyển thành **DONE**. BN A biến mất khỏi hàng chờ.



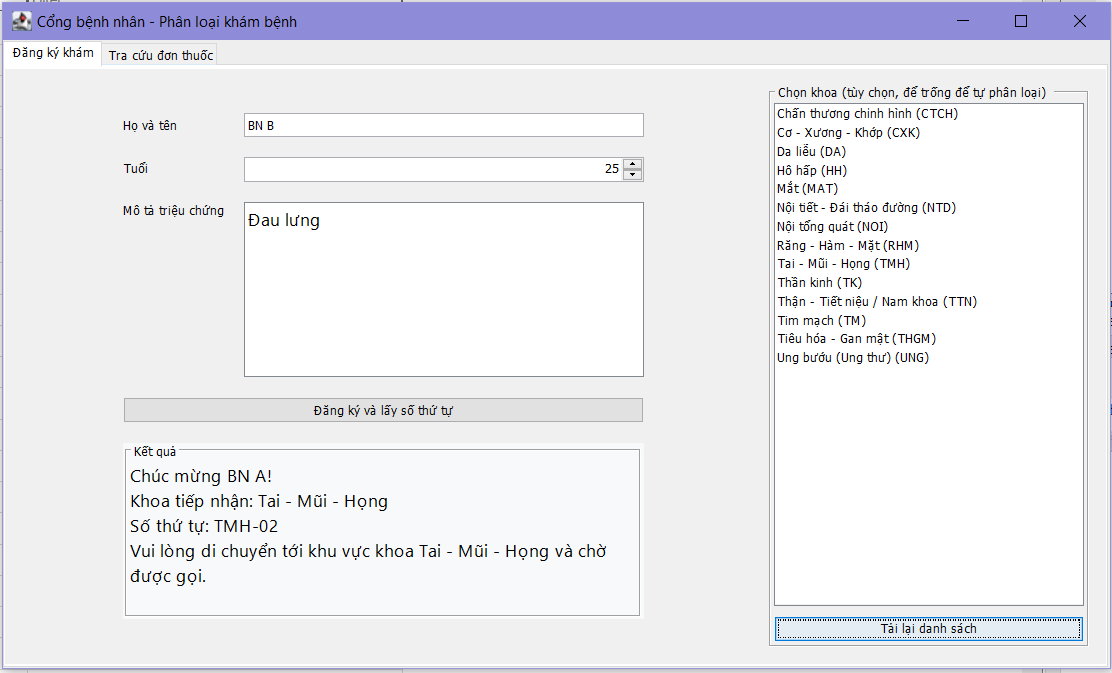
Test case 5: Tra cứu đơn thuốc:

* Trên Patient App ở trang Tra cứu đơn thuốc, tra cứu số TMH-02.
* Hiển thị nội dung đơn thuốc và thông tin khám bệnh chính xác.

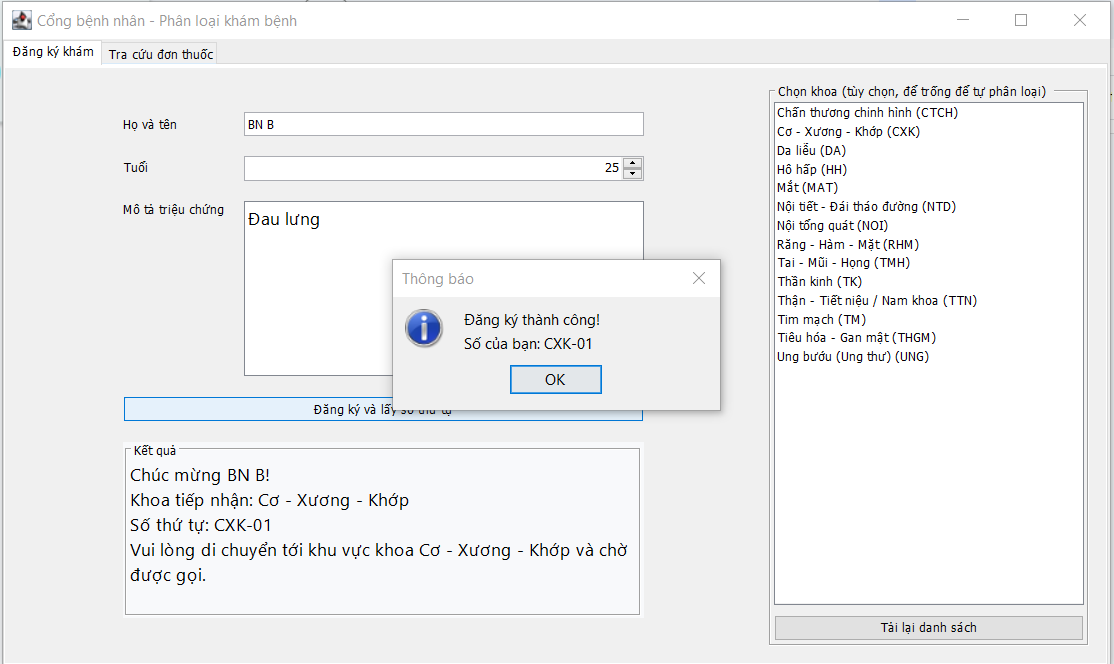


Test case 6: Phân khoa bán tự động:

* Trên **Patient App**, nhập BN B, với triệu chứng: "đau lưng". Để trống lựa chọn khoa ưu tiên.

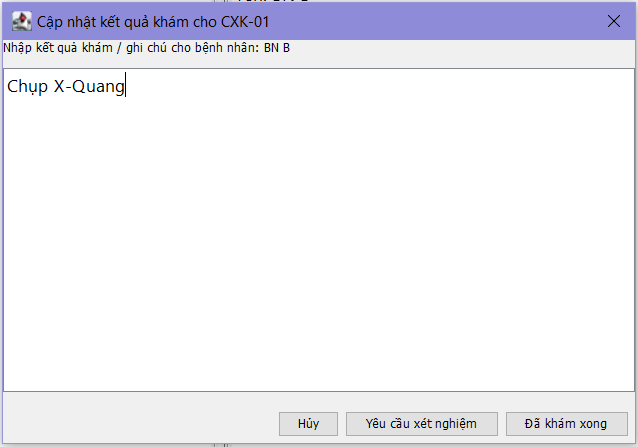


* **SymptomClassifier** gợi ý/gán khoa CXK (dựa trên symptom\_rules). BN B được cấp số CXK-01.

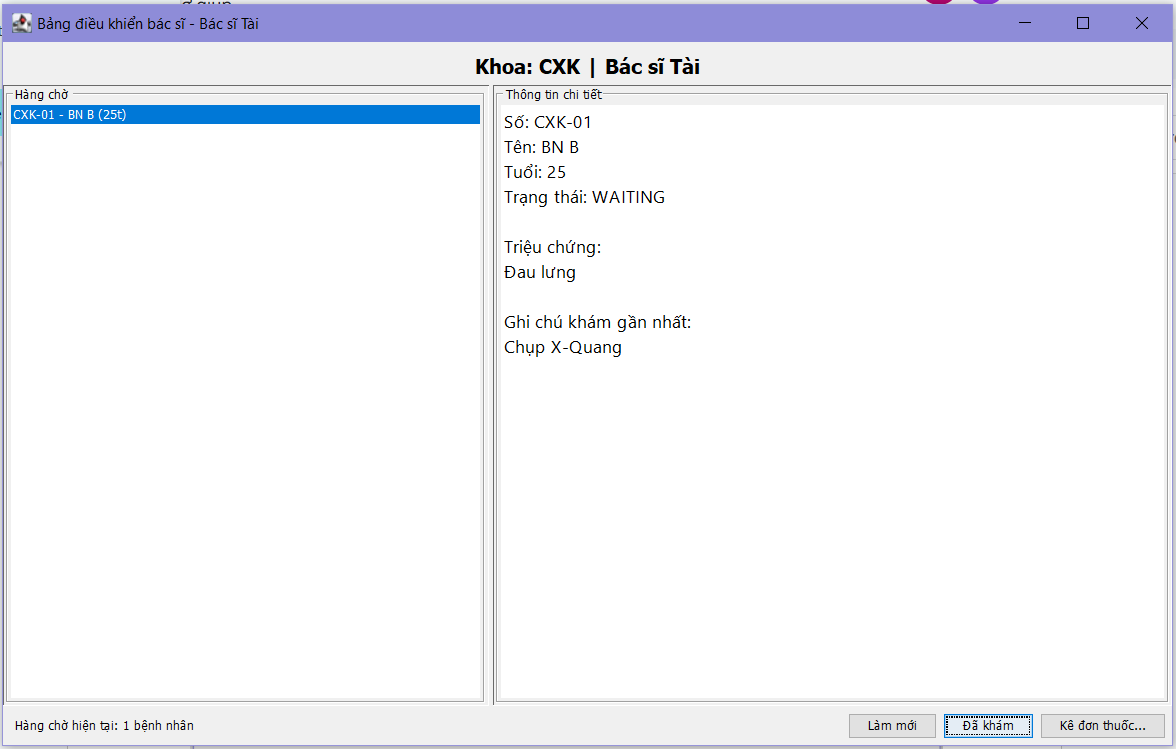


Test case 7: Yêu cầu xét nghiệm:

* Bác sĩ bs\_cxk chọn BN B, bấm nút **Đã khám** để nhập ghi chú để yêu cầu xét nghiệm.



* Sau khi bấm yêu cầu xét nghiệm, BN B được ghi chú yêu cầu chụp X-Quang, nhưng trạng thái vẫn **WAITING.**



# KẾT LUẬN & HƯỚNG PHÁT TRIỂN

1. Kết Luận

**Ưu điểm**

* Đây là một hệ thống có **kiến trúc phân tầng rõ ràng, dễ bảo trì và mở rộng.**
  + Config (AppConfig) tầng trên cùng: chịu trách nhiệm đọc biến môi trường và cung cấp đường dẫn DB.
  + DB Layer ( DatabaseManeger, DatabaseInitializer): quản lí kết nối và khởi tạo dữ liệu.
  + Model (Department, Doctor, Patient, Prescription, SymptomRule, PatientStatus): các lớp dữ liệu thuần.
  + DAO Layer (DepartmentDao, DoctorDao, PatientDao, PrescriptionDao, SymptomRuleDao): truy cập dữ liệu, dùng DatabaseManager.
  + Service Layer (AuthService, SymptomClassifier, QueueNumberService, TriageService, DoctorService): xử lý nghiệp vụ, gọi DAO và util.
  + Util (PasswordHasher, UiUtils): tiện ích chung cho bảo mật và giao diện.
  + UI Layer (PatientApp, DoctorLoginFrame, DoctorDashboardFrame, PrescriptionDialog, ExamResultDialog): giao diện người dùng, gọi service để thực hiện nghiệp vụ.

Nhờ chia tầng (config → db → dao → service → ui) nên việc thay đổi driver DB hoặc update UI đều không ảnh hưởng các tầng khác.

* Tái sử dụng và tiện ích
  + PasswordHasher và UiUtils gom các hàm hỗ trợ chung điều này giúp tái sử dụng, giảm lặp lại code, và chuẩn hóa giao diện.
  + Các DAO dùng PreparedStatement để chống SQL injection, đảm bảo an toàn.
* Trải nghiệm người dùng:
  + giao diện có bố cục hợp lý, dễ thao tác, có thông báo rõ ràng.
  + Refresh tự động mỗi 10 giây giúp đồng bộ nhiều máy cùng sử dụng.

**Nhược điểm**

* SQLite hạn chế mở rộng
  + SQLite phù hợp cho demo hoặc ứng dụng nhỏ, nhưng khi dữ liệu lớn hoặc nhiều người dùng đồng thời thì dễ gặp vấn đề hiệu năng, khóa ghi (write lock).
  + Không hỗ trợ clustering hay replication như PostgreSQL / MySQL.
* PasswordHasher dùng SHA-256 +salt tĩnh
  + Đây chỉ là mức bảo mật cơ bản. Salt tĩnh dễ bị tấn công rainbow table.
  + Nên thay bằng thuật toán hiện đại hơn (BCrypt, Argon2) với salt động.
* DAO viết tay nhiều PreparedStatement
  + Dễ lặp lại code, khó bảo trì khi số bảng tăng.
  + Không có ORM (Hibernate/JPA) nên thiếu tính năng mapping tự động, transaction management nâng cao.
* Swing giao diện cũ
  + Swing không còn phổ biến, thiếu hỗ trợ responsive và hiện đại so với JavaFX hoặc web UI.
  + Khó tích hợp với mobile hoặc web-based client.

PHỤ LỤC