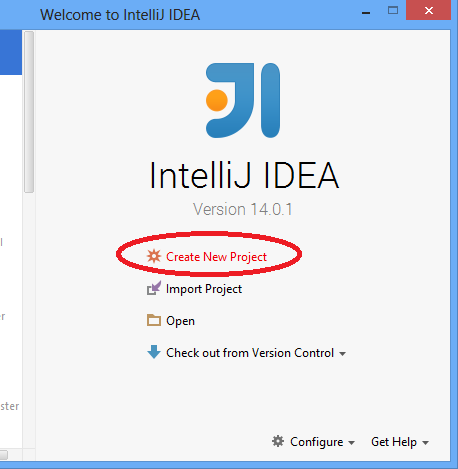
# Hướng dẫn đơn giản sử dụng IntelliJ

**Phần 1: Mở project, Viết chương trình và chạy.**

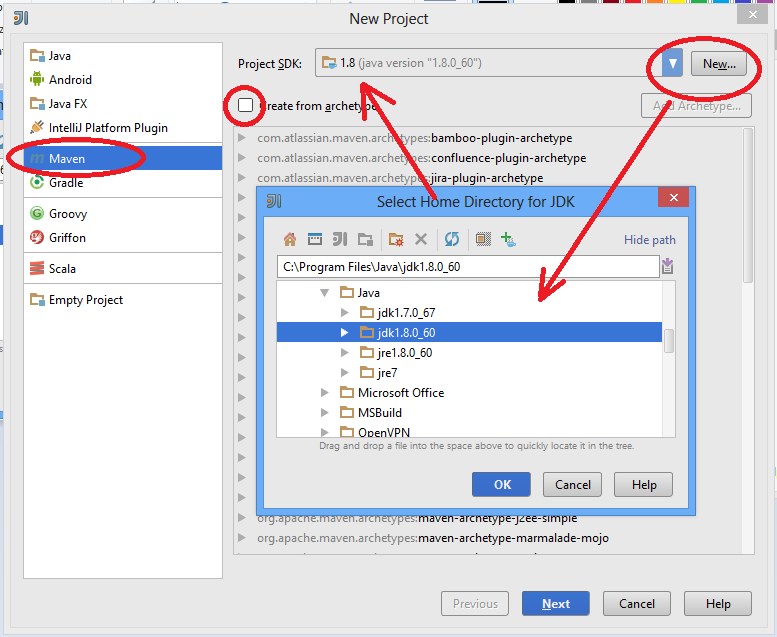
Bước 1.

Chọn mở Project mới

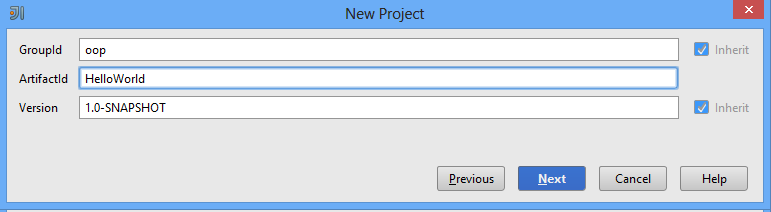


Bước 2.

Tạo project kiểu Maven cho Java 1.8. Chú ý có 03 nơi cần chọn đúng trong như hình vẽ. (1) Maven, (2) KHÔNG tick ‘Create from archetype’, (3) chọn Project SDK 1.8.

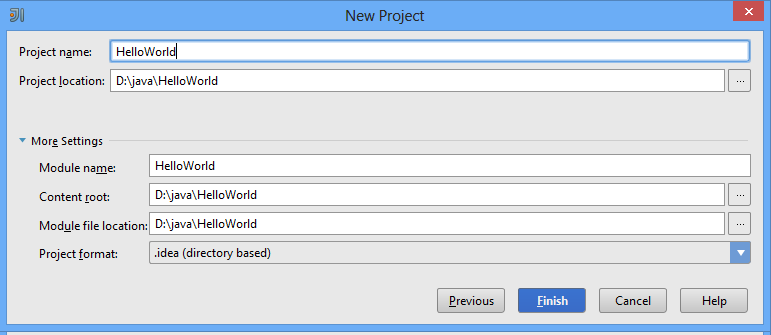


Bước 3. Đặt tên project Maven (HelloWorld) và nhóm của project (ví dụ oop, bạn có thể lấy tên khác, chẳng hạn nick của bạn)

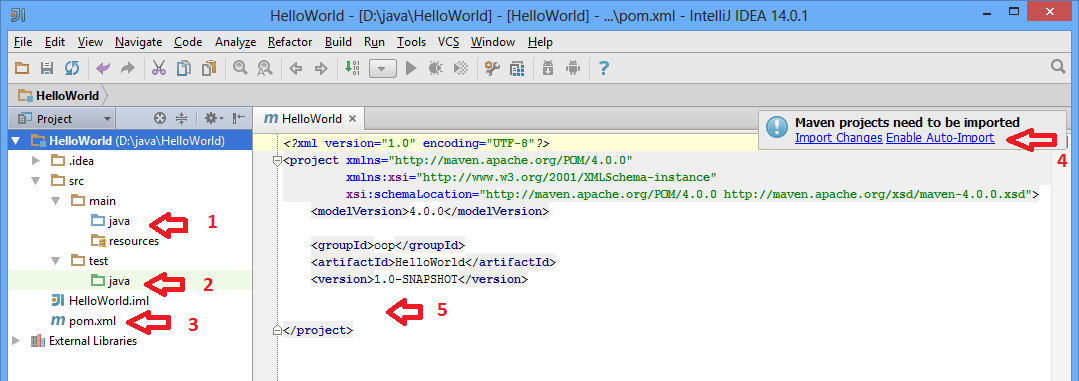


Bước 4.

Đặt tên project IntelliJ, tên này sẽ trùng với tên thư mục. Có thể trùng với tên project Maven. Bạn chỉ cần gõ vào ô Project name, phần còn lại sẽ được tự điền.



Bấm Finish, ta có khung project đã được tạo sẵn, nó được quản lý theo hai kiểu: (1) project kiểu IntelliJ do IntelliJ điều khiển, và (2) project kiểu Maven mà khi ra ngoài môi trường IntelliJ ta vẫn sử dụng được, chẳng hạn chạy bằng dòng lệnh hoặc mở tại Eclipse, NetBean…..



Trong đó,

1. src/main/java: nơi chứa mã nguồn chương trình

2. src/test/java: nơi chứa mã nguồn của test

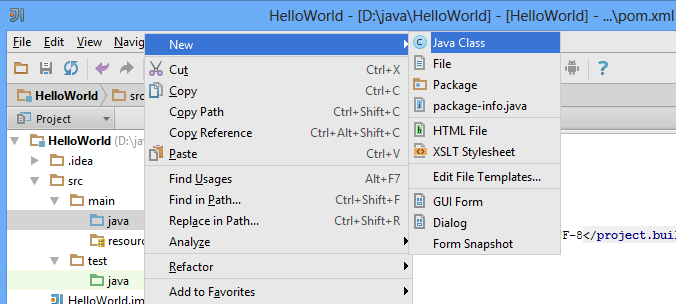
3. pom.xml tại thư mục gốc của project là file cấu hình của project maven, khi cần chỉnh cấu hình project thì sẽ sửa nội dung của những file này. Nội dung file pom.xml đó đang được hiển thị tại tab ‘mHelloWorld’ với ý nghĩa m là Maven, HelloWorld là tên của Maven project.

4. Bạn cần click vào link Enable Auto-Import, để từ nay, mỗi khi bạn sửa nội dung của file pom.xml thì cấu hình project IntelliJ (nằm trong thư mục .idea) sẽ tự thay đổi theo.

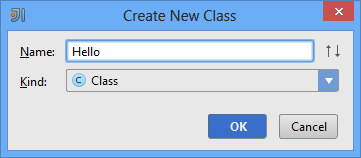
5. Để cấu hình maven chạy cho Java 1.8, quy định mã nguồn dạng UTF-8, và khai báo sử dụng thư viện JUnit để viết test, bạn copy đoạn sau vào vị trí số 5 tại hình trên.

<**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
</**properties**>  
  
<**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-compiler-plugin</**artifactId**>  
 <**version**>2.3.2</**version**>  
 <**configuration**>  
 <**source**>1.8</**source**>  
 <**target**>1.8</**target**>  
 </**configuration**>  
 </**plugin**>  
 </**plugins**>  
</**build**>  
  
<**dependencies**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.11</**version**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
</**dependencies**>

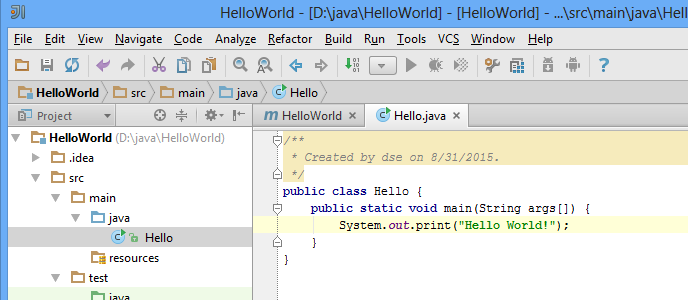
Bước 5. Tạo class đầu tiên: Dùng chuột phải tại thư mục main/java, chọn New / Java Class từ popup.



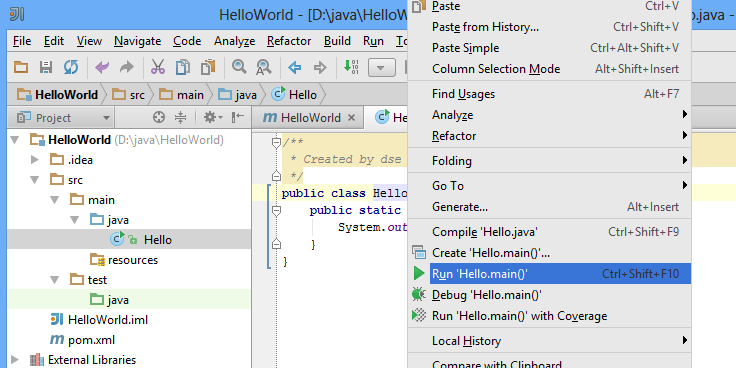
6. Nhập tên class Hello.



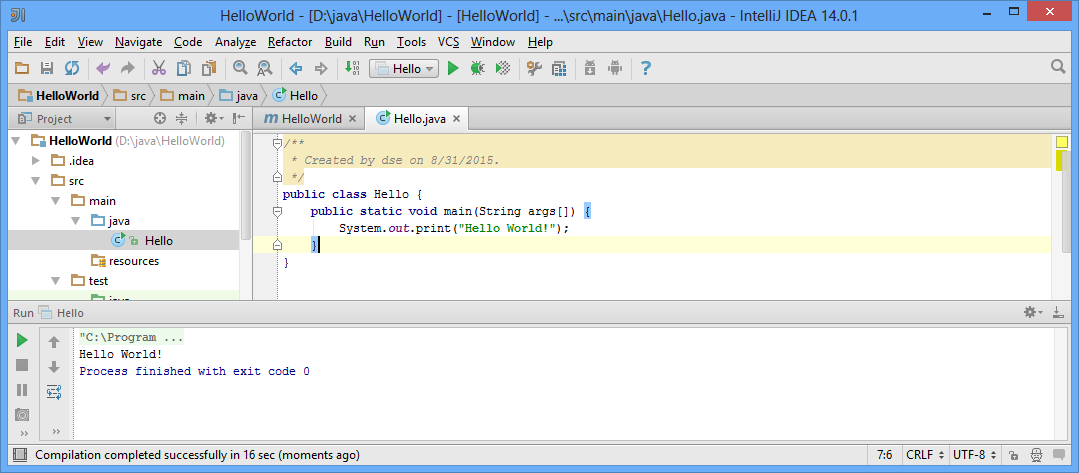
7. Viết chương trình Hello



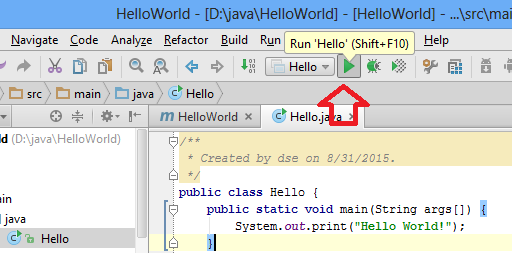
8. Chạy chương trình. Click chuột phải vào tên hàm main, chọn Run “Hello.main()…”. Có thể dùng tổ hợp phím Ctrl+Shift+F10.



Chương trình sẽ được dịch, đóng gói…. Và chạy ra kết quả:

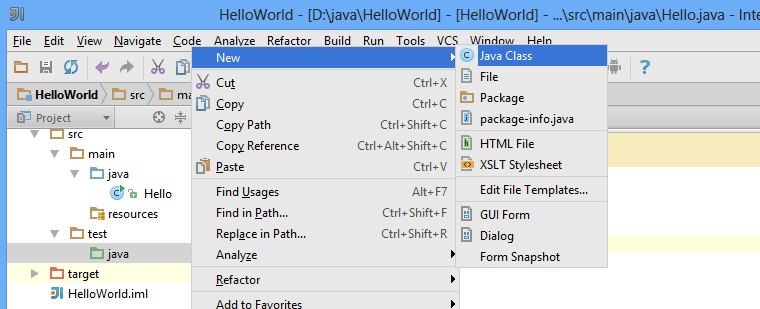


Từ đây, nếu muốn chạy lại, có thể dùng phím tắt, chuột phải như trước, hoặc click vào hình tam giác màu lục bên phải ô Hello.



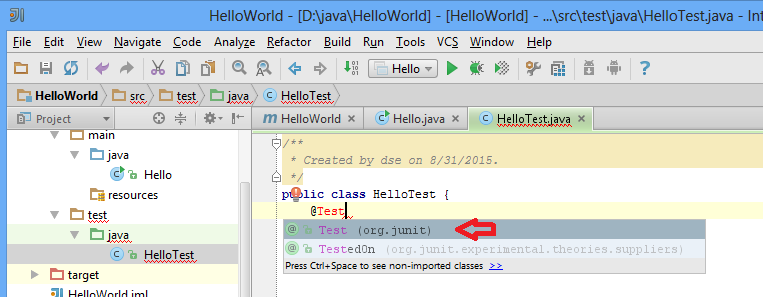
**Phần II: Viết test và chạy test.**

1. Tạo một class nằm trong thư mục test/java (click chuột phải để mở pop up), với tên HelloTest



2. Viết test. Viết code. Sinh mã.

Mỗi test được viết trong 1 hàm, mỗi hàm đó cần được bắt đầu bằng annotation @Test. Gõ vào trong class HelloTest nội dung @Test, IntelliJ sẽ gợi ý các thư viện cần sử dụng.



Chọn (org.junit) Dòng sau sẽ được điền tự động vào file HelloTest.java

import org.junit.Test;

Dòng đó có nghĩa file này sẽ dùng đến class Test trong gói thư viện org.junit. Đây cũng là ví dụ về việc IntelliJ gợi ý người dùng và hỗ trợ điền code.

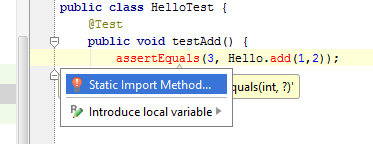
Viết nốt phần còn lại của test, test này ta muốn định nghĩa yêu cầu về một hàm add của lớp Hello có nhiệm vụ tính tổng hai số. Dòng duy nhất trong hàm testAdd có tác dụng gọi hàm Hello.add(1,2), lấy kết quả trả về và so sánh với 3. Ở đây, 3 là kết quả trông đợi (expected) vì 1+2=3. Giá trị trả về của hàm Hello.add(1,2) là kết quả thực tế. Hàm assertEquals (của lớp Assert trong thư viện org.junit) so sánh kết quả trông đợi với kết quả thực tế, nếu bằng nhau thì test đó đạt, nếu không bằng nhau thì test đó không đạt.



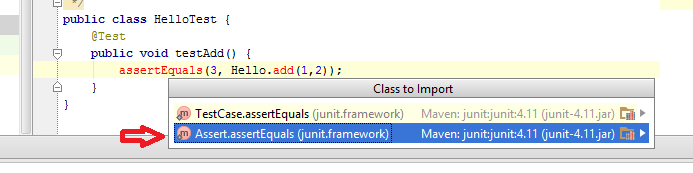
Chú ý các phần màu đỏ, đó là báo hiệu về các lỗi biên dịch. assertEquals là một hàm của lớp Assert trong thư viện org.unit, ta chưa khai báo import nên bị báo đỏ. Hàm add của Hello chưa được viết, nên cũng bị báo lỗi.

Tại mỗi điểm có lỗi, đưa con trỏ vào đó, bấm tổ hợp Alt-Enter sẽ có gợi ý sửa lỗi hoặc sinh mã.

Ví dụ với assertEquals, hãy chọn Static Import….. để IntelliJ thêm lệnh import.



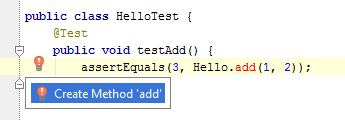
Vì có nhiều thư viện cung cấp hàm assertEquals, ta có vài lựa chọn. Hãy chọn import Assert.assertEquals cho Maven



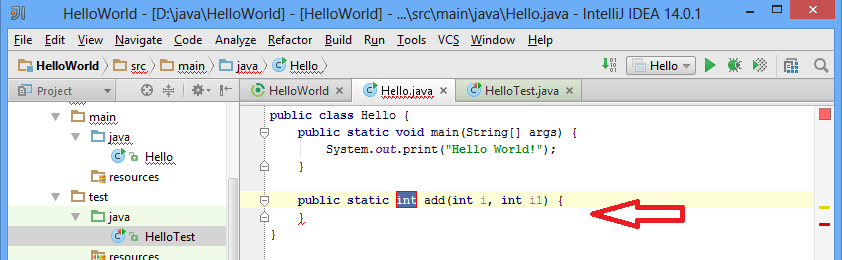
Kết quả là có thêm lệnh import được thêm vào code:

**import static** junit.framework.Assert.assertEquals;

Lỗi còn lại là hàm add chưa có. Hãy đưa con trỏ vào từ add màu đỏ, bấm Alt-Enter. IntelliJ sẽ gợi ý tạo phương thức add.



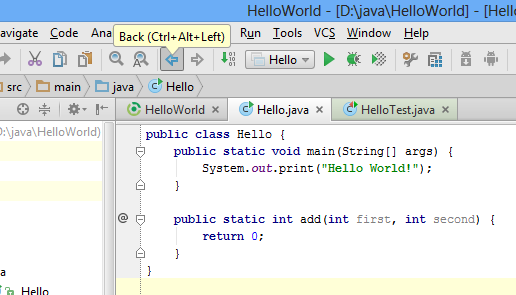
Hãy nhấn tiếp Enter để đồng ý. Bạn sẽ thấy hàm add rỗng được tạo ra tại class Hello. Hãy tiếp tục nhấn Enter theo từng bước để đồng ý với code gợi ý, trước khi Enter có thể gõ nội dung mới thay cho nội dung hiện đang được bôi đen.



Chẳng hạn cuối cùng code của hàm add như sau (ta tạm để nguyên nội dung return 0):

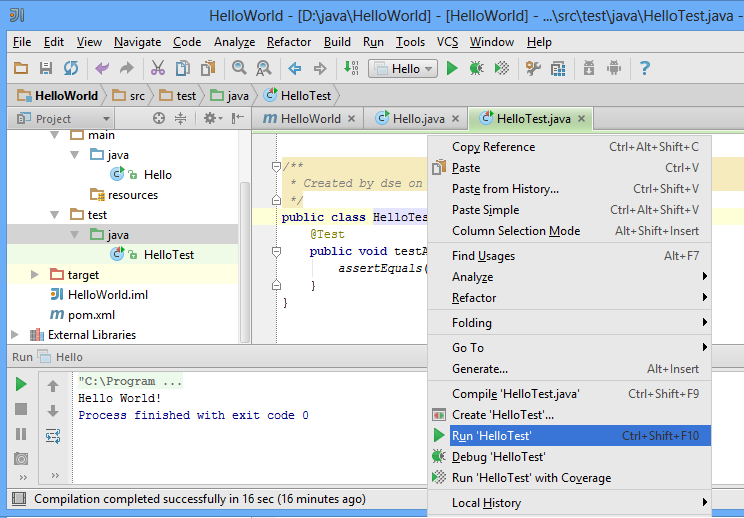
**public static int** add(**int** first, **int** second) {  
 **return** 0;  
}

Để quay về HelloTest, bạn có thể tự chọn tab hoặc click mũi tên Back hoặc dùng tổ hợp phím Ctrl+Alt+Left.

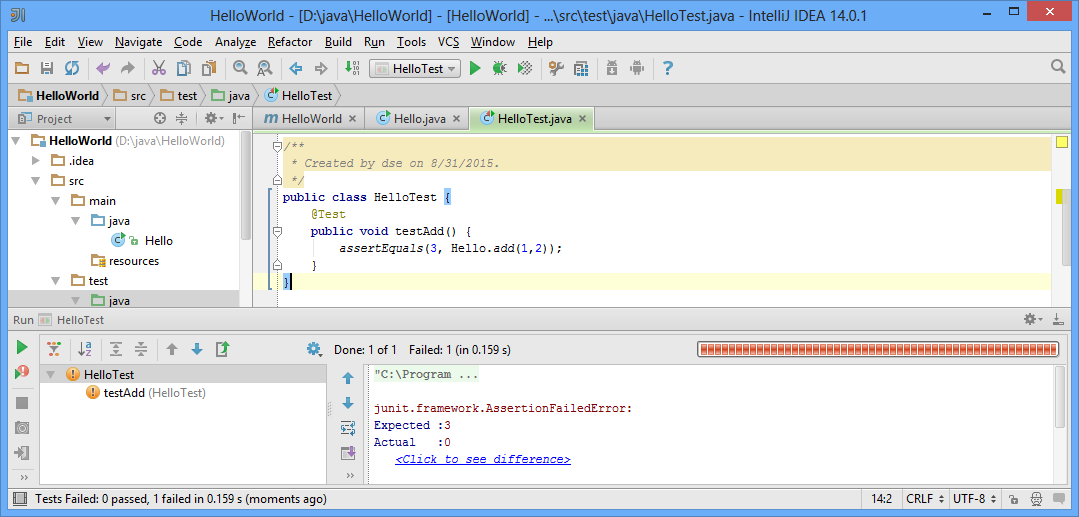


3. Chạy test.

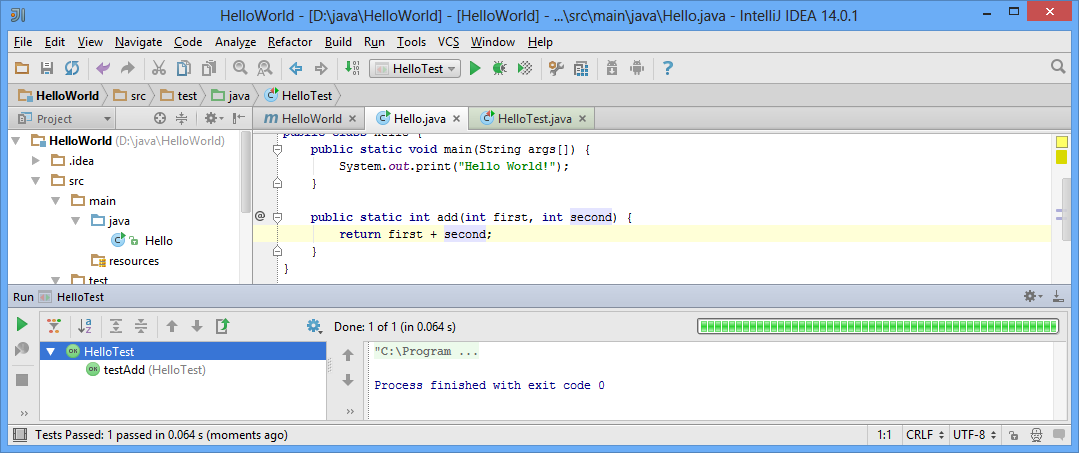
Để chạy test trong HelloTest, click chuột phải vào tên class HelloTest, chọn Run HelloTest.



Kết quả: testAdd không đạt, đáng ra phải trả về 3 thì lại trả về 0



Bạn tự sửa nội dung hàm add cho đúng, rồi chạy lại test để có kết quả đạt như sau:



Đến đây, bạn đã có hai cấu hình chạy: Hello để chạy chương trình Hello (hàm main); HelloTest để chạy các test trong class HelloTest. Chọn cấu hình nào thì nút tam giác bên phải sẽ có tác dụng chạy cấu hình đó.

