NGUYỄN THỊ NGỌC HÂN

N23DCPT091 -D23CQPTTK01-N

BÀI TẬP CỦNG CỐ LÍ THUYẾT 5&6 –ANALYSIS

**I) SWE | TRẮC NGHIỆM**

1. **C** – Lớp thực thể (Entity) giữ dữ liệu & trạng thái.
2. **C** – Lớp biên (Boundary) giao tiếp với người dùng/hệ ngoài.
3. **D** – Inheritance biểu diễn kế thừa.
4. **B –** Class diagram mô tả các lớp và quan hệ.
5. **B** – **Include**: UC A phải gọi UC B để hoàn tất.
6. **B** – Scenario = kịch bản tương tác người dùng–hệ thống.
7. **A** – Aggregation: “có–thuộc”, phần vẫn sống độc lập.
8. **B** – Sequence: thứ tự thông điệp giữa đối tượng.
9. **C** – Lớp điều khiển ↔ Controller (MVC).
10. **C** – Dùng sơ đồ lớp để biểu diễn quan hệ giữa các lớp.

**II) SWE | CÂU HỎI NGẮN**

1. **Lớp thực thể (Entity) là gì?**

* Lớp giữ **dữ liệu & trạng thái nghiệp vụ**; bền vững theo thời gian.
* Thường ánh xạ sang **bảng CSDL** (User, Order, Book…).

1. **Vai trò lớp điều khiển (Control)?**

* Điều phối luồng nghiệp vụ của use case.
* Nhận yêu cầu từ Boundary, gọi Entity/Service rồi trả kết quả.

1. **Scenario là gì?**

* Kịch bản thực thi một use case: Pre/Post, luồng chuẩn, và ngoại lệ.
* Dùng làm **cơ sở viết test**/tiêu chí chấp nhận.

1. **Quan hệ Include giữa các use case?**

* UC A bắt buộc gọi UC B để hoàn tất.
* Dùng để tái sử dụng hành vi chung (VD: Xác thực).

1. **Mục đích sơ đồ lớp (Class Diagram)?**

* Mô tả cấu trúc tĩnh: lớp/thuộc tính/phương thức/quan hệ.
* Nền cho thiết kế code & CSDL.

1. **Khác nhau Aggregation vs Composition?**

* Aggregation: “có–thuộc”, phần sống độc lập (Team–Member).
* Composition: “thành phần”, phụ thuộc vòng đời (Order–OrderItem).

1. **Sơ đồ tuần tự (Sequence) là gì?**

* Biểu diễn thứ tự thông điệp theo thời gian giữa các đối tượng.

1. **Quan hệ Extend giữa các use case?**

* UC phụ mở rộng có điều kiện UC gốc (kích hoạt bởi guard).
* Hành vi không bắt buộc (VD: Áp mã giảm giá).

1. **Vai trò lớp biên (Boundary)?**

* Giao tiếp UI/API với người dùng/hệ ngoài.
* Không chứa logic nghiệp vụ lõi.

1. **Sơ đồ cộng tác (Communication/Collaboration)?**

* Mô tả liên kết giữa đối tượng và đánh số thông điệp (1, 1.1, 2…).
* Nhấn mạnh cấu trúc tương tác hơn là trục thời gian.

**III) SWE | CÂU HỎI THẢO LUẬN NHÓM**

1. **Vai trò Entity–Boundary–Control**

* Entity: dữ liệu bền vững • Boundary: giao tiếp • Control: điều phối UC → tách trách nhiệm, giảm coupling.

1. **Aggregation vs Composition**

* Chọn Composition khi vòng đời phụ thuộc; Aggregation khi phần tử vẫn tồn tại độc lập.

1. **Vì sao cần sơ đồ lớp khi phân tích?**

* Thống nhất ngôn ngữ thiết kế, truy vết yêu cầu → cấu trúc, hỗ trợ mapping DB/code.

1. **Sequence vs Collaboration**

* Sequence = timeline; Collaboration = liên kết + số thông điệp. Hai sơ đồ nên đồng bộ.

1. **Lớp điều khiển trong MVC**

* Chính là Controller: gom luật nghiệp vụ, kết nối View–Model, giữ View mỏng.

1. **Viết scenario khi phân tích**

* Xóa mơ hồ, định nghĩa tiêu chí chấp nhận, mở đường cho test tự động.

1. **Đảm bảo trích đủ use case**

* Ma trận actor ↔ mục tiêu, workshop/phỏng vấn/quan sát, rà ngoại lệ & biên.

1. **Quan hệ giữa use case & scenario**

* UC = mục tiêu chức năng; Scenario = câu chuyện cụ thể thực thi UC (main/alt).

1. **Ưu/nhược dùng Sequence**

* Ưu: rõ flow, dễ tìm điểm lỗi; Nhược: dài & rối với hệ lớn → chia nhỏ theo ngữ cảnh.

1. **Nâng cao chất lượng scenario**

* Format cố định (Pre/Post/Main/Alt), đánh số bước, ngôn ngữ rõ, review với stakeholder.

**IV) SWE | CÂU HỎI TÌNH HUỐNG**

**1) QL thư viện: xuất hiện yêu cầu mới sau khi đã viết xong scenario**

Mục tiêu: kiểm soát thay đổi, tránh làm vỡ kế hoạch.

Cách làm:

1. Ghi Change Request (CR) và mô tả rõ ưu tiên, tiêu chí chấp nhận.
2. Phân tích tác động đến Use Case, Scenario, lớp BCE, cơ sở dữ liệu, và test.
3. Ước lượng và tái ưu tiên backlog hoặc sprint, thống nhất lại phạm vi và deadline.
4. Tổ chức workshop ngắn với khách hàng để xác nhận thay đổi.

Cần cập nhật: Tài liệu SRS, bảng traceability, danh sách Use Case, Scenario V2, Sequence/Collaboration Diagram, Class Diagram/ERD và test case.

**2) Khó xác định lớp điều khiển (Control Class)**

Mục tiêu: tách đúng trách nhiệm của các lớp theo mô hình BCE.

Cách làm:

1. Mỗi Use Case tương ứng với một Control Class chịu trách nhiệm điều phối.
2. Dùng động từ trong Use Case để đặt tên lớp (ví dụ: LoanController, EnrollController).
3. Dựng sơ đồ Sequence sơ bộ (Actor – Boundary – Control – Entity) để xác định vai trò điều phối.
4. Nếu Control quá phức tạp, chia thành nhiều lớp nhỏ; không đặt logic vào Boundary hoặc Entity.

Cần cập nhật: Danh sách lớp BCE, sơ đồ Sequence minh họa.

**3) Sau khi hoàn thành sơ đồ lớp, khách hàng yêu cầu thêm chức năng mới**

Mục tiêu: mở rộng chức năng mà không phá vỡ kiến trúc hiện tại.

Cách làm:

1. Cập nhật Use Case mới hoặc sửa lại Use Case liên quan.
2. Từ Use Case, trích thêm lớp mới hoặc quan hệ mới trong sơ đồ lớp.
3. Kiểm tra lại multiplicity, hướng liên kết, vòng đời lớp để chọn Association/Aggregation/Composition.
4. Cập nhật cơ sở dữ liệu và sơ đồ Sequence nếu cần.

Cần cập nhật: Class Diagram, ERD, Sequence mới, changelog yêu cầu.

**4) UC “Đăng ký khóa học” có nhiều trường hợp ngoại lệ**

Mục tiêu: xử lý rõ ràng các luồng ngoại lệ và tái sử dụng được.

Cách làm:

1. Viết Scenario chuẩn gồm Precondition, Main Flow, Postcondition.
2. Liệt kê luồng ngoại lệ và đánh số [A1], [A2]... gắn vào bước liên quan.
3. Với ngoại lệ lớn hoặc có điều kiện riêng, tách thành Use Case Extend.
4. Dùng Decision Table nếu có nhiều điều kiện rẽ nhánh.

Cần cập nhật: Scenario V2, sơ đồ Use Case (Include/Extend), sơ đồ Sequence chi tiết.

**5) Trong sơ đồ tuần tự, một số đối tượng chưa rõ vai trò**

Mục tiêu: xác định rõ trách nhiệm và dòng thông điệp giữa các đối tượng.

Cách làm:

1. Gán stereotype (Boundary, Control, Entity) cho từng lifeline.
2. Đổi tên đối tượng cho sát chức năng, hợp nhất các lifeline trùng vai trò.
3. Thêm guard condition để làm rõ nhánh lựa chọn.
4. Xóa lifeline không gửi/nhận thông điệp, đảm bảo sơ đồ gọn và chính xác.

Cần cập nhật: Sequence Diagram đã tinh gọn và bảng mô tả vai trò đối tượng.

**6) Phát hiện quan hệ giữa các lớp bị sai**

Mục tiêu: dùng đúng kiểu quan hệ UML cho từng trường hợp.

Cách làm:

1. Xác định loại quan hệ:
   * “is-a” → Inheritance
   * “has-a độc lập” → Aggregation
   * “has-a phụ thuộc” → Composition
2. Kiểm tra lại multiplicity và hướng mũi tên.
3. Đặt role name để mô tả rõ vai trò giữa các lớp.
4. Rà lại bằng CRC cards hoặc walkthrough scenario.

Cần cập nhật: Sơ đồ lớp, ký hiệu quan hệ và phần mô tả lớp.

**7) Khó mô tả quan hệ giữa các Use Case**

Mục tiêu: biểu diễn đúng các mối quan hệ Include, Extend, Generalization.

Cách làm:

1. Hành vi chung lặp lại → Include.
2. Hành vi tùy chọn → Extend (kèm điều kiện).
3. Use Case có biến thể theo vai trò → Generalization.
4. Giữ sơ đồ gọn, tránh lạm dụng Include/Extend gây rối.

Cần cập nhật: Use Case Diagram và phần ghi chú quan hệ.

**8) Xác định lớp biên trong hệ thống bán hàng**

Mục tiêu: nhận diện rõ các lớp giao tiếp với người dùng và hệ thống ngoài.

Cách làm:

* Các lớp biên gợi ý:
  + CustomerUI (đăng ký, đăng nhập, đặt hàng)
  + CheckoutUI (địa chỉ, thanh toán)
  + AdminUI (quản lý sản phẩm, đơn hàng, khuyến mãi)
  + PaymentAPIAdapter, ShippingAPIAdapter (giao tiếp bên ngoài)
* Lớp biên chỉ giao tiếp, mọi xử lý nghiệp vụ thuộc lớp Control.

Cần cập nhật: Danh sách Boundary Class, sơ đồ Sequence minh họa.

**9) Sau khi hoàn thiện scenario, khách hàng muốn thêm chức năng mới**

Mục tiêu: kiểm soát thay đổi mà không phá vỡ tài liệu đã hoàn thiện.

Cách làm:

1. Ghi nhận Change Request với mô tả và mức ưu tiên.
2. Cập nhật Scenario V2 cho Use Case liên quan.
3. Đồng bộ traceability, Sequence, Class, và ERD.
4. Điều chỉnh kế hoạch, chi phí và deadline nếu cần.

Cần cập nhật: Tài liệu SRS mới, backlog, test case, Scenario V2.

**10) Trong sơ đồ cộng tác, một số đối tượng không tương tác đúng**

Mục tiêu: đảm bảo thông điệp và liên kết giữa các đối tượng chính xác.

Cách làm:

1. Dựa vào Scenario liệt kê các thông điệp 1, 1.1, 2,...
2. Chỉ giữ liên kết thật sự có trao đổi, bỏ liên kết dư.
3. Đánh lại số thứ tự thông điệp và kiểm tra hướng gửi – nhận.
4. Đối chiếu với sơ đồ Sequence để đảm bảo đồng bộ.

Cần cập nhật: Collaboration Diagram đã chỉnh sửa, bảng đối chiếu với Sequence.

-----------------------------------------------------------------------------------------------------------------

**B. SWE | CÂU HỎI TRẮC NGHIỆM**

1. **C** – Pha lấy yêu cầu.
2. **C** – Agile (lặp & cải tiến liên tục).
3. **C** – Quan sát cách người dùng làm việc.
4. **C** – 3NF loại bỏ phụ thuộc bắc cầu.
5. **B** – Sơ đồ tuần tự.
6. **B** – Sơ đồ tuần tự (nhấn theo thời gian).
7. **C** – CMMI level 5 = cải tiến quy trình.
8. **A** – Agile linh hoạt & đổi nhanh.
9. **A** – DRY = hạn chế viết lặp.
10. **B** – Thiết kế lớp: đóng gói & tái sử dụng.
11. **C** – Kiểm thử có thể chạy song song phát triển.

**II) Câu hỏi trả lời ngắn (Short Answer)**

1. **Phần mềm**: chương trình + dữ liệu + tài liệu. **CNPM**: quy trình/phương pháp/công cụ để xây–vận hành–bảo trì SP.
2. **SDLC phổ biến**: Waterfall (tuần tự), V-Model (song hành kiểm thử), Spiral (lặp + rủi ro), Agile/Scrum (lặp ngắn – giao hàng sớm).
3. **3 loại yêu cầu**:
   * **Chức năng**: hệ thống làm gì (UC/luồng nghiệp vụ).
   * **Phi chức năng**: hiệu năng, bảo mật, khả dụng, tuân thủ…
   * **Ràng buộc/miền**: luật pháp, nền tảng, chuẩn tích hợp.
4. **Sơ đồ lớp**: bản vẽ cấu trúc tĩnh (lớp/thuộc tính/phương thức/quan hệ) – nền cho thiết kế & DB.
5. **Vì sao chúng ta cần kiểm thử**: phát hiện lỗi sớm, xác nhận hiểu đúng yêu cầu, giảm chi phí sửa.
6. **SOLID**: S–Single Responsibility; O–Open/Closed; L–Liskov; I–Interface Segregation; D–Dependency Inversion.
7. **Hộp đen**: dựa input/output, không cần biết code. **Hộp trắng**: dựa cấu trúc bên trong (nhánh/đường đi).
8. **Thiết kế DB từ lớp**: lớp→bảng, thuộc tính→cột, quan hệ→FK/bảng ghép; đặt khóa; chuẩn hóa tới **3NF**.
9. **3 ưu điểm Agile**: phản hồi thay đổi nhanh • giao hàng liên tục • tăng hợp tác & minh bạch.
10. **Chuẩn hóa DB**: 1NF → 2NF → **3NF** (→ BCNF nếu cần).

**III) CÂU HỎI THẢO LUẬN NHÓM (DISCUSSION QUESTIONS)**

**1) So sánh mô hình phát triển phần mềm Agile và Waterfall**

* **Waterfall:** tuần tự theo pha (yêu cầu → thiết kế → triển khai → kiểm thử → bảo trì).  
  ➜ Phù hợp dự án yêu cầu rõ, ít thay đổi.
* **Agile:** chia nhỏ thành sprint, giao sản phẩm từng phần, nhận phản hồi sớm.  
  ➜ Phù hợp môi trường thay đổi nhanh, khách hàng tham gia thường xuyên.
* **Tóm lại:** Agile linh hoạt – Waterfall ổn định; nên chọn theo quy mô & đặc thù yêu cầu.

**2) Lợi ích của việc sử dụng UML trong thiết kế phần mềm**

* Tạo **ngôn ngữ chung** giữa nhà phân tích – lập trình viên – khách hàng.
* Giúp **trực quan hóa hệ thống**, giảm sai lệch hiểu biết.
* Hỗ trợ **truy vết yêu cầu** và **bảo trì dễ dàng** nhờ mô hình hóa cấu trúc, hành vi.
* **Áp dụng:** dùng sơ đồ Use Case, Class, Sequence, Activity xuyên suốt SDLC.

**3) Làm thế nào để đảm bảo phần mềm có thể bảo trì tốt trong tương lai**

* Thiết kế theo nguyên tắc **SOLID** và **DRY**.
* Áp dụng **mẫu thiết kế (Design Patterns)** để dễ mở rộng, giảm phụ thuộc.
* Tài liệu, comment, và kiểm thử đơn vị đầy đủ.
* **Áp dụng:** quản lý mã nguồn bằng Git, CI/CD, và viết test tự động.

**4) Tại sao các công ty phần mềm thường sử dụng mô hình phát triển lặp (Iterative Development)**

* Cho phép **phát hành sớm**, nhận phản hồi thực tế từ người dùng.
* Mỗi vòng lặp giúp **phát hiện lỗi và cải tiến** nhanh hơn.
* Giảm rủi ro vì hệ thống được kiểm thử liên tục.
* **Áp dụng:** Agile, Scrum, hay RUP đều là biến thể của phát triển lặp.

**5) Vai trò của kiến trúc phần mềm trong việc xây dựng hệ thống phức tạp**

* Là **khung cấu trúc tổng thể**, định hướng tổ chức module, tầng (layer).
* Giúp **phân tách trách nhiệm**, tăng khả năng mở rộng và bảo trì.
* Là cơ sở cho **quyết định kỹ thuật dài hạn**.
* **Áp dụng:** mô hình nhiều tầng (MVC, Layered Architecture, Microservices...).

**6) Làm thế nào để đảm bảo hệ thống đáp ứng yêu cầu bảo mật**

* Áp dụng **nguyên tắc bảo mật từ giai đoạn phân tích** (Security by Design).
* Kiểm soát quyền truy cập, mã hóa dữ liệu, xác thực & logging đầy đủ.
* Đánh giá bảo mật định kỳ bằng **pentest hoặc code review**.
* **Áp dụng:** OWASP Top 10, bảo vệ API bằng token và HTTPS.

**7) So sánh kiểm thử đơn vị (Unit Testing) và kiểm thử tích hợp (Integration Testing)**

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Unit Testing** | **Integration Testing** |
| **Mục tiêu** | Kiểm tra logic của 1 hàm/lớp riêng lẻ | Kiểm tra sự tương tác giữa các module |
| **Thời điểm** | Sớm (sau khi viết code xong) | Sau khi ghép module |
| **Công cụ** | JUnit, NUnit, PyTest... | Postman, Selenium, REST Assured... |
| **Kết quả** | Phát hiện lỗi cục bộ | Phát hiện lỗi giao tiếp |

* **Kết luận:** Hai loại test bổ trợ nhau trong quy trình CI/CD.

**8) Những thách thức chính trong việc thu thập yêu cầu phần mềm**

* Khách hàng **không diễn đạt rõ** nhu cầu thật sự.
* Yêu cầu **thay đổi liên tục** trong quá trình phát triển.
* Khác biệt giữa **ngôn ngữ kỹ thuật và nghiệp vụ**.
* **Giải pháp:** phỏng vấn, quan sát, mô phỏng Use Case, prototype, review định kỳ.

**9) Cách áp dụng quy trình Scrum vào dự án thực tế**

* Xây dựng **Product Backlog** gồm các User Story có ưu tiên.
* Chia dự án thành **Sprint (2–4 tuần)** với mục tiêu cụ thể.
* Thực hiện họp Daily Stand-up, Sprint Review, và Retrospective.
* **Vai trò:** Product Owner (quản lý yêu cầu), Scrum Master (điều phối), Development Team (thực hiện).
* **Kết quả:** tăng tính minh bạch, phản hồi sớm, cải tiến liên tục.

**10) Tại sao tối ưu hóa mã nguồn lại quan trọng trong phát triển phần mềm**

* Giúp **nâng cao hiệu suất** và **giảm tiêu hao tài nguyên**.
* Mã gọn, dễ đọc → **bảo trì nhanh, ít lỗi**.
* Giảm chi phí vận hành và tăng khả năng mở rộng hệ thống.
* **Áp dụng:** refactor định kỳ, loại bỏ lặp code, áp dụng clean code & linting.

**IV) SWE | CÂU HỎI TÌNH HUỐNG**

1. **Yêu cầu mới sau khi đã có scenario**

* Mở **Change Request** → phân tích tác động UC/Scenario/Model → ước lượng & **tái ưu tiên** backlog/sprint → xác nhận lại.

1. **Bug nghiêm trọng nhưng lead muốn bỏ qua**

* Trình bày **rủi ro/ảnh hưởng**; đề xuất fix/mitigation; **ghi issue** & escalate khi cần; chốt lại **Definition of Done**.

1. **Trễ tiến độ vì yêu cầu đổi liên tục**

* Cố định scope theo **iteration**, giảm phạm vi giao hàng, thêm **buffer**, kiểm soát CR.

1. **Thiết kế CSDL TMĐT không dư thừa**

* Mô hình ER đúng; FK/unique; bảng ghép n–n; **chuẩn hóa 3NF/BCNF**; index hợp lý.

1. **Startup: chọn Waterfall hay Agile?**

* **Agile** (biến động cao, cần feedback & time-to-market nhanh).

1. **Tối ưu hiệu suất**

* Đo (profiling/APM) → xác định nút thắt → **cache**, tối ưu query/index → batch/async → scale → **load test**.

1. **Nhiều lỗ hổng bảo mật, không muốn gián đoạn**

* Vá theo mức rủi ro; bật **WAF/rate-limit**, quản lý secret; log/monitor; triển khai dần bằng **feature toggle**.

1. **Đội ngũ bất đồng về cách triển khai**

* Đặt **tiêu chí quyết định**, chạy **POC/A-B test**, quyết định bởi PO/architecture board dựa dữ liệu.

1. **Đảm bảo UI thân thiện**

* Theo guideline UX + accessibility; **usability test**; tiêu chí chấp nhận UI rõ; phản hồi tức thì.

1. **Dự án cũ không tài liệu**

* Reverse-engineer: đọc mã/DB/log; phỏng vấn vận hành; viết **tài liệu tối thiểu** (SRS rút gọn/ADR); thêm **test hồi quy**, rồi tiếp tục phát triển.

**BÀI TẬP THỰC HÀNH**

1. **Java OOP – Quản lý sinh viên**

**// Domain**

**class Student {**

**private final String id;**

**private String name;**

**private String email;**

**public Student(String id, String name, String email) {**

**this.id = id; this.name = name; this.email = email;**

**}**

**public String getId() { return id; }**

**public String getName() { return name; }**

**public String getEmail() { return email; }**

**public void changeEmail(String newEmail){ this.email = newEmail; }**

**}**

**class Course {**

**private final String code;**

**private final String title;**

**public Course(String code, String title){ this.code = code; this.title = title; }**

**public String getCode(){ return code; }**

**public String getTitle(){ return title; }**

**}**

**class Enrollment {**

**private final Student student;**

**private final Course course;**

**private Double grade; // nullable**

**public Enrollment(Student s, Course c){ this.student = s; this.course = c; }**

**public void recordGrade(double g){ this.grade = g; }**

**public String toString(){**

**return student.getId()+" - "+course.getCode()+" : "+(grade==null? "N/A" : grade);**

**}**

**}**

**// Service (đơn giản)**

**import java.util.\*;**

**class StudentService {**

**private final Map<String, Student> students = new HashMap<>();**

**private final Map<String, Course> courses = new HashMap<>();**

**private final List<Enrollment> enrollments = new ArrayList<>();**

**public void addStudent(Student s){ students.put(s.getId(), s); }**

**public void addCourse(Course c){ courses.put(c.getCode(), c); }**

**public Enrollment enroll(String studentId, String courseCode){**

**Student s = students.get(studentId);**

**Course c = courses.get(courseCode);**

**if(s==null || c==null) throw new IllegalArgumentException("Invalid id");**

**Enrollment e = new Enrollment(s,c);**

**enrollments.add(e);**

**return e;**

**}**

**public List<Enrollment> listEnrollments(){ return Collections.unmodifiableList(enrollments); }**

**}**

**// Demo nhỏ**

**public class Main {**

**public static void main(String[] args) {**

**StudentService service = new StudentService();**

**service.addStudent(new Student("SV001","Nguyen An","an@uni.edu"));**

**service.addStudent(new Student("SV002","Tran Binh","binh@uni.edu"));**

**service.addCourse(new Course("SE101","Nhập môn CNPM"));**

**service.addCourse(new Course("DB201","Cơ sở dữ liệu"));**

**Enrollment e1 = service.enroll("SV001","SE101");**

**e1.recordGrade(8.5);**

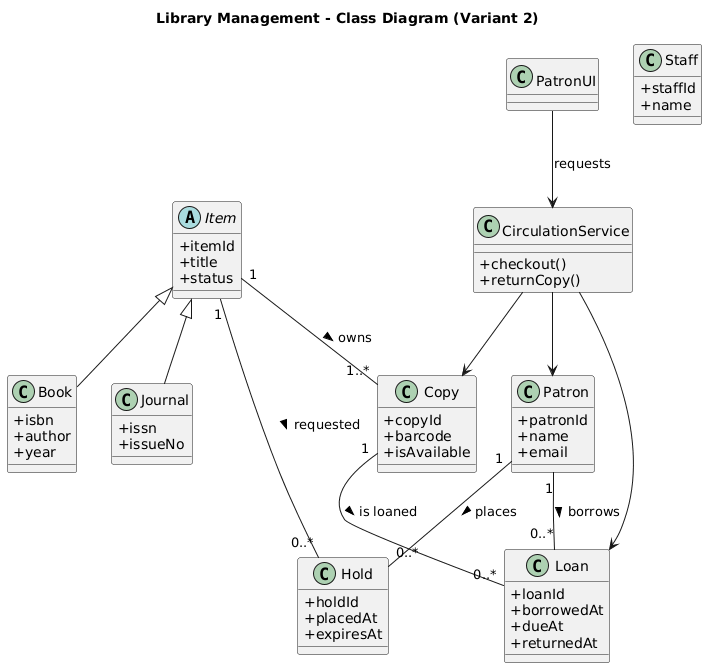
**service.enroll("SV002","DB201");**

**for(Enrollment e : service.listEnrollments()) System.out.println(e);**

**}**

**}**

1. **CLASS DIAGRAM: HỆ THỐNG QUẢN LÍ THƯ VIỆN**

****

1. **Test case – Đăng ký tài khoản**

**Bảng test tóm tắt**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Mục tiêu** | **Bước chính** | **Kỳ vọng** |
| TC01 | Đăng ký thành công | Nhập email hợp lệ, mật khẩu ≥8, xác nhận đúng, bấm Đăng ký | Tạo tài khoản, gửi email xác thực |
| TC02 | Trùng email | Nhập email đã tồn tại | Báo “Email đã được sử dụng” |
| TC03 | Mật khẩu yếu | Nhập pass 6 ký tự | Báo “Mật khẩu tối thiểu 8 ký tự” |
| TC04 | Xác nhận sai | Pass và Confirm khác nhau | Báo lỗi & không tạo tài khoản |
| TC05 | Email không hợp lệ | “abc@” | Báo “Email không hợp lệ” |
| TC06 | Chặn spam | Submit liên tục >5 lần/1 phút | Bị rate-limit, hiển thị thử lại sau |

**Gherkin**

**Feature: User registration**

**Scenario: Successful registration**

**Given I am on the registration page**

**When I submit "han@example.com" and password "Abcd#1234"**

**Then I should see "Đăng ký thành công"**

**And an activation email is sent**

**Scenario: Duplicate email**

**When I submit "han@example.com" again**

**Then I should see "Email đã được sử dụng"**

1. **Thiết kế CSDL bán hàng trực tuyến – DDL**

**- - SCHEMA: ecommerce\_v2 (3NF)**

**CREATE TABLE customers(**

**customer\_id BIGSERIAL PRIMARY KEY,**

**full\_name VARCHAR(120) NOT NULL,**

**email VARCHAR(120) UNIQUE NOT NULL,**

**phone VARCHAR(20),**

**created\_at TIMESTAMP NOT NULL DEFAULT now()**

**);**

**CREATE TABLE categories(**

**category\_id BIGSERIAL PRIMARY KEY,**

**category\_name VARCHAR(100) NOT NULL UNIQUE**

**);**

**CREATE TABLE products(**

**product\_id BIGSERIAL PRIMARY KEY,**

**category\_id BIGINT NOT NULL REFERENCES categories(category\_id),**

**sku VARCHAR(40) UNIQUE NOT NULL,**

**title VARCHAR(200) NOT NULL,**

**price NUMERIC(12,2) NOT NULL CHECK (price>=0),**

**status VARCHAR(20) NOT NULL DEFAULT 'ACTIVE'**

**);**

**CREATE TABLE orders(**

**order\_id BIGSERIAL PRIMARY KEY,**

**customer\_id BIGINT NOT NULL REFERENCES customers(customer\_id),**

**order\_status VARCHAR(20) NOT NULL DEFAULT 'PENDING',**

**placed\_at TIMESTAMP NOT NULL DEFAULT now()**

**);**

**CREATE TABLE order\_items(**

**order\_id BIGINT NOT NULL REFERENCES orders(order\_id) ON DELETE CASCADE,**

**product\_id BIGINT NOT NULL REFERENCES products(product\_id),**

**qty INT NOT NULL CHECK (qty>0),**

**unit\_price NUMERIC(12,2) NOT NULL CHECK (unit\_price>=0),**

**PRIMARY KEY(order\_id, product\_id)**

**);**

**CREATE TABLE payments(**

**payment\_id BIGSERIAL PRIMARY KEY,**

**order\_id BIGINT NOT NULL REFERENCES orders(order\_id),**

**method VARCHAR(30) NOT NULL, -- CARD/WALLET/COD**

**amount NUMERIC(12,2) NOT NULL CHECK (amount>=0),**

**status VARCHAR(20) NOT NULL, -- APPROVED/FAILED**

**tx\_ref VARCHAR(80),**

**paid\_at TIMESTAMP**

**);**

**-- Gợi ý index**

**CREATE INDEX idx\_products\_category ON products(category\_id);**

**CREATE INDEX idx\_order\_items\_product ON order\_items(product\_id);**

1. **Java – Minh họa**

**final class BankAccount {**

**private final String accountNo;**

**private long balance; // lưu VND**

**public BankAccount(String accountNo, long init){**

**if(init < 0) throw new IllegalArgumentException("balance >= 0");**

**this.accountNo = accountNo; this.balance = init;**

**}**

**public String getAccountNo(){ return accountNo; }**

**public long getBalance(){ return balance; }**

**// API an toàn, không lộ state bên trong**

**public void deposit(long amount){**

**if(amount <= 0) throw new IllegalArgumentException("amount > 0");**

**balance += amount;**

**}**

**public boolean withdraw(long amount){**

**if(amount <= 0) return false;**

**if(balance < amount) return false;**

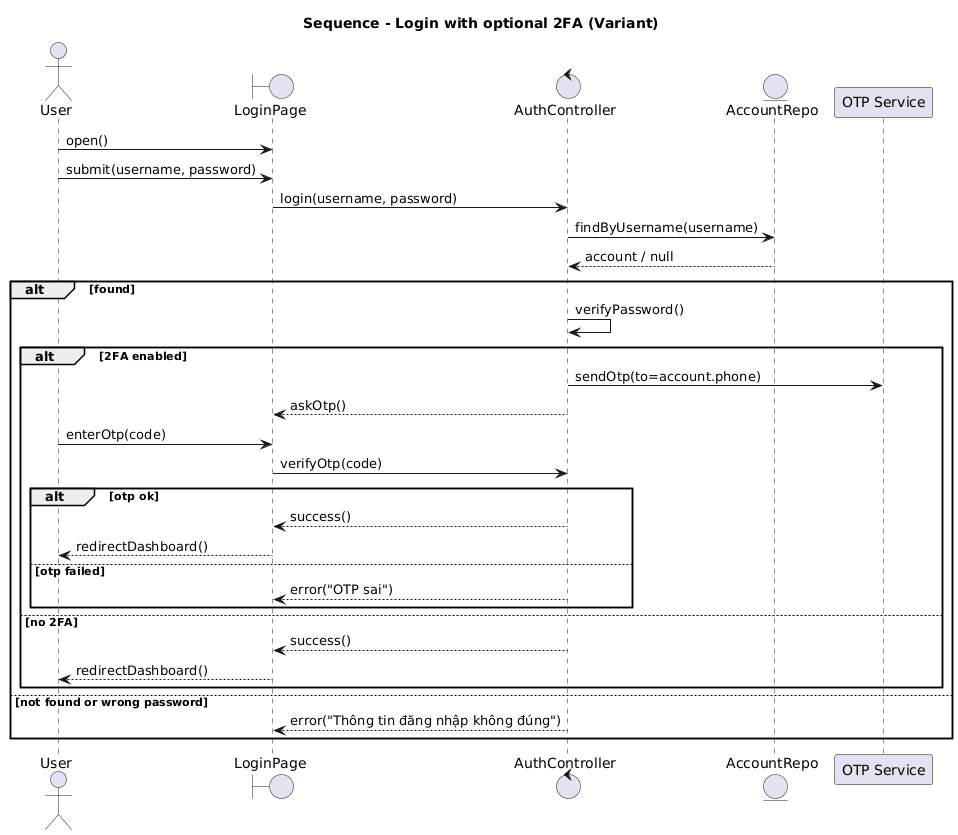
**balance -= amount;**

**return true;**

**}**

**}**

1. **Sequence UML – “Đăng nhập hệ thống”**

****

1. **Unit Test Java – Hàm tính tổng**

**// Class cần test**

**class Calculator {**

**public static int sum(int a, int b){ return a + b; }**

**}**

**// Test (JUnit 5)**

**import static org.junit.jupiter.api.Assertions.\*;**

**import org.junit.jupiter.api.\*;**

**import org.junit.jupiter.params.\*;**

**import org.junit.jupiter.params.provider.CsvSource;**

**class CalculatorTest {**

**@ParameterizedTest**

**@CsvSource({**

**"0,0,0",**

**"1,2,3",**

**"-5,5,0",**

**"-3,-7,-10"**

**})**

**void sum\_basic(int a, int b, int expected){**

**assertEquals(expected, Calculator.sum(a,b));**

**}**

**@Test**

**void sum\_boundary() {**

**assertEquals(Integer.MIN\_VALUE, Calculator.sum(Integer.MAX\_VALUE, 1)); // tràn số (hành vi Java)**

**}**

**}**

1. **Kịch bản kiểm thử “Thanh toán” TMĐT**

Scenario chuẩn

* Tiền điều kiện: giỏ hàng hợp lệ, tổng tiền đã tính, người dùng đăng nhập.

1. User chọn “Thanh toán”.
2. Hệ thống tạo phiên thanh toán và hiển thị phương thức: Thẻ hoặc Ví nội bộ.
3. Người dùng chọn phương thức và xác nhận.
4. Hệ thống nhận kết quả, phát hành hóa đơn, cập nhật trạng thái.

Ngoại lệ chính

* E1 – Thẻ bị từ chối: gateway trả *declined* → đơn giữ *UNPAID*, hiển thị thử lại/đổi phương thức.
* E2 – Ví không đủ số dư: báo lỗi, gợi ý nạp ví.
* E3 – Timeout: retry 2 lần, nếu vẫn lỗi → đánh dấu *PENDING\_PAYMENT*.
* E4 – Không khớp số tiền: rollback, log cảnh báo.

**Gherkin**

**Scenario: Pay by internal wallet**

**Given I have 1,000,000 VND in wallet**

**When I pay order 450,000 VND by wallet**

**Then order status should be "PAID"**

**And wallet balance decreased by 450,000 VND**

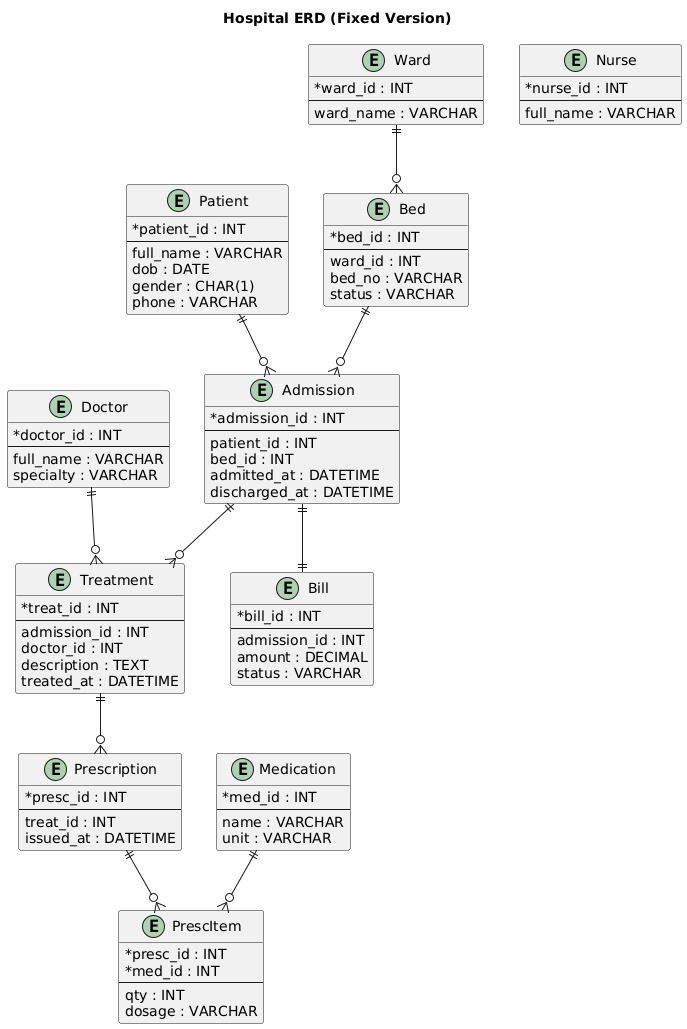
**Scenario: Pay by card but gateway declines**

**When I pay by card and gateway returns "DECLINED"**

**Then order status should be "UNPAID"**

**And I see "Thẻ bị từ chối"**

1. **ERD – Quản lý bệnh viện**

****

1. **Use case – “Đổi lịch chuyến bay” (Rebook) trong hệ thống đặt vé máy bay**

**ID & Tên: UC-AIR-REBOOK - Đổi lịch chuyến bay**  
Actor chính: Hành khách đã có vé (PNR)  
Tiền điều kiện: PNR hợp lệ, vé còn trong điều kiện cho phép đổi; có chỗ trên chuyến mới.  
Hậu điều kiện thành công: PNR cập nhật sang chuyến mới; thu chênh lệch/phí; gửi vé mới.  
Hậu điều kiện thất bại: PNR giữ nguyên; log lý do.

Luồng chuẩn

1. Hành khách nhập mã đặt chỗ và xác thực.
2. Hệ thống hiển thị các chuyến thay thế và mức giá/chi phí chênh lệch.
3. Hành khách chọn chuyến mới.
4. Hệ thống tính phí đổi và tổng chênh lệch.
5. Hành khách xác nhận và thanh toán phần chênh (nếu có).
6. Hệ thống cập nhật PNR, phát hành vé mới, gửi email/SMS.

Ngoại lệ

* E1 – Hết chỗ: không còn ghế phù hợp → đề xuất ngày/giờ khác.
* E2 – Không đủ điều kiện đổi: vé khuyến mãi không cho đổi → hiển thị quy định.
* E3 – Thanh toán chênh lệch thất bại: giữ trạng thái *Pending Rebook*, cho phép thanh toán lại.
* E4 – Trùng yêu cầu: nếu rebook đã hoàn tất trước đó → hiển thị kết quả đã cập nhật.