|  |
| --- |
| Hà Nội - 2016  **TRƯỜNG ĐẠI HỌC QUỐC GIA HÀ NỘI**  **TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**  🙞🕮🙜  **BÁO CÁO BÀI TẬP NHÓM**  **MÔN: Các Vấn Đề Hiện Đại Của Phát Triển Phần Mềm**  ***Đề tài:***  ***Kiểm Thử Dùng AspectJ***  *Giảng viên hướng dẫn:* TS. Nguyễn Thị Huyển Châu  *Sinh viên thực hiện :* Nhóm 1  Cao Thị Vân Anh  Nguyễn Nam Chung  Hoàng Thụy  Trần Thị Huyền Trang |

**MỤC LỤC**

[LỜI NÓI ĐẦU](#_Toc448547656)

[DANH MỤC CÁC THUẬT NGỮ VIẾT TẮT](#_Toc448547657)

[Chương I: Cơ sở lý thuyết 1](#_Toc448547658)

[1. Giới thiệu về AOP (Lập trình hưởng khía cạnh) 1](#_Toc448547659)

[1.1 Tổng quan về AOP 1](#_Toc448547660)

[1.2 Các đặc điểm của AOP 3](#_Toc448547661)

[1.3 Điểm mạnh, điểm yếu của AOP 5](#_Toc448547662)

[1.4 Ứng dụng AOP 6](#_Toc448547663)

[2. Ngôn ngữ AspectJ 7](#_Toc448547664)

[2.1 Giới thiệu chung về AspectJ 7](#_Toc448547665)

[2.2 Các khái niệm cơ bản 7](#_Toc448547666)

[2.3 Lập trình với AspectJ 9](#_Toc448547667)

[3. Sử dụng AspectJ cho kiểm thử phần mềm 10](#_Toc448547668)

[3.1 Giới thiệu về kỹ thuật Design by Contract(DBC) 10](#_Toc448547669)

[3.2 Contract checking 12](#_Toc448547670)

[4. So sánh AspectJ với các công nghệ khác 14](#_Toc448547671)

[4.1 Nhận xét về AspectJ 14](#_Toc448547672)

[4.2 So sánh AspectJ với các công nghệ khác trong mục đích kiểm thử: 15](#_Toc448547673)

[Chương 2: Phân tích và thiết kế ứng dụng 23](#_Toc448547674)

[1. Phát biểu bài toán 23](#_Toc448547675)

[2. Thiết kế hệ thống 24](#_Toc448547676)

[2.1. Biểu đồ Usecase của hệ thống 24](#_Toc448547677)

[2.2 Mô tả biểu đồ tuần tự của quá trình mua và bán cổ phiếu: 28](#_Toc448547678)

[2.3 Xác định các lớp 30](#_Toc448547679)

[3. Cài đặt và phát triển ứng dụng 31](#_Toc448547680)

[Chương 3: Tích hợp AspectJ để kiểm thử ứng 34](#_Toc448547681)

[1. Tạo các AspectJ class 34](#_Toc448547682)

[2. Tạo các kịch bản kiểm thử AspectJ cho demo mua bán chứng khoán 34](#_Toc448547683)

[2.1 Xác định JoinPoint 34](#_Toc448547684)

[2.2 Tạo các PointCut và Advice để thực thi các Pointcut tương ứng với các JointPoint đã xác định 35](#_Toc448547685)

[Chương 4: Kết luận 40](#_Toc448547686)

[TÀI LIỆU THAM KHẢO 41](#_Toc448547687)

# **LỜI NÓI ĐẦU**

Việc chuyển đổi các yêu cầu người dùng vào trong một hệ thống phần mềm bao giờ cũng rất khó khăn, mặc dù hiện nay đã có rất nhiều phương pháp tiếp cận như lập trình hướng đối tượng, hướng thành phần, các design pattern.... Chúng cũng đã giải quyết được một số vấn đề những vẫn chưa có một phương pháp nào thảo mãn việc giải quyết các yêu cầu đan xen ở mức hệ thống. Các nhà nghiên cứu lý thuyết đã đưa ra mô hình AOP để giải quyết các vấn đề mà mô hình lập trình hiện tại chưa đáp ứng được hoặc đáp ứng được nhưng việc thực hiện nó quá phức tạp. Lập trình hướng khía cạnh (Aspect Oriented Programming - AOP) là một kiểu lập trình mới nhanh chóng thu hút được các nhà phát triển trong giới công nghệ thông tin. AOP là một mô hình lập trình tách biệt các chức năng phụ với logic nghiệp vụ của chương trình chính. Các chức năng phụ nằm rải rác xuyên suốt trong hệ thống được tách thành các đơn vị duy nhất, gọi là aspect(khía cạnh). Một aspect là một đơn vị modun cho sự thi hành cắt ngang chương trình. Nó đóng gói các hành vi mà ảnh hưởng đến nhiều lớp vào các modun có khả năng sử dụng lại. Đây là một phương pháp lập trình phát triển dựa trên lập trình hướng đối tượng. Đề tài về AOP cũng rất còn mới nên nhóm quyết định lấy tài “Kiểm thử với AspectJ” để tìm hiểu.

# **DANH MỤC CÁC THUẬT NGỮ VIẾT TẮT**

|  |  |
| --- | --- |
| Thuật ngữ viết tắt | Viết đầy đủ |
| AOP | Aspect-oriented programming |
| CP | Cổ Phiếu |
| DBC | Design by contract |
| HĐH | Hệ điều hành |
| JDK | Java Development Kit |

# 

# **Chương I: Cơ sở lý thuyết**

## **1. Giới thiệu về AOP (Lập trình hưởng khía cạnh)**

### ***1.1 Tổng quan về AOP***

AOP được xây dựng trên các phương pháp lập trình hiện tại như lập trình hướng đối tượng, lập trình có cấu trúc, bổ sung thêm các khái niệm và cấu trúc để modun hóa các chức năng cắt ngang của hệ thống (crosscutting concern). Với AOP, các quan hệ cơ bản sử dụng các phương pháp cơ bản. Nếu sử dụng OOP sẽ thực thi các quan hệ cơ bản dưới hình thức lơp (class). Các aspect trong hệ thống đóng gói các chức năng cắt ngang hệ thống lại với nhau. Chúng sẽ quy định các modun khác nhau gắn kết với nhau để hình thành nên hệ thống cuối cùng.

Nền tảng cơ bản của AOP với OOP là cách quản lý các chức năng cắt ngang hệ thống. Việc thực thi của từng chức năng cắt ngang AOP bỏ qua các hành vi được tích hợp vào nó. AOP có thể xem là một sự bổ sung cho OOP, cho phép cũng giải quyết các bài toán phức tạp hơn và hiệu quả hơn.

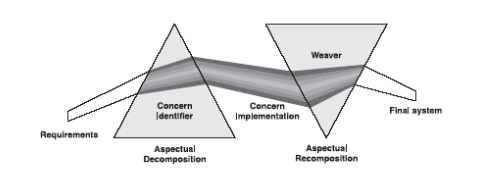
Vấn đề cốt lõi của AOP là cho phép chúng ta thực hiện các vấn đề riêng biệt một cách linh hoạt và kết hợp chúng lại. Nhờ các mã được tách riêng, vẫn đề đan xen trở nên dễ kiểm soát hơn. Các aspect của hệ thống có thể thay đổi, thêm hoặc xóa lúc biên dịch và có thể tái sử dụng. Một dạng biên dịch đặc biệt có tên là Aspect Weaver thực hiện kết hợp các thành phần riêng lẻ lại thành hệ thống hợp nhất.

AOP gồm 3 bước thực hiện:

- Aspectual decomposition: Trong bước này chúng ta phân tách các yêu cầu nhằm xác định các concern lõi và concern đan xen. Các concern lõi được tách ra khỏi các concern đan xen.

- Concern Implementation: Thực thi các concern một cách độc lập

- Aspectual Recomposotion: Trong bước này chúng ta chỉ ra các quy lật kết hợp bằng cách tạp ra các aspect. Quá trình này còn được gọi là quá trình dệt mã, sử dụng các thông tin trong aspect để cấu thành hệ thống đích.



Hình 1: Các giai đoạn pháp triển sử dụng phương pháp AOP

Giống như các phương pháp lập trình khác, việc thực hiện AOP bao gồm hai phần: đặc tả ngôn nhữ mô tả cú pháp và cấu trúc ngôn ngữ, biên dịch kiểm tra sự đứng đắn của mã lệnh theo đặc tả ngôn ngữ và cấu trúc ngôn ngữ, biên dịch kiểm tra sự đứng đắn của mã lệnh theo đặc tả ngôn ngữ và chuyển đổi sang dạng mã máy có thể thực thi.

Đặc tả ngôn ngữ AOP xét ở mức cao, ngôn ngữ AOP chia làm 2 thành phần:

- Thể hiện vấn đề: Ánh xạ yêu cầu riêng lẻ ỏ dạng mã để trình biên dịch có thể chuyển sang dạng thực thi. Thể hiện vấn đề có dạng thủ tục cụ thể, bạn có thể dụng ngôn ngữ truyền thông như C/C++ hay java

- Đặc tả quy tắc đan kết: Xác định cách thức tổng hợp các vấn đề riêng lẻ. Muốn vậy, thể hiện cần sử dụng hoặc tạo một ngôn ngữ để xác định các quy tắc tổng hợp các thể hiện khác nhau. Ngôn ngữ xác định quy tắc đan kết có thể là sự mở rộng của ngôn ngữ thể hiện, hoặc một ngôn ngữ hoàn toàn khác.

Biên dịch AOP: trình biên dịch AOP thực hiện theo trình tự 2 bước sau:

- Kết hợp các hành vi riêng lẻ.

- Chuyển đổi thông tin kết quả sang dạng mã thực thi.

Việc biên dịch ứng dụng AOP có thể thực hiện theo nhiều cách, trong đó có cách dịch mã sang mã. Theo cách này, mã nguồn được tiền xử lý các aspect riêng lẻ để tạo ra mã đã được đan kết với nhau. Sau đó, trình biên dịch AOP đưa mã đã được chuyển đổi này sang trình biên dịch ngôn ngữ cơ sở để tạo dạng mã thực thi cuối cùng. Ví dụ, theo cách thức này, các aspect riêng lẻ của ứng dụng AOP dùng ngôn ngữ Java trước tiên sẽ đc đan kết ở dạng mã nguồn Java, kế tiếp trình biên dịch Java sẽ chuyển mã nguồn này sang dạng mã thực thi(byte code). Tương tự, cũng có thể hiện đan kết ở giai đoạn mã byte code, byte code vẫn là 1 dạng mã nguồn, nếu như hệ thống thực thi chế độ nền – JVM – hỗ trợ aspect. Với cách này, đầu tiên JVM sẽ nạp các quy tắc đan kết rồi áp dụng các quy tắc này cho các lớp được nạp sau đó trình biên dịch thực hiện việc đan kết các aspect trong khi thực thi.

### ***1.2 Các đặc điểm của AOP***

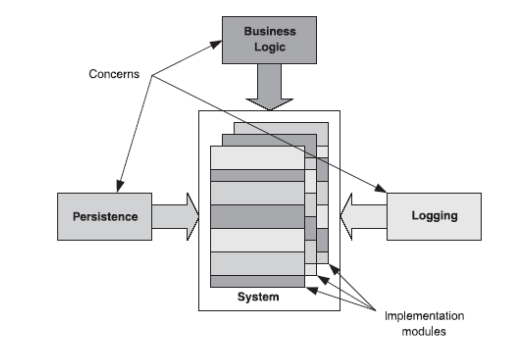
Concern là các yêu cầu cụ thể hay mối quan tâm đặc trưng được xác định để thoảmãn mục tiêu chung của hệ thống. Hệ thống phần mềm là sự gắn kết của tập các concern.Ví dụ, hệ thống ngân hàng bao gồm các concern sau đây: quản lý khách hàng và quản lýtài khoản, các giao dịch nội ngân hàng, các giao dịch ATM, chăm sóc khách hàng, lưugiữ các thực thể trong hệ thống, xác nhận truy cập các dịch vụ, …Ngoài ra một phầnmềm còn phải đảm bảo khả năng dễ hiểu, dễ bảo hành và duy trì, dễ phát triển. Concern được chia làm hai loại:

- Concern thành phần: Thể hiện các chức năng nội tại của một module.

- Concern đan xen: Thể hiện các quan hệ ràng buộc giữa các module trong hệ thống

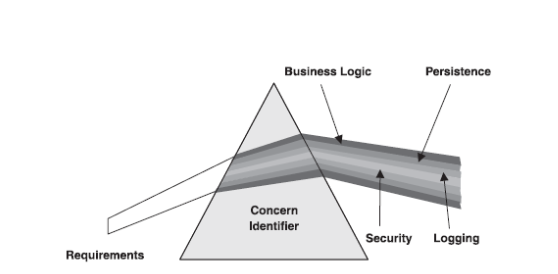
Một ứng dụng doanh nghiệp điển hình có thể bao gồm các concern đan xen sau: authentication, logging, resource pooling, performance, storage management, datapersistence, security, multithread safety, transaction integrity, error checking…

Các concern được phục vụ cho một vài module. Ví dụ, logging tác động tới tất cả moduletrong hệ thống, authencication tác động tới module có yêu cầu kiểm soát truy cập

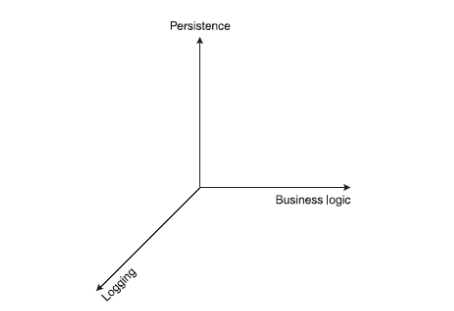


*Hình 2: Mô hình các concern mức hệ thống*

Việc xác định được các concern trong hệ thống, chúng ta sẽ tập trung vào các concern một cách độc lập và sẽ giảm độ phức tạp của hệ thống.

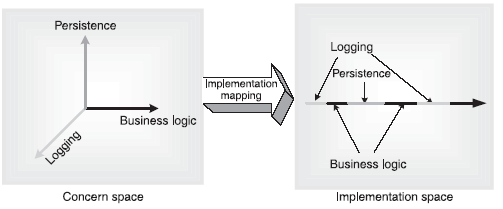


*Hình 3: Mô hình ánh xạ yêu cầu người dùng sử dụng AOP*



*Hình 4: Mô hình đa chiều về sự phụ thuộc giữa các modulo với nhau*

Các concern đan xen nhau giữa các module, các kỹ thuật thi công hiện tại sẽ trộn chúng vào một module. Hình sau minh hoạ sự thực hiện này: Với mô hình biểu diễn nhiều chiều của các concern được ánh xạ trên các ngôn ngữ một chiều như sau.



*Hình 5: Mô hình ánh xạ từ các concern hệ thống sang các phương pháp lập trình thống*

### ***1.3 Điểm mạnh, điểm yếu của AOP***

1.3.1 Điểm mạnh của AOP

Khi sử dụng AOP để giải quyết các bài toán chúng ta cần thiết kế và thi công hệ thống theo một phương pháp mới , với AOP có những điểm mạnh sau:

- Modul hóa những vấn đề đan nhau: AOP xác định các vấn đề 1 cách tách biệt, hạn chế tối thiểu việc nhập nhằng mã, cho phép modul hóa cả vấn đề liên quan đến lớp trừu tượng.

- Dễ dàng phát triển hệ thống: việc thêm chức năng mới có thể thực hiện dễ dàng bằng cách tạo aspect mới mà ko cần quan tâm đến vấn đề đan nhau. Khi thêm các modul mới vào hệ thống, các aspect hiện có sẽ đan kết với chúng và tạo nên sự phát triển chặt chẽ.

- Cho phép để lại quyết định thiết kế tương lai: một thiết kế tốt phải tính đến cả yêu cầu hiện tại và tương lai, việc xác định yêu cầu tương lai là một công việc khó khăn. Nếu bỏ sót những yêu cầu tương lai có thể bạn sẽ phải thay đổi hay thực hiện lại nhiều phần hệ thống. Với AOP, người thiết kế hệ thống có thể để lại các quyết định thiết kế cho những yêu cầu tương lai nhờ thực hiện theo các aspect riêng biệt.

- Tái sử dụng mã tốt hơn: các aspect là những modul riêng biệt, được kết hợp linh động, đây chính là yếu tố quan trọng để tái sử dụng mã. AOP cho phép tái sử dụng mã tốt hơn OOP.

1.3.2 Điểm yếu của AOP

Mặc dù AOP có các ưu điểm trên khi thiết kế và thi công một hệ thống,nhưng trên thực tếAOP cũng có một vài đặc điểm gay trở ngại sau:

- AOP thực ra không giải quyết các vấn đề mới, không giải quyết được vấn đề vẫn chưa được giải quyết. AOP giải quyết các bài toán theo cách tốt hơn.

- AOP không là cứu cánh cho các thiết kế cẩu thả.

- AOP phá vỡ tính đóng gói (encapsulation).

### ***1.4 Ứng dụng AOP***

AOP được ứng dụng trên hầu hết các ngôn ngữ:

- Java, C, Python, PHP.

- .NET framework (C#, VB.NET), Ruby, UML 2.0, XML.

- Delphi, ColdFusion, ..

Các dự án về AOP: AspectJ, AspectWerkz, Jboss AOP, Sping AOP, Aspect#, AspectC++, JAC

## **2. Ngôn ngữ AspectJ**

### ***2.1 Giới thiệu chung về AspectJ***

AspectJ là ngôn ngữ lập trình hướng khía cạnh (AOP) trên nền ngôn ngữ Java. Code AspectJ được viết độc lập với code chương trình Java thông thường. AspectJ compiler là ajc.

Code AspectJ hỗ trợ cho chương trình viết bằng ngôn ngữ Java có thể:

- Monitoring/ Tracing/ Debugging

- Testing

- Bổ sung chức năng

### ***2.2 Các khái niệm cơ bản***

2.2.1 JoinPoint

Joinpoint là bất kỳ điểm nào có thể xác định được khi thực hiện chương trình. Ví dụ: lời gọi hàm, khởi tạo đối tượng, exception, tham chiếu đối tượng, truy cập biến... Joinpoint chính là vị trí mà các hành động thực thi cắt ngang được đan vào. Trong AspectJ mọi thứ đều xoay quang joinpoint.

Một số loại JoinPoint chính trong AspectJ:

- Joinpoint tại hàm khởi tạo (constructor)

- Joinpoint tại các phương thức

- Joinpoint tại các điểm truy cập thuộc tính

- Joinpoint tại các điểm điều kiển ngoại lệ: được điều khiển trong các khối điều khiển ngoại lệ.

2.2.2 PointCut

Pointcut là một cấu trúc chương trình mà nó chọn các joinpoint và ngữ cảnh tại các joinpoint đó. Ví dụ, một pointcut có thể chọn một joinpoint là lời gọi đến một phương thức và lấy thông tin ngữ cảnh của phương thức đó như đối tượng chứa các phương thức đó, các tham số của phương thức đó.

Cú pháp của pointcut được khai báo như sau:

*[access specifier]* pointcut *pointcut-name*([args]) : *pointcut-definition*;

2.2.3 Advice

Advice là mã thực hiện tại một joinpoint mà được chọn bởi pointcut. Hau nói cách khác, nếu ta coi pointcut là khai báo tên phương thức thì advice là phần thân của phương thức đó. Pointcut và advice sẽ hình thành nên các luật đan kết các quan hệ đan xen.

Advice được chia thành ba loại sau:

- Before: Được thực hiện trước joinpoint

- After: Được thực hiện sau joinpoint

- Around: Thực thi xung quang joinpoint

2.2.4 Aspect

Aspect là phần từ trung tâm của AspectJ, giống như class trong Java. Aspect chứa mã thể hiện các luật đan kết cho các concern. Joinpoint, pointcut, advice được kết hợp trong aspect

Aspect được khai báo theo mẫu sau:

[access specification] aspect <AspectName>

[extends class-or-aspect-list]

[implements interface-list]

[<association-specifier>(Pointcut)] {

... aspect body

}

Tuy có gần giống các đặc điểm của class trong Java như: chứa thuộc tính, phương thức, có thể khai báo trừu tượng, có thể kế thừa... Tuy nhiên, Aspect có một số khác biệt cơ bản sau:

- Aspect không thể khởi tạo trực tiếp.

- Aspect không thể kế thừa từ một aspect cụ thể (không phải trừu tượng)

Aspect có thể được đánh dâu là có quyền bằng định danh privileged. Nhờ đó nó có thể truy cập đến các thành viên của lớp mà chúng cắt ngang.

Tóm lại, ta có thể định nghĩa tổng quan về mối quan hệ của Joinpoint, Pointcut, advice, aspect bằng phương trình như sau:

Program:

Joinpoint = well-defined point in the program

AspectJ:

Pointcut = Joinpoint-set

Advice = Kind x Pointcut x Code

Where Kind = {before, after, around}

Aspect = Advice - list

### ***2.3 Lập trình với AspectJ***

Tạo một Aspect class gồm các bước:

- Xác định các Joinpoint

- Thiết kế PointCut

- Viết code thực thi cho các advice

Trong giai đoạn phát triển Development, aspectJ được dùng để debugging, testing, turning:

- Tracing

- Profiling and logging

- Testing: kiểm thử pre-condition, post-condition, invariant

- Testing: kiểm soát ràng buộc (contract enforcement)

Trong giai đoạn Product, aspectJ được dùng để:

- Change Monitoring

- Context passing

- Consistent behavior

## **3. Sử dụng AspectJ cho kiểm thử phần mềm**

### ***3.1 Giới thiệu về kỹ thuật Design by Contract(DBC)***

Design by contract(DBC) là một thiết kế phần mềm hướng đối tượng với mục tiêu là để đảm bảo chất lượng phần mềm, tăng độ tin cậy và có thể dùng lại. Sử dụng ngôn ngữ AspectJ trả về xác nhận là một biểu thức Boolean chèn vào một điểm cụ thể của chương trình thực thi, phải trả về giá trị True, nếu trả về False khẳng định có lỗi trong phần mềm.

Giải pháp này mang lại những thuận lợi:

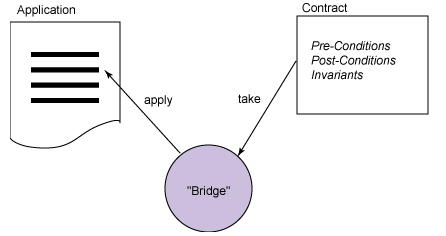
- Tính minh bạch: code pre và post-condition không trộn lẫn với logic nghiệp vụ

- Tính tái sử dụng: hầu hết các phần đều có thể tái sử dụng

- Tính linh hoạt: các modulo có thể thêm sửa xóa một cách đơn giản

- Tính đơn giản: cú pháp sử dụng đơn giản

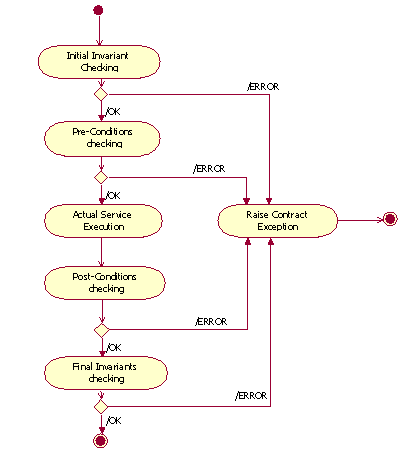
Cấu trúc thực thi bao gồm các ứng dụng mã(những phần không chứa các yếu tố liên quan đến BDC); thành phần contract (gồm kiểm tra pre-condition, post-condition, invariant) và một đối tượng hoạt động như một cầu nối giữa code và contract, có khả năng áp dụng các contract ở phần bên phải của code với logic đúng. Được mô tả trong hình sau:



*Hình 6: Modun hóa cho giải pháp Design by Contract*

Các thiết kế được minh họa trong hình trên đảm bảo một giải pháp linh hoạt cao, cho phép bạn ứng dụng và loại bỏ các contract mà không cần thay đổi hoặc thực hiện các contract khác hoặc mã ứng dụng. Và nó hoàn toàn trong suốt đối với các ứng dụng.

Contract là một class Java mà thực hiện một interface cụ thể và “bridge” là một khía cạnh của AspectJ. Aspect này chỉ ra vị trí chính xác nơi mà các contract được áp dụng và điều khiển logic cần thiết để áp dụng các contract. Hình .... thể hiện thành phần chi tiết nó thể hiện tổng quát tất cả các contract vì vậy bạn có thể phát triển một khía cạnh trừu tượng nào đó mà bạn xác định.



*Hình 7: Sơ đồ hoạt động theo logic của Design by contract*

### ***3.2 Contract checking***

3.2.1 Pre-conditions (Tiền điều kiện)

- Kiểm tra sự hợp lệ của tham số

Ví dụ: với source code và aspect tương ứng

class Power: boolean withdraw(int amount) { Pre condition:

balance - amount > 50;

aspect WithDrawPreCond {

final int MIN\_BALANCE = 50;

before(Power power, int amount) :

call(boolean Power.withdraw(int)) && target(power) && args(amount) {

assert power.balance - amount > MIN\_BALANCE:

“withdrawal too big:” + amount;

}

}

3.2.2 Post-conditions (Hậu điều kiện)

- Kiểm tra sự chính xác của kết quả

Ví dụ: với source code và aspect tương ứng

class Power: boolean withdraw(int amount): Post condition:

result == (old(balance) - amount) >= 0 &&

balance == (result ? old(balance) - amount : old(balance))

aspect WithDrawPostCond {

int old\_balance;

before(Power power) : call(boolean Power.withdraw(int)) && target(power) {

old\_balance = power.balance;

}

after(Power power, int amount) returning(boolean changed) :

call(boolean Power.withdraw(int)) && target(power) && args(amount) {

assert changed == (old\_balance - amount) >= 0

&& power.balance == (changed ? old\_balance-amount : old\_balance);

}

}

3.2.3 Policy enforcement (Chính sách thực thi)

- Kiểm tra và sửa nếu việc kiểm tra thất bại

Ví dụ: với source code và aspect tương ứng

boolean withdraw(int amount){

pre balance - amount > 50 {

System.out.println (“withdraw rejected”);

return false;

}

aspect WithdrawCorrect {

final int MIN\_BALANCE = 50;

boolean around (Power power, int amount):

call (boolean Power.withdraw(int) && target (power) && args(amount)

{

if (power.balance - amount >= MIN\_BALANCE)

return proceed (power, amount);

else {

System.out.println (“withdraw rejected”);

return false;

}

}

}

3.2.4 Invariants (Khả năng bất biến)

- Kiểm tra một số điều kiện luôn ở trong trạng thái giữ, các điều kiện vẫn không thay đổi sau khi thực hiện các thành phần bên ngoài

Ví dụ: với source code và aspect tương ứng

class Power: boolean withdraw(int amount): Pre condition:

balance >= 500;

aspect Invariant {

boolean invariant(int balance) { return balance >= 500; }

pointcut write(int balance) : set(int Power.balance) && args(balance);

before(int balance) : write(balance) {

if (!invariant(balance)) { System.out.println("invariant violated"); }

}

}.

## **4. So sánh AspectJ với các công nghệ khác**

### ***4.1 Nhận xét về AspectJ***

Do các mã của aspect độc lập với mã nguồn chính của chương trình, có thể sửa đổi tùy ý theo ý muốn của lập trình viên, chính vì vậy AOP còn được sử dụng vào các loại kiểm chúng trong quá trình thiết kế phần mềm. Ví dụ: Kiểm chứng giao thức, kiểm tra việc gọi tuần tự các hàm trong chương trình..

Kiểm chứng sự nhất quán giữa cài đặt so với thiết kế ở thời điểm thực thi. Trong đó các phương thức trong giao thức có thể được thực hiện song song với nhau và phải thỏa mãn các mệnh đề tiền và hậu điều kiện (kiểm chứng sự tuân thủ giữa thực thi và đặc tả giao thức tương tác trong các chương trình đa luồng). Trong việc cài đặt mã nguồn chương trình có thể vi phạm các đặc tả thiết kế của giao thức, thông thường các vi phạm này khó được phát hiện trong bước kiểm thử bằng các bộ dữ liệu đầu vào và đầu ra

Phương pháp kiểm chứng dựa trên AOP là dựa vào những kiểu khái niệm cơ bản của AOP như: jont pont, point cut, advice, aspect để xây dựng các modun kiểm chứng từ các chức năng cắt ngang hệ thống. Các aspect này sẽ được đan vào khung mã nguồn chương trình thông qua trình biên dịch AspectJ để thực hiện chức năng kiểm chứng

Một aspect phải có đầy đủ các thông tin về các quy tắc đan: join point và point cut và mã kiểm chứng được cài đặt trong advice. Trình biên dịch AspectJ mới có thể đan chúng vào chương trình chính phục vụ quá trình kiểm thử.

AspectJ cho phép đan xen mã aspect với các chương trình Java ở ba mức khác nhau: mức mã nguồn, mã bytecode và tại thời điểm nạp chương trình khi chương trình gốc chuẩn bị được thực hiện. Với việc đan xen ở mức mã bytecode vào tại thời điểm nạp chương trình thì phương pháp này có thể được sử dụng mà không yêu cầu phải có mã nguồn. Khi thay đổi đặc tả thì mới phải sinh và biên dịch lại mã aspect

Các mã aspect này có thể được tự động sinh ra từ các đặc tả giao thức tương tác IP. Đặc tả này được xây dựng thông qua các biểu thức chính quy mở rộng hoặc máy trạng thái giao thức.

Hạn chế khi sử dụng AspectJ là phải thực thi chương trình, các vi phạm (VD: vi phạm về ràng buộc thời gian) chỉ được phát hiện trong bước kiểm thử. Mã aspect được đan vào chương trình chính nên sẽ làm tăng kích thước và thời gian thực thi của chương trình

Các aspect không làm thay đổi hành vi của chương trình gốc, ngoại trừ thời gian chạy và kích thước của chương trình. Thời gian chạy sau khi đan mã aspect sẽ tăng tỷ lệ thuận với số luồng trong chương trình và số phương thức được mô tả trong giao thức

### ***4.2 So sánh AspectJ với các công nghệ khác trong mục đích kiểm thử:***

4.1.1 So sánh kiểm thử java sử dụng Xstream, AspectJ, Vsdf:

*Đặt vấn đề:*

Test tích hợp hoặc test đơn vị có thể yêu cầu tạo ra các cấu trúc đối tượng phức tạp. Điều này thậm chí khó giải quyết và không có khả năng test trong các hệ thống kế thừa.

*Giải pháp:*

Một phương pháp ngắn hạn có thể sử dụng để giải quyết vấn đề trên là nối tiếp hóa đối tượng để nắm bắt các đối tượng quan trọng khi chúng được sử dụng trong hệ thống thực tế. Sau đó kiểm thử có thể xây dựng lại các đối tượng và sử dụng chúng trong các kiểm thử khác.

*Xstream*:

Có nhiều phương pháp trong java có thể được sử dụng để nối tiếp hóa các đối tượng như: sự nối tiếp hóa của JAXB, JavaBean hoặc kết hợp cả hai. Tất cả chúng đều có vấn đề hạn chế. Ví dụ: chúng yêu cầu đối tượng được tuần tự hóa phải phù hợp với các yêu cầu nhất định, giống như các đặc điểm kỹ thuật của JavaBean. Nếu xung quanh đối tượng không có đối tượng nào, điều này sẽ trở nên phức tạp, không chỉ đối với đối tượng mức cao đã được xử lý và sau đó chúng sẽ được lồng vào đồ thị đối tượng. Thư viện của Xstream không có các yêu cầu này và cũng có khả năng xử lý một tỷ lệ lớn các trường hợp.

Trong đoạn code dưới đây là một hàm kiểm thử đơn vị sử dụng Xstream. Một lớp được gọi là Xapp chứa một phương thức scenario1 mà chúng ta dự định test. Để gọi phương thức scenario1, một số dữ liệu phải được tạo và cấu trúc thành một trật tự yêu cầu của đối tượng, ở đây chúng ta chỉ cần sử dụng một bản đồ đơn giản để thể hiện điều đó. Vì vậy, kiểm thử tạo ra một đối tượng từ một tuần tự hóa Xstream XML và sau đó gọi phương thức scenario1() của hệ thống để kiểm tra.

*package com.octodecillion.testing;*

*import static org.hamcrest.core.Is.is;*

*import static org.junit.Assert.assertThat;*

*import java.io.File;*

*import java.io.IOException;*

*import org.junit.Test;*

*import org.junit.runner.RunWith;*

*import org.junit.runners.JUnit4;*

*import com.octodecillion.testing.XApp.Inventory;*

*import com.thoughtworks.xstream.XStream;*

*import com.thoughtworks.xstream.io.xml.StaxDriver;*

*/\*\**

*\* @author jbetancourt*

*\**

*\*/*

*@RunWith(JUnit4.class)*

*public class InventoryTest {*

*/\*\**

*\* @throws IOException*

*\**

*\*/*

*@SuppressWarnings("boxing")*

*@Test*

*public void scenario1\_test() throws IOException {*

*Inventory inventory = (Inventory) new XStream(new StaxDriver())*

*.fromXML(new File(*

*"inventory-scenario1.xml"));*

*XApp app = new XApp();*

*app.setInventory(inventory).scenario1();*

*assertThat(app.getInventory().stock.size(), is(1));*

*}*

*}*

*AOP*

Vì không có sự ánh xạ và sửa đổi tới các đối tượng nên việc tuần tự hóa của Xstream cần tới hai dòng mã. Chúng ta có thể chèn đoạn mã nguồn này (hoặc qua một tiện ích) vào nơi mà hệ thống được kiểm tra để nắm bắt trạng thái của thử nghiệm tiếp theo. Tuy nhiên, điều này yêu cầu chúng ta xả rác mã với các mối quan tâm tuần tự hóa này. Chúng ta có thể xóa bỏ chúng sau khi chúng ta nắm bắt được đối tượng.

Một cách khác là sử dụng phương pháp lập trình hướng khía cạnh. Với AOP chúng ta có thể tư vấn ở các joinpoints để nắm bắt các đối tượng. Điều này có thể thực hiện với Load Time Weaving. Các nguồn gốc chưa bị sửa đổi và để nắm bắt lại các đối tượng giống nhau chúng ta phải chạy lại hệ thống và áp dụng lại AOP LTW

Trong đoạn code dưới, AspectJ AOP Java framework được sử dụng để tạo một khía cánh nắm bắt các đối tượng dữ liệu ở phương thức setScenarioData() trong lớp kiểm thử Xapp dưới.

*package com.octodecillion.testing;*

*import java.io.File;*

*import java.io.FileWriter;*

*import com.thoughtworks.xstream.XStream;*

*import com.thoughtworks.xstream.io.xml.StaxDriver;*

*/\*\**

*\* Aspect to allow XStream streaming of application data object.*

*\**

*\* @author jbetancourt*

*\**

*\*/*

*public aspect XStreamObject {*

*/\*\**

*\* Capture the App inventory object during creation of data*

*\* and stream to file.*

*\**

*\*/*

*@SuppressWarnings("javadoc")*

*after(final XApp app) :*

*execution( protected void setScenarioData()) &&*

*!within(XStreamObject) && target(app) {*

*try {*

*FileWriter writer = new FileWriter(*

*new File("inventory-scenario1.xml"));*

*new XStream(new StaxDriver()).toXML(app.getInventory(), writer);*

*writer.close();*

*} catch (Exception e) {*

*e.printStackTrace();*

*}*

*}*

*}*

*Việc lưu trữ sử dụng Vsdf :*

Một vấn đề với việc tuần tự hóa kiểm thử là lưu trữ nó ở đâu. Nếu chúng ta có một ứng dụng phức tạp, có thể có nhiều đối tượng và có nhiều trạng thái khác nhau phụ thuộc vào các yêu cầu của kịch bản test.

Với Vsdf, nhiều sự nối tiếp hóa có thể được lưu trữ trong một file. Vì vây chúng ta có thể lưu trữ tất cả các luồng đối tượng trong một file, hoặc các trạng thái tuần tự khác nhau của một loại đối tượng có thể được lưu trữ trong mỗi file VSdf. Một số ưu điểm trong phương pháp này là làm giảm file kiểm thử, có khả năng thay đổi và cập nhật các tuần tự hóa độc lập.

Ví dụ ta có đoạn code kiểm thử sau:

*package com.octodecillion.testing;*

*import java.io.IOException;*

*import java.util.HashMap;*

*import java.util.Map;*

*/\*\**

*\* Simple XStream example.*

*\**

*\* @author jbetancourt*

*\**

*\*/*

*public class XApp {*

*private Inventory inventory;*

*/\*\**

*\* The scenario that needs testing.*

*\*/*

*public void scenario1() {*

*// do the business case here ...*

*}*

*/\*\* default constructor \*/*

*public XApp() {*

*setScenarioData();*

*}*

*/\*\**

*\* The Application entry point.*

*\* @param args*

*\* @throws IOException*

*\*/*

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

*new XApp();*

*}*

*/\*\**

*\* set up the data for scenario1*

*\*/*

*protected void setScenarioData() {*

*inventory = new Inventory();*

*inventory.add("322", new Vehicle() {*

*@Override*

*public int getNumWheels() {*

*return 4;*

*}*

*});*

*}*

*/\*\**

*\* @author jbetancourt*

*\**

*\*/*

*public class Inventory {*

*/\*\*      \*/*

*Map<String, Vehicle> stock = new HashMap<String, Vehicle>();*

*/\*\**

*\* @param id*

*\* @param veh*

*\*/*

*public void add(final String id, final Vehicle veh) {*

*stock.put(id, veh);*

*}*

*} // end class Inventory*

*/\*\**

*\* @author jbetancourt*

*\**

*\*/*

*public interface Vehicle {*

*/\*\**

*\* @return num of wheels*

*\*/*

*public int getNumWheels();*

*} // end Vehicle*

*/\*\**

*\* @param inventory*

*\* @return the app object*

*\*/*

*public XApp setInventory(final Inventory inventory) {*

*this.inventory = inventory;*

*return this;*

*}*

*/\*\**

*\* @return the inventory*

*\*/*

*public Inventory getInventory() {*

*return inventory;*

*}*

*} // end class Xapp*

4.2.2 So sánh AspectJ với Jmockit trong kiểm thử đơn vị

Trong kiểm thử đơn vị khi phát triển và bảo trì mã, các đơn vị mã nhỏ nhất được kiểm thử trong sự cô lập từ việc cộng tác giữa các đơn vị hoặc các hệ thống con. Trong các hệ thống được thiết kế tốt, việc kiểm thử đơn giản hơn nhiều: giao diện đầu ra dễ dàng sử dụng. trong các hệ thống kế thừa với các đoạn mã tồi thì đây có thể không phải là một trường hợp. Hơn nữa, khi kiểm thử các hệ thống kế thừa, cơ hội để tăng khả năng kiểm thử không phải lúc nào cũng có thể làm được.

Vì vậy, nhều mô hình, khác nhau, các nền tảng và các công cụ đã được sử dụng để hỗ trợ sự cô lập này. Nền tảng Mock Object là một ý tưởng trong số đó. AOP cũng có khả năng cung cấp sự tách biệt này nhưng hiếm khi được đề cập đến. Sự khác biệt là gì? Cái nào tốt hơn trong kiểm thử?

Mock Object mô phỏng các đối tượng, bắt trước các hành vi của đối tượng trong thực tế với nhiều cách kiểm soát.

Về cầu trúc và cú pháp cả hai đều là định nghĩa một lớp. Các hành vi mới được xây dựng cũng là một phương thức

Đặc điểm khác nhau chủ yếu của AspectJ và Jmockit là ở pointcut: Trong JMockit các pointcut và các đặc tả advice được kết hợp với nhau, các ký hiệu phương thức phải phù hợp với mục tiêu của phương thức. Trong AspectJ chúng tách biệt nhau.

*AspectJ:*

*@Aspect*

*static class TestAspect {*

*@Around("call( \* AspectJUnit.Service.query(int)) && args(i)")*

*public String doQuery(ProceedingJoinPoint jp, int i){*

*return (i == 1) ? "Hello world!"*

*: (String) jp.proceed(new Object[]{i});*

*}*

*}*

*JMockit*

*class MockService extends MockUp<Service> {*

*@Mock*

*public String query(Invocation invocation, int n) {*

*return (n == 1) ? "Hello world!"*

*: ((Service) invocation.*

*getInvokedInstance()).query(n);*

*}*

*}*

Cú pháp kết hợp trong Mock đơn giản hơn, một phương thức test thường hướng tới một phương thức cụ thể. Các lớp Mock thậm chí có thể được tạo ra trên một dòng test.

Aspect phải được tạo ra như một lớp tĩnh bao gồm các advice liên quan được đan tại thời điểm biên dịch. Các mock được tích hợp trong thời gian chạy JUNit nên các advice của nó chỉ hoạt động cho phương thức test hiện tại, và bạn có nhiều cơ hội hơn cho các kiểm thử khác nhau mà có cùng mục tiêu. Vì vậy, mã nguồn đầy đủ của việc kiểm thử sử dụng Aspects không có nhiều xác nhận như được viết bằng Mocks

# **Chương 2: Phân tích và thiết kế ứng dụng**

## **1. Phát biểu bài toán**

Bài toán được sử dụng để xây dựng ứng dụng demo cho bài báo cáo là một phần nghiệp vụ chức năng của giao dịch mua bán chứng khoán. Bài toán được phát biểu như sau:

Khi một nhà đầu tư muốn đầu tư vào chứng khoản tại một công ty chứng khoán, người dùng đăng ký online vào hệ thống. Thông tin tài khoản của nhà đầu tư bao gồm: Tên đăng nhập, Mật khẩu, Họ tên đầy đủ, CMND, số tài khoản ngân hàng.

Cổ phiếu của các doanh nghiệp được niêm yết trên sàn có các thông tin: Mã cổ phiếu (gồm 3-5 ký tự số hoặc chữ cái la tinh), Loại cổ phiếu (cổ phiếu thông thường, cổ phiếu ưu đãi,…), Tên tham chiếu, Giá trần, Giá sàn, Giá hiện tại cảu cổ phiếu (Giá này được update định kỳ).

Một nhà đầu tư khi đăng nhập thành công vào chương trình giao dịch chứng khoán, nhà đầu tư có thể thực hiện các tính năng sau:

* Xem danh sách cổ phiếu sở hữu
* Xem danh sách cổ phiếu theo dõi
* Tìm kiếm/thêm/xóa cổ phiếu theo dõi
* Mua cổ phiếu
* Bán cổ phiếu.

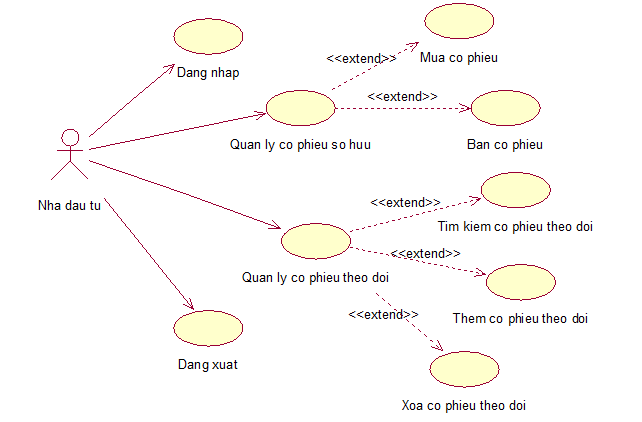
Trong khi mua/bán cổ phiếu có một số điều kiện ràng buộc như sau:

* Nhà đầu tư khi mua cổ phiếu phải đảm bảo đúng các thông tin đầu vào như mã cổ phiếu. Tài khoản phải có đủ số tiền mặt để mua số lượng cổ phiếu và loại cổ phiếu mà nhà đầu tư mong muốn ( số tiền mặt trước khi mua >= khối lượng cổ phiếu\*giá). Khối lượng cổ phiếu mua vào < số lượng cổ phiếu đang còn của các doanh nghiệp. Giá mua phải thỏa mãn: giá sàn<= giá mua<= giá trần. Sau khi mua thành công cổ phiếu, tiền trong tài khoản của khách hàng sẽ bị trừ đi.
* Nhà đầu tư khi bán cổ phiếu phải đảm bảo những điều kiện sau: Đảm báo đúng thông tin mã cổ phiếu, thỏa mãn các thông tin như khối lượng bán giá và giá bán. Khối lượng bán ra bé hơn hoặc bằng khối lượng cổ phiếu mà nhà đầu tư có. Giá bán cũng phải thỏa mãn: sàn<= giá bán<= giá trần. Nếu khối lượng cổ phiếu sau khi bán bằng 0 thì cổ phiếu sẽ bị xóa khỏi danh sách sở hữu. Sauk hi thực hiện bán cổ phiếu thành công, số lượng cổ phiếu sở hữu sẽ bị trừ đi số lượng cổ phiếu bán ra. Đồng thời tiền mặt trong tài khoản của nhà đầu tư tăng lên sao số lượng cổ phiếu bán \* giá bán.

Theo định kỳ, hệ thống mua bán cổ phiếu sẽ tự động update giá và số lượng từ các doanh nghiệp để người dùng có thể theo dõi tình hình cổ phiếu.

## **2. Thiết kế hệ thống**

### ***2.1. Biểu đồ Usecase của hệ thống***



*Hình 8: Biểu đồ usecase của hệ thống*

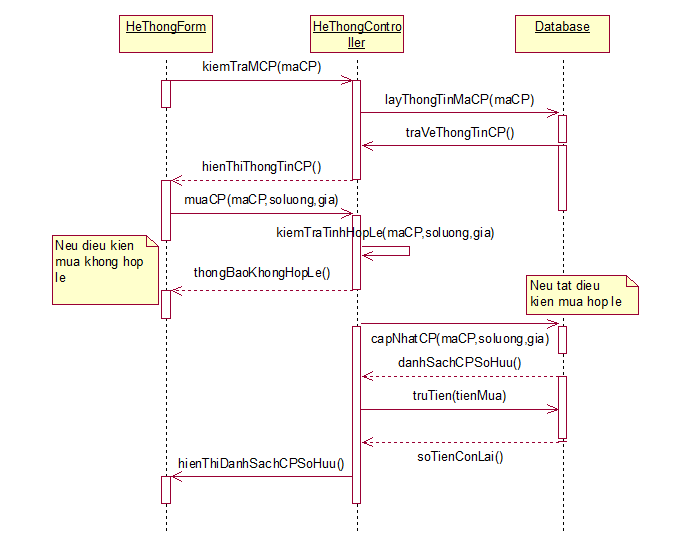
* Tác nhân chính của hệ thống là : Nhà đầu tư
* Các usecase chính của hệ thống:
  + Đăng nhập
  + Đăng xuất
  + Quản lý cổ phiếu sở hữu: Mua của phiếu, bán cổ phiếu
  + Quản lý cổ phiếu theo dõi: Tìm kiếm cổ phiếu, thêm cổ phiếu theo dõi, xóa cổ phiếu theo dõi

*Đặc tả các usecase chính của hệ thống:*

* *Usecase Đăng nhập*
  + Mô tả: Usecase mô tả các người dùng đăng nhập vào hệ thống giao dịch chứng khoán
  + Luồng sự kiện:
    - Người dùng lựa chọn tính năng đăng nhập của hệ thống
    - Hệ thống yêu cầu nhập thông tin Username và Password để được xác thực
    - Người dùng nhập Username và Password
    - Hệ thống kiểm tra thông tin tính xác thực của người dùng và trả lại thông báo user không hợp lệ nếu xác thực là false và yêu cầu nhập lại thông tin, ngược lại nếu xác thực là true thì hiển thị màn hình giao dịch chứng khoán
  + Yêu cầu đặc biệt: Không có
  + Tiền điều kiện: Hệ thống đang ở trạng thái login và hiển thị màn hình login
  + Hậu điều kiện: Nếu đăng nhập thành công thì hiển thị màn hình giao dịch chứng khoán. Ngược lại nếu đăng nhập không thành công thì thông báo tới người dùng là “Đăng nhập không thành công, vui lòng nhập lại các thông tin"
  + Điểm mở rộng: Không có.
* *Usecase Mua cổ phiếu*
  + Mô tả: Usecase mô tả quy trình nhà đầu tư mua cổ phiếu
  + Luồng sự kiện:
    - Hệ thống hiển thị danh sách cổ phiếu nhà đầu tư sở hữu.
    - Nhà đầu tư lựa chọn phương thức mua cổ phiếu, nhập mã cổ phiếu muốn mua
    - Hệ thống kiểm tra mã cổ phiếu có hợp lệ không. Nếu hợp lệ, hệ thống hiển thị các thông tin liên quan đến mã cổ phiếu đó: giá trần của cổ phiếu, giá sàn của cổ phiếu, số lượng cổ phiếu còn lại mà nhà đầu tư có thể mua
    - Nhà đầu tư nhập số cổ phiếu muốn mua và giá cổ phiếu mà nhà đầu tư sẽ mua
    - Hệ thống kiểm tra số phiếu mua có vượt quá số lượng cổ phiếu còn lại hay không? Kiểm tra giá mua có nằm giữa giá sàn và giá trần không? Kiểm tra số tiền mặt mà nhà đầu tư có trong tài khoản có đủ để thực hiện việc mua số lượng cổ phiếu đó hay không?
    - Hệ thống đưa ra các thông báo nếu một trong các điều kiện trên không thỏa mãn và không thực hiện giao dịch mua cổ phiếu. Ngược lại, nếu thỏa mãn tất cả các điều kiện trên, hệ thống sẽ cập nhật thông tin cổ phiếu mua này vào trong danh sách cổ phiếu sở hữu của nhà đầu tư và thực hiện trừ tiền mặt trong tài khoản của nhà đầu tư
  + Yêu cầu đặc biệt: không có.
  + Tiền điều kiện: Nhà đầu tư đã đăng nhập thành công vào hệ thống
  + Hậu điều kiện: Cổ phiếu mua vào sẽ được cập nhật vào trong danh sách cổ phiếu sở hữu của nhà đầu tư. Số cổ phiếu của doanh nghiệp sẽ bị giảm đi và cập nhật lên hệ thống giao dịch chứng khoán.
  + Điểm mở rộng: Không có.
* *Usecase Bán cổ phiếu*
  + Mô tả: Usecase mô tả quy trình thực hiện giao dịch bán cổ phiếu.
  + Luồng sự kiện:
    - Hệ thống hiển thị danh sách cổ phiếu nhà đầu tư sở hữu.
    - Người dùng lựa chọn giao dịch bán cổ phiếu và nhập thông tin mã cổ phiếu muốn bán.
    - Hệ thống kiểm tra mã cổ phiếu có hợp lệ và có trong danh sách cổ phiếu sở hữu của nhà đầu tư hay không. Nếu hợp lệ, hệ thống hiển thị các thông tin liên quan đến mã cổ phiếu đó: giá trần của cổ phiếu, giá sàn của cổ phiếu, số lượng cổ phiếu có thể bán.
    - Người dùng nhập số lượng cổ phiếu muốn bán và giá bán
    - Hệ thống kiểm tra các điều kiện: Số cổ phiếu bán ra có vượt quá số cổ phiếu sở hữu của nhà đầu tư không? Giá bán có nằm trong khoảng trần sàn của mã cổ phiếu đó không?
    - Hệ thống đưa ra thông báo nếu một trong các điều kiện trên không thỏa mãn Ngược lại các điều kiện trên đều thỏa mãn thì hệ thống sẽ thực hiện giao dịch bán số cổ phiếu người dùng muốn bán. Cập nhật số tiền mà nhà đầu tư có được sau khi bán xong cổ phiếu, giảm trừ số cổ phiếu còn lại của mã cổ phiếu mà nhà đầu tư bán. Nếu số cổ phiếu được bán hết, hệ thống sẽ tự động xóa mã cổ phiếu đó trong danh sách cổ phiếu sở hữu của nhà đầu tư.
  + Yêu cầu đặc biệt: Không có.
  + Tiền điều kiện: Nhà đầu tư đã đăng nhập thành công vào hệ thống.
  + Hậu điều kiện: Nếu bán thành công số cổ phiếu còn lại của nhà đầu tư sẽ bị giảm trừ, số tiền của nhà đầu tư trong tài khoản sẽ được cộng thêm số tiền mà cổ phiếu bán được.
  + Điểm mở rộng: Không có.
* *Usecase Tìm kiếm và thêm cổ phiếu theo dõi*
  + Mô tả: Usecase mô tả quy trình tìm kiếm cổ phiếu mà nhà đầu tư muốn theo dõi.
  + Luồng sự kiện:
    - Hệ thống hiển thị danh sách cổ phiếu nhà đầu tư đang theo dõi.
    - Hệ thống hiển thị danh sách mã cổ phiếu mà các doanh nghiệp đăng ký
    - Người dùng tìm kiếm và lựa chọn mã cổ phiếu
    - Hệ thống hiển thị thông tin cổ phiếu
    - Người dùng lựa chọn thêm cổ phiếu vào danh sách theo dõi
    - Thông tin liên quan đến mã cổ phiếu mà nhà đầu tư tìm kiếm được thêm vào danh sách cổ phiếu theo dõi của nhà đầu tư
  + Yêu cầu đặc biệt: Không có
  + Tiền điều kiện: Nhà đầu tư đã đăng nhập thành công vào hệ thống.
  + Hậu điều kiện: Nếu thành công, cổ phiếu sẽ được cập nhật vào danh sách cổ phiếu được theo dõi của nhà đầu tư và thường xuyên cập nhật các thông tin giá sàn, giá trần, số lượng cổ phiếu có thể mua của mã cổ phiếu đó.
  + Điểm mở rộng: Không có.
* *Usecase Xóa cổ phiếu theo dõi*
  + Mô tả: Usecase mô tả hành động xóa cổ phiếu theo dõi của nhà đầu tư.
  + Luồng sự kiện:
    - Hệ thống hiển thị danh sách cổ phiếu nhà đầu tư đang theo dõi.
    - Nhà đầu tư lựa chọn mã cổ phiếu muốn xóa khỏi danh sách theo dõi
    - Hệ thống loại bỏ các thông tin liên quan đến mã cổ phiếu ra khỏi danh sách theo dõi của nhà đầu tư
  + Yêu cầu đặc biệt: Không có
  + Tiền điều kiện: Nhà đầu tư đã đăng nhập thành công vào hệ thống.
  + Hậu điều kiện: Nếu hành động xóa thành công, mã cổ phiếu sẽ bị xóa khỏi danh sách theo dõi.
  + Điểm mở rộng: Không có.

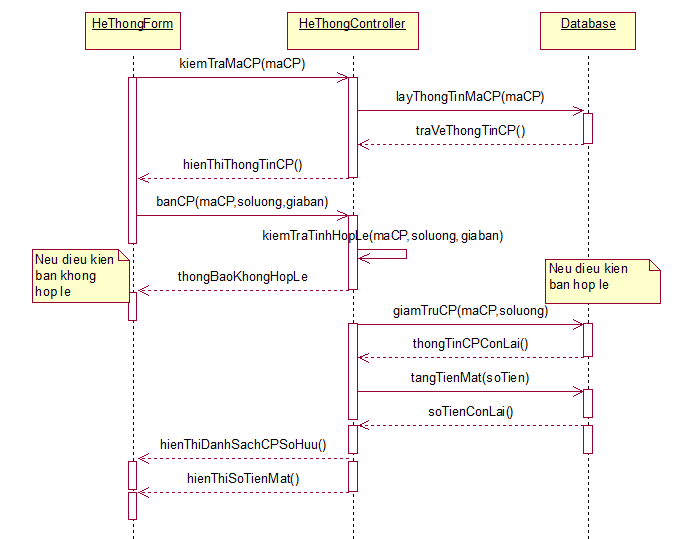
### ***2.2 Mô tả biểu đồ tuần tự của quá trình mua và bán cổ phiếu:***

* Biểu đồ tuần tự của mua cổ phiếu:



*Hình 9: Biểu đồ tuần tự của quá trình mua cổ phiếu*

* Biểu đồ tuần tự của bán cổ phiếu:



*Hình 10: Biểu đồ tuần tự của quá trình bán cổ phiếu*

### ***2.3 Xác định các lớp***

*Class HeThongForm*: Giao diện tương tác người dùng

*Class giả lập server HeThongController:*

- Đọc DB từ file

- Update giá random

- Login: xác thực người dùng

- Kiểm tra tính hợp lệ cổ phiếu

- Mua cổ phiếu

- Bán cổ phiếu

- Lấy thông tin cổ phiếu

- Thêm/xóa cổ phiếu theo dõi

*Class User:*

- username

- pass

- hoTen

- tienmat

- dsCPSohuu (Map<MaCP, [khoiluong, giaTrungbinh]>)

- dsCPTheodoi (List<String>)

*Class Cophieu*:

- maCP

- thamchieu

- tran

- san

- giaHientai (giá này update định kỳ)

## **3. Cài đặt và phát triển ứng dụng**

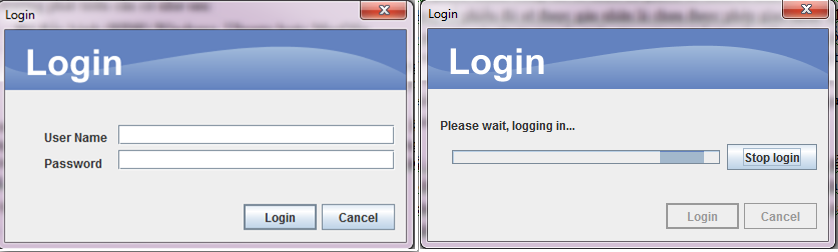
Ứng dụng Demo sẽ được viết bằng ngôn ngữ Java, nêu chúng ta cần cài đặt các môi trường phát triển cần có như sau:

* Hệ điều hành (HĐH) Windows, Ubuntu hoặc MacOSx
* Java Development Kit (JDK) cho HĐH tương ứng.
* Công cụ lập trình Eclipse, Android Studio, … cho HĐH tương ứng.
* Để kiểm thử với AscpectJ cần cài đặt thêm AspjectJ plugin phiên bản 1.8 trở lên cho Eclipse

Tạo một Object loại AspectJ để lập trình và kiểm thử bằng AspectJ. Khi lập trình chúng ta sẽ viết code cho ứng dụng riêng và viết các lớp aspect riêng để test các concern mà chúng ta đã xác định được trong hệ thống. Phần mô tả cách kiểm thử bằng AspectJ sẽ được bài báo cáo mô tả trong phần 3.

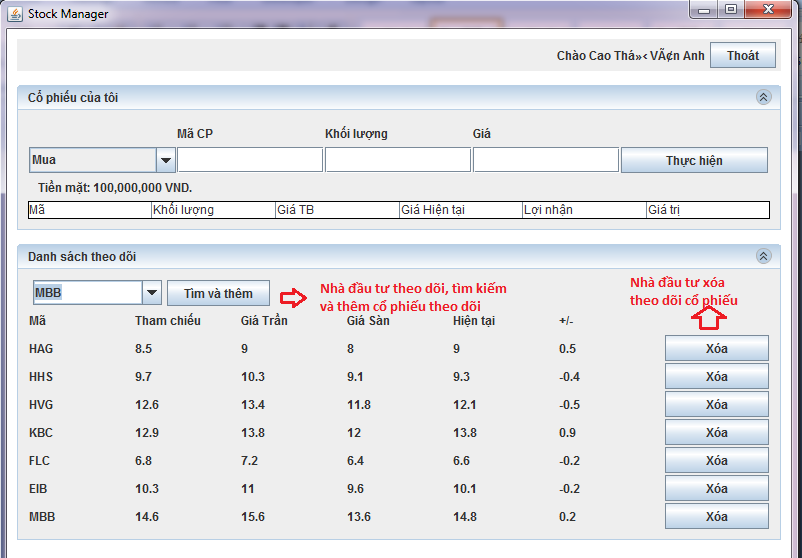
Kết quả của ứng dụng đã xây dựng và cài đặt như sau:

* Màn hình đăng nhập của ứng dụng dành cho nhà đầu tư:

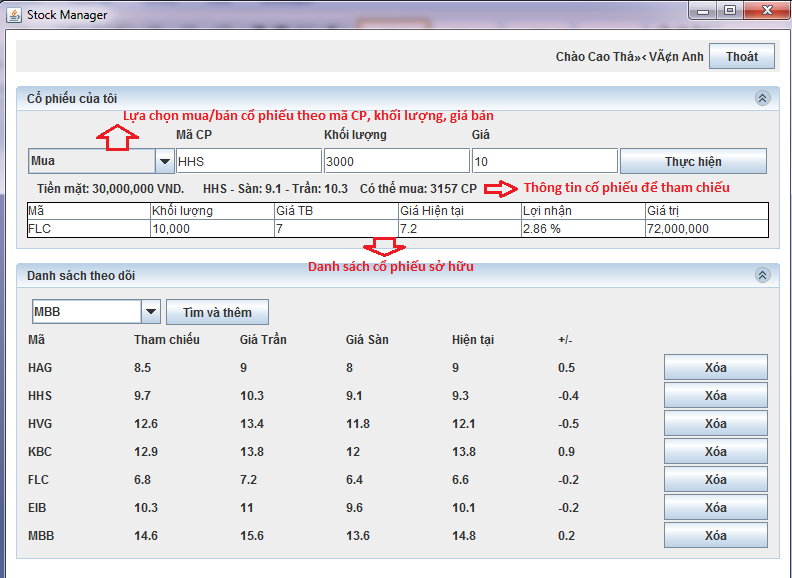


*Hình 11: Chức năng đăng nhập*

* Màn hình chức năng theo dõi và mua bán cổ phiếu của nhà đầu tư:



*Hình 12: Theo dõi cổ phiếu*

**

*Hình 13: Mua/bán cổ phiếu mà nhà đầu tư sở hữu*

# **Chương 3: Tích hợp AspectJ để kiểm thử ứng**

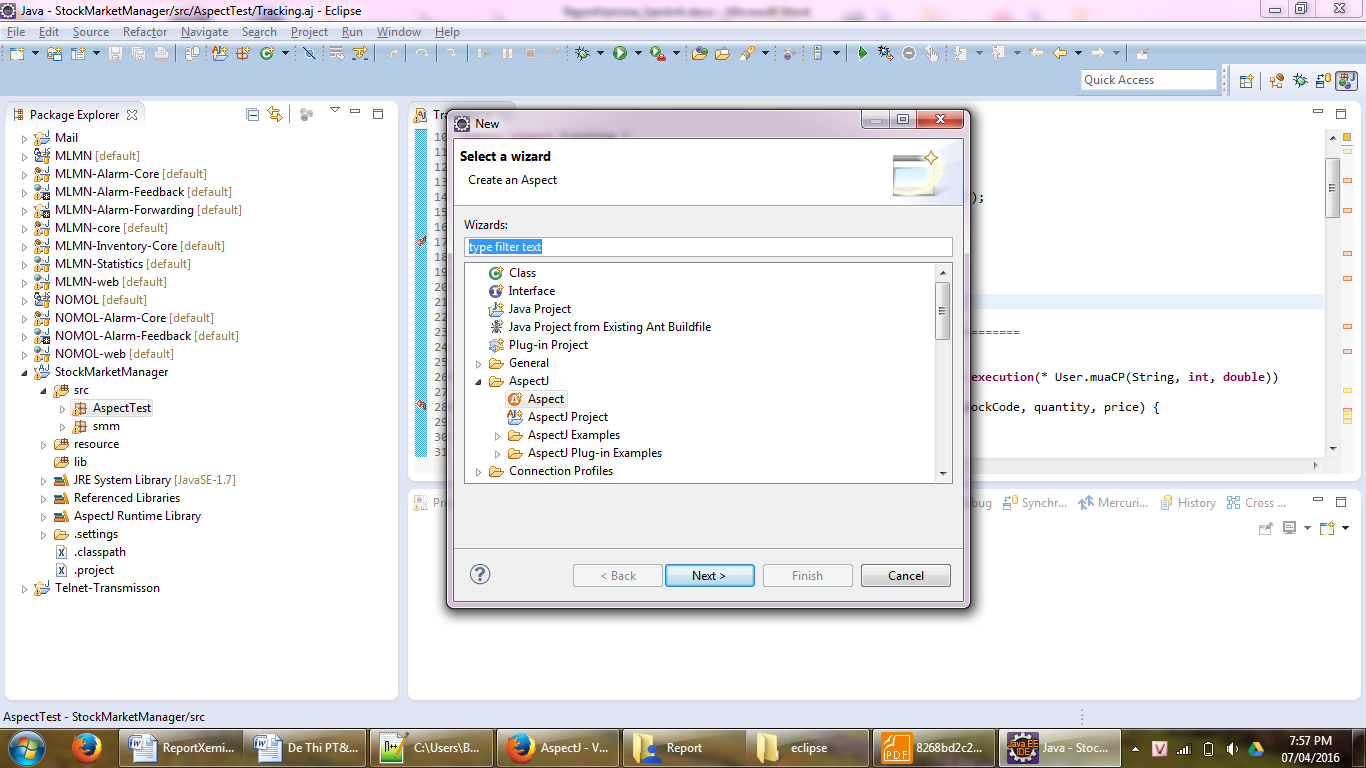
## **Tạo các AspectJ class**

Các bước tạo AspjectJ class như sau:

Bước 1: Mở source code chương trình Demo

Bước 2: Thêm một Package có tên bất kỳ (ví dụ: TestApjectJ)

Bước 3: Tạo AspjectJ class trong package vừa tạo.



*Hình 14: Cách tạo các lớp aspect*

## **Tạo các kịch bản kiểm thử AspectJ cho demo mua bán chứng khoán**

Trong quá trình viết chương trình demo, nhóm đã cố tình cài đặt một số lỗi trong chương trình. Mục đích để sử dụng AspectJ tìm ra các vấn đề lỗi đó.

### ***2.1 Xác định JoinPoint***

Trong ứng dụng demo có 2 JointPoint được xác định là có thể xảy ra lỗi:

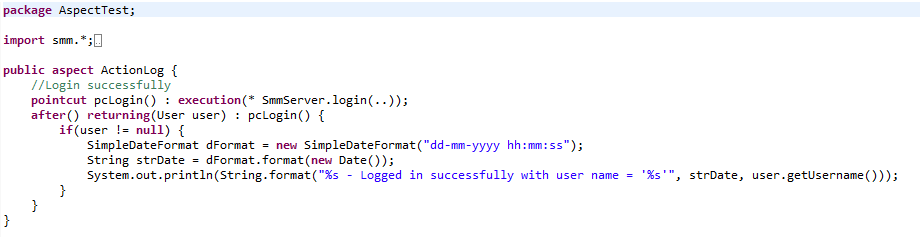
* Phương thức mua cổ phiếu : muaCP(String maCP, int khoiluong, double giamua)
* Phương thức bán cổ phiếu: banCP(String maCP, int khoiluong, double giaban)

Và 2 JointPoint dùng để Tracking trong ứng dụng:

* Login
* Control Flow

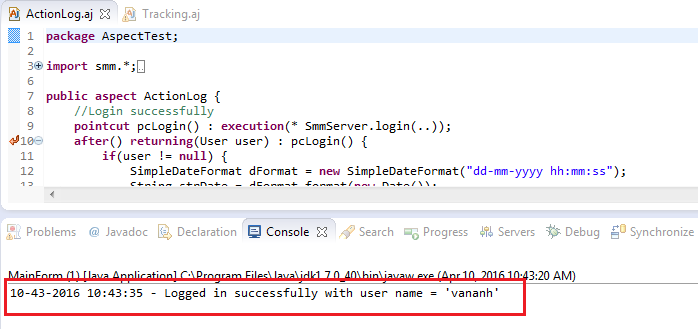
### ***2.2 Tạo các PointCut và Advice để thực thi các Pointcut tương ứng với các JointPoint đã xác định***

* Xây dựng PointCut và Advice để tracking quá trình login của người dùng:



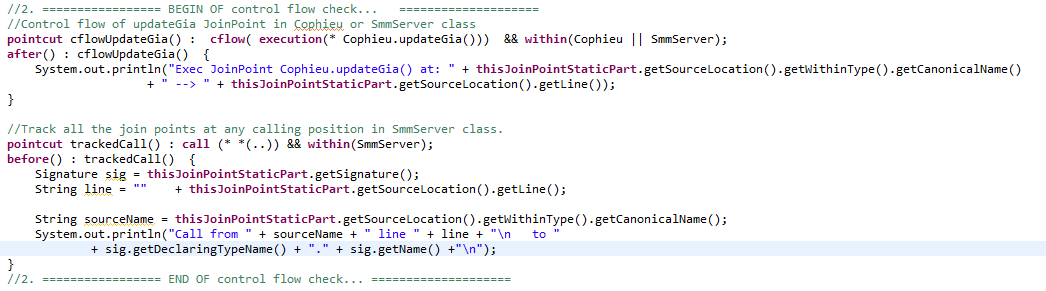
*Hình15 : Mã AspectJ cho Tracking Login*

Kết quả thu được khi chạy chương trình:

**

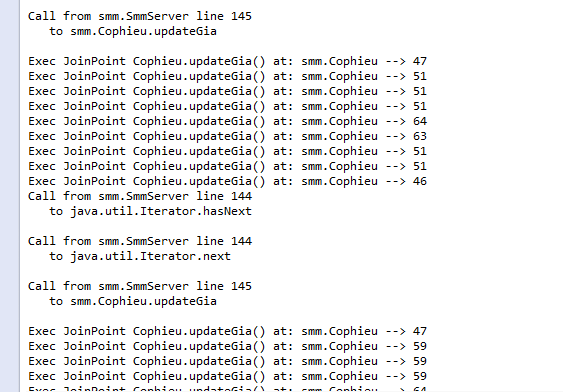
*Hình 16: Kết quả tracking login*

* Xây dựng PointCut và Advice cho Control Flow: là một pointcut hỗ trợ kiểm tra thực thi một phương thức (hàm) của một lớp (class) xác định được gọi từ các class khác nhau. Control flow có thể lấy các thông tin thực thi hàm như: thời điểm thực thi, điểm gọi hàm từ từ các class bên ngoài, …



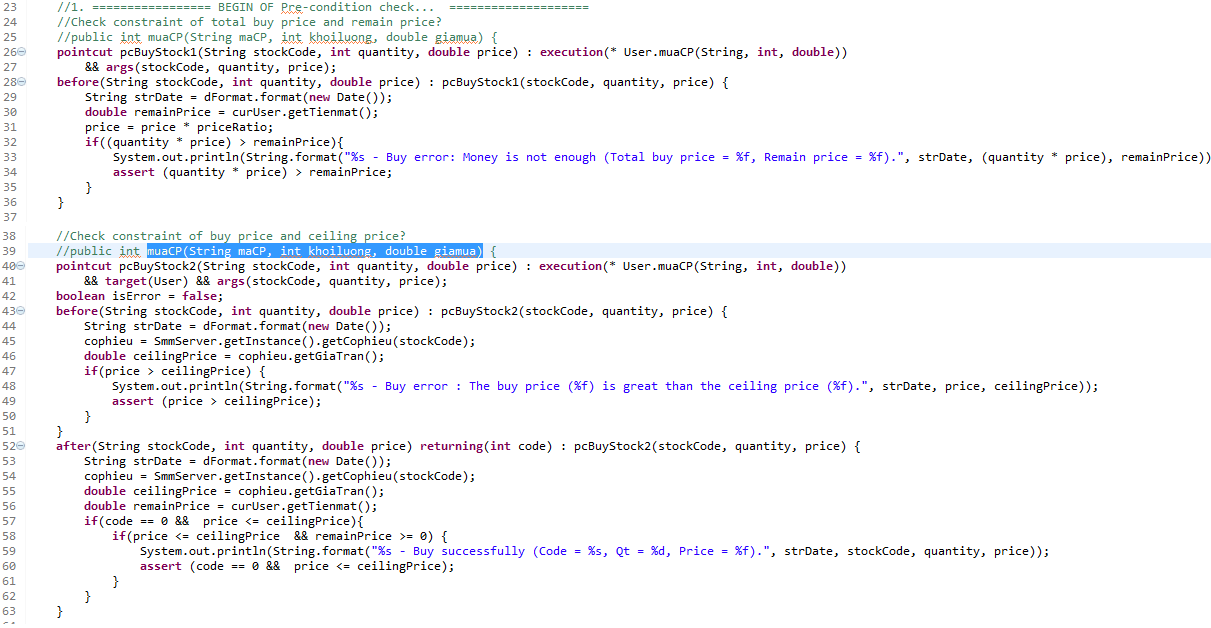
*Hình 17: Mã Aspect để tracking ControlFlow*

Kết quả thu được khi chạy chương trình:

**

*Hình 18: Kết quả tracking ControlFlow*

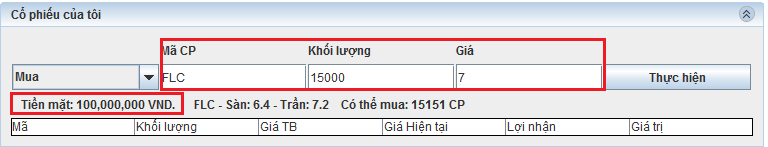
* Xây dựng PointCut và Advice để kiểm thử phương thức mua cổ phiếu:



*Hình 19: Mã Aspect để testing muaCP*

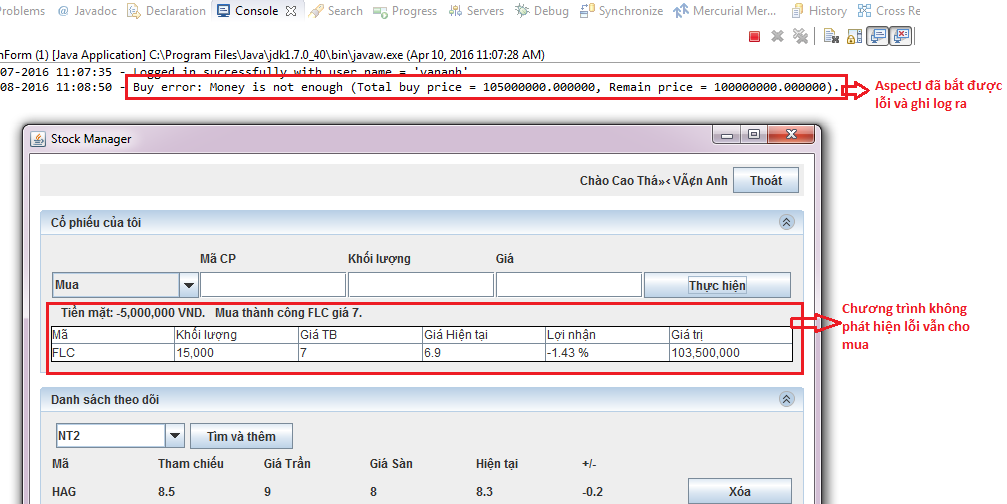
Kết quả thu được khi chạy chương trình:

* + Ban đầu số tiền mặt của nhà đầu tư là 100.000.000 VNĐ. Nhà đầu tư mua 15000 cổ phiếu của FLC với giá là 7



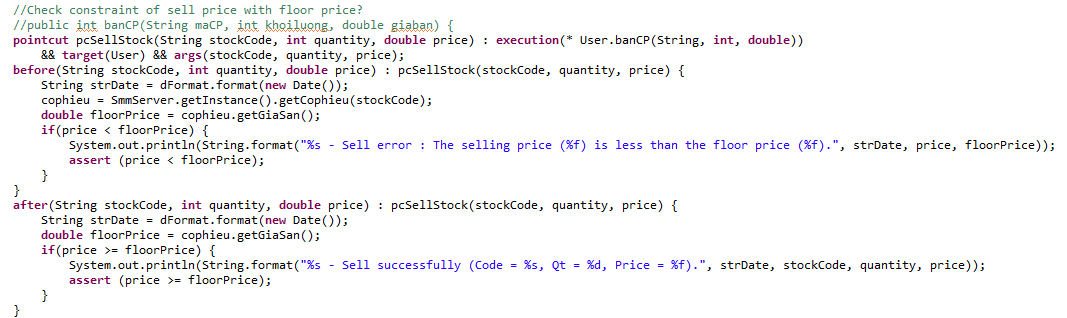
*Hình 20: Thao tác mua cổ phiếu cua nhà đầu tư*

* + Chương trình vẫn thực hiện phương thức mua 15000 cổ phiếu của nhà đầu tư, cho dù số tiền mặt của nhà đầu tư không đủ. Và số tiền mặt sau khi nhà đầu tư mua thành công là âm.=> Đây là một lỗi.
  + Khi cài đặt aspect cho phương thức mua cổ phiếu, AspectJ đã bắt được lỗi trong điều kiện mua không hợp lệ này.



*Hình 21: Kết quả sau khi thực hiện mua CP*

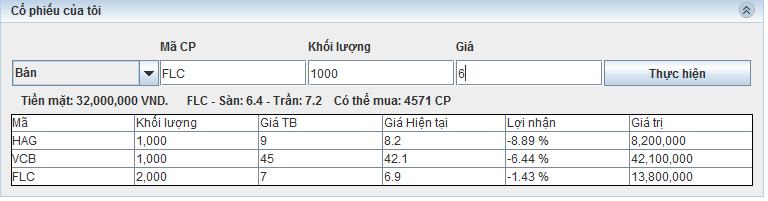
* Xây dựng PointCut và Advice để kiểm thử phương thức bán cổ phiếu:



*Hình 22: Mã Aspect để testing banCP*

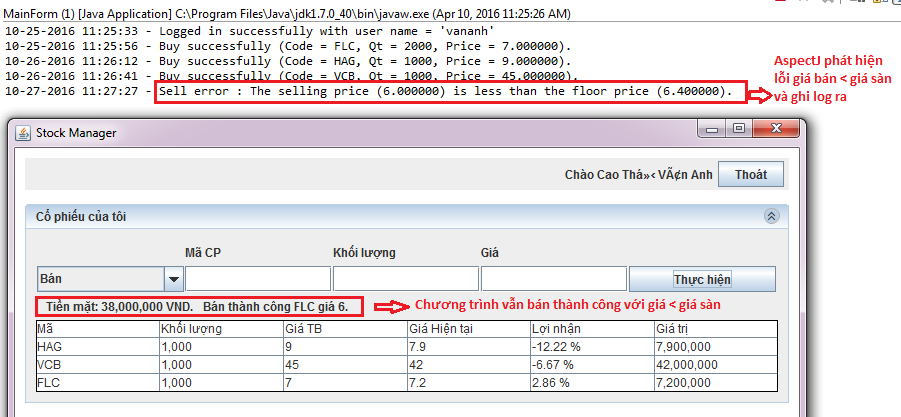
Kết quả thu được khi chạy chương trình:

* + Nhà đầu tư thực hiện bán 1000 cổ phiếu với giá bán thấp hơn giá sàn cho phép là 6.4



*Hình 23:Thao tác bán cổ phiếu của nhà đầu tư*

* + Chương trình vẫn thực hiện bán 1000 cổ phiếu FLC với giá bán là 6 < giá sàn là 6.4 => Đây là một trường hợp lỗi
  + Khi cài đặt mã aspect bắt điều kiện giá bán phải lớn hơn giá sàn thì chương trình đã phát hiện ra lỗi và ghi log ra cho người lập trình thấy.



*Hình 24: Kết quả sau khi thực hiện bán CP*

Kết luận: Như vậy, có những lỗi trong quá trình lập trình, nhà phát triển không thể kiểm soát hết được. Điều đó rất khó phát hiện nếu chương trình vẫn hoạt động bình thường. Khi xác định các jointpoint và xây dựng được aspect để kiểm soát vấn đề lỗi này, người lập trình có thể phát hiện được ra các vấn đề nghiệp vụ còn lỗi xảy ra trong chương trình của mình và khắc phục một cách nhanh chóng hơn.

# **Chương 4: Kết luận**

AOP là một ý tưởng mới, vẫn còn cần thời gian để đánh giá, tìm hiểu mối quan hệ các kỹ thuật hiện có và để phát triển, ứng dụng rộng rãi.

Cần nhấn mạnh điều quan trọng là AOP xây dựng trên các công nghệ hiện hữu. Nó chỉ làm việc với thiết kế tốt. Để có thiết kế tốt, bạn cần phải áp dụng đúng kỹ thuật cho đúng vấn đề, có thể là AOP, OOP, phân tích theo chức năng, lập trình cấu trúc…

Từ khả năng modun hóa các quan hệ đan xen, các chức năng cắt ngang hệ thống; tách rời sự hoạt động của các modun cũng như nhiều ưu điểm khác của AOP so với OOP mà hiện nay AOP đã trở thành sự lựa chọn phù hợp cho rất nhiều hệ thống phần mềm; đặc biệt là trong các chức năng lưu vết, bảo mật, xác thực của hệ thống phần mềm. Ngoài ra, do các mã aspect độc lập với mã nguồn chính của chương trình , có thể sửa đổi tùy ý theo ý muốn của người lập trình vì vậy mà AOP còn được ứng dụng rất lớn vào các loại kiểm chứng trong quá trình thiết kế phần mềm. Ví dụ như: kiểm chứng giao thức, kiểm tra việc gọi tuần tự các hàm trong chương trình...

Nội dung chính của các phương pháp kiểm chứng dựa trên AOP là dựa vào những khái niệm cơ bản của AOP như: join point, pointcut, advice, aspect để xây dựng lên các modun kiểm chứng (các aspect) từ các chức năng cắt ngang hệ thống. Các aspect này sẽ được đan vào các khung mã nguồn chương trình thông qua trình biên dịch AspectJ để thực hiện chức năng kiểm chứng.

Qua nghiên cứu và thực nghiệm nhóm thấy việc kiểm thử bằng AspectJ khá còn mới mẻ nên cần nhiều nỗ lực hơn nữa đến từ các cá nhân, tổ chức để ứng dụng kiểm thử hướng khía cạnh vào thực tế. Việc viết các aspect kiểm thử có cấu trúc đơn giản, dễ học có khả năng chỉnh sửa cao.

Trong thời gian tới, nhóm cũng như cá nhân em rất muốn tìm hiểu sâu hơn nữa những khía cạnh khác, do thời gian có hạn nên với báo trên nhóm mới chỉ tìm hiểu chung về mặt lý thuyết để từ đó ứng dụng vào bài tập demo.

# **TÀI LIỆU THAM KHẢO**

[1] <http://octodecillion.com/blog/aop-advice-as-mocks/>

[2] <http://octodecillion.com/blog/java-test-xstream-serialization/>

[3] https://eclipse.org/aspectj/doc/next/progguide/index.html

[4] http://www.runtime-verification.org/course09/lecture2/lecture2.pdf

[5] http://laptrinh.vn/d/6091-aspect-oriented-programming-aop-la-gi-aop-voi-aspectj.html

[6] http://www.eclipse.org/aspectj/doc/released/faq.php

[7] http://www.fernuni-hagen.de/ps/prjs/IIIA/examples-checksum.html

[8] https://blog.idrsolutions.com/2015/02/8-useful-java-testing-tools-frameworks-programmers-developers-coders/

[9] http://programmers.stackexchange.com/questions/210168/can-aspectj-or-aop-in-general-be-used-to-test-exception-handling

[10] http://www.objectteams.org/publications/WTAOP05.pdf

[11] http://denis-zhdanov.blogspot.com/2009/09/aspectj-call-vs-execution-pointcuts.html

[12] http://www.ibm.com/developerworks/library/j-ceaop/

[13]http://lore.ua.ac.be/Teaching/SE3BAC/practicum/designByContract/ADBC\_README.pdf