**Bài thực hành 04 – Tuần 11**

* Link đã nộp bài lên Git/GitHub:

<https://github.com/SuPerEle14/TranTheLam-20183937-710809>

* Link Source code:

https://github.com/SuPerEle14/TranTheLam-20183937-710809.git

Mục lục

[1. MỤC ĐÍCH VÀ NỘI DUNG 1](#_Toc90072872)

[2. CHUẨN BỊ 1](#_Toc90072873)

[3. NỘI DUNG CHI TIẾT 2](#_Toc90072874)

[3.1. Kiểm thử đơn vị 2](#_Toc90072875)

[3.2. Lập trình 8](#_Toc90072876)

[3.3. BÀI TẬP 15](#_Toc90072877)

**Báo Cáo 04: Lập trình và kiểm thử đơn vị**

## MỤC ĐÍCH VÀ NỘI DUNG

* Trước tiên, người học được hướng dẫn thực hành sử dụng Eclipse IDE, JUnit Framework và thực hành thiết kế các Testcases. Sau đó, người học sẽ được học cách tạo và sinh Javadoc tự động với Eclipse
* Trong phần Unit test, này người học sẽ được hướng dẫn về:
  + Các bước thiết kế TestCase cho UnitTest
  + Tạo TestCase cho các class bằng Eclipse và Junit
  + Tiếp cận phương pháp Test Driven Development (TDD) trong quá trình xây dựng phần mềm
* Tiếp theo người học sẽ được học về cách refactoring code. Người học sẽ được hướng dẫn nhận diện một class hoặc method cần được refactoring và sau đó sẽ thực hành refactoring một class
* Sau bài học người học sẽ có thể áp dụng những kiến thức mình đã được học vào bài tập về nhà cũng như Capstone Project

## CHUẨN BỊ

* Người học cần cài đặt sẵn Eclipse và môi trường Java11 trên máy cá nhân
* Clone project từ: <https://github.com/leminhnguyen/AIMS-Student>

## NỘI DUNG CHI TIẾT

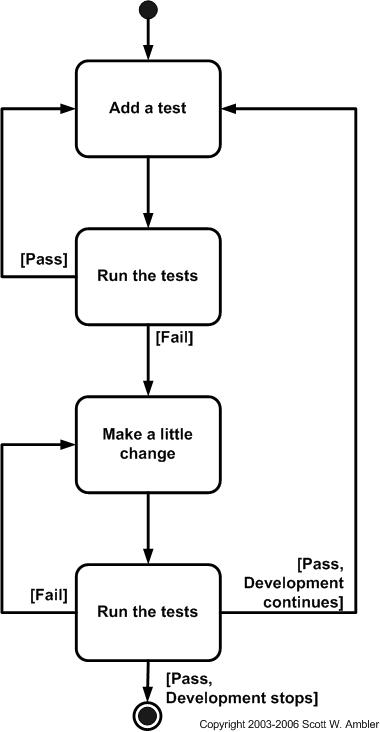
### Kiểm thử đơn vị

#### Bắt đầu với Kiểm thử đơn vị

* Test là một phần nằm trong phần mềm phát triển, thực thi các phần khác nhau của phần mềm và xác nhận kết quả có đúng như mong đợi (expected result) hoặc thực thi đúng theo luồng các sự kiện mong muốn (behavior testing)
* Unit Test là các đoạn code hoặc chương trình được viết bởi lâp trình viên nhằm kiểm tra hành vi hoặc trạng thái của một chức năng cụ thể trong PM
* Unit Test nhắm tới một đơn vị nhỏ của phần mềm như method hoặc class. Những phụ thuộc bên ngoài nên loại bỏ ra khỏi unit test, thay vào đó sử dụng mock được tạo ra bởi các test framework
* Unit Test không phù hợp cho việc testing cho những thành phần có giao diện người dùng phức tạp hoặc có sự giao tiếp giữa các thành phần khác nhau trong chương trình.

#### Làm quen với TDD (test driven development)

* TDD là quá trình phát triển phầm mềm dựa trên việc các yêu cầu chức năng của phầm mềm được chuyển thành các testcase trước khi phần mềm được phát triển hoàn thiện và theo dõi quá trình phát triển phần mềm bằng cách kiểm thử phần mềm với các testcase đó. Hay nói một cách đơn giản là chúng ta sẽ tiến hành viết test trước khi code và điều này trái ngược với việc một phần mềm được hoàn thiện rồi mới bắt đầu viết test



* Các bước thực hiện TDD
  + **B1 - Add a Test:**  bước đầu tiên trước khi bắt đầu phát triển một tính năng mới sẽ là viết test cho chức năng cần test (thông thường test ban đầu sẽ fail do có thể class cần Test chưa được viết)
  + **B2 - Run the Tests:** chạy các đoạn test đã viết
  + **B3 - Make A Litte Change:** tiến hành viết code hoặc cập nhật chức năng để có thể vượt qua các Tests
  + **B4 - Run the Tests:** thực hiện chạy lại các test, nếu như failed thì ta cần quay lại cập nhật code và chạy lại test cho đến khi pass. Sau khi vuợt qua các test thì ta bắt đầu lặp lại quá trình từ đầu

#### Làm quen với JUNIT5

* Junit5 là phiên bản thứ 5 của Junit, một framework sử dụng annotation để nhận diện ra các phương thức test và Junit là một phần mềm mã nguồn mở
* Các phiên bản của Junit đi kèm theo với Eclipse, vì vậy chúng ta không cần phải cài đặt thủ công
* Chi tiết về bước add Junit5 vào trong Eclipse sẽ được trình bày chi tiết ở bước tiếp theo
* Junit nhận diện các phương thức cần test bằng các annotation (bắt đầu bằng @), một vài annotation được sử dụng phổ biến trong Junit5
  + @Test: Biểu thị một phương thức test (Test Method)
  + @DisplayName: Khai báo tên cho Test Class hoặc Test Methodethod
  + @BeforeEach: phương thức được thực thi trước khi bắt đầu mỗi Test Method
  + @AfterEach: phương thức được thực thi sau khi chạy xong mỗi Test Method
  + @BeforeAll: phương thức được thực thi trước khi tất cả các Test Method được thực hiện (VD: connect tới DB,..)
  + @AfterAll: phương thức được thực thi sau khi tất cả các Test Method được thực hiện (VD: đóng connection tới DB,...)
* Để tìm hiểu chi tiết hơn và cụ thể hơn về Junit5, người học có thể tham khảo theo các link sau
  + <https://www.journaldev.com/20834/junit5-tutorial>
  + <https://junit.org/junit5/docs/current/user-guide/>

#### Thực hành thiết kế unit tests theo hướng TDD

1. **Đặc tả yêu cầu**

* Trong phần này chúng ta sẽ thực hành việc thiết kế các testcases cho phương thức *validateDeliveryInfo* nằm trong controller *PlaceOrder*
* Input đầu vào của phương thức *validateDeliveryInfo* sẽ là thông tin người dùng nhập vào như: *name, phone, address*
* Trong đó:

|  |  |  |
| --- | --- | --- |
| **STT** | **Tên tham số** | **Yêu cầu** |
| 1 | name | chỉ bao gồm chữ cái, không chứa ký tự đặc biệt, không được phép null |
| 2 | phone | chỉ bao gồm chữ số, độ dài 10 ký tự và bắt đầu là số 0 |
| 3 | address | không được phép null, không chứa ký tự đặc biệt |

* Công việc chúng ta cần phải thực hiện là thiết kế unit tests dựa trên đặc tả và xây dựng phương thức *validateDeliveryInfo* theo quá trình TDD
* Chúng ta có thể tách nhỏ *validateDeliveryInfo* thành 3 phương thức nhỏ hơn là *validateAdress*, *validateName* và *validatePhoneNumber*

1. **Tạo UnitTest bằng Eclipse**

* Tạo một package tên controller ở trong folder test, sau này khi tiến hành tạo các testcase cho các class khác thì chúng ta nên tuân theo cấu trúc của class đó ở trong src folder
* Tiếp theo chúng ta sẽ tiến hành xây dựng các class Test cho các phương thức này bằng Junit5. Click chuột phải vào Project -> New -> Junit Testcase. Sau đó điền các thông tin cần thiết như:
* ***Source folder:*** là nơi các class Test được tạo ra, chúng ta sẽ đặt các testcase ở trong folder test
* ***Package*:** là package ở trong folder test mà chúng ta muốn đặt các test case
* ***Name:*** là tên của class Test
* ***Class Under Test:*** là tên class mà chúng ta đang cần test
* Sau đó click Finish, nếu như project chưa có Junit thì Eclipse sẽ hiện lên popup yêu cầu add Junit -> Click ok
* Đến đây chúng ta đã học được cách tạo thành công testcase cho một class, và nếu như bấm run thì tất nhiên là kết quả sẽ failed bởi vì chúng ta chưa implement bất kỳ phần code cần test nào

1. **Thực hành xây dựng các phương thức theo TDD**

* Bây giờ chúng tay sẽ bắt tay vào quá trình TDD, nếu như bây giờ chúng ta khởi tạo một đối tương của PlaceOrderController ở trong class ValidatePhoneNumber và test phương thức validatePhoneNumber với một số điện thoại hợp lệ (VD: 0123456789) thì chúng ta sẽ bị failed, lý do là code ban đầu của chúng ta đang mặc định return false)
* Vì vậy chúng ta sẽ quay lại class PlaceOrder và implement phương thức validatePhoneNumber. Số điện thoại chỉ được bao gồm 10 ký tự số vậy nên ta sẽ implement, và nếu chúng ta quay lại để chạy lại testcase thì chúng ta sẽ pass

Ảnh có chứa văn bản

Mô tả được tạo tự động

implement validPhoneNumber

* Chúng ta có thể tối ưu dữ liệu test bằng cách đưa vào một list các cặp dữ liệu - kết quả mong đợi (input - expected outcome). Chúng ta có thể làm được điều này bằng cách sử dụng annotation @Parameterized và @CsvSource

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

iimplement validatePhoneNumberTest và kết quả chạy

* ValidateName theo TDD

Ảnh có chứa văn bản

Mô tả được tạo tự động

implement validateName

test và kết quả

* ValidateAddress theo TDD
* implement validateName

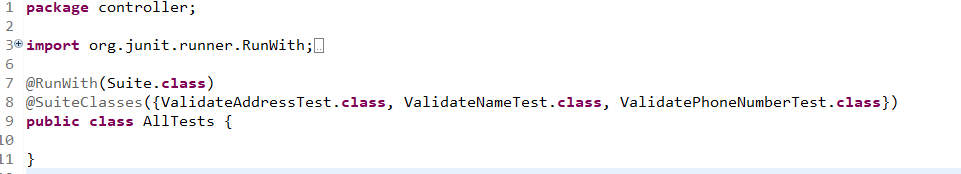
Ảnh có chứa văn bản

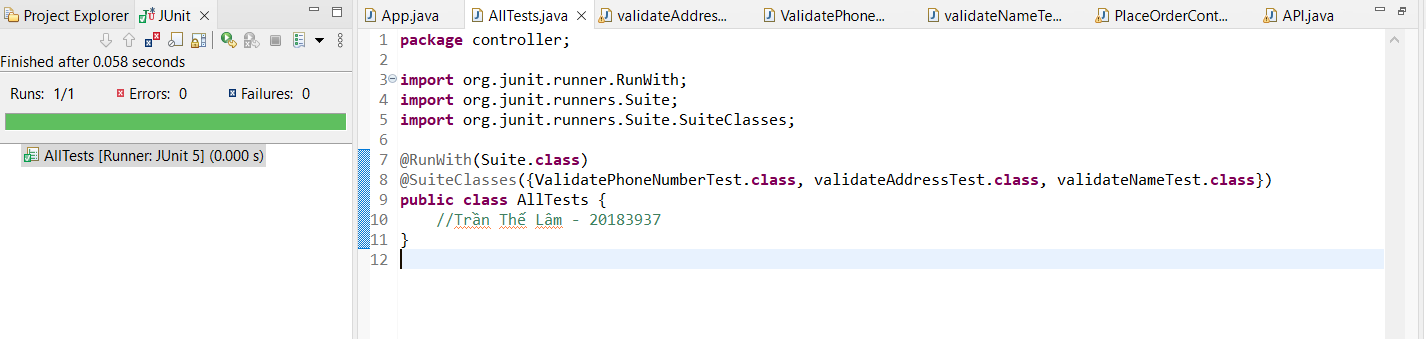
Mô tả được tạo tự động

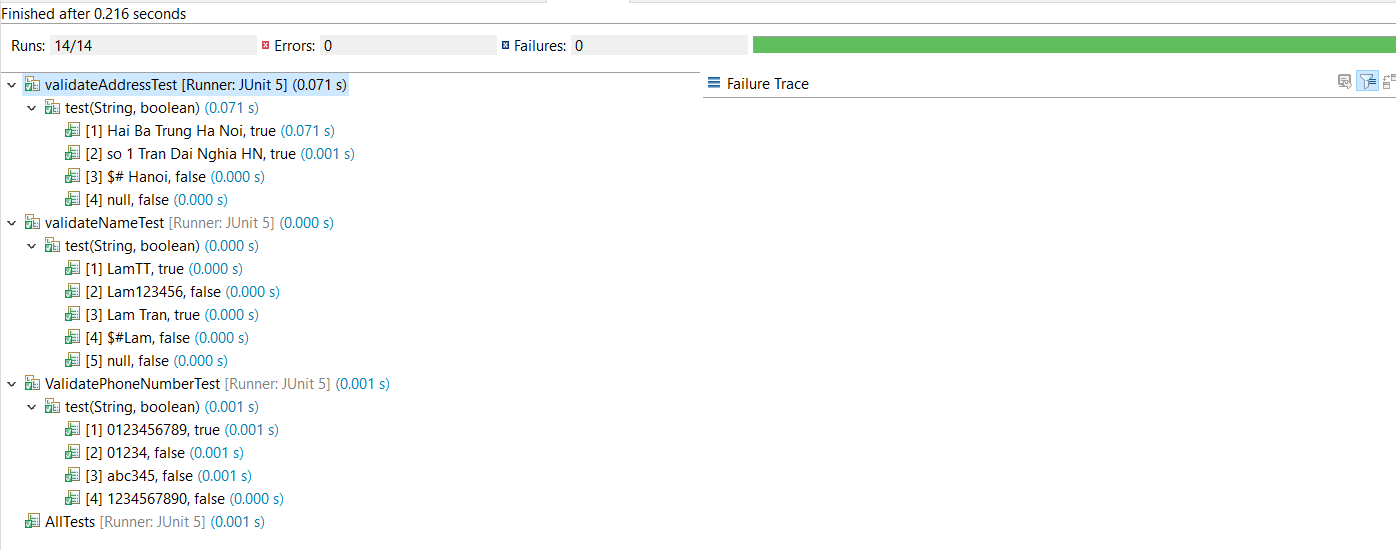
test và kết quả

1. **Thực hành tạo Test Suite**

* Sau khi hoàn thiện chúng ta sẽ có 3 class Test, và chúng ta có thể tiến hành tạo Test Suite. Test Suite là một tập các testcases có liên quan đến một nghiệp vụ nào đó. Click chuột phải vào project -> New -> Test Suite
* Sau đó chúng ta thêm các class cần test vào class AllTests



* Nếu muốn nhìn kết quả một cách trực quan click vào trong folder test/controller sau đó bấm Run thì Eclipse sẽ chạy hết tất cả các testcase nằm ở bên trong và show kết quả của từng TestCase một cách rất trực quan
* 



### Lập trình

#### Bắt đầu với code sample

* Trước khi đến với bài học các bạn hãy xem qua một class có tên ***API.java*** khoảng 15-20p và hãy suy ngẫm các câu hỏi:
  + Class này mục đích là gì ?
  + Class này do ai viết ?
  + Sử dụng class này như thế nào ?
* Nếu như sau khi xem xong và bạn đảm bảo hiểu hết những gì viết viết trong class thì chắc hẳn bạn phải là một lập trình viên Java nhiều kinh nghiệm
* Tuy nhiên đa phần các bạn sẽ bị choáng ngợp và cảm thấy khó hiêu ở class khoảng gần 100 dòng này. Bạn sẽ không hiểu class này mục đích là gì, dùng như thế nào và ai viết
* Không những class trên gây khó hiểu mà nó đang còn tồn đọng nhiều vấn đề như tái cấu trúc, tối ưu code,..
* Vây bài học hôm nay sẽ giúp các bạn giải quyết từng vấn đề và bạn có thể áp dụng vào trong Project của mình

#### Làm quen với Javadoc

* Chắc hẳn trong chúng ta thì ai cũng đã nghe tới khái niệm như thêm comment hoặc documentation cho các class hoặc method mình viết
* Documentation là gì? Documentation đơn giản là các đoạn text được thêm vào trong mã nguồn của dự án phầm mềm với mục đích giải thích những cái bạn đang làm, thực hiện như thế nào và làm thế nào để sử dụng nó.
* Việc tạo thói quen thêm comment và documentation vào code thì có vai trò rất quan trọng, nó sẽ giúp cho người khác hiểu code của bạn, và cũng có thể là chính bạn sau này khi đọc lại code của mình. Hơn thế nữa việc thêm documentation vào code sẽ chứng tỏ bạn là một lập trình viên chuyên nghiệp.
* JAVADOC là documentation cho ngôn ngữ Java, mục đích chính của nó cũng là giúp cho bạn giải thích những đoạn code được viết. Việc thêm JAVADOC có thể thực hiện thủ công bằng cách gõ từng ký tự nhưng cũng có rất nhiều công cụ IDE hỗ trợ bạn tạo doc tự động như: Eclipse, intellij, VSCode,…
* Cú pháp của JAVADOC: các documentation của Java được đặt ở trong cặp /\*\* \*/ và có thể thêm nhiều dòng trong giữa cặp dấu.
* JAVADOC thường được mô tả bởi các annotation (bắt đầu bởi @), một vài loại annotatio phổ biến trong JAVADOC
  + ***@author***: chỉ tên tác giả của đoạn code hoặc có thể là người đóng góp nhiều nhất. Thường được áp dụng cho các level: class hoặc package
  + ***@param***: Mô tả tham số truyền vào một phương thức hoặc constructor
  + ***@return***: mô tả giá trị trả về của một class hoặc phương thức
  + ***@since***: phiên bản mà thuộc tính được thêm vào
  + ***@throws***: loại exception mà phương thức có thể tung ra
  + ***@deprecated***: chỉ cho người khác biết là phương thức hoặc class này không còn được sử dụng nữa
  + {***@link***}: tạo liên kết tới những phương thức hoặc phần nội dung khác
* Người học có thể tham khảo chi tiết thêm những loại annotation Java cung cấp ở link sau: <https://idratherbewriting.com/java-javadoc-tags/>

#### Thực hành tạo Javadoc với Eclipse

1. **Import project vào Eclipse**

* Mở Eclipse và import Project đã clone về máy, hoặc nếu bạn đã clone thì sử dụng git pull để lấy code mới nhất về nhánh master
* Mở class src/utils/API.java
* Tiếp theo chúng ta sẽ tiến hành thêm javadoc cho class và method ở trong class API.java này

1. **Thực hành thêm doc cho API class**

* Để thêm doc vào trong Java code thông qua Eclipse thì chúng ta có 3 cách:
  + Gõ thủ công
  + Dùng thanh công cụ
  + Dùng các shortcuts
* Để thêm javadoc cho class, method hoặc attribute thông qua thanh công cụ thì ta sẽ click con trỏ chuột lên ngay bên trên class, method hoặc attribute đó. Sau đó click chuột phải -> Source -> Generate Element Comment
* Cách nhanh nhất để thêm java doc chính là thông qua shortcurt. Để thêm javadoc thì ta click chuột lên phía trên của class, method hoặc attribute sau đó gõ /\*\* và ấn Enter, sau đó Eclipse sẽ tạo Javadoc tự động cho chúng ta
* Ví dụ với class API

Ảnh có chứa văn bản

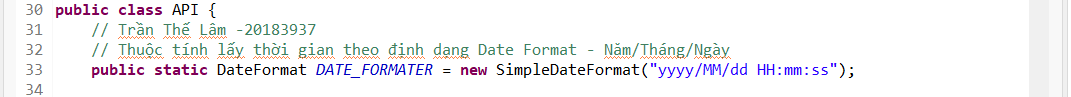
Mô tả được tạo tự động

* Việc thêm javadoc ở trên mới cho class API chỉ là bước đầu và khá sơ khai, vậy nên chúng ta cần bổ sung thêm các thông tin khác như: mô tả, @version, ngày viết chương trình
* Việc thêm mô tả cho một class, method hoặc attribute nên để lên đầu của documentation. Sau khi thêm các thông tin ta sẽ có thông tin documentation như sau:

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Sau khi thêm documentation cho class API thì chúng ta đã có một cái nhìn tổng quan về mục đích của class này . Tiếp theo chúng ta sẽ tiếp tục thêm documentation cho các attribute và method
* ***Attributes***



* ***get method***

Ảnh có chứa văn bản

Mô tả được tạo tự động

* ***post method***

Ảnh có chứa văn bản

Mô tả được tạo tự động

* ***allowMethods***

Ảnh có chứa văn bản

Mô tả được tạo tự động

1. **Thực hành thêm comment cho các phương thức**
   * Để làm rõ hơn các câu lện trong từng method thì chúng ta có thể thêm comment (//) vào trong các method như sau

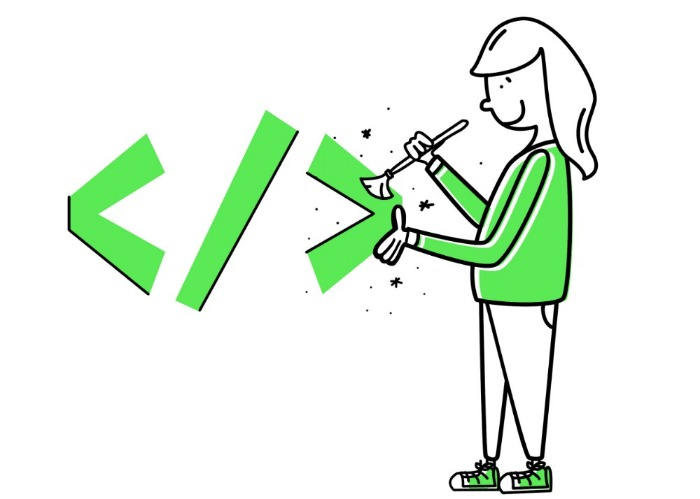
Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

#### Làm quen với refactoring



1. **Mục đích của refactoring**

* Refactoring là việc thay đổi hoặc tái cấu trúc lại mã nguồn nhằm mục đích chuyển code rối rắm (mess code) về dạng code dễ hiểu hơn (clean code).
* Refactoring giúp cho việc bảo trì một cách dễ dàng, giảm bớt đi sự phức tạp rườm rà từ đó giúp chúng ra dễ quản lý mã nguồn và đưa được product của mình ra thị trường một cách nhanh chóng

1. **Khi nào cần refactoring code**

* Khi code trong project của mình bị lặp đi lặp lại nhiều lần
* Khi code quá rườm rà, dài dòng, khó hiểu và khó thêm thuộc tính mới
* Trong quá trình fix bug, khi chúng ta liên tục gặp phải bugs thì việc refactoring cho code cleaner hơn sẽ giúp chúng ta nhận diện ra những vấn đề trong code
* Trong quá trình review code trước khi đưa ra sản phẩm, đây có thể là cơ hội cuối cùng để tinh chỉnh code trước khi đưa ra sản phẩm

1. **Một vài loại refactoring phổ biến**

* ***Extract method:*** phương pháp này có nghĩa là nếu như bạn thấy đoạn code nào đó lặp đi lặp lại nhiều lần trong các phương thức khác thì hãy tách đoạn code đó ra thành một phương thức riêng
* ***Extract class***: đây là cách làm tương tự như extract method nhưng ở mức độ class, những phương thức nào có liên quan tới nhau và hay được sử dụng thì chúng ta có thể tách ra một class khác và tái sử dụng bằng cách kế thừa hoặc kết tập

#### Thực hành refactoring code với class API

1. **Nhận diện vấn đề**

* Sau khi nhìn lại 2 methods và có sự phân cách giữa các dòng chúng ta sẽ thấy giữa 2 phương thức post và get này có các đoạn code chung như setup connection và đọc dữ liệu trả về từ server,
* Ta có thể tiến hành refactor đoạn mã nguồn này bằng cách chọn đoạn mã nguồn cần extract, ấn chuột phải, chọn Refactor 🡪 Extract method.

1. **Thực hành refactoring**

* Chúng ta sẽ tiến hành trích xuất 2 phương thức đặt tên là setupConnection và readResponse
* Code chúng ta sau khi refactoring sẽ như sau
  + ***setupConnection()***

Ảnh có chứa văn bản

Mô tả được tạo tự động

* + ***ReadResponse()***

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Khi đó 2 phương thức get và post sẽ còn lại như sau

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

## BÀI TẬP

* **Thiết kế các testcase cho UC Place Rush Order bằng phương pháp TDD như đã học và được hướng dẫn thực hành**
* **Thực hành thêm javadoc cho các class, method và attribute cho UC Place Rush Order. Sau đó tiến hành review code và refactoring lại những điểm code chưa hợp lý (lưu ý cần chỉ rõ cần refactoring ở điểm nào và tại sao)**

**Thiết kế cho UC PlaceRushOrder**

**Thực hiện 2 testcase validateAddressRushOrder (chỉ được nhận ở hà nội), validateDateRushOrder( kiểm tra ngày tháng giao hàng nhanh) bằng phương pháp TDD**

* validateAddressRushOrder

Ảnh có chứa văn bản

Mô tả được tạo tự động

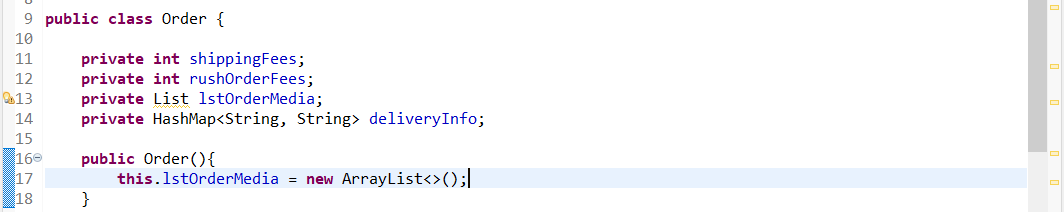
* **Thực hành thêm javadoc cho các class, method và attribute cho UC Place Rush Order. Sau đó tiến hành review code và refactoring lại những điểm code chưa hợp lý (lưu ý cần chỉ rõ cần refactoring ở điểm nào và tại sao)**

**API.java:**

code cho class API token được lưu trong Configs.java lên không cần truyền tham số token cho các method get và post trong lớp API.java

**Order.java**

Thêm thuộc tính rushOrderFees



Sửa lại cách tính

**PlaceOrderController.java**

Thêm cách tính phí giao hàng hàng hóa trên 1.000.000 được freeship

Ảnh có chứa văn bản

Mô tả được tạo tự động

**Thêm code:**

Để minh họa cho việc đặt hàng nhanh, thêm thuộc tính spRushOrder trong entity Media

<support Rush Order>

Ảnh có chứa văn bản

Mô tả được tạo tự động

**PlaceRushOrderController.java**

Được kế thừ từ PlacerOrderController.java

Ảnh có chứa văn bản

Mô tả được tạo tự động

**ShippingScreenHandler.java**

Tối ưu code

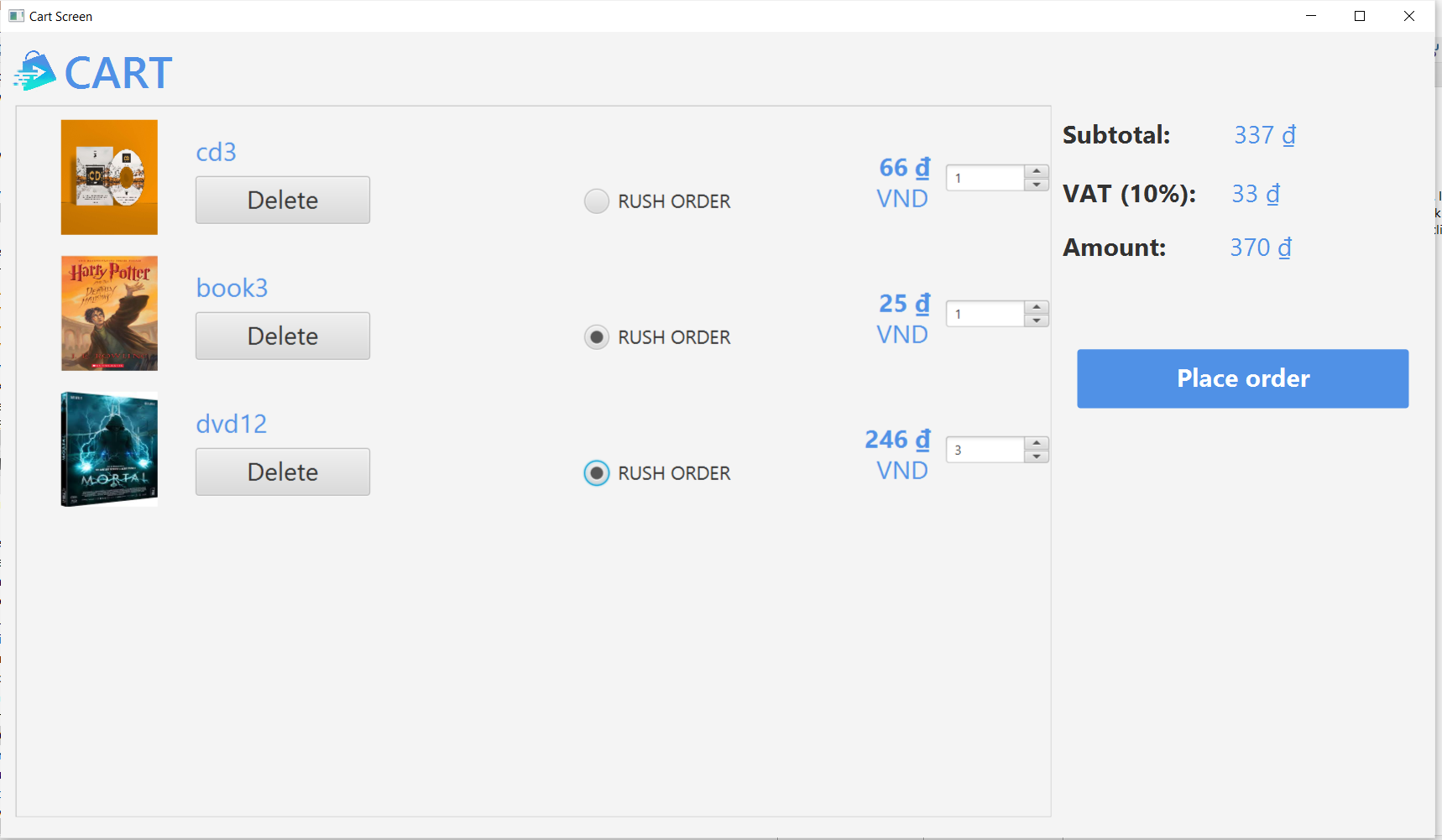
Ảnh có chứa văn bản

Mô tả được tạo tự độngẢnh có chứa văn bản

Mô tả được tạo tự độngẢnh có chứa văn bản

Mô tả được tạo tự động

**Một số hình ảnh chạy thử**



Ảnh có chứa văn bản

Mô tả được tạo tự động