**TRƯỜNG ĐẠI HỌC PHENIKAA**

**KHOA CÔNG NGHỆ THÔNG TIN**

**A logo for a university

Description automatically generated**

**BÁO CÁO ĐỒ ÁN CƠ SỞ**

Nhóm 4

Đề tài:  
Áp dụng JML trong phương pháp hình thức kiểm chứng mô hình ATM.

Thành Viên:

Bùi Lê Minh-22010141-22010141@st.phenikaa-uni.edu.vn

Bạch Quang Anh-22010434-22010434@st.phenikaa-uni.edu.vn

Phạm Thế Minh-22010075-22010075@st.phenikaa-uni.edu.vn

Ngô Quang Anh-22010388-22010388@st.phenikaa-uni.edu.vn

Giảng viên hướng dẫn:Vũ Quang Dũng

## PHÂN CÔNG NHIỆM VỤ ĐỒ ÁN

|  |  |
| --- | --- |
| **Danh sách các công việc/nhiệm vụ** | **Mô tả tóm tắt công việc** |
| **Cài đặt OpenJMl** | **Cài đặt JDK, OpenJML trên ubuntu** |
| **Thiết kế mô phỏng mô hình ATM** | **Vẽ sơ đồ usecase, class diagram, statediagram, Biểu đồ ERD** |
| **Viết chương trình mô phỏng ATM** | **Lập trình java mô hình ATM tập trung vào tính năng rút tiền và kiểm tra số dư tài khoản và lập trình giao diện đơn giản** |
| **Thu thập và tìm hiểu** | **Tìm kiếm các tài liệu về JML và kiểm chứng bằng OpenJML** |
| **Đặc tả và kiểm chứng** | **Viết đặc tả JML cho chương trình Java và kiểm chứng bằng OpenJML** |
| **Tổng hợp và viết báo cáo** | **Ghi lại các kết quả kiểm chứng, tiến độ công việc, tổng hợp quá trình và viết báo cáo** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STT** | **Mã Sinh Viên** | **Họ và tên** | **Nhiệm vụ phân công** | **Nội Dung đã thực hiện** | **Đánh giá** |
| 1 | 22010434 | Bạch Quang Anh | -Cài đặt JDK,Ubuntu -Thu thập và tìm hiểu tài liệu  -Viết đặc tả  -Tông hợp và viết báo cáo | -Cài đặt thành công  -Cung cấp tài liệu và hoàn thiện báo cáo -Chạy thử dữ liệu trên hệ điều hành ubuntu | 25% |
| 2 | 22010075 | Phạm Thế Minh | -Thiết kế các sơ đồ đặc tả -Viết báo cáo phần sơ đồ đặc tả và mô hình ATM  -Kiểm chứng mô hình bằng Openjml  -Viết đặc tả JML | -Thiết kế sơ đồ usecase, ERD  -Hoàn thiện phần 3 báo cáo -Đang tiến hành kiểm chứng -Hoàn thiện đặc tả JML cho mô hình | 25% |
| 3 | 22010141 | Bùi Lê Minh | -Cài đặt JDK,Ubuntu - Hoàn thiện báo cáo và biên bản -Thu thập và tìm kiếm tài liệu | -Cài đặt và chạy thử dữ liệu thành công -Thu thập và tìm kiếm đầy đủ tài liệu | 25% |
| 4 | 22010388 | Ngô Quang Anh | -Code chương trình và giao diện cho ATM -Thiết kế các sơ đồ đặc tả -Kiểm chứng mô hình bằng OpenJML  -Viết đặc tả JML | -Hoàn thiện chương trình và giao diện ATM  -Thiết kế và đặc tả state diagram, dataflow diagram -Đang tiến thành kiểm chứng  -Hoàn thiện đặc tả JML cho mô hình | 25% |

# **LỜI MỞ ĐẦU**

# Trong thời đại số hóa, hệ thống máy rút tiền tự động (ATM) đóng vai trò quan trọng trong lĩnh vực ngân hàng và tài chính, giúp người dùng thực hiện các giao dịch một cách nhanh chóng, tiện lợi và an toàn. Tuy nhiên, để đảm bảo tính đúng đắn, bảo mật và toàn vẹn dữ liệu của hệ thống, việc kiểm chứng phần mềm trước khi triển khai là điều cần thiết.

# Mô hình kiểm chứng bằng OpenJML giúp phát hiện các lỗi logic trong chương trình, xác minh các điều kiện tiền đề, hậu điều kiện và bất biến của hệ thống. Điều này giúp đảm bảo rằng các giao dịch trên ATM diễn ra đúng quy trình, tránh rủi ro về bảo mật và sai sót trong xử lý dữ liệu.

# Báo cáo này trình bày quá trình xây dựng mô hình ATM, các yêu cầu kiểm chứng phần mềm dựa trên OpenJML và đánh giá kết quả thực nghiệm. Qua đó, nhóm nghiên cứu mong muốn nâng cao chất lượng phần mềm, đảm bảo tính chính xác và đáng tin cậy của hệ thống.

# **LÝ DO CHỌN ĐỀ TÀI**

# Tính thực tiễn cao

# ATM là một trong những hệ thống quan trọng của ngân hàng, phục vụ hàng triệu người dùng mỗi ngày. Việc kiểm chứng giúp phát hiện và phòng tránh các lỗi có thể gây ra rủi ro tài chính hoặc ảnh hưởng đến trải nghiệm khách hàng.

# Nâng cao tính chính xác và bảo mật

# Hệ thống ATM yêu cầu tính chính xác tuyệt đối trong các giao dịch như kiểm tra số dư, rút tiền và xác thực người dùng. Bất kỳ sai sót nào cũng có thể dẫn đến hậu quả nghiêm trọng.

# Ứng dụng OpenJML trong kiểm chứng phần mềm

# OpenJML là công cụ mạnh mẽ giúp kiểm tra tính đúng đắn của chương trình Java bằng các chú thích JML. Việc áp dụng OpenJML vào hệ thống ATM giúp xác định lỗi logic trước khi triển khai thực tế.

# Hỗ trợ phát triển phần mềm chất lượng cao

# Kiểm chứng phần mềm giúp lập trình viên phát hiện và sửa lỗi ngay từ giai đoạn phát triển, tiết kiệm thời gian và chi phí bảo trì sau này.

# Củng cố kiến thức về kiểm chứng và lập trình hướng đối tượng

# Đề tài này không chỉ giúp hiểu rõ hơn về cách xây dựng hệ thống ATM mà còn cung cấp kiến thức thực tế về kiểm chứng phần mềm, giúp sinh viên nâng cao kỹ năng lập trình và phân tích hệ thống.

# 

# 

# **LỜI CẢM ƠN**

Em muốn bày tỏ lòng biết ơn sâu sắc và chân thành đến tất cả các quý thầy cô đã hỗ trợ và giúp đỡ em trong quá trình học tập và nghiên cứu đề tài này. Từ khi bắt đầu học tập tại trường đến nay, em đã nhận được rất nhiều sự quan tâm, giúp đỡ của các giảng viên và bạn bè, và em sẽ mãi mãi ước ao được giữ gìn những mối quan hệ này.

Em muốn gửi lời cảm ơn đặc biệt đến các Thầy Cô ở Ngành CNTT, đã truyền đạt cho em những kiến thức quý báu trong suốt quá trình học tập tại trường. Nhờ có những lời hướng dẫn và dạy bảo của các Thầy Cô, em mới có thể hoàn thiện được đề tài nghiên cứu của mình. Em xin chân thành cảm ơn những Thầy Cô đã trực tiếp giúp đỡ, quan tâm và hướng dẫn em hoàn thành tốt bài báo cáo này trong thời gian qua. Sự giúp đỡ của các Thầy Cô đã truyền cảm hứng cho em để tiếp tục nghiên cứu và học tập.

Em thừa nhận rằng bài tập lớn của nhóm em còn hạn chế và còn nhiều thiếu sót. Tuy nhiên, em rất mong nhận được những ý kiến đóng góp quý báu của các Thầy Cô để kiến thức của em trong lĩnh vực này được hoàn thiện hơn, đồng thời có điều kiện bổ sung và nâng cao ý thức của mình.

Một lần nữa, em xin chân thành cảm ơn tất cả mọi người đã giúp đỡ em trong quá trình học tập và nghiên cứu đề tài này. Em sẽ luôn trân trọng những kinh nghiệm và kiến thức quý báu mà em đã học được từ các Thầy Cô.

*Em xin chân thành cảm ơn!*

## Mục Lục:

[**1.**](https://docs.google.com/document/d/1k1d6ow1eWXyvjbHmE0AaJsuwjmkj5CyH/edit#heading=h.30j0zll) [**Giới thiệu**](https://docs.google.com/document/d/1k1d6ow1eWXyvjbHmE0AaJsuwjmkj5CyH/edit#heading=h.30j0zll) [**2.**](https://docs.google.com/document/d/1k1d6ow1eWXyvjbHmE0AaJsuwjmkj5CyH/edit#heading=h.tyjcwt) **Tổng quan về Java Modeling Language (JML)**

2.1 Giới thiệu về JML

2.2 Công cụ hỗ trợ JML

2.3 Một số nghiên cứu liên quan

[**3.**](https://docs.google.com/document/d/1k1d6ow1eWXyvjbHmE0AaJsuwjmkj5CyH/edit#heading=h.z337ya)**Hệ thống ATM** Các sơ đồ và đặc tả

[**4.**](https://docs.google.com/document/d/1k1d6ow1eWXyvjbHmE0AaJsuwjmkj5CyH/edit#heading=h.3whwml4)**Các yêu cầu kiểm chứng về mô hình ATM**

4.1 Tính đúng đắn   
 4.2 Tính bảo mật

4.3 Tính vẹn toàn dữ liệu

4.4 Tính khả dụng

4.5 Tính bảo trì

[**5.**](https://docs.google.com/document/d/1k1d6ow1eWXyvjbHmE0AaJsuwjmkj5CyH/edit#heading=h.2bn6wsx) **Cài đặt mô hình kiểm chứng**

5.1 Đặc tả các chức năng chính bằng JML

5.2 Kiểm chứng tự động một số tính chất của hệ thống ATM với OpenJML

5.3 Kết quả dự kiến

5.4 Kết quả thực nghiệm

**6. Kết luận và hướng phát triển**

**7. Tài liệu tham khảo**

**1. Giới thiệu**

Kiểm thử phần mềm đang được xem là giải pháp chủ yếu nhằm đảm bảo chất lượngcho các sản phẩm. Trong thực tế, các hoạt động kiểm chứng chủyếu được thực hiện thủ công và tiêu tốn khoảng 30-50% tài nguyên (thờigian, nhân lực, chi phí, v.v.) của quá trình phát triển sản phẩm. Hơn nữa, độ phức tạp của các phần mềm ngày càng tăng và trong môi trường cạnhtranh như hiện nay đòi hỏi các công ty phần mềm phải áp dụng các phươngpháp và công cụ nhằm tự động hóa các hoạt động kiểm thử .

Kiểm thử phần mềm bao gồm trong nó nhiều kỹ thuật khác nhau (kiểm thử hộp đen, kiểm thử hộp trắng, kiểm thử mô hình, kiểm thử đơn vị, kiểm thử tích hợp, v.v.). Mỗi kỹ thuật kiểm thử được sử dụng trong một hoặc một vài các giai đoạn của quá trình phát triển phần mềm (phân tích, thiết kế, cài đặt và bảo trì phần mềm). Kiểm chứng một trong kỹ thuật thường được sử dụng trong kiểm thử phần mềm, để đánh giá sự tuân thủ giữa các giai đoạn khác nhau trong quá trình phát triển phần mềm (ví dụ, kiểm chứng sự tuân thủ giữa thực thi và đặc tả trong phát triển phần mềm).

Để kiểm thử tính đúng đắn của hệ thống phần mềm, điều kiện cần đặt ra là phải có một đặc tả chi tiết và chính xác. Đã có nhiều ngôn ngữ đặc tả ra đời và phát triển như OCL, Event B v.v. sử dụng để hỗ trợ đặc tả hệ thống trong giai đoạn phân tích, thiết kế và cài đặt, trong đó JML là ngôn ngữ đặc tả được xây dựng dành riêng cho ngôn ngữ lập trình Java và được kiểm chứng một cách tự động bởi phần mềm OpenJML tích hợp trong môi trường lập trình Eclipse.

Trong những năm gần đây, Hệ thống rút tiền tự động ATM (**Automatic Teller Machine**) được sử dụng một cách phổ biến bởi sự tiện lợi của nó trong các giao dịch liên quan đến tiền tệ (rút tiền, chuyển tiền, thanh toán hóa đơn, đặt vé máy bay, v.v.). Tuy nhiên, bên cạnh những ưu điểm mà hệ thống này mang lại như sự dễ dàng, nhanh chóng thực thi các số giao dịch thì nó cũng đặt ra một số vấn đề cần giải quyết như: *Làm thế nào để ngăn chặn khách hàng rút số tiền vượt quá số dư thực sự của tài khoản? Số tiền trong hạn mức một giao dịch rút tiền là bao nhiêu? Hạn mức rút tiền của một tài khoản trong một ngày là bao nhiêu?* v.v. Bởi vậy, cần có một chính sách đặc tả và kiểm chứng hệ thống một cách rõ ràng, chặt chẽ giúp giảm thiểu các rủi ro gặp phải trong quá trình vận hành hệ thống.

Báo cáo này tập trung đặc tả và kiểm chứng một số tính chất cơ bản của hệ thống rút tiền tự động ATM. Các đóng góp chính của bài báo là: *Đặc tả hệ thống ATM bằng JML* và *Kiểm chứng tự động một số tính chất của hệ thống ATM bằng phần mềm OpenJML*. Nội dung mà bài báo trình bày cho thấy được tính khả thi cả về phương diện lý thuyết và thực nghiệm đối với quá trình xây dựng và phát triển một hệ thống phần mềm.

**2. Tổng quan về Java Modeling Language (JML)  
2.1. Giới thiệu về JML**

JML (Java Modeling Language) là ngôn ngữ đặc tả hình thức được sử dụng để hỗ trợ cho ngôn ngữ lập trình Java trên các khía cạnh: đặc tả hành vi của các **lớp** (**Class**) trong Java và ghi lại các quyết định về thiết kế và thực thi. JML được thiết kế để:  
 - Dễ đọc và dễ sử dụng cho cả lập trình viên và chuyên gia kiểm chứng.  
 - Tích hợp trực tiếp với mã nguồn Java, không cần thay đổi cú pháp ngôn ngữ.  
 - Hỗ trợ kiểm chứng hình thức (**Formal Verification**) và kiểm thử tự động.JML giúp giải quyết nhiều vấn đề trong phát triển phần mềm, đặc biệt là:  
 - **Mô tả chính xác hành vi chương trình**: Định nghĩa rõ ràng cách các phương thức hoạt động.  
 - **Phát hiện lỗi sớm**: Tích hợp với công cụ như OpenJML để phát hiện lỗi trước khi chương trình chạy.  
 - **Tự động kiểm chứng**: Hỗ trợ kiểm thử chính thức mà không cần viết nhiều test case thủ công.  
 - **Cải thiện tài liệu**: Code tự mô tả giúp lập trình viên mới dễ hiểu hơn.

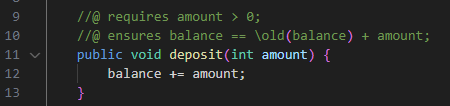
JML thực hiện các công việc trên thông qua bổ sung các **khẳng định** (**Assertions**) vào tệp mã nguồn Java. Cú pháp và ngữ nghĩa của JML đã được xây dựng đầy đủ và chi tiết, bao gồm các khía cạnh đặc tả và kiểm chứng các ràng buộc về **bất biến** (**Invariants**), **tiền điều kiện** (**Pre-conditions**) và **hậu điều kiện** (**Post-conditions**) của các phương thức trong chương trình. Đặc tả JML được nhúng vào mã nguồn Java (có thể ở cùng một tệp mã nguồn Java hoặc ở trong một tệp riêng biệt). Một đặc tả JML được bắt đầu bởi ký hiệu //@<JML specification> hoặc /\*@<JML specification>\*@/ theo sau là các thuộc tính cần đặc tả. Bảng 1 mô tả một số từ khóa cơ bản được sử dụng một cách phổ biến trong JML.

Bảng 1. Một số từ khóa thông dụng của JML

|  |  |
| --- | --- |
| **Từ khóa** | **Tình huống áp dụng** |
| *requires* | Mô tả các ràng buộc tiền điều kiện (pre-condition) |
| *ensures* | Mô tả các ràng buộc hậu điều kiện (post-condition) |
| *signals* | Giá trị ngoại lệ của hậu điều kiện (post-condition) |
| *invarian*t | Mô tả các ràng buộc bất biến |
| *non\_null* | Giá trị của tham số khác rỗng |
| *\old* | Lấy giá trị cũ của một biến được tham chiếu |
| *\forall* | Với tất cả giá trị có trong tập hợp cho trước |
| *\exists* | Tồn tại giá trị của tham số |
| *\result* | Tham chiếu đến kết quả của chương trình con |

Sau đây là một đoạn mã trong ngôn ngữ lập trình Java mô tả phương thức ***deposit(int amount)***. Chú ý rằng, đối với phương thức ***deposit (int amount)*** tiền điều kiện đối với thuộc tính amount (số lượng tiền cần rút) phải là một số dương (amount > 0) và hậu điều kiện ***balance == \old(balance) + amount*** → Số dư mới phải bằng số dư cũ cộng với số tiền gửi vào.

**Đặc tả 1. Minh họa cú pháp của JML**



Các ràng buộc trên được mô tả trong JML như trong Đặc tả 1, trong đó các dòng mã lệnh 9 và 10 là mã nguồn viết bằng ngôn ngữ JML, các dòng lệnh 11 và 12 là câu lệnh trong ngôn ngữ lập trình Java. Đây là một ví dụ đơn giản minh họa sự hỗ trợ của ngôn ngữ mô hình hóa JML đối với ngôn ngữ lập trình Java trong việc đặc tả các ràng buộc ngữ nghĩa của hệ thống tại giai đoạn cài đặt chương trình phần mềm.

**2.2. Công cụ hỗ trợ JML**OpenJML là một công cụ để kiểm chứng (Kiểm tra tính nhất quán của mã nguồn và các đặc tả) của các chương trình Java. Nó được triển khai từ Ngôn ngữ mô hình hóa Java (JML) và được xây dựng bằng trình biên dịch OpenJDK. OpenJML có thể được sử dụng như một công cụ dòng lệnh (**Command-line Tool**) hoặc thông qua giao diện đồ họa (GUI) được tích hợp trên môi trường lập trình Eclipse.

Công cụ này được phát triển một cách mạnh mẽ từ năm 2009. Ban đầu, dự án chỉ đơn giản là một thử nghiệm để xem liệu OpenJDK có thể thay thế tốt cho trình phân tích cú pháp tùy chỉnh dựa trên ESC/Java và trình biên dịch MultiJava dựa trên các công cụ JML2. OpenJML đã thực thi được một số tác vụ sau:

* Phân tích cú pháp (**Parsing**) và kiểm tra kiểu (**TypeChecking**) khi kết hợp mã JML với mã Java tương ứng.
* Kiểm tra tĩnh (**Static**) mã nguồn và bản đặc tả. OpenJML chuyển các đặc tả Java + JML thành các điều kiện kiểm chứng, sau đó được kiểm tra bởi các bộ giải SMT.
* Kiểm tra kết quả của các khẳng định (**Assertion**) tại thời điểm chạy chương trình (**Runtime**) bằng cách biên dịch các đặc tả thành các tệp .class Java thông thường. OpenJML sử dụng trình biên dịch OpenJDK, tăng cường xử lý các tệp nguồn được bổ sung các ràng buộc vào mã nguồn Java tương ứng trong suốt quá trình thực thi chương trình.

Các bước chuẩn bị sử dụng OpenJML

**a. Cài đặt OpenJML**

1.Yêu cầu hệ thống:

- JDK (Java Development Kit) phiên bản 8 hoặc cao hơn.

2. Tải xuống OpenJML:

-Truy cập trang web chính thức của OpenJML và tải phiên bản mới nhất.

-Giải nén và cấu hình đường dẫn nếu cần.

3.Kiểm tra cài đặt:

-Mở terminal hoặc command prompt, chạy lệnh:

*openjml -version*

-Nếu hiển thị phiên bản, cài đặt đã thành công.

**b. Cấu hình IDE**

Eclipse:

- Cài đặt plugin OpenJML nếu có (hoặc sử dụng CLI để kiểm chứng).

- Đảm bảo dự án Java đang chạy phiên bản tương thích với JML.

**c. Sử dụng OpenJML để viết và kiểm chứng đặc tả** 1. Viết đặc tả JML

Đặc tả JML được thêm vào mã Java dưới dạng comment (/\*@ ... \*/ hoặc //@).

Các phần cơ bản trong đặc tả JML:

-Precondition (Điều kiện trước):

Mô tả điều kiện phải đúng trước khi thực hiện phương thức

-Postcondition (Điều kiện sau):

Đảm bảo kết quả trả về đúng như kỳ vọng

-Invariant (Bất biến):

Mô tả các điều kiện phải luôn đúng trong suốt vòng đời của đối tượng

Ví dụ:

*public class ATM {*

*private int balance;*

*/\*@ requires amount > 0;*

*@ ensures balance == \old(balance) - amount;*

*@\*/*

*public void withdraw(int amount) {*

*balance -= amount;*

*}*

*}*

2. Kiểm chứng mã với OpenJML

2.1 . Chạy OpenJML:

Sử dụng CLI:

*openjml --esc ATM.java*

⇒Lệnh trên sử dụng công cụ ESC (**Extended Static Checker**) của OpenJML để kiểm tra các đặc tả JML.

2.2 . Xử lý lỗi:

Nếu mã không tuân thủ đặc tả, OpenJML sẽ trả về thông báo lỗi chi tiết.

2.3 . Kết hợp kiểm thử (Runtime Assertion Checking):

Biên dịch mã với OpenJML để chèn các kiểm tra thời gian chạy:

*openjml -rac ATM.java*

*java ATM*

**d. Quy trình thực hiện**

1. Viết đặc tả JML cho các chức năng chính của hệ thống ATM (rút tiền, nạp tiền, kiểm tra số dư).

2. Sử dụng OpenJML để kiểm chứng tính đúng đắn của từng đặc tả.

3.Xử lý các lỗi logic hoặc vi phạm đặc tả nếu có.

**2.3 Một số nghiên cứu liên quan**

Đã có một vài các phương pháp và công cụ được đề xuất để đặc tả và kiểm chứng chương trình phần mềm. Trong mục này, bài báo trình bày và đánh giá một số nghiên cứu liên quan tới đặc tả và kiểm chứng bài toán rút tiền tự động (ATM). Các nghiên cứu này được chia thành hai hướng chính: (i) kiểm chứng thiết kế và (ii) kiểm chứng mã nguồn.

Hướng nghiên cứu kiểm chứng thiết kế sử dụng một số ngôn ngữ như OCB (ngôn ngữ đặc tả trung gian), UML (ngôn ngữ mô hình hóa thống nhất UML), Event-B, v.v. Các nghiên cứu này thường tập trung chuyển đổi mô hình thiết kế sang một ngôn ngữ trung gian gần với mã nguồn. Sau đó tiến hành kiểm chứng thông qua các phương pháp kinh điển như kiểm chứng mô hình (Model Checking) hoặc chứng minh định lý (Theorem Proving). Phương pháp này đòi hỏi chi phí tương đối tốn kém và sự thực thi bởi các chuyên gia.

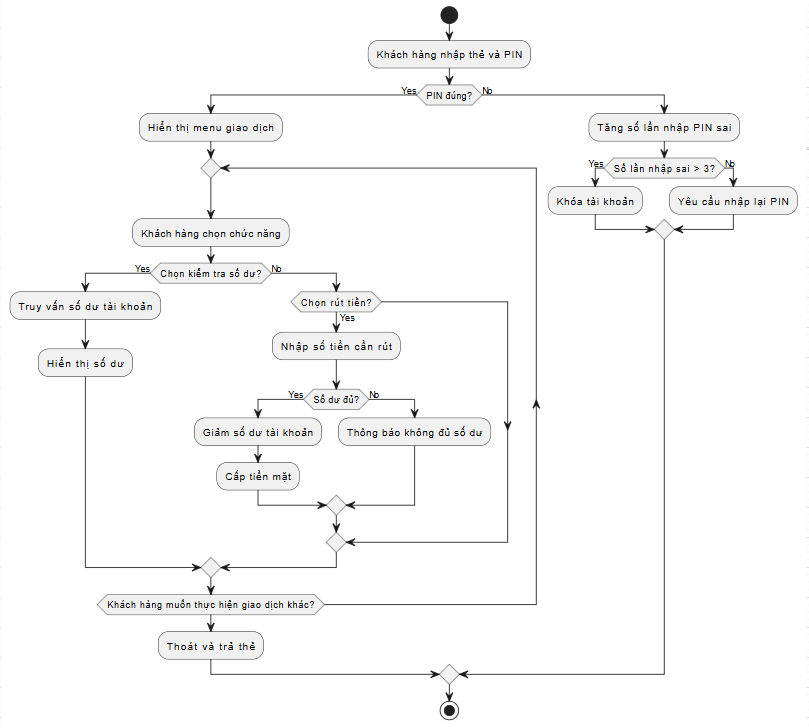
Hướng nghiên cứu kiểm chứng mã nguồn lại có xu hướng sử dụng các công cụ đã có sẵn như JLO (Java Logical Observer) hoặc các phương pháp sử dụng JML (các yêu cầu được đặc tả dưới dạng tiền/hậu điều kiện của các biến hay các phương thức trong chương trình), các yêu cầu sẽ được phát hiện vào thời điểm chạy chương trình.

Một trong các nghiên cứu rất gần với công việc trong bài báo này được tác giả Iqbal và các cộng sự thực hiện trong . Họ xây dựng mô hình ATM như một sơ đồ chuyển trạng thái để phục vụ mục tiêu kiểm chứng. Các yêu cầu của hệ thống được mô tả bằng Logic thời gian tuyến tính (LTL), kế tiếp họ sử dụng công cụ SPIN để kiểm tra xem mô hình có thỏa mãn các đặc tả hay không. Công cụ SPIN chấp nhận các mô hình được viết bằng ngôn ngữ PROMELA và các đặc tả bằng các công thức LTL. Tuy nhiên, với phương pháp này, các nhà phát triển phần mềm cần phải nắm rõ về Logic thời gian tuyến tính cũng như cách vận dụng nó vào đặc tả phần mềm. Trong khi, nghiên cứu của chúng tôi sử dụng JML để đặc tả (có cú pháp rất gần gũi với ngôn ngữ lập trình Java) và công cụ OpenJML có sẵn, tích hợp vào trong môi trường lập trình Eclipse là một trong những ưu điểm của nghiên cứu này.

**3. Hệ thống ATM**

**Mô tả hệ thống ATM**

**//activity diagram**



Sơ đồ hoạt động mô tả luồng giao dịch của khách hàng tại máy ATM:

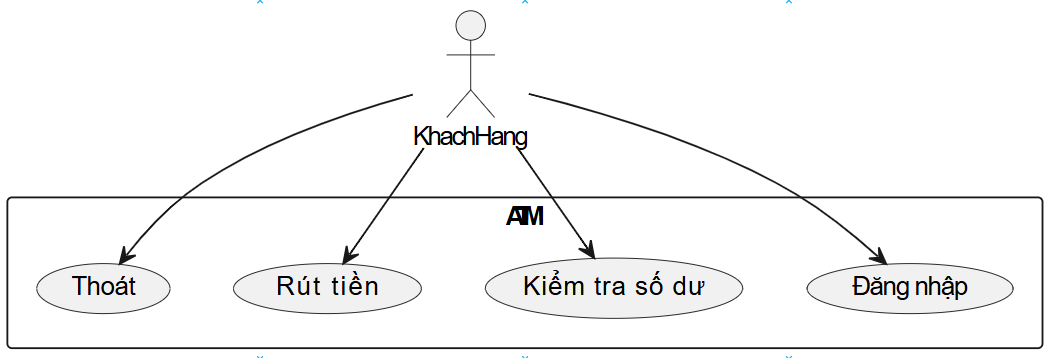
1. Bắt đầu:

Khách hàng nhập thẻ và mã PIN.

1. Xác thực PIN:  
   * Nếu đúng, hệ thống hiển thị menu giao dịch.
   * Nếu sai, tăng số lần nhập sai. Nếu quá 3 lần, tài khoản bị khóa.
2. Lựa chọn giao dịch:  
   * Kiểm tra số dư: Truy vấn và hiển thị số dư.
   * Rút tiền: Nhập số tiền cần rút, kiểm tra số dư.  
     + Nếu đủ, giảm số dư và cấp tiền mặt.
     + Nếu không đủ, thông báo lỗi.
3. Thoát:

Nếu không có giao dịch khác, ATM trả thẻ và kết thúc.

**//Biểu đồ Usecase**



#### **1. Tác nhân (Actor)**

* **Khách hàng**: Là người sử dụng máy ATM để thực hiện các giao dịch như đăng nhập, kiểm tra số dư, rút tiền và thoát khỏi hệ thống.

#### **2. Các Use Case (Chức năng của hệ thống)**

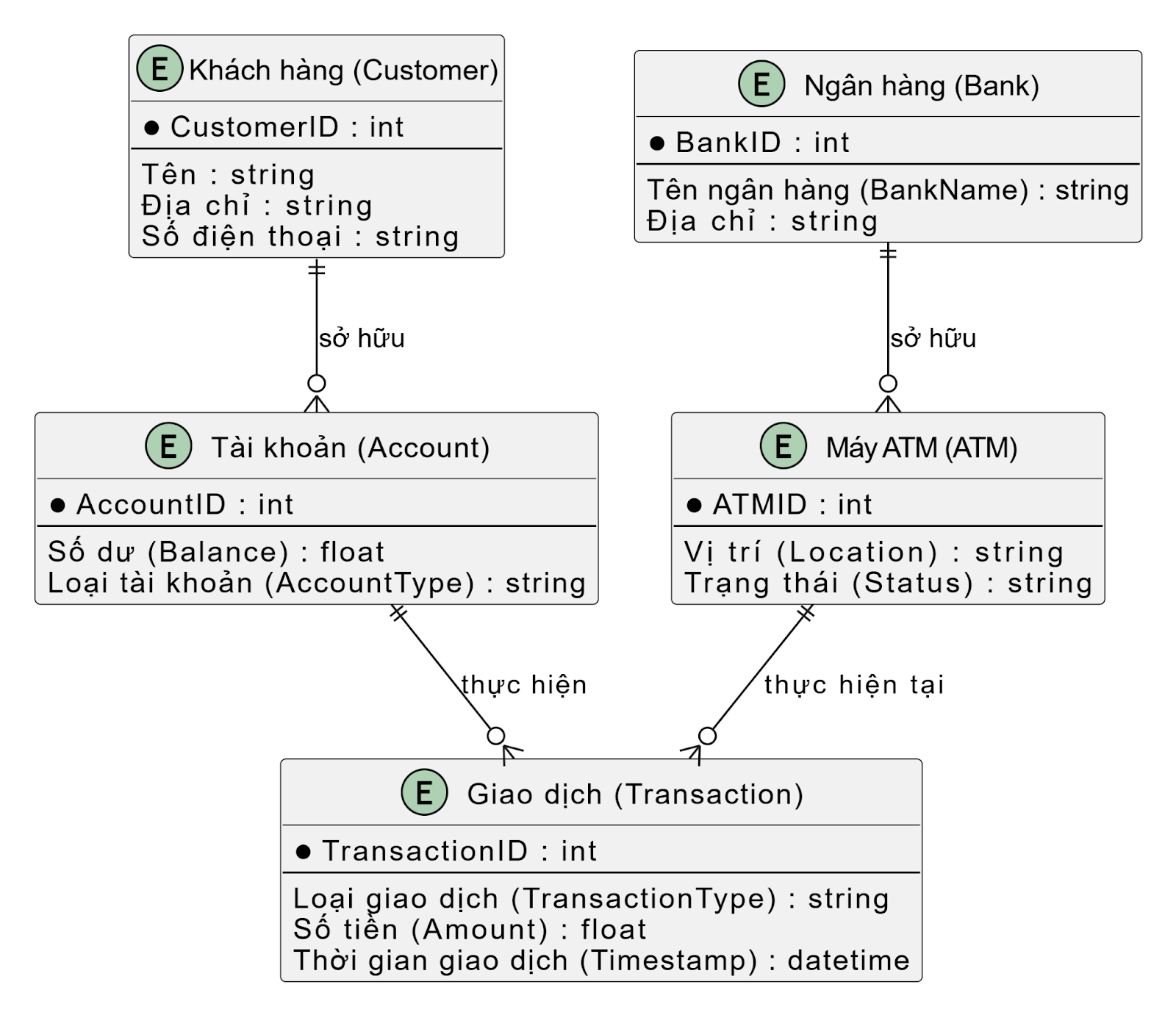
1. **Đăng nhập**
   * Khách hàng nhập mã PIN để truy cập tài khoản.
   * Hệ thống kiểm tra mã PIN với cơ sở dữ liệu ngân hàng.
   * Nếu đúng, hệ thống cho phép thực hiện giao dịch.
   * Nếu sai quá số lần quy định, tài khoản bị khóa.
2. **Kiểm tra số dư**
   * Khách hàng chọn chức năng kiểm tra số dư.
   * Hệ thống truy vấn cơ sở dữ liệu để lấy thông tin số dư hiện tại của tài khoản.
   * Hiển thị số dư lên màn hình ATM.
3. **Rút tiền**
   * Khách hàng nhập số tiền muốn rút.
   * Hệ thống kiểm tra số dư trong tài khoản:  
     + Nếu đủ tiền: Trừ số tiền trong tài khoản và cấp tiền mặt.
     + Nếu không đủ tiền: Hiển thị thông báo lỗi.
4. **Thoát**
   * Khách hàng chọn chức năng thoát.
   * Hệ thống kết thúc phiên làm việc và đưa máy ATM về trạng thái sẵn sàng cho người dùng tiếp theo.

#### **3. Quan hệ giữa các Use Case**

* **"Đăng nhập"** là bước bắt buộc để thực hiện các chức năng khác.
* **"Kiểm tra số dư"** và **"Rút tiền"** là các chức năng độc lập nhưng phụ thuộc vào tài khoản đã đăng nhập.
* **"Thoát"** có thể được thực hiện bất kỳ lúc nào để kết thúc phiên làm việc.

Bạn có muốn tôi bổ sung chi tiết nào khác không?

**// Biểu đồ ER**



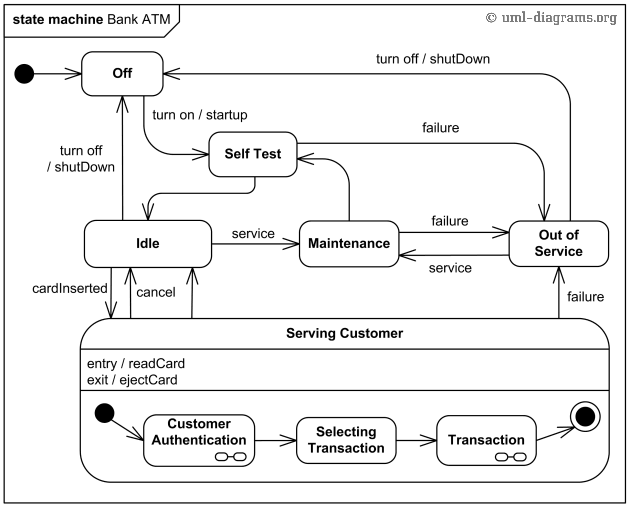
### **Thực thể chính:**

1. **Khách hàng (Customer):**
   * Thuộc tính: CustomerID, Tên, Địa chỉ, Số điện thoại.
2. **Tài khoản (Account):**
   * Thuộc tính: AccountID, Số dư (Balance), Loại tài khoản (AccountType).
3. **Giao dịch (Transaction):**
   * Thuộc tính: TransactionID, Loại giao dịch (TransactionType), Số tiền (Amount), Thời gian giao dịch (Timestamp).
4. **Máy ATM (ATM):**
   * Thuộc tính: ATMID, Vị trí (Location), Trạng thái (Status).
5. **Ngân hàng (Bank):**
   * Thuộc tính: BankID, Tên ngân hàng (BankName), Địa chỉ.

### **Mối quan hệ:**

1. **Khách hàng - Tài khoản:**
   * Một khách hàng có thể sở hữu nhiều tài khoản.
   * Một tài khoản thuộc về một khách hàng.
   * Quan hệ: **1:N**.
2. **Tài khoản - Giao dịch:**
   * Một tài khoản có thể thực hiện nhiều giao dịch.
   * Một giao dịch thuộc về một tài khoản.
   * Quan hệ: **1:N**.
3. **ATM - Giao dịch:**
   * Một máy ATM có thể thực hiện nhiều giao dịch.
   * Một giao dịch được thực hiện trên một máy ATM.
   * Quan hệ: **1:N**.
4. **Ngân hàng - Máy ATM:**
   * Một ngân hàng sở hữu nhiều máy ATM.
   * Một máy ATM thuộc về một ngân hàng.
   * Quan hệ: **1:N**.

**//state diagram**



-ATM ban đầu được tắt. Sau khi bật nguồn, ATM thực hiện hành động khởi động và chuyển sang trạng thái **Tự kiểm tra (Self Test)**. Nếu kiểm tra thất bại, ATM chuyển sang trạng thái **Ngoài dịch vụ (Out of Service)**, nếu không sẽ có một chuyển tiếp không kích hoạt (triggerless transition) sang trạng thái **Chờ (Idle)**. Trong trạng thái này, ATM chờ khách hàng tương tác.

-Trạng thái của ATM thay đổi từ **Chờ (Idle)** sang **Phục vụ khách hàng (Serving Customer)** khi khách hàng đưa thẻ ngân hàng hoặc thẻ tín dụng vào đầu đọc thẻ của ATM. Khi vào trạng thái **Phục vụ khách hàng (Serving Customer)**, hành động nhập (entry action) **đọc thẻ (Read Card)** được thực hiện. Lưu ý rằng, chuyển tiếp từ trạng thái **Phục vụ khách hàng (Serving Customer)** trở lại trạng thái **Chờ (Idle)** có thể được kích hoạt bởi sự kiện **hủy (Cancel)**, vì khách hàng có thể hủy giao dịch bất cứ lúc nào.

-Trạng thái **Phục vụ khách hàng (Serving Customer)** là một trạng thái tổng hợp với các trạng thái con tuần tự: **Xác thực khách hàng (Customer Authentication)**, **Chọn giao dịch (Selecting Transaction)** và **Giao dịch (Transaction)**. Trong đó, **Xác thực khách hàng (Customer Authentication)** và **Giao dịch (Transaction)** cũng là các trạng thái tổng hợp, được biểu diễn bằng **Biểu tượng chỉ báo phân rã** (**Decomposition Indicator Icon**) bị ẩn. Trạng thái **Phục vụ khách hàng (Serving Customer)** có một chuyển tiếp không kích hoạt (triggerless transition) trở lại trạng thái **Chờ (Idle)** sau khi giao dịch kết thúc. Trạng thái này cũng có một **Hành động thoát** (**Exit Action**), **Trả thẻ (Eject Card)** để trả lại thẻ của khách hàng khi rời khỏi trạng thái, bất kể nguyên nhân nào gây ra sự chuyển đổi khỏi trạng thái này.

**4. Các yêu cầu kiểm chứng về mô hình ATM**

Hệ thống mô phỏng ATM được phát triển nhằm hỗ trợ người dùng thực hiện các giao dịch tài chính như:

* Đăng nhập bằng mã PIN.
* Kiểm tra số dư tài khoản.
* Rút tiền, gửi tiền một cách an toàn.
* Đăng xuất khỏi hệ thống để bảo mật thông tin.

Tuy nhiên, trong các hệ thống tài chính, tính chính xác, bảo mật và toàn vẹn dữ liệu đóng vai trò cực kỳ quan trọng. Vì vậy, việc kiểm chứng hình thức là cần thiết để đảm bảo rằng hệ thống tuân thủ chặt chẽ các quy tắc thiết kế, tránh lỗi logic hoặc bảo mật ngay từ giai đoạn phát triển.

Phần 4 của báo cáo sẽ tập trung vào phân tích và kiểm chứng hình thức hệ thống theo 5 tiêu chí chính:

1. Tính đúng đắn (Correctness)  
 2. Tính bảo mật (Security)  
 3. Tính toàn vẹn dữ liệu (Data Integrity)

4. Tính khả dụng (Availability)  
 5. Tính bảo trì (Maintainability)

Bằng cách áp dụng OpenJML, chúng tôi có thể xác minh tính chính xác của hệ thống thông qua các yêu cầu kiểm chứng hình thức dưới dạng hợp đồng JML (Java Modeling Language)

### **4.1 Tính đúng đắn (Correctness) Mục tiêu:**

Hệ thống phải hoạt động chính xác theo yêu cầu đặt ra. Điều này bao gồm:

* Đảm bảo đăng nhập đúng khi nhập mã PIN hợp lệ.
* Khi nhập mã PIN sai, số lần thử (loginAttempts) phải tăng lên nhưng không vượt quá giới hạn.
* Khi đăng nhập thành công, hệ thống phải gán đúng currentAccountNumber.
* Khi rút tiền, gửi tiền, số dư tài khoản phải thay đổi chính xác.

#### **Kiểm chứng hình thức**

* Khi hệ thống khởi động:

*//@ ensures getLoginAttempts() == 0;*

*//@ ensures !userDung;*

*public ATM() {*

*loginAttempts = 0;*

*userDung = false;*

*}*

* Đăng nhập thành công:

*//@ requires bankDatabase.ktUser(pin);*

*//@ ensures userDung;*

* Đăng nhập thất bại:

*//@ requires !bankDatabase.ktUser(pin);*

*//@ ensures loginAttempts == \old(loginAttempts) + 1;*

* Rút tiền đúng quy tắc:

*//@ requires amount > 0;*

*//@ requires bankDatabase.getSoDu(currentAccountNumber) >= amount;*

*//@ ensures bankDatabase.getSoDu(currentAccountNumber) == \old(bankDatabase.getSoDu(currentAccountNumber)) - amount;*

### **4.2 Tính bảo mật (Security)**

#### **Mục tiêu:**

* Hệ thống phải bảo vệ **thông tin tài khoản**.
* Ngăn chặn **truy cập trái phép** vào tài khoản của người khác.
* Giới hạn số lần nhập sai mã PIN.

#### **Kiểm chứng hình thức**

* **Không cho phép lấy thông tin số dư khi chưa đăng nhập:**

*//@ requires userDung;*

*//@ ensures \result == bankDatabase.getSoDu(currentAccountNumber);*

* **Giới hạn số lần nhập PIN sai:**

*//@ requires loginAttempts < 3;  
 //@ ensures loginAttempts == \old(loginAttempts) + 1;*

* **Tài khoản bị khóa sau 3 lần nhập sai:**

*//@ requires loginAttempts >= 3;*

*//@ ensures accountLocked;*

### **4.3 Tính toàn vẹn dữ liệu (Data Integrity)**

#### **Mục tiêu**

* Dữ liệu tài khoản không bị sai lệch hoặc mất mát sau mỗi giao dịch.
* Khi rút tiền hoặc gửi tiền, số dư phải cập nhật chính xác.

#### **Kiểm chứng hình thức**

* Tránh lỗi thao tác trên tài khoản không tồn tại:

*//@ requires getAccount(userAccountNumber) != null;*

* Rút tiền không làm số dư âm:

*//@ requires bankDatabase.getSoDu(currentAccountNumber) >= amount;*

* Gửi tiền cập nhật đúng số dư:  
    
    *//@ ensures bankDatabase.getSoDu(currentAccountNumber) ==\old(bankDatabase.getSoDu(currentAccountNumber)) + amount;*

### **4.4 Tính khả dụng (Availability)**

#### **Mục tiêu**

* Hệ thống luôn đáp ứng nhanh chóng, không bị gián đoạn.
* Không xảy ra lỗi nghiêm trọng như NullPointerException.

#### **Kiểm chứng hình thức**

* Trước khi truy xuất tài khoản, phải kiểm tra tài khoản tồn tại:  
    
  *//@ requires bankDatabase.getAccount(currentAccountNumber) != null;*
* Nếu rút tiền vượt quá số dư, giao dịch không được thực hiện:

*//@ requires bankDatabase.getSoDu(currentAccountNumber) >= amount;*

*//@ ensures \result == true ==> (bankDatabase.getSoDu(currentAccountNumber) == \old(bankDatabase.getSoDu(currentAccountNumber)) - amount);***4.5 Tính bảo trì (Maintainability)**

#### **Mục tiêu**

* Hệ thống dễ dàng bảo trì và mở rộng.
* Đảm bảo mã nguồn dễ đọc, dễ hiểu, tuân theo nguyên tắc thiết kế tốt.

#### **Kiểm chứng hình thức**

* Xác định bất biến hệ thống:

*//@ invariant getLoginAttempts() >= 0 && getLoginAttempts() <= 3;*

*//@ invariant userDung ==> (currentAccountNumber >= 0);*

*//@ invariant !userDung ==> (currentAccountNumber == -1);*

* Tách biệt các phương thức pure và impure:

*//@ pure*

*public boolean ktUser(int userPIN) {*

*Account userAccount = getAccountpin(userPIN);*

*return userAccount != null && userAccount.validatePIN(userPIN);*

*}*

**5.Cài đặt mô hình kiểm chứng**

**5.1 Đặc tả các chức năng chính bằng JML**

Như đã đề cập, hệ thống ATM khi triển khai sẽ đặt ra một số vấn đề cần giải quyết như: Làm thế nào để ngăn chặn khách hàng rút số tiền vượt quá số dư thực sự của tài khoản? Số tiền trong hạn mức một giao dịch rút tiền là bao nhiêu? Hạn mức rút tiền của một tài khoản trong một ngày là bao nhiêu?

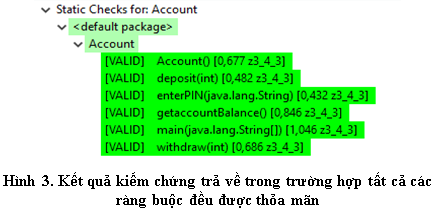
Để giải quyết những câu hỏi này, người phát triển hệ thống cần có một chính sách đặc tả chi tiết, chính xác cũng như cần phải kiểm chứng được sự tuân thủ của quá trình thực thi đối với các đặc tả này. Trong Tiểu mục này, bài báo sẽ mô tả các ràng buộc ngữ nghĩa của một số phương thức quan trọng có trong hệ thống và bản đặc tả của nó thông qua ngôn ngữ JML như được trình bày trong Bảng 2

**Bảng 2. Đặc tả các ràng buộc cần kiểm chứng của hệ thống ATM**

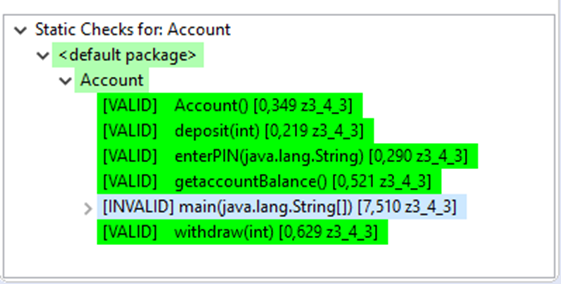
|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Đối tượng đặc tả** | **Mô tả ràng buộc bằng ngôn ngữ tự nhiên** | **Đặc tả bằng JML** |
| **1.** | **AuthenticateUser(int pin)(lớp ATM)** | **PIN phải là số hợp lệ (0-9999).**  **- Số lần thử không vượt quá 2.**  **- Nếu PIN đúng, xác thực thành công, reset số lần thử và cập nhật tài khoản.**  **- Nếu sai, tăng số lần thử.**  **- Sau 3 lần sai, khóa 30 giây và reset số lần thử.** | *@ requires pin >= 0 && pin <= 9999;*  *@ requires loginAttempts <= 2;*  *@ ensures \result == bankDatabase.ktUser(pin);*  *@ ensures userDung == bankDatabase.ktUser(pin);*  *@ ensures loginAttempts == (bankDatabase.ktUser(pin) ? 0 : \old(loginAttempts) + 1);*  *@ ensures bankDatabase.ktUser(pin) ==> currentAccountNumber == bankDatabase.getaccpin(pin);*  *@ ensures bankDatabase.ktUser(pin) ==> AdminCheck == bankDatabase.getadmin(pin);*  *@ assignable loginAttempts, userDung, currentAccountNumber, AdminCheck, screen.\*;*  *@ also*  *@ public exceptional\_behavior*  *@ requires pin >= 0 && pin <= 9999;  @ requires loginAttempts > 2;*  *@ signals (InterruptedException e) true;  @ ensures loginAttempts == 0;*  *@ assignable loginAttempts, screen.\*;*  *@\*/* |
| **2.** | **Transaction(int amount) (lớp Withdrawal)** | **- Số tiền rút phải dương và nhỏ hơn hoặc bằng số dư tài khoản.**  **- ATM phải có đủ tiền.**  **- Nếu thành công, giảm số dư và thông báo rút tiền.**  **- Nếu không đủ số dư hoặc ATM không đủ tiền, hiển thị lỗi mà không thay đổi số dư.** | *@ requires amount > 0 && amount <= bankDatabase.getSoDu(getAccountNumber());*  *@ requires cashDispenser.ktSoDuATM(amount);  @ ensures bankDatabase.getSoDu(getAccountNumber()) == \old(bankDatabase.getSoDu(getAccountNumber())) - amount; @ ensures**bankDatabase.getTongDu(getA***ccou***ntNumber()) == \old(bankDatabase.getTongDu(getAccountNumber())) - amount;*  *@ ensures screen.messageJLabel7.getText().equals("\nTiền của bạn đã được rút Vui lòng nhận tiền!"); @ ensures screen.messageJLabel8.getText().equals("Available Balance: " + bankDatabase.getSoDu(getAccountNumber()) + "vnd");  @ assignable bankDatabase.getSoDu(getAccountNumber()), bankDatabase.getTongDu(getAccountNumber()), screen.messageJLabel7, screen.messageJLabel8;*  *@ also<br> @ public exceptional\_behavior*  *@ requires amount > bankDatabase.getSoDu(getAccountNumber()); @ ensures screen.messageJLabel7.getText().equals("\nSố dư trong tài khoản không đủ.\n\nHãy chọn số nhỏ hơn.");  @ ensures screen.messageJLabel8.getText().equals("Available Balance: " + bankDatabase.getSoDu(getAccountNumber()) + "vnd");  @ assignable screen.messageJLabel7, screen.messageJLabel8;*  *@ also  @ public exceptional\_behavior  @ requires amount <= bankDatabase.getSoDu(getAccountNumber()) && !cashDispenser.ktSoDuATM(amount);  @ ensures screen.messageJLabel7.getText().equals("\nXin lỗi! SỐ tiền trong ATM không đủ.\nVui long nhap so tien nho hon."); @ ensures screen.messageJLabel8.getText().equals("Available Balance: " + bankDatabase.getSoDu(getAccountNumber()) + "vnd");  @ assignable screen.messageJLabel7, screen.messageJLabel8;@\*/* |
| **3.** | **Execute() (lớp BalanceInquiry)** | **- Tài khoản phải hợp lệ.**  **- Hiển thị số dư khả dụng và tổng dư của tài khoản hiện tại.**  **- Không thay đổi số dư sau khi kiểm tra.** | *@ requires getAccountNumber() > 0;  @ ensures screen.messageJLabel.getText().equals("Thông Tin Số Dư:"); @ ensures screen.createBalanceGUI(getBankDatabase().getSoDu(getAccountNumber()), getBankDatabase().getTongDu(getAccountNumber()));  @ ensures \old(getBankDatabase().getSoDu(getAccountNumber())) == getBankDatabase().getSoDu(getAccountNumber());*  *@ ensures \old(getBankDatabase().getTongDu(getAccountNumber())) == getBankDatabase().getTongDu(getAccountNumber());  @ assignable screen.\*; @\** |

**5.2 Kiểm chứng tự động một số tính chất của hệ thống ATM với OpenJML**

Trong tiểu mục trên, báo cáo đã đặc tả các ràng buộc ngữ nghĩa của hệ thống bằng ngôn ngữ JML. Tuy nhiên, các nhà phát triển phần mềm cần phải chỉ ra rằng, bản đặc tả này được tuân thủ bởi bản cài đặt chương trình phần mềm. OpenJML được phát triển dành riêng cho JML trong kiểm chứng các đặc tả tại giai đoạn cài đặt phần mềm. Sau quá trình thực nghiệm, tác giả đã cài đặt và kiểm chứng thành công một số ràng buộc của hệ thống ATM. Đây là một trong những kết quả lý tưởng được mong muốn của nhà phát triển phần mềm.

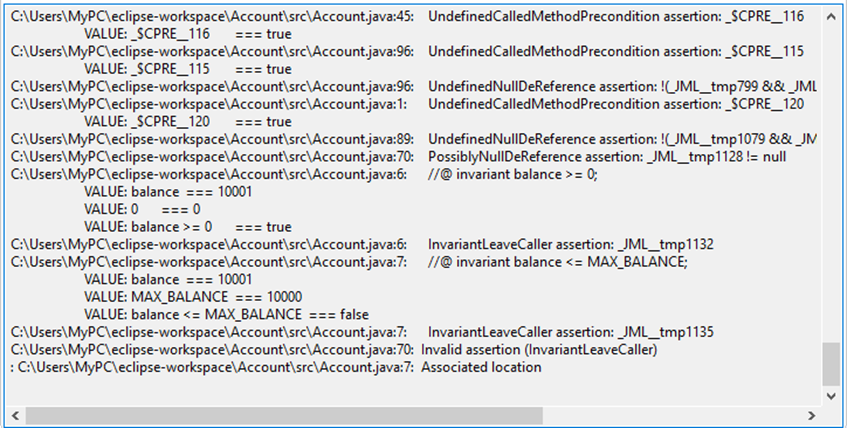


Tuy nhiên, trong rất nhiều trường hợp, kết quả kiểm chứng không như mong muốn, tức là các ràng buộc của hệ thống bị vi phạm, khi đó OpenJML sẽ trả lại cho chúng ta một phản ví dụ (*CounterExample*), dựa trên phản ví dụ này, người phát triển dễ dàng lần vết để sửa lỗi sai trong chương trình của mình



**Hình 4. Minh họa kết quả kiểm chứng Hệ thống ATM khi có ràng buộc bị vi phạm**

Hình 4 và 5 minh họa kết quả kiểm chứng và phản ví dụ trong trường hợp ràng buộc bị vi phạm. Trong đó, biến **MAX\_BALANCE** vượt quá giá trị giới hạn cho phép là 10000 dẫn đến trường hợp INVALID như được thông báo.



**Hình 5. Phản ví dụ mô tả vị trí trong chương trình xảy ra vi phạm ràng buộc**

## **5.3 Kết quả dự kiến**

Khi kiểm chứng mô hình ATM bằng OpenJML, ta mong đợi nhận được các kết quả tương ứng với năm tiêu chí chính:

### **1. Tính đúng đắn (Correctness)**

* Kỳ vọng:  
  + Mọi chức năng như xác thực người dùng, kiểm tra số dư, rút tiền hoạt động đúng như thiết kế.
  + Ràng buộc bất biến (invariant) về số dư tài khoản được bảo toàn, không có trường hợp rút tiền vượt mức số dư.
  + Giao dịch chỉ thực hiện khi xác thực thành công.
  + Sau khi rút tiền, số dư tài khoản phải giảm đúng số tiền yêu cầu.
* Lỗi có thể xảy ra:  
  + Rút tiền dù số dư không đủ (vi phạm điều kiện soDu >= soTienRut).
  + Xác thực người dùng thành công dù nhập sai mã PIN.
  + Giá trị số dư sau giao dịch không chính xác.
  + Không xử lý đúng trạng thái thẻ bị khóa khi nhập sai PIN nhiều lần.

### **2. Tính bảo mật (Security)**

* Kỳ vọng:  
  + Người dùng không thể rút tiền hoặc kiểm tra số dư mà không xác thực thành công.
  + Không tiết lộ thông tin tài khoản của người khác.
  + Không lưu trữ hoặc hiển thị mã PIN sau khi nhập.
* Lỗi có thể xảy ra:  
  + Truy xuất số dư mà không qua xác thực (getSoDu() không kiểm tra quyền truy cập).
  + Mã PIN hiển thị trực tiếp trên giao diện hoặc lưu trữ không mã hóa.
  + Thực hiện giao dịch khi thẻ bị khóa.

### **3. Tính toàn vẹn dữ liệu (Data Integrity)**

* Kỳ vọng:  
  + Số dư không bị thay đổi ngoài các thao tác hợp lệ (rút tiền, gửi tiền).
  + Hệ thống không xảy ra lỗi khi giao dịch bị gián đoạn.
* Lỗi có thể xảy ra:  
  + Không khôi phục trạng thái ban đầu nếu giao dịch thất bại.
  + Cùng một tài khoản có thể bị trừ tiền hai lần nếu giao dịch thực hiện song song mà không có cơ chế khóa dữ liệu.

### **4. Tính khả dụng (Availability)**

* Kỳ vọng:  
  + Hệ thống vẫn hoạt động khi nhiều người dùng truy cập cùng lúc.
  + Xử lý các trường hợp ngoại lệ (thẻ bị nuốt, ATM hết tiền, mất kết nối).
* Lỗi có thể xảy ra:  
  + Không có xử lý khi ATM hết tiền.
  + Không trả lại thẻ khi giao dịch thất bại.
  + Hệ thống bị treo nếu nhập sai mã PIN nhiều lần.

### **5. Tính bảo trì (Maintainability)**

* Kỳ vọng:  
  + Code dễ mở rộng và sửa lỗi.
  + Các phương thức và bất biến được mô tả rõ ràng bằng OpenJML.
* Lỗi có thể xảy ra:  
  + Thiếu chú thích (//@ requires, //@ ensures, //@ invariant).
  + Code khó hiểu, khó mở rộng khi thêm chức năng mới.

**5.4 Kết quả thực nghiệm**

### **1. Ghi nhận tiến độ và lý do chưa hoàn thành**

Hiện tại, quá trình kiểm chứng mô hình ATM bằng OpenJML chưa hoàn tất do:

* Một số lỗi kiểm chứng chưa được khắc phục hoàn toàn, cần điều chỉnh thuật toán và chú thích.

### **2. Ghi nhận kết quả sơ bộ**

Nếu bạn đã thử nghiệm một số phần nhưng chưa hoàn chỉnh, có thể mô tả kết quả tạm thời:

* Đã kiểm chứng thành công các điều kiện bất biến (invariant) của tài khoản và số dư.
* Còn lỗi liên quan đến phạm vi truy cập biến (private với public).

**6. Kết luận và hướng phát triển**

Đặc tả và kiểm chứng hệ thống phần mềm là các giai đoạn quan trọng, quyết định chất lượng của hệ thống phần mềm. Trong bài báo này, chúng tôi đã sử dụng phương pháp hình thức để đặc tả và kiểm chứng các ràng buộc ngữ nghĩa của hệ thống ATM. Cụ thể, chúng tôi sử dụng ngôn ngữ mô hình hóa cho Java (JML) để mô tả các ràng buộc của hệ thống và tiến hành kiểm thử các tính chất này một cách tự động bằng phần mềm OpenJML tích hợp trên môi trường lập trình Eclipse. Phương pháp mà chúng tôi sử dụng cho thấy được độ tin cậy của kết quả kiểm chứng.

Đã có nhiều ngôn ngữ đặc tả (OCL, Event –B, v.v.) ra đời và phát triển. Việc lựa chọn ngôn ngữ và công cụ nào sử dụng trong các giai đoạn đặc tả và kiểm chứng phụ thuộc vào từng miền ứng dụng cụ thể, kiến thức cũng như kinh nghiệm của nhà phát triển phần mềm. Có thể nói, không có phương pháp hay công cụ kiểm chứng nào phù hợp với tất cả các chương trình phần mềm. Trong bài báo này, tác giả sử dụng JML kết hợp cùng với OpenJML và cho thấy đây cũng là một trong những phương pháp hiệu quả, giúp tăng tính tin cậy cũng như giảm thời gian và công sức phát triển hệ thống phần mềm.

Trong bài báo này, chúng tôi mới chỉ có những nghiên cứu ban đầu về việc đặc tả và kiểm chứng các ràng buộc ngữ nghĩa của hệ thống phần mềm; minh họa trực tiếp trên Hệ thống rút tiền tự động ATM. Trong tương lai, chúng tôi sẽ tiếp tục thực nghiệm phương pháp này trên các hệ thống có tính chất phức tạp và có tính chuyên biệt miền hơn (các hệ thống mạng quân sự, hệ thống máy bay không người lái, v.v.).

**7. Tài liệu tham khảo**

1. Abrial Jean-Raymond (2010), *Modeling in Event-B: system and software engineering*, Cambridge University Press.

2. Aldajani Mansour A., and Hesham K. Alfares (2009), *“Location of banking automatic teller machines based on convolution”*. Computers & Industrial Engineering, Elsevier, vol. 57, no. 4, pp. 1194-1201.

3. Cabot Jordi, and Martin Gogolla (2012). *“Object constraint language (OCL): a definitive guide”*. International School on Formal Methods for the Design of Computer, Communication and Software Systems.

4. Cheon Yoonsik, and Gary T. Leavens (2002), *A runtime assertion checker for the Java Modeling Language (JML)*, 2002.

5. Cok David R (2011), *“OpenJML: JML for Java 7 by extending OpenJDK”*. NASA Formal Methods Symposium, Springer, Berlin, Heidelberg, pp. 472-479.

6. Cok David R., and Joseph R. Kiniry (2004), *“Esc/java2: Uniting esc/java and jml”*. International Workshop on Construction and Analysis of Safe, Secure, and Interoperable Smart Devices. Springer, Berlin, Heidelberg, pp. 108-128.

7. Du Bousquet Lydie, et al (2010), *“Reusing a JML specification dedicated to verification for testing, and vice-versa: case studies”*, Journal of Automated Reasoning, vol. 45, no. 4, pp. 415-435.

8. Iqbal, Ikhwan Mohammad and Adzkiya, Dieky and Mukhlash, Imam (2017). *“Formal verification of automated teller machine systems using SPIN”,* AIP Conference Proceedings, AIP Publishing LLC, p. 020045.

9. Jorgensen Paul C (2013), “*Software testing: a craftsman's approach”*, Auerbach Publications.

10. Maropoulos, Paul G and Ceglarek, Dariusz (2010), “Design verification and validation in product lifecycle”, CIRP annals, Elsevier, Vol. 59, no. 2, pp. 740 - 759.

11. Paul & Ryan (2010). “Java wars: IBM joins OpenJDK as Oracle shuns Apache Harmony”, Ars Technica. Archived from the original on 19.

12. Smith Graeme (2012), “The Object-Z specification language”, Springer Science & Business Media, Vol. 1.

13. Sieber, Kurt (2013). The foundations of program verification. Springer-Verlag.

14. Ullman R., and R. Dunn (1982). Quality Assurance for Computer Soft-ware, New York: McGraw-Hill.

15. Vogel & Lars (2014), Eclipse IDE tutorial, Vogella.com.