# Junit

## Giới thiệu về JUnit

**JUnit là gì?**  
  
JUnit là một mã nguồn mở Java thử nghiệm khung được sử dụng để viết và chạy thử nghiệm lặp lại.Ông là một hệ thống khung xUnit kiểm tra đơn vị cho một cá thể (cho java ngôn ngữ).Nó bao gồm các tính năng sau đây:  
1, để kiểm tra kết quả mong đợi của sự khẳng định (Assertion)  
2, để chia sẻ các công cụ thử nghiệm phổ biến dữ liệu thử nghiệm  
3, để thuận tiện cho tổ chức và hoạt động của các bài kiểm tra bộ kiểm tra  
4, đồ họa và Á hậu kiểm tra văn bản  
  
JUnit được phát triển bởi Erich Gamma (GOF Một) và Kent Beck (xp và một trong những cấu trúc lại đi tiên phong) chuẩn bị.  
  
Cần lưu ý rằng JUnit thường được sử dụng để kiểm tra đơn vị, và do đó cần phải hiểu được cấu trúc bên trong của mã này đang được thử nghiệm (được gọi là màu trắng-box thử nghiệm), kia là xp JUnit lập trình và tái cấu trúc (cấu trúc lại) được khuyến khích mạnh mẽcông cụ, tự động kiểm tra đơn vị vì những trường hợp rất có thể tăng hiệu quả của phát triển, nhưng trong thực tế là viết mã kiểm tra mất rất nhiều thời gian và công sức, sau đó những lợi ích của việc sử dụng công cụ này ở đâu?  
  
Tôi tin rằng là:  
1. Cho chương trình xp, các cuộc gọi cho các bài kiểm tra văn bản trước khi viết mã, có thể buộc bạn phải suy nghĩ đúng đắn trước khi viết mã trong mã (phương pháp) của hàm và logic, hay viết mã này là không ổn định, sau đó bạn cần trong khi duy trìkiểm tra mã và mã thực tế, khối lượng công việc này sẽ làm tăng đáng kể.Do đó, trong lập trình xp, quá trình cơ bản là: ý tưởng - "viết mã kiểm tra -" viết mã - "thử nghiệm, và kiểm tra văn bản và mã văn bản được gia tăng, và viết một bài kiểm tra nhỏ mà mã trong việc chuẩn bị của tương laiNếu vấn đề được tìm thấy nhiều khối có thể được truy nguồn từ vấn đề, giảm bớt khó khăn trong việc điều chỉnh các lỗi lại  
2. Cho tái cấu trúc, lợi ích, và chương trình xp là tương tự, bởi vì tái cấu trúc cũng phải thay đổi một thử nghiệm nhỏ, ít tốn thời gian do trả lại lỗi.  
3. Cho người không ở trên hai trường hợp, khi chúng tôi phát triển việc sử dụng thích hợp của bài kiểm tra viết JUnit là cần thiết, bởi vì các thử nghiệm nói chung chúng ta cũng cần phải viết mã, bạn không thể sử dụng JUnit ban đầu, nếu bạn đang sử dụng JUnit để chuẩn bị thử nghiệm mã sẽ giảm bớt công việc bảo trì trong tương lai, chẳng hạn như thay đổi trong phương pháp sau (điều này tương đương với việc tái thiết).   
  
JUnit khẳng định vì còn có một chức năng, nếu kết quả thử nghiệm không vượt qua bài kiểm tra sẽ cho chúng ta rằng không vượt qua, tại sao, và nếu thực hành nói chung trước đây là viết một số mã kiểm tra là để xem kết quả của họ, và sau đó tự xác định kết quả của việc sử dụng đúngsử dụng lợi thế JUnit là kết quả này là chính xác để đánh giá nó để hoàn thành, chúng ta chỉ cần nhìn vào nó quyền cho chúng tôi biết kết quả có thể, nói chung, rất nhiều sẽ cải thiện hiệu quả.  
  
**Cài đặt JUnit**  
  
Cài đặt rất đơn giản, đầu tiên các địa chỉ sau để tải về một gói zip mới:  
<http://download.sourceforge.net/junit/>  
  
Sau khi tải về, giải nén vào thư mục yêu thích của bạn, giả định là JUNIT\_HOME, sau đó JUNIT\_HOME junit.jar theo gói bổ sung vào hệ thống của bạn CLASSPATH biến môi trường, cho môi trường IDE, nhu cầu sử dụng tăng lên của dự án JUnitlib, IDE khác nhau của các thiết lập có các cài đặt khác nhau, mà không nói gì nhiều.  
 **Làm thế nào để sử dụng JUnit để viết các bài kiểm tra?**  
  
**Kiểm tra với một test case**  
Giả sử chúng ta có một class Number như dưới đây. Mỗi object của class đại diện cho một số nguyên. Method isEven kiểm tra tính chẵn lẻ của số nguyên đó. Mục tiêu của chúng ta là kiểm tra tính đúng đắn của method isEvent.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public class Number {      private int number;      public Number(int number) { this.number = number; }      public boolean isEven() { return (number % 2 == 0)?true:false; }  } |

Việc đầu tiên là tải tập tin [junit-4.x.x.jar](http://www.junit.org/) và cho nó vào CLASSPATH. Đoạn mã dưới đây sẽ kiểm tra isEvent với 2.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import static org.junit.Assert.\*;  import org.junit.Test;  public class NumberTest {      @Test public void testIsEven() {          Number number = new Number(2);          assertTrue(number.isEven());      }  } |

Dịch hai đoạn mã trên và chạy lệnh java org.junit.runner.JUnitCore NumberTest. Khi đó JUnit sẽ chạy các method được đánh dấu @Test. Kết quả là:

Mã:

JUnit version 4.8.2

.

Time: 0.008

OK (1 test)

**Kiểm tra với nhiều test case**  
Bây giờ ta muốn kiếm tra với nhiều số hơn. Chúng ta sẽ không chèn thêm assertTrue vào testIsEven (vì như vậy sẽ không thể biết được số nào làm cho test bị sai). Thay vào đó, chúng ta sẽ tham số hóa method testIsEven. Các tham số được lấy từ bộ test tạo ra bởi method getTestParameters.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | import static org.junit.Assert.\*;  import org.junit.Test;  import org.junit.runner.RunWith;  import org.junit.runners.Parameterized;  import org.junit.runners.Parameterized.Parameters;  import java.util.Arrays;  import java.util.Collection;    @RunWith(value=Parameterized.class)  public class NumberParameterizedTest {      private int value;      private boolean expected;      public NumberParameterizedTest(int value, boolean expected) {          this.value = value;          this.expected = expected;      }      @Parameters public static Collection&lt;Object[]&gt; getTestParameters() {          return Arrays.asList(new Object[][] { {2, true}, {3, false}, {5, true}, });      }      @Test public void testIsEven() {          Number number = new Number(value);          assertEquals(number.isEven(), expected);      }  } |

Lần này, Junit sẽ lần lượt lấy từng mẫu ra để kiểm tra. Kết quả là (trích):

Mã:

JUnit version 4.8.2

...E

Time: 0.022

There was 1 failure:

1) testIsEven[2](NumberParameterizedTest)...

Như vậy là test sai với mẫu thứ ba (5, true) (vì đếm từ 0).  
  
Thời gian cho biết, kiểm tra về số lượng các điểm nhỏ, và nếu thử nghiệm cho thấy OK.Nếu không, những điểm nhỏ phía sau F đánh dấu, cho thấy rằng các thử nghiệm không thành công.  
  
Mỗi kết quả kiểm tra phải được OK, vì vậy để giải thích các thử nghiệm thành công, nếu không thành công các thông tin sẽ ngay lập tức theo các hướng dẫn sửa đổi.  
  
  
**Kiểm tra Exception**  
Trong một số trường hợp, một method sẽ throw Exception thay vì trả về kết quả tính toán. Để đảm bảo hành vi của method đúng như mong đợi, chúng ta phải test cả những trường hợp này để xem chúng có throw đúng Exception cần throw không. @Test(expected=YourException.class) là annotation phục vụ cho mục đích này.  
  
  
**Dùng test làm hướng dẫn sử dụng**  
Hãy nhìn lại đoạn code trên. Khi đọc test (nếu bạn hiểu JUnit), bạn sẽ suy ra ngay cách sử dụng method isEven: đầu tiên tạo một object Number với constructor nhận vào giá trị int, sau đó gọi isEven(). Vậy là, test code còn một tác dụng khác là làm hướng dẫn sử dụng (tương tự như các đoạn code trong tutorial trên mạng). Ưu điểm của việc lấy test làm hướng dẫn sử dụng là nhanh gọn và bám sát theo thay đổi của chương trình.  
  
  
**Lưu ý**  
Dưới đây là một số kỹ năng cụ thể bằng văn bản hoặc mã kiểm tra thực hành tốt:  
1. Không sử dụng các nhà xây dựng TestCase khởi Lịch thi đấu, nhưng thay vì sử dụng các thiết lập () và Teardown (phương pháp).  
2. Không dựa vào hoặc giả định rằng để chạy kiểm tra, bởi vì các phương pháp thử nghiệm bằng cách sử dụng JUnit Vector lưu.Vì vậy, các nền tảng khác nhau sẽ được theo một thứ tự khác nhau, loại bỏ từ các phương pháp thử Vector.  
3. Để tránh các tác dụng phụ của việc chuẩn bị một TestCase.Ví dụ: Nếu các xét nghiệm tiếp theo phụ thuộc vào các dữ liệu giao dịch cụ thể, không gửi dữ liệu giao dịch.Sẽ chỉ đơn giản là cuộn trên đó.  
4. Khi kế thừa một lớp thử nghiệm, hãy nhớ gọi thiết lập superclass () và Teardown (phương pháp).  
5. Sẽ kiểm tra mã và mã làm việc cùng nhau, trong khi đồng thời biên soạn và cập nhật.(Sử dụng Ant JUnit có để hỗ trợ nhiệm vụ.)  
6. Lớp học thử nghiệm và phương pháp thử nghiệm cần phải có một đề án đặt tên phù hợp.Nếu tên lớp làm việc với các thử nghiệm để tạo thành tên lớp thử nghiệm.  
7. Để đảm bảo rằng các thử nghiệm độc lập về thời gian, không dựa vào dữ liệu đã lỗi thời được sử dụng để thử nghiệm.Dẫn đến việc duy trì tiếp theo của quá trình là khó khăn để tái sản xuất thử nghiệm.  
8. Nếu bạn đang viết phần mềm cho thị trường quốc tế, việc chuẩn bị thử nghiệm để xem xét khi các yếu tố quốc tế.Ngôn ngữ bản địa không chỉ để thử nghiệm.  
9. Càng nhiều càng tốt để cung cấp việc sử dụng JUnit để khẳng định / không phương pháp và phương thức xử lý ngoại lệ có thể làm cho mã ngắn gọn hơn.  
10 Các thử nghiệm. Để được như nhỏ nhất có thể, việc thực hiện tốc độ.

## JUnit là gì?

Testing là vấn đề sống còn của software  
  
Khi phát triển một sản phẩm phần mềm, testing là một công việc quan trọng. Nó đảm bảo rằng phần mềm khi ra đời có hoạt động như ban đầu đã định không? Test được thực hiện trong suốt quá trình, trong tất cả các workflow. Mỗi workflow sẽ có những hình thức test và công cụ test thích hợp.  
  
JUnit là một framework đơn giản dùng cho việc tạo các unit testing tự động, và chạy các test có thể lặp đi lặp lại. Nó chỉ là một phần của họ kiến trúc xUnit cho việc tạo các unit testing. JUnit là một chuẩn trên thực tế cho unit testing trong Java. JUnit về nguồn gốc được viết bởi 2 tác giả Erich Gamma và Kent Beck 1.  
  
**Lợi ích của JUnit**  
  
JUnit tránh cho người lập trình phải làm đi làm lại những việc kiểm thử nhàm chán bằng cách tách biệt mã kiểm thử ra khỏi mã chương trình, đồng thời tự động hóa việc tổ chức và thi hành các bộ số liệu kiểm thử.   
  
Thoạt tiên, khi sử dụng JUnit, ta có thể có cảm giác là JUnit chỉ làm mất thêm thời gian cho việc kiểm thử: Thay vì phải viết thêm các lớp và phương thức mới phục vụ cho công tác kiểm thử, ta có thể soạn nhanh một bộ số liệu rồi viết ngay vào trong phương thức main() và quan sát ngay kết quả kiểm thử. Vì quá trình soạn số liệu và quá trình kiểm thử diễn ra đồng thời, nên ta sẽ dễ dàng nhận biết được ngay chương trình đã chạy đúng trên bộ số liệu kiểm thử hay không, mà không cần nhìn vào tín hiệu "xanh" mà JUnit có thể hỗ trợ.

## Tìm hiểu về Testing Junit

**I. Giới thiệu JUnit**  
  
JUnit là một framework đơn giản dùng cho việc tạo các unit testing tự động, và chạy các test có thể lặp đi lặp lại. Nó chỉ là một phần của họ kiến trúc xUnit cho việc tạo các unit testing. JUnit là một chuẩn trên thực tế cho unit testing trong Java. JUnit về nguồn gốc được viết bởi 2 tác giả Erich Gamma và Kent Beck 1  
  
**Giới thiệu:**  
  
JUnit có những đặc điểm đáng lưu tâm như sau:  
• Xác nhận (assert) việc kiểm tra kết quả được mong đợi  
• Các Test Suite cho phép chúng ta dễ dàng tổ chức và chạy các test  
• Hỗ trợ giao diện đồ họa và giao diện dòng lệnh  
  
Các test case của JUnit là các lớp của Java, các lớp này bao gồm một hay nhiều các phương thức unit testing, và những test này lại được nhóm thành các Test Suite.  
  
Mỗi phương thức test trong JUnit phải được thực thi nhanh chóng. Tốc độ là điều tối quan trọng vì càng nhiều test được viết và tích hợp vào bên trong quá trình xây dựng phần mềm, cần phải tốn nhiều thời gian hơn cho việc chạy toàn bộ Test Suite. Các lập trình viên không muốn bị ngắt quãng trong một khoãng thời gian dài trong khi các test chạy, vì thế các test mà chạy càng lâu thì sẽ có nhiều khả năng là các lập trình viên sẽ bỏ qua bước cũng không kém phần quan trọng này.  
  
Các test trong JUnit có thể là các test được chấp nhận hay thất bại, các test này được thiết kế để khi chạy mà không cần có sự can thiệp của con người. Từ những thiết kế như thế, bạn có thể thêm các bộ test vào quá trình tích hợp và xây dựng phần mềm một cách liên tục và để cho các test chạy một cách tự động  
  
**1. Đặt vấn đề**  
  
Mọi chương trình đều phải đi kèm với bộ số liệu kiểm thử. Tuy nhiên, hoạt động này chưa thực sự phổ biến trong chương trình giảng dạy lập trình và công nghệ phần mềm tại một số trường đại học ở nước ta. Người lập trình thường xem nhẹ việc kiểm thử, đơn giản vì đó là một công việc nhàm chán, ít gây hứng thú. Nhưng kiểm thử là một hoạt động quan trọng và không thể thiếu được nhằm phát hiện lỗi trong chương trình, từ đó nâng cao năng suất và đảm bảo chất lượng sản phẩm phần mềm. Beck và Gamma là những người đầu tiên phát triển công cụ mã nguồn mở JUnit để hỗ trợ việc kiểm thử.   
  
Bài viết này sẽ trình bày lại một ví dụ minh họa việc áp dụng JUnit bằng việc đưa ra một thiết kế đơn giản và hợp lý để giải quyết bài toán đặt ra.   
  
**2. Lợi ích của JUnit**  
  
JUnit tránh cho người lập trình phải làm đi làm lại những việc kiểm thử nhàm chán bằng cách tách biệt mã kiểm thử ra khỏi mã chương trình, đồng thời tự động hóa việc tổ chức và thi hành các bộ số liệu kiểm thử.   
  
Thoạt tiên, khi sử dụng JUnit, ta có thể có cảm giác là JUnit chỉ làm mất thêm thời gian cho việc kiểm thử: Thay vì phải viết thêm các lớp và phương thức mới phục vụ cho công tác kiểm thử, ta có thể soạn nhanh một bộ số liệu rồi viết ngay vào trong phương thức main() và quan sát ngay kết quả kiểm thử. Vì quá trình soạn số liệu và quá trình kiểm thử diễn ra đồng thời, nên ta sẽ dễ dàng nhận biết được ngay chương trình đã chạy đúng trên bộ số liệu kiểm thử hay không, mà không cần nhìn vào tín hiệu "xanh" mà JUnit có thể hỗ trợ.  
  
**3. Ví dụ minh họa**  
  
Sau đây là một ví dụ minh họa với những yêu cầu mới dần dần được thêm vào: Hãy thiết kế lớp tiền tệ. Tiền tệ được đặc trưng bằng số tiền và đơn vị tiền (chẳng hạn VND hoặc USD). Trước yêu cầu này, ta dễ dàng viết ra lớp Money:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | public class Money {      private double amount;      private String currency;        public Money(double amount, String currency) {          this.amount = amount;          this.currency = currency;      }  } |

Bây giờ giả sử rằng ta chỉ cần xử lý một loại tiền tệ duy nhất, chẳng hạn tiền Việt Nam. Hãy hiện thực phương thức dưới đây để cộng hai số tiền cùng loại với nhau, và dùng JUnit để kiểm thử chương trình.

|  |  |
| --- | --- |
| 1  2 | Money add(Money money) |

Do đó sẽ phải hiệu chỉnh class Money để giải quyết vấn đề đặt ra:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | import junit.framework.Assert;  import junit.framework.TestCase;    public class MoneyTest extends TestCase {      public void testAdd() {      Money m1 = new Money(200, "VND");      Money m2 = new Money(1000, "VND");      Money result = m1.add(m2); // đối tượng lưu kết quả tính toán      Money expected = new Money(1200, "VND"); // kết quả dự kiến      Assert.assertTrue(result.equals(expected)); // lệnh kiểm thử  } |

## Hướng dẫn sử dụng JUnit 4 - Phần 1

Hướng dẫn này giới thiệu các annotation cơ bản hỗ trợ trong Junit 4.  
  
**JunitTest1.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | import org.junit.\*;  import static org.junit.Assert.\*;  import java.util.\*;    public class JunitTest1 {        private Collection collection;        @BeforeClass      public static void oneTimeSetUp() {          // one-time initialization code          System.out.println("@BeforeClass - oneTimeSetUp");      }        @AfterClass      public static void oneTimeTearDown() {          // one-time cleanup code          System.out.println("@AfterClass - oneTimeTearDown");      }        @Before      public void setUp() {          collection = new ArrayList();          System.out.println("@Before - setUp");      }        @After      public void tearDown() {          collection.clear();          System.out.println("@After - tearDown");      }        @Test      public void testEmptyCollection() {          assertTrue(collection.isEmpty());          System.out.println("@Test - testEmptyCollection");      }        @Test      public void testOneItemCollection() {          collection.add("itemA");          assertEquals(1, collection.size());          System.out.println("@Test - testOneItemCollection");      }  } |

**Kết quả**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @BeforeClass - oneTimeSetUp  @Before - setUp  @Test - testEmptyCollection  @After - tearDown  @Before - setUp  @Test - testOneItemCollection  @After - tearDown  @AfterClass - oneTimeTearDown |

## Hướng dẫn sử dụng JUnit 4 - Phần 2

Expected Exception Test: Được sử dụng để kiểm tra ném ngoại lệ của phương pháp này.  
  
**JunitTest2.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import org.junit.\*;    /\*\*   \* JUnit Expected Exception Test   \*   \*/  public class JunitTest2 {        @Test(expected = ArithmeticException.class)      public void divisionWithException() {        int i = 1/0;      }    } |

Trong ví dụ trên, divisionWithException () sẽ ném một ngoại lệ ArithmeticException, vì đây là một ngoại lệ dự kiến​​, vì vậy các unit test sẽ pass.

## Hướng dẫn sử dụng JUnit 4 - Phần 3

Trong phần này chúng ta sẽ tìm hiểu về Ignore Test. "Ignore" có nghĩa là phương thức này là chưa sẵn sàng để thử nghiệm, JUnit sẽ bỏ qua phương thức này.  
  
**JunitTest3.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import org.junit.\*;    /\*\*   \* JUnit Ignore Test   \*   \*/  public class JunitTest3 {        @Ignore("Not Ready to Run")      @Test      public void divisionWithException() {        System.out.println("Method is not ready yet");      }    } |

## Hướng dẫn sử dụng JUnit 4 - Phần 4

Trong phần này chúng ta sẽ tìm hiểu về Time Test. "Time Test" có nghĩa là nếu một đơn vị kiểm tra mất nhiều thời gian hơn so với số quy định của mili giây để chạy, thử nghiệm sẽ kết thúc và đánh dấu thất bại.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import org.junit.\*;    /\*\*   \* JUnit TimeOut Test   \*   \*/  public class JunitTest4 {        @Test(timeout = 1000)      public void infinity() {          while (true);      }    } |

Trong ví dụ trên, ở phương thức infinity() phương pháp sẽ không return, vì vậy JUnit sẽ đánh dấu nó như là thất bại và ném một ngoại lệ.

|  |  |
| --- | --- |
| 1  2 | java.lang.Exception: test timed out after 1000 milliseconds |

## Hướng dẫn sử dụng JUnit 4 - Phần 5

Trong phần này chúng ta sẽ tìm hiểu Suite Test. "Suite Test" có nghĩa là bó một vài trường hợp đơn vị kiểm tra và chạy nó với nhau. Trong Junit, cả @RunWith và @Suite annotation được sử dụng để chạy thử nghiệm bộ.   
  
Ví dụ dưới đây có nghĩa là cả hai JunitTest1 và JunitTest2 sẽ chạy cùng nhau sau khi JunitTest5 được thực hiện.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import org.junit.runner.RunWith;  import org.junit.runners.Suite;    /\*\*   \* JUnit Suite Test   \*   \*/    @RunWith(Suite.class)  @Suite.SuiteClasses({          JunitTest1.class,          JunitTest2.class  })  public class JunitTest5 {  } |

**Kết quả**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @BeforeClass - oneTimeSetUp  @Before - setUp  @Test - testEmptyCollection  @After - tearDown  @Before - setUp  @Test - testOneItemCollection  @After - tearDown  @AfterClass - oneTimeTearDown |

## Hướng dẫn sử dụng JUnit 4 - Phần 6

Trong phần này chúng ta sẽ tìm hiểu Parameterized Test. Trong JUnit, bạn có thể sử dụng cả hai @RunWith and @Parameter annotation để truyền tham số vào một Unit Test.  
  
**1. MathUtils - Parameterized Test**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public class MathUtils {        public static int add(int a, int b) {          return a + b;      }  } |

Dưới đây là một thử nghiệm JUnit cho lớp trên.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | import static org.junit.Assert.assertEquals;  import java.util.Arrays;  import org.junit.Test;  import org.junit.runner.RunWith;  import org.junit.runners.Parameterized;  import org.junit.runners.Parameterized.Parameters;    @RunWith(value = Parameterized.class)  public class MathUtilsTest {        private int numberA;      private int numberB;      private int expected;        //parameters pass via this constructor      public MathUtilsTest(int numberA, int numberB, int expected) {          this.numberA = numberA;          this.numberB = numberB;          this.expected = expected;      }        //Declares parameters here      @Parameters(name = "{index}: add({0}+{1})={2}")      public static Iterable<Object[]> data1() {          return Arrays.asList(new Object[][] {              { 1, 1, 2 },              { 2, 2, 4 },              { 8, 2, 10 },              { 4, 5, 9 }          });      }        @Test      public void test\_add() {          assertEquals(expected,MathUtils.add(numberA, numberB));      }    } |

**2. DomainUtils - Parameterized Test**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import java.util.regex.Pattern;    public class DomainUtils {        private static Pattern pDomainName;        private static final String DOMAIN\_NAME\_PATTERN = "^((?!-)[A-Za-z0-9-]{1,63}(?<!-)\\.)+[A-Za-z]{2,6}$";        static {          pDomainName = Pattern.compile(DOMAIN\_NAME\_PATTERN);      }        //is this a valid domain name?      public static boolean isValidDomainName(String domainName) {          return pDomainName.matcher(domainName).find();      }    } |

Dưới đây là một thử nghiệm JUnit cho lớp trên.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | import static org.junit.Assert.assertEquals;  import java.util.Arrays;  import org.junit.Test;  import org.junit.runner.RunWith;  import org.junit.runners.Parameterized;  import org.junit.runners.Parameterized.Parameters;    @RunWith(value = Parameterized.class)  public class DomainUtilsTest {        private String domain;      private boolean expected;        public DomainUtilsTest(String domain, boolean expected) {          this.domain = domain;          this.expected = expected;      }        @Parameters(name= "{index}: isValid({0})={1}")       public static Iterable<Object[]> data() {           return Arrays.asList(new Object[][] {           { "google.com", true },          { "laptrinh.vn", true },                  { "-laptrinh.vn", false },                  { "laptrinh-.vn", false },                  { "3423kjk", false },                  { "mk#$kdo.com", false }              }       );       }        @Test      public void test\_validDomains() {          assertEquals(expected,DomainUtils.isValidDomainNam  e(domain));      }    } |