**Nội dung**[[Ẩn](https://gpcoder.com/5234-kiem-thu-voi-junit-trong-java/)]

* [1 JUnit là gì?](https://gpcoder.com/5234-kiem-thu-voi-junit-trong-java/#JUnit_la_gi)
* [2 Các tính năng của JUnit](https://gpcoder.com/5234-kiem-thu-voi-junit-trong-java/#Cac_tinh_nang_cua_JUnit)
* [3 Một số khái niệm cần biết trong Unit Test](https://gpcoder.com/5234-kiem-thu-voi-junit-trong-java/#Mot_so_khai_niem_can_biet_trong_Unit_Test)
* [4 Cài đặt JUnit](https://gpcoder.com/5234-kiem-thu-voi-junit-trong-java/#Cai_dat_JUnit)
* [5 Ví dụ sử dụng JUnit trên Eclipse](https://gpcoder.com/5234-kiem-thu-voi-junit-trong-java/#Vi_du_su_dung_JUnit_tren_Eclipse)
* [6 Kiểm tra độ bao phủ của Unit Test với plugin EclEmma](https://gpcoder.com/5234-kiem-thu-voi-junit-trong-java/#Kiem_tra_do_bao_phu_cua_Unit_Test_voi_plugin_EclEmma)
* [7 Một số bài hướng dẫn Unit Test trong Java](https://gpcoder.com/5234-kiem-thu-voi-junit-trong-java/#Mot_so_bai_huong_dan_Unit_Test_trong_Java)

**JUnit là gì?**

Trong Java, để thực hiện viết code cho Unit Test chúng ta có thể sử dụng một trong hai Framework: [JUnit](https://junit.org/) và [TestNG](https://testng.org/).

**JUnit** là một framework mã nguồn mở, miễn phí, đơn giản dùng để unit test cho ngôn ngữ lập trình Java. Trong Java, chúng ta thường sẽ sử dụng method để làm unit test.

Hiện tại, hầu hết trong các dự án chúng ta vẫn đang sử dụng [JUnit 4](https://junit.org/junit4/), nhưng [JUnit 5](https://junit.org/junit5) đã được phát hành và có lẽ nó sẽ trở thành xu hướng trong thời gian sắp tới. Chúng ta có thể sử dụng JUnit để viết code test cho cả unit testing và integration testing.

**Các tính năng của JUnit**

* JUnit là một framework mã nguồn mở, được sử dụng để viết và chạy kiểm thử.
* Cung cấp các annotation để định nghĩa các phương thức kiểm thử.
* Cung cấp các Assertion để kiểm tra kết quả mong đợi.
* Cung cấp các test runner để thực thi các test script.
* Test case JUnit có thể được chạy tự động.
* Test case JUnit có thể được tổ chức thành các test suite.
* JUnit cho thấy kết quả test một cách trực quan: pass (không có lỗi) là màu xanh và fail (có lỗi) là màu đỏ.
* ….

**Một số khái niệm cần biết trong Unit Test**

* **Unit Test case**: là 1 chuỗi code để đảm bảo rằng đoạn code được kiểm thử làm việc như mong đợi. Mỗi function sẽ có nhiều test case, ứng với mỗi trường hợp function chạy.
* **Setup**: Đây là hàm được chạy trước khi chạy các test case, thường dùng để chuẩn bị dữ liệu để chạy test.
* **Teardown**: Đây là hàm được chạy sau khi các test case chạy xong, thường dùng để xóa dữ liệu, giải phóng bộ nhớ.
* **Assert**: Mỗi test case sẽ có một hoặc nhiều câu lệnh Assert, để kiểm tra tính đúng đắn của hàm.
* **Mock**: là một đối tượng ảo, mô phỏng các tính chất và hành vi giống hệt như đối tượng thực được truyền vào bên trong khối mã đang vận hành nhằm kiểm tra tính đúng đắn của các hoạt động bên trong. Giả sử chương trình của chúng ta được chia làm 2 module: A và B. Module A đã code xong, B thì chưa. Để test module A, ta dùng mock để làm giả module B, không cần phải đợi tới khi module B code xong mới test được.
* **Test Suite** : Test suite là một tập các test case và nó cũng có thể bao gồm nhiều test suite khác, test suite chính là tổ hợp các test.

**Cài đặt JUnit**

Ngày nay, JUnit được tích hợp sẵn trong hầu hết các Java IDE (Eclipse, NetBeans và IntelliJ). Nếu không có sẵn, các bạn có thể tạo một [project Maven](https://gpcoder.com/2916-huong-dan-su-dung-apache-maven-voi-eclipse/) và thêm thư viện JUnit vào file **pom.xml** như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <!-- https://mvnrepository.com/artifact/junit/junit -->  <dependency>      <groupId>junit</groupId>      <artifactId>junit</artifactId>      <version>4.12</version>      <scope>test</scope>  </dependency> |

Trong bài viết này, chúng ta sẽ sử dụng phiên bản [JUnit 4](https://junit.org/junit4/). Đối với [JUnit 5](https://junit.org/junit5), chúng ta sẽ cùng tìm hiểu ở một bài viết khác.

**Ví dụ sử dụng JUnit trên Eclipse**

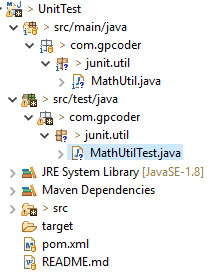
Giả sử chúng ta có một class util có 2 phương thức devide() và add().

* Phương thức divide() : thực hiện chia phần nguyên của 2 số. Phương thức này nhận 2 đối số: số bị chia (dividend) và số chia (divisor). Nếu số chia là 0 thì chương trình sẽ throw một ngoại lệ, ngược lại chương trình sẽ trả về kết quả sau khi thực hiện chia nguyên.
* Phương thức add() : sẽ thực hiện tính tổng của 2 số nguyên.

Chương trình của chúng ta như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | package com.gpcoder.junit.util;    public class MathUtil {        private MathUtil() {          throw new UnsupportedOperationException("Cannot call constructor directly!");      }        public static int divide(int dividend, int divisor) {          if (divisor == 0) {              throw new IllegalArgumentException("Cannot divide by zero (0).");          }          return dividend / divisor;      }        public static int add(int number1, int number2) {          return number1 - number2;      }  } |

Bây giờ chúng ta sẽ sử dụng JUnit để kiểm tra phương thức trên với các đầu vào khác nhau.

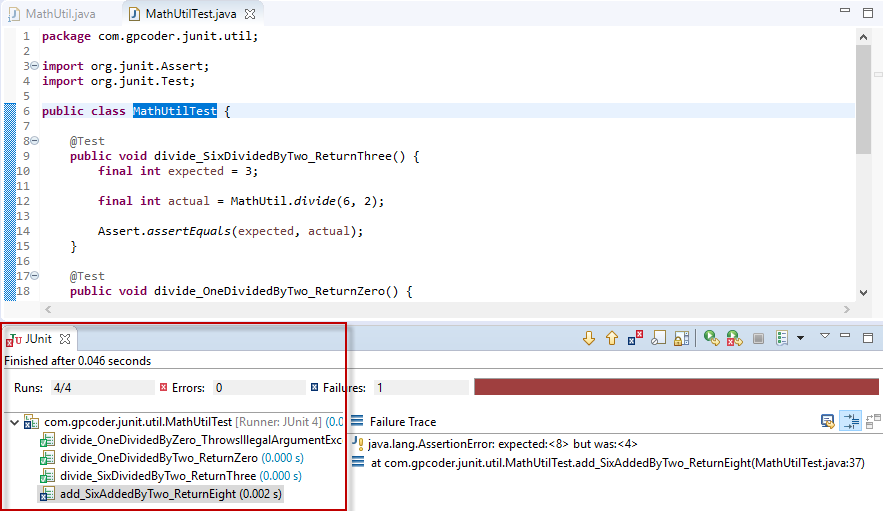
[](https://gpcoder.com/wp-content/uploads/2019/02/junit-hello-world-src-structure.png)

* Đầu tiên chúng ta sẽ tạo một class mới với suffix là xxx**Test**. Đây là một naming convention cho Unit Test. Class này nên được đặt trong thư mục test, cùng package name với tên của class cần viêt unit test để dễ dàng quản lý.
* Tiếp theo, chúng ta sẽ tạo test case để test các trường hợp có thể có của một phương thức. Mỗi test case nên tạo một phương thức để kiểm tra:
  + Phương thức devide() có thể có một số trường hợp test sau: trường hợp kết quả phép chia ra một số nguyên, trường hợp kết quả phép chia ra số lẻ, trường hợp số chia là số 0.
  + Phương thức add() : chỉ đơn giản kiểm tra kết quả cộng 2 số.

|  |  |
| --- | --- |
| 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 | package com.gpcoder.junit.util;    import org.junit.Assert;  import org.junit.Test;    public class MathUtilTest {        @Test      public void divide\_SixDividedByTwo\_ReturnThree() {          final int expected = 3;            final int actual = MathUtil.divide(6, 2);            Assert.assertEquals(expected, actual);      }        @Test      public void divide\_OneDividedByTwo\_ReturnZero() {          final int expected = 0;            final int actual = MathUtil.divide(1, 2);            Assert.assertEquals(expected, actual);      }        @Test(expected = IllegalArgumentException.class)      public void divide\_OneDividedByZero\_ThrowsIllegalArgumentException() {          MathUtil.divide(1, 0);      }        @Test      public void add\_SixAddedByTwo\_ReturnEight() {          final int expected = 8;            final int actual = MathUtil.add(6, 2);            Assert.assertEquals(expected, actual);      }  } |

Để chạy kiểm tra các test case trên, chúng ta sẽ chọn chuột phải trên class tương ứng cần test, sau đó chọn **Run As** –> **Unit Test**. Tương tự, chúng ta cũng có thể thực thi test cho một phương thức hoặc cả project.

Chúng ta có kết quả như sau:

[](https://gpcoder.com/wp-content/uploads/2019/02/junit-hello-world.png)

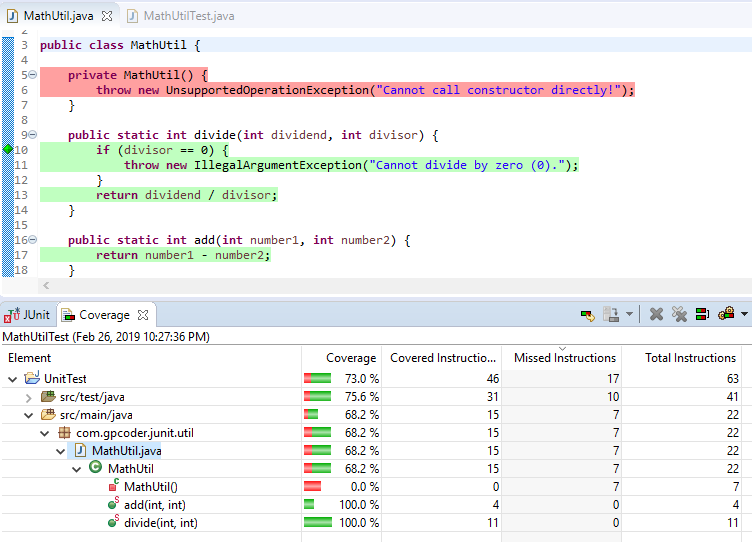
Trên kết quả test, chúng ta có thể thấy được tổng thời gian thực hiện tất cả các test case (0.055 seconds), thời gian thực thi mỗi test case, kết quả các test case tương ứng.

Như trong ví dụ trên, chúng ta có 3 phương thức pass (có màu xanh) cho phương thức divide(). Điều này có nghĩa là code logic của phương thức devide() đã đúng như mong đợi.

Phương thức add\_SixAddedByTwo\_ReturnEight() có màu đỏ, điều này có nghĩa là logic của phương thức add() đã có gì đó không đúng như mong đợi. Khi chúng ta click chuột vào test case bị lỗi, IDE sẽ hiển thị một vài thông tin chi tiết tại sao kết quả lại fail. Như hình bên trái, chúng ta có thể thấy là phương thức add() đang mong muốn là 8 nhưng kết quả là 4. Từ đó, chúng ta có thể kiểm tra lại code của chương trình để tìm nguyên nhân xảy ra kết quả không mong đợi.

**Kiểm tra độ bao phủ của Unit Test với plugin EclEmma**

Trong bài viết [Hướng dẫn sử dụng plugin EclEmma trong Eclipse](https://gpcoder.com/1943-huong-dan-su-dung-plugin-eclemma-trong-eclipse/), tôi đã giới thiệu với các bạn plugin EclEmma (Coverage – Jacoco), một công cụ để kiểm tra độ bao phủ của code Unit Testcase trong chương trình. EclEmma có thể đánh dấu những đoạn code nào mà testcase chưa đáp ứng được, tính tỷ lệ phần trăm độ bao phủ của testcase trên từng file, package, project. Từ đó chúng ta có thể đánh giá được chất lượng của Unit Testcase, cũng như dễ dàng bổ sung testcase cho những đoạn code còn thiếu.

[](https://gpcoder.com/wp-content/uploads/2019/02/junit-hello-world-coverage-jacoco.png)

EclEmma sử dụng màu để đánh dấu kết quả bao phủ của code Unit Test. Từ đó có thể bổ sung một số trường hợp còn thiếu nếu cần thiết.

* Màu đỏ: code Unit Test chưa bao phủ (chưa kiểm tra) được dòng code này.
* Màu vàng: chưa bao phủ hết trường hợp.
* Màu xanh: đã được kiểm tra.

0360 6787 801

**Lời kết:**

Trên đây là một số thông tin cơ bản về JUnit trong Java. Trong các bài viết tiếp theo, chúng ta sẽ cùng tìm hiểu sâu hơn về cách thực hiện unit test với JUnit cho các trường hợp phức tạp và giải thích các annotation của JUnit đã sử dụng trong bài viết này.

## JUnit Hello World in NetBeans IDE

This article is intended for Java developers with basic knowledge of core Java and Object Oriented Principles. You need not be familiar with JUnit framework or NetBeans IDE to follow this article.

This article provides an introduction to the test-first, or test-driven, development (TDD) philosophy which recommends that unit testing and coding should go hand-in-hand to ensure the stability of your code.

This is an actionable article which guides a user to write a very simple unit test while developing a simple application. It would hardly take 10 minutes for a beginner level Java developer to try out the code in this article.

Generally, it takes a couple of years for a Java developer working on professional projects to appreciate the benefits of writing unit tests for his/her code. Even the NetBeans documentation on creating Unit test requires familiarity with Java Collections classes. This article demonstrates the benefit of Unit testing with very less effort and would immediately encourage developers in getting in to the habit of writing unit tests simultaneously with production code.

## What Is JUnit?

JUnit is a simple, free and open source framework to write repeatable unit tests using Java. Erick Gamma and Kert Beck originally wrote it. It is a regression-testing framework, which allows the developer to write code faster with high quality. JUnit tests increase the stability of the software.

The main philosophy behind this testing framework is to make coding and testing move hand in hand.

## NetBeans IDE Overview

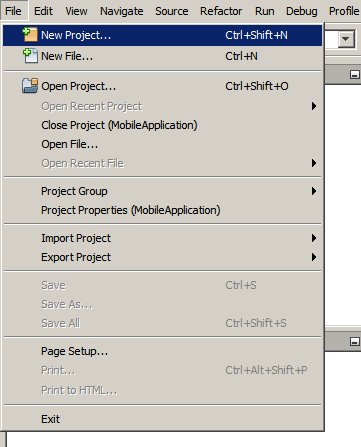
In this article, we will apply this philosophy of testing and coding side by side to an object-oriented version of Hello World program by creating a test case for it in [NetBeans IDE](https://netbeans.org/). NetBeans IDE is the official IDE for Java 8 for development of desktop, mobile, and web applications, as well as HTML5 (including Oracle JET) applications with HTML, JavaScript, and CSS.

Let us apply this philosophy of testing and coding side by side to the object-oriented version of Hello World program by creating a test case for it in NetBeans IDE v8.2. It available for free download from [this URL](https://netbeans.org/downloads/).

## Writing Java and Test Classes Using NetBeans IDE

Please follow the steps below:

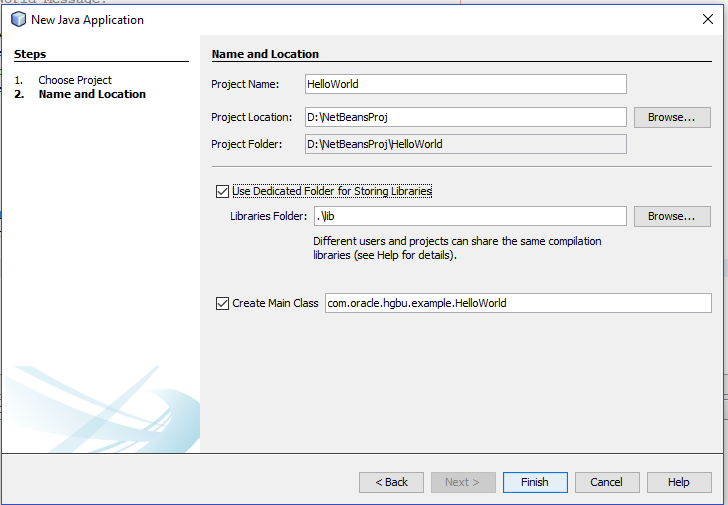
1. Launch NetBeans IDE. In Windows 10, it can be started using > NetBeans > NetBeans 8.2 IDE.
2. In the IDE, choose File > New Project, as shown in the figure below.



3. In the New Project wizard, expand the Java category and select Java Application as shown in the figure below. Then click Next.

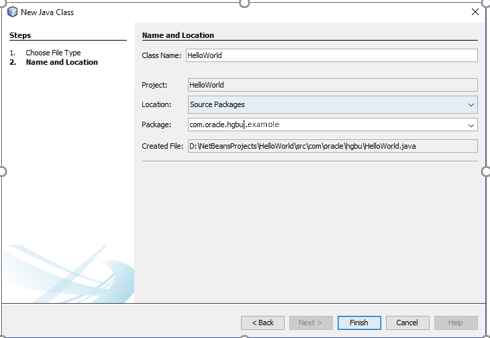
4. In the Name and Location page of the wizard, do the following (as shown in the figure below):

* In the Project Name field, type HelloWorld.
* Check the Use Dedicated Folder for Storing Libraries checkbox.
* In the Create Main Class field, type com.dzone.example.HelloWorld



5. Click Finish to create the project.

6. Now go to context menu of the project and Select **New > Java Class…**to create a new Java class.



6. Type in the following code for the class.

package com.oracle.hgbu.example;

/\*\*

\* An Object Oriented Hello World Program.

\* @version 1.0

\*\*/

public class HelloWorld {

private String hello;

private String world;

/\*\*

\* Constructor

\*/

HelloWorld(){

hello = “Hello”;

world = “World”;

}

/\*\*

\* forms the Hello World Message.

\*/

public String formMessage(){

String message;

message = hello + ” ” + world;

return message;

}

public static void main(String[] args) {

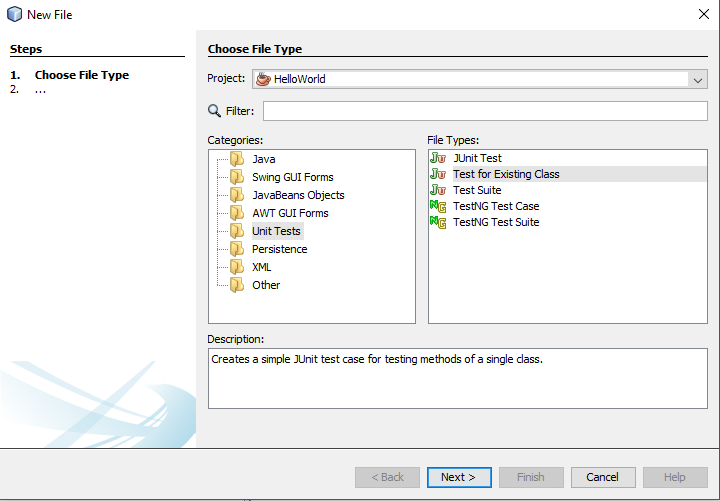
HelloWorld helloWorld = new HelloWorld();

System.out.println(helloWorld.formMessage());

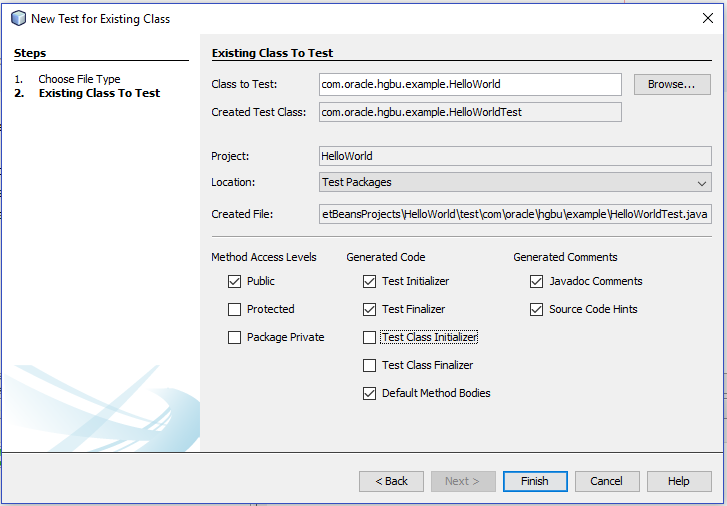
}

}

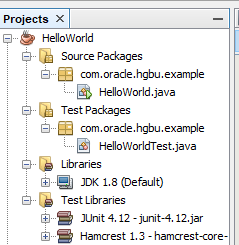
7. Now select **New File...** from the context menu of the Project and Choose **Test for Existing Class**.



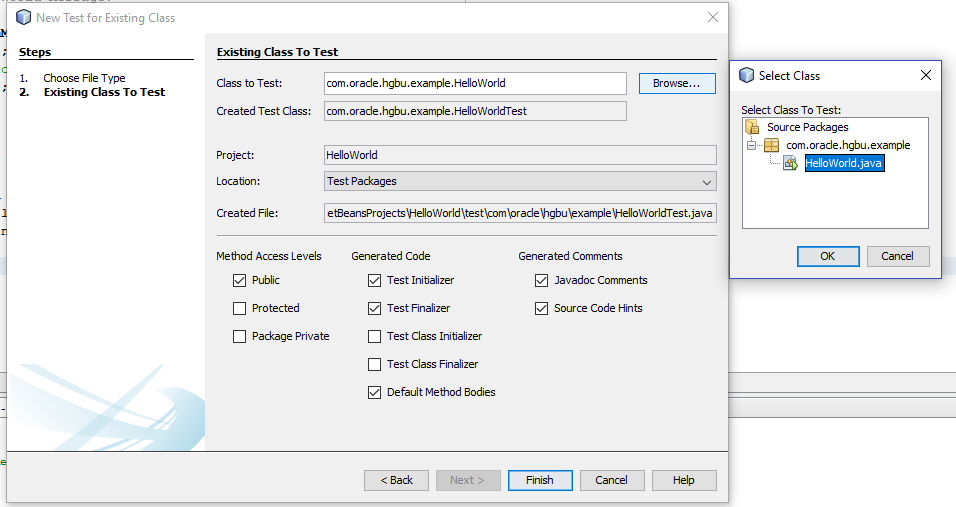
8. Enter the following details and Click **Next**.



See below the project structure for your reference.



9. Now select the **HelloWorld.java** from **Browse…** as shown in the figure below and click **Finish**.



10. Write the following code for the HelloWorldTest class.

package com.oracle.hgbu.example;

import org.junit.After;

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

/\*\*

\* Test class for HelloWorld

\* @version 1.0

\*/

public class HelloWorldTest {

HelloWorld helloWorldInstance;

@Before

public void setUp() {

System.out.println("\* HelloWorldTest: Before method setUp()");

helloWorldInstance = new HelloWorld();

}

@After

public void tearDown() {

System.out.println("\* HelloWorldTest: After method tearDown()");

helloWorldInstance = null;

}

/\*\*

\* unit test for formMessage method.

\*/

@Test

public void testFormMessage() {

//Asserts that two Strings are equal

String expResult = "Hello World!";

String result = helloWorldInstance.formMessage();

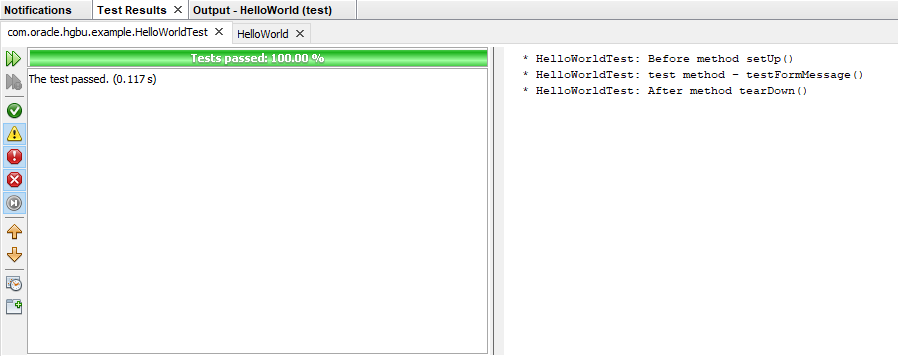
System.out.println("\* HelloWorldTest: test method 1 testFormMessage()");

Assert.assertEquals(expResult, result);

}

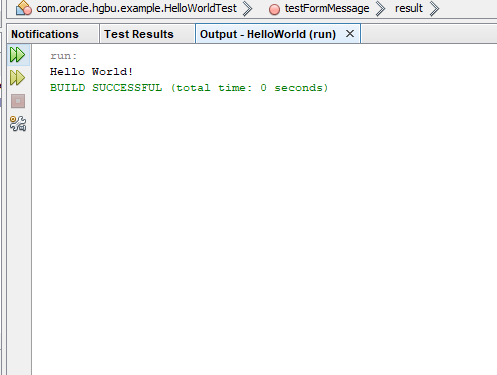
}

11. Select **Test** from the context menu of the project or press (Alt + F6).



Congrats, the green bar indicates that the test is successful. If the test case fails, a red bar and error message are displayed.

12. Now select **Run** from the context menu of the project to Run the Hello World Program.



The string “Hello World” is printed to the console. This finishes the Hello World program with Unit Test in NetBeans IDE.

# **MavenCodeCoverage**

# **Examining code coverage in Maven projects**

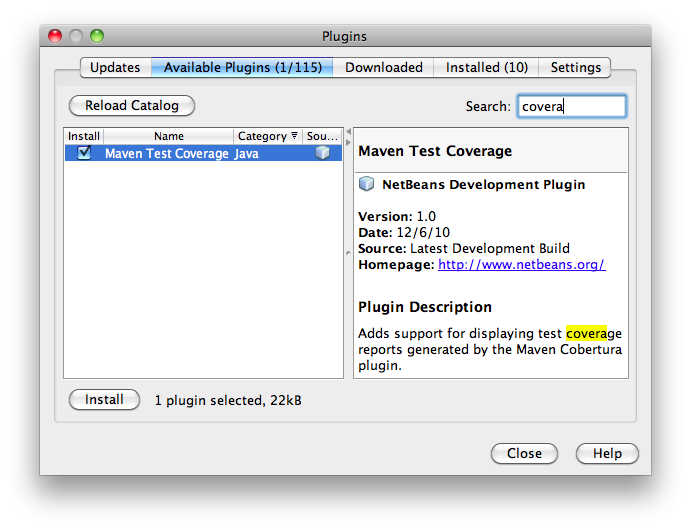
This document describes how to display test coverage for your Maven project in NetBeans, via the Cobertura Maven plugin.

|  |
| --- |
| Contents [[hide](javascript:toggleToc())]   * [1 Examining code coverage in Maven projects](http://wiki.netbeans.org/MavenCodeCoverage#Examining_code_coverage_in_Maven_projects)   + [1.1 Installation and setup](http://wiki.netbeans.org/MavenCodeCoverage#Installation_and_setup)   + [1.2 Configuring your project](http://wiki.netbeans.org/MavenCodeCoverage#Configuring_your_project)   + [1.3 Examining Code Coverage](http://wiki.netbeans.org/MavenCodeCoverage#Examining_Code_Coverage)   + [1.4 Using JaCoCo](http://wiki.netbeans.org/MavenCodeCoverage#Using_JaCoCo) |

## Installation and setup

This document assumes NetBeans 7.0 or later.

The test coverage functionality is provided by the Maven Test Coverage plugin, which is available for 7.0 on the dev update center and also the plugin portal. Start up the IDE, go to Tools | Plugins and search for "coverage". If NetBeans finds the plugin, install it by pressing the Install button and following the wizard. You can also download it from the [Plugin Portal](http://plugins.netbeans.org/PluginPortal/faces/PluginDetailPage.jsp?pluginid=37496).

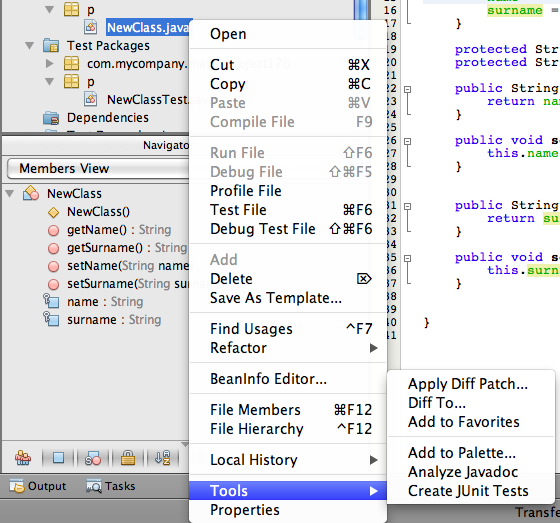
[](http://wiki.netbeans.org/File:Maven_test_coverage_plugin.png)

NetBeans 7.1 and later include the plugin by default, so you do not need to download anything.

## Configuring your project

We now need a suitable Maven project. If you already have your own Maven project which contains some tests, just open it using File | Open Project.

If not, just create one using File | New Project, selecting one of the templates in the **Maven** category and following the wizard. You will also need some code and tests for this code: the easiest way is to do this is right-clicking an existing Java class, and choosing Tools | Create JUnit Tests.

[](http://wiki.netbeans.org/File:Create_junit_tests.png)

The most important part of the setup is to configure the Cobertura Maven plugin in your project. The simplest way is to add the following snippet of code to the pom.xml file of your project:

<reporting>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>cobertura-maven-plugin</artifactId>

<version>2.4</version>

</plugin>

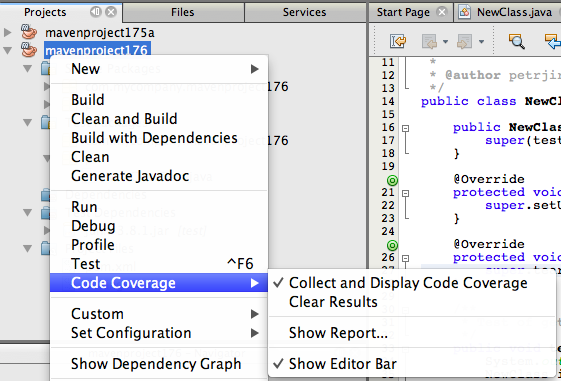
</plugins>

</reporting>

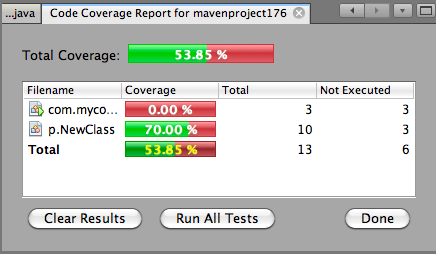
For a more comprehensive documentation, see the [cobertura-maven-plugin website](http://mojo.codehaus.org/cobertura-maven-plugin/usage.html).

## Examining Code Coverage

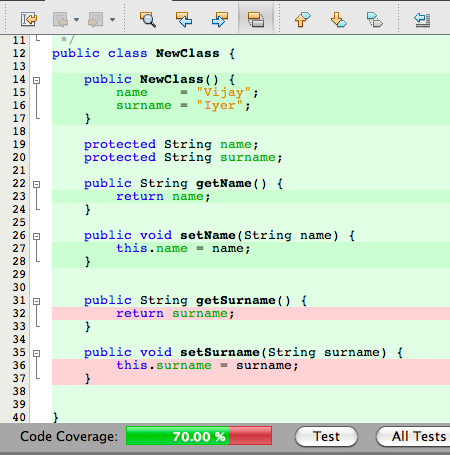
Now you are ready to run the code coverage report for your project. In the pop-up menu of the project node in the Projects tab, you will now see a new Code Coverage submenu. In this submenu, choose Show Report....

[](http://wiki.netbeans.org/File:Cc_popup_menu.png)

The **Code Coverage Report** window appears. When you press the Run All Tests button in this window, you will see the code coverage summary for the project.

[](http://wiki.netbeans.org/File:Code_coverage_summary.png)

Now you can also double-click individual items to get code coverage details for individual classes. In this detailed view, lines covered by tests are displayed with green background, whereas lines that are not covered have pink background.

[](http://wiki.netbeans.org/File:File_line_coverage.png)

To submit feedback on the Maven Code Coverage plugin, post to the [NetBeans forums or mailing lists](http://netbeans.org/community/lists/index.html), or log bugs in the [bug tracker](http://netbeans.org/community/issues.html), category projects/Maven.

## Using JaCoCo

In NetBeans 7.2, JaCoCo is supported as a code coverage engine, since Cobertura does not work with JDK 7 language constructs. Usage is essentially the same as for Cobertura; you just need to configure JaCoCo as per its recommendations:

<build>

<plugins>

<plugin>

<groupId>org.jacoco</groupId>

<artifactId>jacoco-maven-plugin</artifactId>

<version>0.7.0.201403182114</version>

<executions>

<execution>

<goals>

<goal>prepare-agent</goal>

</goals>

</execution>

<execution>

<id>report</id>

<phase>prepare-package</phase>

<goals>

<goal>report</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>