**Nguyễn Năng Sơn 20194158**

**Báo cáo thực hành buổi 4**

**Lập trình và kiểm thử đơn vị**

Link github: <https://github.com/nangson2001/ThucHanhTK-XDPM>

**Mục lục**

Contents

[**I.** **Mục đích** 1](#_Toc90149186)

[**II.** **Kiểm thử đơn vị** 2](#_Toc90149187)

[**1.** **Test Driven Development (TDD)** 2](#_Toc90149188)

[**a)** **Khái niệm** 2](#_Toc90149189)

[**b)** **Các bước thực hiện TDD** 2](#_Toc90149190)

[**2.** **JUnits** 3](#_Toc90149191)

[**3.** **Thực hành thiết kế unit tests theo hướng TDD** 3](#_Toc90149192)

[**a)** **Viết code các phương thức còn thiếu** 3](#_Toc90149193)

[**b)** **Tạo Unit Test** 4](#_Toc90149194)

[**III.** **Lập trình** 5](#_Toc90149195)

[**1.** **Tạo JAVADOC** 5](#_Toc90149196)

[**a)** **Khái niệm** 5](#_Toc90149197)

[**b)** **Tạo JAVADOC** 6](#_Toc90149198)

[**2.** **Refactoring code** 6](#_Toc90149199)

# **Mục đích**

Trong bài thực hành này, em sẽ lập trình phần mềm và kiển thử đơn vị. Trong đó, em lập trình bằng IDE IntelliJ và kiển thử đơn vị bằng JUnit Framework. Cụ thể, em sẽ thiết kế Test Case cho Unit Test và tạo Test Case bằng JUnit, code các phần còn thiếu và refactoring code một caachs hợp lý.

# **Kiểm thử đơn vị**

## **Test Driven Development (TDD)**

### **Khái niệm**

 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**

Diagram

Description automatically generated

Figure 1: Các bước thực hiện TDD

* + Bước 1 - 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)
  + Bước 2 - Run the Tests: chạy các đoạn test đã viết
  + Bước 3 - Make A Litte Change: tiến hành viết code hoặc cập nhật chức năng để có thể vượt qua các Tests
  + Bước 4 - 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

## **JUnits**

* Junit là một framework sử dụng annotation để nhận diện ra các phương thức test và Junit là một phần mềm mã nguồn mở
* Junit nhận diện các phương thức cần test bằng các annotation (bắt đầu bằng @), một vài annotation được sử dụng phổ biến:
  + @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,...)

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

### **Viết code các phương thức còn thiếu**

Yêu cầu:

|  |  |  |
| --- | --- | --- |
| **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 |

* Phương thức validatePhoneNumber:

Text

Description automatically generated

* Phương thức validateName:

Graphical user interface

Description automatically generated

* Phương thức validateAddress:

Graphical user interface, text

Description automatically generated

### **Tạo Unit Test**

* Test cho validatePhoneNumber:

Text

Description automatically generated

* Test cho validateName:

Text

Description automatically generated

* Test cho validateAddress:

Text

Description automatically generated

## **Lập trình**

## **Tạo JAVADOC**

### **Khái niệm**

* Documentation 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.
* 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

### **Tạo JAVADOC**

JAVADOC cho lớp API.java:

Text

Description automatically generated

## **Refactoring code**

Class API.java sau khi refactoring code:

Text

Description automatically generated