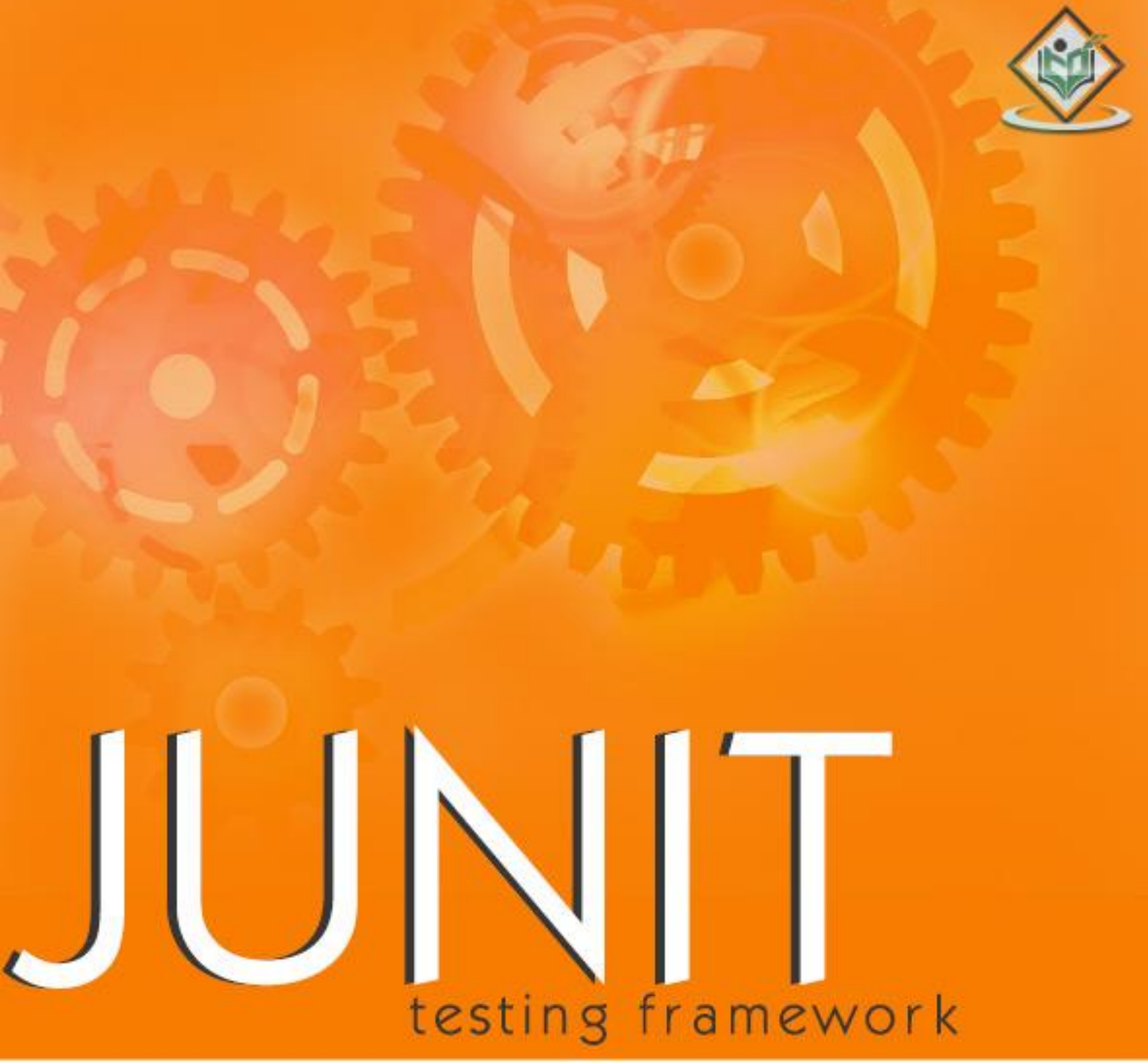

JUNIT

testing framework

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

JUnit is a unit testing framework for Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks collectively known as xUnit, that originated with JUnit.

This tutorial explains the use of JUnit in your project unit testing, while working with Java. After completing this tutorial you will gain sufficient knowledge in using JUnit testing framework from where you can take yourself to next levels.

Audience

This tutorial has been prepared for beginners to help them understand the basic functionality of JUnit tool.

Prerequisites

We assume you are going to use JUnit to handle all levels of Java projects development. So it will be good if you have the knowledge of software development using any programming language, especially Java programming and software testing process.

Copyright & Disclaimer

Table of Contents

1. OVERVIEW

Testing is the process of checking the functionality of an application to ensure it runs as per requirements. Unit testing comes into picture at the developers' level; it is the testing of single entity (class or method). Unit testing plays a critical role in helping a software company deliver quality products to its customers.

Unit testing can be done in two ways: manual testing and automated testing.

Manual Testing	Automated Testing
Executing a test cases manually without any tool support is known as manual testing.	Taking tool support and executing the test cases by using an automation tool is known as automation testing.
Time-consuming and tedious: Since test cases are executed by human resources, it is very slow and tedious.	**Fast**: Automation runs test cases significantly faster than human resources.
Huge investment in human resources: As test cases need to be executed manually, more testers are required in manual testing.	**Less investment in human resources**: Test cases are executed using automation tools, so less number of testers are required in automation testing.
Less reliable: Manual testing is less reliable, as it has to account for human errors.	**More reliable**: Automation tests are precise and reliable.
Non-programmable: No programming can be done to write sophisticated tests to fetch hidden information.	**Programmable**: Testers can program sophisticated tests to bring out hidden information.

What is JUnit?

JUnit is a unit testing framework for Java programming language. It plays a crucial role test-driven development, and is a family of unit testing frameworks collectively known as xUnit.

JUnit promotes the idea of "first testing then coding", which emphasizes on setting up the test data for a piece of code that can be tested first and then implemented. This approach is like "test a little, code a little, test a little, code a little." It increases the productivity of the programmer and the stability of program code, which in turn reduces the stress on the programmer and the time spent on debugging.

Features of JUnit

- JUnit is an open source framework, which is used for writing and running tests.
- Provides annotations to identify test methods.
- Provides assertions for testing expected results.
- Provides test runners for running tests.
- JUnit tests allow you to write codes faster, which increases quality.
- JUnit is elegantly simple. It is less complex and takes less time.
- JUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results.
- JUnit tests can be organized into test suites containing test cases and even other test suites.
- JUnit shows test progress in a bar that is green if the test is running smoothly, and it turns red when a test fails.

What is a Unit Test Case?

A Unit Test Case is a part of code, which ensures that another part of code (method) works as expected. To achieve the desired results quickly, a test framework is required. JUnit is a perfect unit test framework for Java programming language.

A formal written unit test case is characterized by a known input and an expected output, which is worked out before the test is executed. The known input should test a precondition and the expected output should test a post-condition.

There must be at least two unit test cases for each requirement: one positive test and one negative test. If a requirement has sub-requirements, each sub-requirement must have at least two test cases as positive and negative.

2. ENVIRONMENT SETUP

Try it Online Option

We already have set up Java programming environment online, so that you can compile and execute all the available examples online at the same time while you are doing your theory work. It gives you confidence in what you are reading and verify the programs with different options. Feel free to modify any example and execute it online.

Try the following example using our online compiler option available at http://www.compileonline.com/.

```
public class MyFirstJavaProgram {

    public static void main(String []args) {

       System.out.println("Hello World");

    }

}
```

For most of the examples given in this tutorial, you will find a **Try it** option in our website code sections at the top right corner that will take you to the online compiler. So just make use of it and enjoy your learning.

Local Environment Setup

JUnit is a framework for Java, so the very first requirement is to have JDK installed in your machine.

System Requirement

JDK	1.5 or above.
Memory	No minimum requirement.

Disk Space	No minimum requirement.
Operating System	No minimum requirement.

Step 1: Verify Java Installation in Your Machine

First of all, open the console and execute a java command based on the operating system you are working on.

OS	Task	Command
Windows	Open Command Console	c:\> java -version
Linux	Open Command Terminal	$ java -version
Mac	Open Terminal	machine:~ joseph$ java -version

Let's verify the output for all the operating systems:

OS	Output
Windows	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
Linux	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07)

	Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
Mac	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM)64-Bit Server VM (build 17.0-b17, mixed mode, sharing)

If you do not have Java installed on your system, then download the Java Software Development Kit (SDK) from the following link:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

We are assuming Java 1.6.0_21 as the installed version for this tutorial.

Step 2: Set JAVA Environment

Set the **JAVA_HOME** environment variable to point to the base directory location where Java is installed on your machine. For example,

OS	Output
Windows	Set the environment variable JAVA_HOME to C:\Program Files\Java\jdk1.6.0_21
Linux	export JAVA_HOME=/usr/local/java-current
Mac	export JAVA_HOME=/Library/Java/Home

Append Java compiler location to the System Path.

OS	Output

Windows	Append the string **C:\Program Files\Java\jdk1.6.0_21\bin** at the end of the system variable, **Path**.
Linux	export PATH=$PATH:$JAVA_HOME/bin/
Mac	not required

Verify Java installation using the command **java -version** as explained above.

Step 3: Download Junit Archive

Download the latest version of JUnit jar file from http://www.junit.org. At the time of writing this tutorial, we have downloaded Junit-4.10.jar and copied it into C:\>JUnit folder.

OS	Archive name
Windows	junit4.10.jar
Linux	junit4.10.jar
Mac	junit4.10.jar

Step 4: Set JUnit Environment

Set the **JUNIT_HOME** environment variable to point to the base directory location where JUNIT jar is stored on your machine. Let's assuming we've stored junit4.10.jar in the JUNIT folder.

OS	Description
Windows	Set the environment variable JUNIT_HOME to C:\JUNIT

Linux	export JUNIT_HOME=/usr/local/JUNIT
Mac	export JUNIT_HOME=/Library/JUNIT

Step 5: Set CLASSPATH Variable

Set the **CLASSPATH** environment variable to point to the JUNIT jar location.

OS	Description
Windows	Set the environment variable CLASSPATH to %CLASSPATH%;%JUNIT_HOME%\junit4.10.jar;.;
Linux	export CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.10.jar:.
Mac	export CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.10.jar:.

Step 6: Test JUnit Setup

Create a java class file name TestJunit in **C:\>JUNIT_WORKSPACE**

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;
public class TestJunit {
   @Test
   public void testAdd() {
      String str= "Junit is working fine";
      assertEquals("Junit is working fine",str);
   }
}
```

Create a java class file name TestRunner in **C:\>JUNIT_WORKSPACE** to execute test case(s).

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(TestJunit.class);
      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

Step 7: Verify the Result

Compile the classes using **javac** compiler as follows:

```
C:\JUNIT_WORKSPACE>javac TestJunit.java TestRunner.java
```

Now run the Test Runner to see the result as follows:

```
C:\JUNIT_WORKSPACE>java TestRunner
```

Verify the output.

```
true
```

3. TEST FRAMEWORK

JUnit is a **Regression Testing Framework** used by developers to implement unit testing in Java, and accelerate programming speed and increase the quality of code. JUnit Framework can be easily integrated with either of the following:

- Eclipse
- Ant
- Maven

Features of Junit Test Framework

JUnit test framework provides the following important features:

- Fixtures
- Test suites
- Test runners
- JUnit classes

Fixtures

Fixtures is a fixed state of a set of objects used as a baseline for running tests. The purpose of a test fixture is to ensure that there is a well-known and fixed environment in which tests are run so that results are repeatable. It includes:

- setUp() method, which runs before every test invocation.
- tearDown() method, which runs after every test method.

Let's check one example:

```
import junit.framework.*;


public class JavaTest extends TestCase {
   protected int value1, value2;
```

```
   // assigning the values
   protected void setUp(){
      value1=3;
      value2=3;
   }

   // test method to add two values
   public void testAdd(){
      double result= value1 + value2;
      assertTrue(result == 6);
   }
}
```

Test Suites

A test suite bundles a few unit test cases and runs them together. In JUnit, both @RunWith and @Suite annotation are used to run the suite test. Given below is an example that uses TestJunit1 & TestJunit2 test classes.

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

//JUnit Suite Test
@RunWith(Suite.class)
@Suite.SuiteClasses({
   TestJunit1.class ,TestJunit2.class
})
public class JunitTestSuite {
}
import org.junit.Test;
```

```
import org.junit.Ignore;
import static org.junit.Assert.assertEquals;

public class TestJunit1 {

   String message = "Robert";
   MessageUtil messageUtil = new MessageUtil(message);

   @Test
   public void testPrintMessage() {
      System.out.println("Inside testPrintMessage()");
      assertEquals(message, messageUtil.printMessage());
   }
}
import org.junit.Test;
import org.junit.Ignore;
import static org.junit.Assert.assertEquals;

public class TestJunit2 {

   String message = "Robert";
   MessageUtil messageUtil = new MessageUtil(message);

   @Test
   public void testSalutationMessage() {
      System.out.println("Inside testSalutationMessage()");
      message = "Hi!" + "Robert";
      assertEquals(message,messageUtil.salutationMessage());
   }
```

```
}
```

Test Runners

Test runner is used for executing the test cases. Here is an example that assumes the test class **TestJunit** already exists.

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(TestJunit.class);
      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

JUnit Classes

JUnit classes are important classes, used in writing and testing JUnits. Some of the important classes are:

- Assert - Contains a set of assert methods.
- TestCase - Contains a test case that defines the fixture to run multiple tests.
- TestResult - Contains methods to collect the results of executing a test case.

4. BASIC USAGE

Let us now have a basic example to demonstrate the step-by-step process of using Junit.

Create a Class

Create a java class to be tested, say, MessageUtil.java in **C:\> JUNIT_WORKSPACE**

```
/*
* This class prints the given message on console.
*/
public class MessageUtil {

   private String message;

   //Constructor
   //@param message to be printed
   public MessageUtil(String message){
      this.message = message;
   }

   // prints the message
   public String printMessage(){
      System.out.println(message);
      return message;
   }
}
```

Create Test Case Class

1. Create a java test class, say, TestJunit.java.
2. Add a test method testPrintMessage() to your test class.
3. Add an Annotaion @Test to method testPrintMessage().
4. Implement the test condition and check the condition using assertEquals API of Junit.

Create a java class file name TestJunit.java in C:\>JUNIT_WORKSPACE.

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;
public class TestJunit {

   String message = "Hello World";
   MessageUtil messageUtil = new MessageUtil(message);

   @Test
   public void testPrintMessage() {
      assertEquals(message,messageUtil.printMessage());
   }
}
```

Create Test Runner Class

1. Create a TestRunner java class.
2. Use runClasses method of JUnitCore class of JUnit to run the test case of the above created test class.
3. Get the result of test cases run in Result Object.
4. Get failure(s) using the getFailures() method of Result object.
5. Get Success result using the wasSuccessful() method of Result object.

Create a java class file named TestRunner.java in C:\>JUNIT_WORKSPACE to execute test case(s).

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(TestJunit.class);
      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

Compile the MessageUtil, Test case and Test Runner classes using javac.

```
C:\JUNIT_WORKSPACE>javac MessageUtil.java TestJunit.java TestRunner.java
```

Now run the Test Runner, which will run the test case defined in the provided Test Case class.

```
C:\JUNIT_WORKSPACE>java TestRunner
```

Verify the output.

```
Hello World
true
```

Now update TestJunit in C:\>JUNIT_WORKSPACE so that the test fails. Change the message string.

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;
```

```
public class TestJunit {

   String message = "Hello World";
   MessageUtil messageUtil = new MessageUtil(message);

   @Test
   public void testPrintMessage() {
      message = "New Word";
      assertEquals(message,messageUtil.printMessage());
   }
}
```

Let's keep the rest of the classes as is, and try to run the same Test Runner.

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(TestJunit.class);
      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

Now run the Test Runner, which will run the test case defined in the provided Test Case class.

```
C:\JUNIT_WORKSPACE>java TestRunner
```

Verify the output.

```
Hello World
testPrintMessage(TestJunit): expected:<[New Wor]d> but was:<[Hello Worl]d>
false
```

End of ebook preview

If you liked what you saw…

Buy it from our store @ **https://store.tutorialspoint.com**