JUNIT

Introduction:

JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.Unit testing can be done in two ways − manual testing and automated testing.

JUnit shows test progress in a bar that is green if the test is running smoothly, and it turns red when a test fails.

How to install JUNIT in your machine?

## 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 –versioonn |

If you want to check wether java is available in your system then open your command prompt and type as java –version. If it exist then it will give output as follows.

C:\Users\Mounika>java -version

java version "1.8.0\_291"

Java(TM) SE Runtime Environment (build 1.8.0\_291-b10)

Java HotSpot(TM) 64-Bit Server VM (build 25.291-b10, mixed mode)

|  |  |
| --- | --- |
| Linux | java version "1.8.0\_101"  Java(TM) SE Runtime Environment (build 1.8.0\_101) |
| Mac | java version "1.8.0\_101"  Java(TM) SE Runtime Environment (build 1.8.0\_101) |

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

[https://www.oracle.com](https://www.oracle.com/technetwork/java/javase/downloads/index.html).

## 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\jdk1.8.0\_101 |
| 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\jdk1.8.0\_101\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.

How to download JUnit Archive?

Follow the below process to download junit.

1.go to through this link <https://github.com/junit-team/junit4/wiki/Download-and-Install>

and you will find in that pages as below.

# Plain-old JAR

Download the following JARs and add them to your test classpath:

* [junit.jar](https://search.maven.org/search?q=g:junit%20AND%20a:junit)
* [hamcrest-core.jar](https://search.maven.org/artifact/org.hamcrest/hamcrest-core/1.3/jar)

click on junit.jar

In next page you will find junit and downloading option so that you can easily download junit.jar.

After installing the junit the next step is to set Environmental variable and System Variable path

After setting the paths we can able to launch eclipse properly by setting paths correctly.

In Junit we are going to run projects as dynamic.i.e we have to create dynamic projects to create them the process is

CASE-1:File->New->DynamicProject select it if in case dynamic is not available there it self you must follow as below

Go to File->new->other->Web->Dynamic Web Project->next

Then you will get a window you have to give project name and click next next and select web.xml checkbox and click finish.

After creating a project we have to add some libraries and path follows as

Buildpath->configurebuildpath->java buildpath:libraries->add external jar->c:select jarfile and open next click on add libraries->junit5->finish->next ->apply.

Let us go into our 1st program in junit this program can able to show how the program is going to give the message which we want to display

Program code:

package junitTesting;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class TestJunit {

String message = "Hello World";

MessageUtil messageUtil = new MessageUtil(message);

@Test

public void testPrintMessage() {

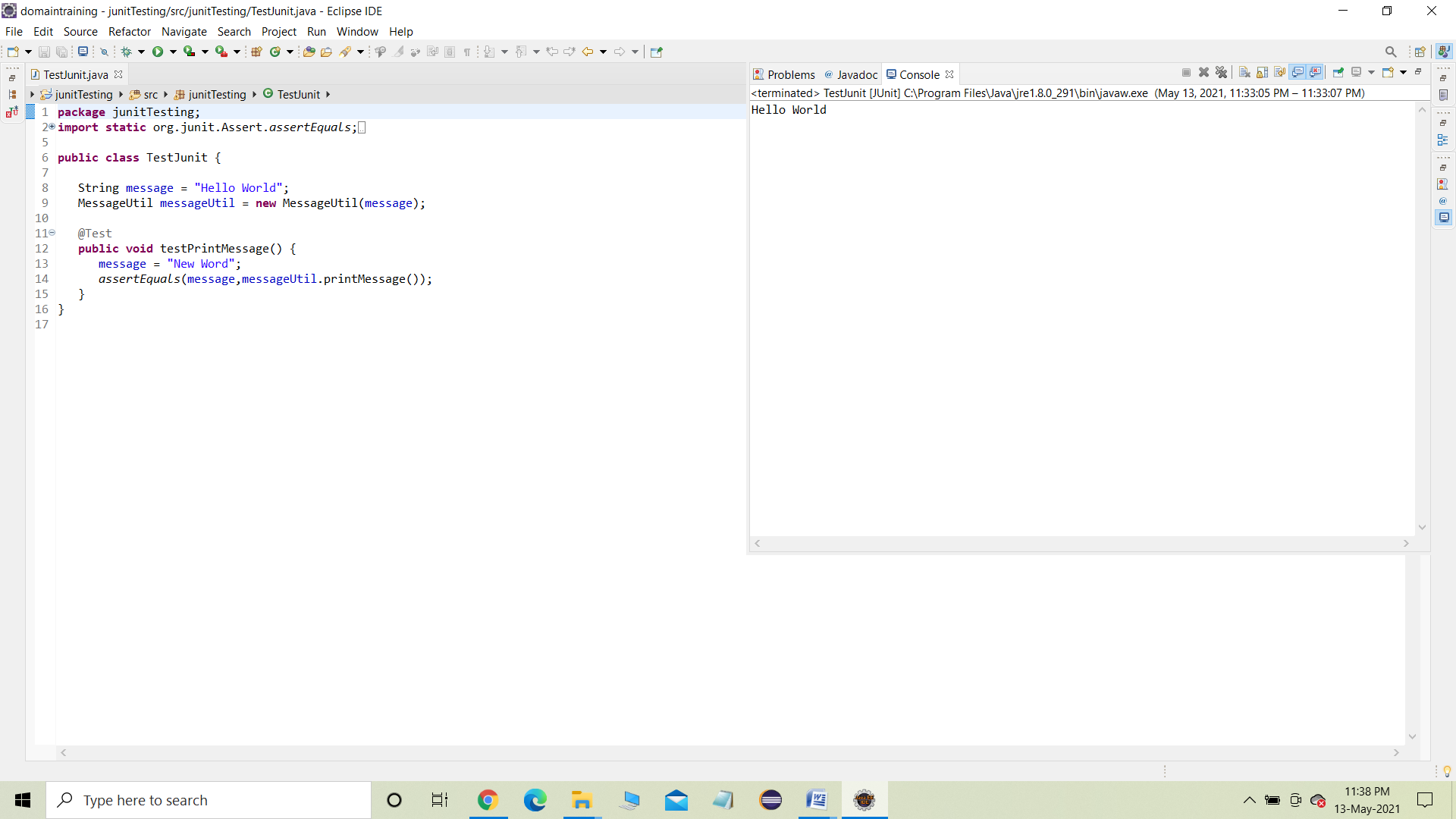
message = "New Word";

assertEquals(message,messageUtil.printMessage());

}

}

Output:



2.program code:

**package** junitTesting;

**import** org.junit.runner.JUnitCore;

**import** org.junit.runner.Result;

**import** org.junit.runner.notification.Failure;

**public** **class** Runner {

**public** **static** **void** main(String[] args) {

Result result = JUnitCore.*runClasses*(TestJunit.**class**);

**for** (Failure failure : result.getFailures()) {

System.***out***.println(failure.toString());

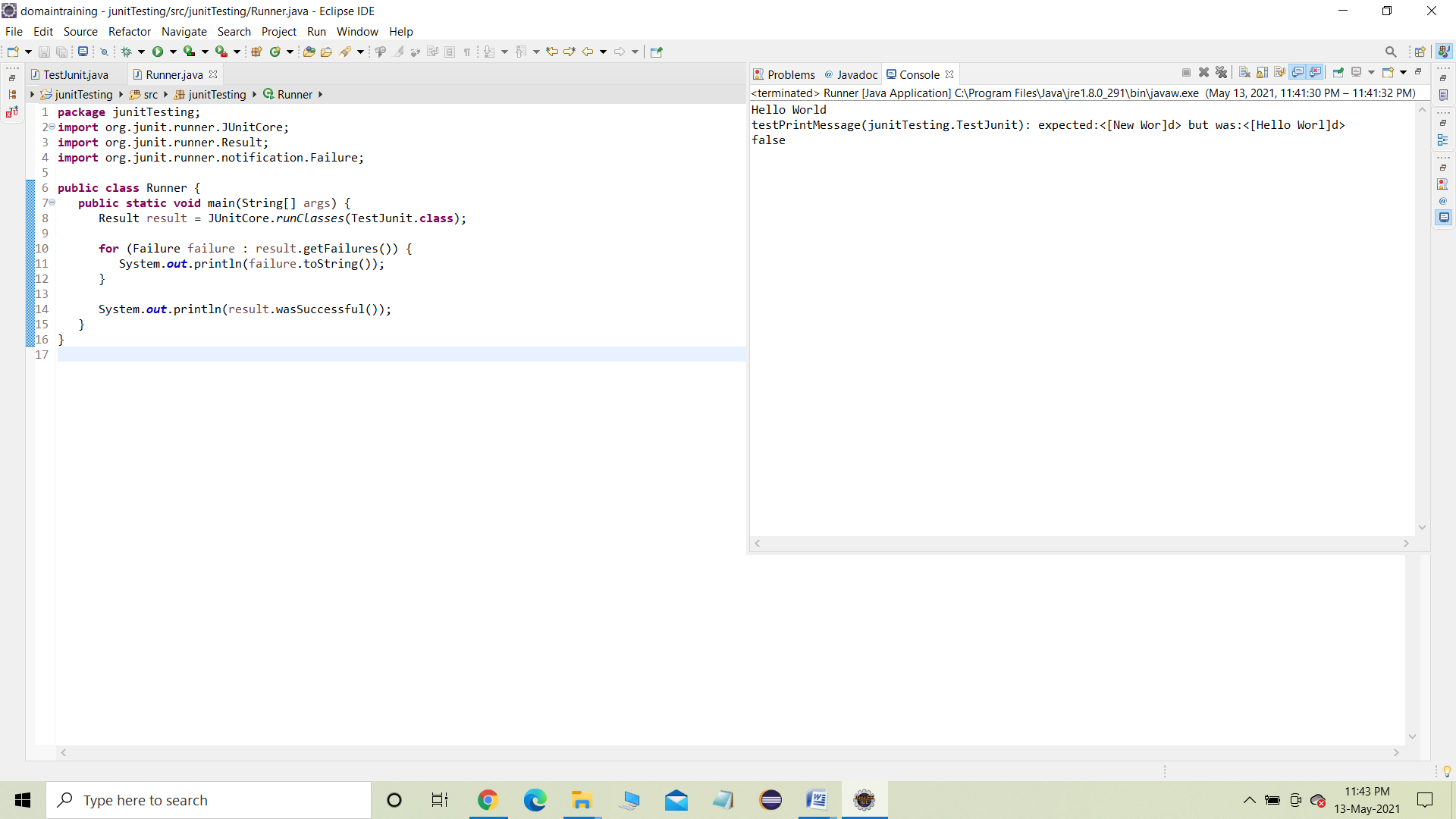
}

System.***out***.println(result.wasSuccessful());

}

}

Output:



## 3.Test Suites:

## package junitTesting;

## 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 { } output: InsidetestSalutationMessage

## 4.Program Code:

package junitTesting;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class TestJunit2 {

String message = "Robert";

MessageUtil messageUtil = new MessageUtil(message);

@Test

public void testSalutationMessage() {

System.out.println("InsidetestSalutationMessage");

message = "Hi" + "Robert";

assertEquals(message,messageUtil.salutationMessage());

}

}

Output:

InsidetestSalutationMessage

package junitTesting;

import org.junit.runner.JUnitCore;

import org.junit.runner.Result;

import org.junit.runner.notification.Failure;

public class TestRunner1 {

public static void main(String[] args) {

Result result = JUnitCore.runClasses(TestJunit1.class);

for (Failure failure : result.getFailures()) {

System.out.println(failure.toString());

}

System.out.println(result.wasSuccessful());

}

}

Output:

True

TestCase Class

A test case defines the fixture to run multiple tests. Some of the

important methods of TestCase class are as follows −

Sr.No. Methods &amp; Description

1.int countTestCases()

Counts the number of test cases executed by run(TestResult result).

2.TestResult createResult()

Creates a default TestResult object.

3.String getName()

Gets the name of a TestCase.

4.TestResult run()

A convenience method to run this test, collecting the results with a default

TestResult object.

5. void run(TestResult result)

Runs the test case and collects the results in TestResult.

6.void setName(String name)

Sets the name of a TestCase.

7. void setUp()

Sets up the fixture, for example, open a network connection.

8.void tearDown()

Tears down the fixture, for example, close a network connection.

9.String toString()

Returns a string representation of the test case.

TestResult Class:

1.void addError(Test test, Throwable t)

Adds an error to the list of errors.

2.void addFailure(Test test, AssertionFailedError t)

Adds a failure to the list of failures.

3.void endTest(Test test)

Informs the result that a test was completed.

4.int errorCount()

Gets the number of detected errors.

5. Enumeration&lt;TestFailure&gt; errors()

Returns an Enumeration for the errors.

6.int failureCount()

Gets the number of detected failures.

7.void run(TestCase test)

Runs a TestCase.

8.int runCount()

Gets the number of run tests.

9.void startTest(Test test)

Informs the result that a test will be started.

10.void stop()

Marks that the test run should stop.

TestSuite Class:

A TestSuite is a Composite of tests. It runs a collection of test cases.

1.void addTest(Test test)

Adds a test to the suite.

2.void addTestSuite(Class&lt;? extends TestCase&gt; testClass)

Adds the tests from the given class to the suite.

3.int countTestCases()

Counts the number of test cases that will be run by this test.

4.String getName()

Returns the name of the suite.

5.void run(TestResult result)

Runs the tests and collects their result in a TestResult.

6.void setName(String name)

Sets the name of the suite.

7.Test testAt(int index)

Returns the test at the given index.

8.int testCount()

Returns the number of tests in this suite.

9.static Test warning(String message)

Returns a test which will fail and log a warning message.

import junit.framework.\*;

public class JunitTestSuite {

public static void main(String[] a) {

// add the test&#39;s in the suite

TestSuite suite = new TestSuite(TestJunit1.class,

TestJunit2.class, TestJunit3.class );

TestResult result = new TestResult();

suite.run(result);

System.out.println(&quot;Number of test cases = &quot; +

result.runCount());

}

}

Compile the Test suite classes using javac.

Now run the Test Suite.

C:\JUNIT\_WORKSPACE&gt;java JunitTestSuite

Output:

No of Test Case = 1

Test Case Name = testAdd

Updated Test Case Name = testNewAdd

Number of test cases = 3

we will see one complete example of JUnit testing using POJO

class, Business logic class, and a test class, which will be run by the

test runner.

**public** **class** EmployeeDetails {

**private** String name;

**private** **double** monthlySalary;

**private** **int** age;

/\*\*

\* **@return** the name

\*/

**public** String getName() {

**return** name;

}

/\*\*

\* **@param** name the name to set

\*/

**public** **void** setName(String name) {

**this**.name = name;

}

/\*\*

\* **@return** the monthlySalary

\*/

**public** **double** getMonthlySalary() {

**return** monthlySalary;

}

/\*\*

\* **@param** monthlySalary the monthlySalary to set

\*/

**public** **void** setMonthlySalary(**double** monthlySalary) {

**this**.monthlySalary = monthlySalary;

}

/\*\*

\* **@return** the age

\*/

**public** **int** getAge() {

**return** age;

}

/\*\*

\* **@param** age the age to set

\*/

**public** **void** setAge(**int** age) {

**this**.age = age;

}

}

**public** **class** EmpBusinessLogic {

// Calculate the yearly salary of employee

**public** **double** calculateYearlySalary(EmployeeDetails employeeDetails) {

**double** yearlySalary = 0;

yearlySalary = employeeDetails.getMonthlySalary() \* 12;

**return** yearlySalary;

}

// Calculate the appraisal amount of employee

**public** **double** calculateAppraisal(EmployeeDetails employeeDetails) {

**double** appraisal = 0;

**if** (employeeDetails.getMonthlySalary() < 10000) {

appraisal = 500;

} **else** {

appraisal = 1000;

}

**return** appraisal;

}

}

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class TestEmployeeDetails {

EmpBusinessLogic empBusinessLogic = new EmpBusinessLogic();

EmployeeDetails employee = new EmployeeDetails();

//test to check appraisal

@Test

public void testCalculateAppriasal() {

employee.setName("Rajeev");

employee.setAge(25);

employee.setMonthlySalary(8000);

double appraisal = empBusinessLogic.calculateAppraisal(employee);

assertEquals(500, appraisal, 0.0);

}

// test to check yearly salary

@Test

public void testCalculateYearlySalary() {

employee.setName("Rajeev");

employee.setAge(25);

employee.setMonthlySalary(8000);

double salary = empBusinessLogic.calculateYearlySalary(employee);

assertEquals(96000, salary, 0.0);

}

}

Output:

True

Assertion:

This class provides a set of assertion methods, useful for writing

tests. Only failed assertions are recorded.

1.void assertEquals(boolean expected, boolean actual)

Checks that two primitives/objects are equal.

2.void assertTrue(boolean condition)

Checks that a condition is true.

3.void assertFalse(boolean condition)

Checks that a condition is false.

4.void assertNotNull(Object object)

Checks that an object isn&#39;t null.

5.void assertNull(Object object)

Checks that an object is null.

6.void assertSame(object1, object2)

The assertSame() method tests if two object references point to the same object.

7.void assertNotSame(object1, object2)

The assertNotSame() method tests if two object references do not point to the same object.

8.void assertArrayEquals(expectedArray, resultArray);

The assertArrayEquals() method will test whether two arrays are equal to each other.

import org.junit.Test;

import static org.junit.Assert.\*;

public class TestAssertions {

@Test

public void testAssertions() {

// test data

String str1 = new String("abc");

String str2 = new String("abc");

String str3 = null;

String str4 = "abc";

String str5 = "abc";

int val1 = 5;

int val2 = 6;

String[] expectedArray = { "one", "two", "three" };

String[] resultArray = { "one", "two", "three" };

// Check that two objects are equal

assertEquals(str1, str2);

// Check that a condition is true

assertTrue(val1 < val2);

// Check that a condition is false

assertFalse(val1 > val2);

// Check that an object isn't null

assertNotNull(str1);

// Check that an object is null

assertNull(str3);

// Check if two object references point to the same object

assertSame(str4, str5);

// Check if two object references not point to the same object

assertNotSame(str1, str3);

// Check whether two arrays are equal to each other.

assertArrayEquals(expectedArray, resultArray);

}

}

->Create another test runnerclass.

import org.junit.runner.JUnitCore;

import org.junit.runner.Result;

import org.junit.runner.notification.Failure;

public class TestRunner4 {

public static void main(String[] args) {

Result result = JUnitCore.runClasses(TestAssertions.class);

for (Failure failure : result.getFailures()) {

System.out.println(failure.toString());

}

System.out.println(result.wasSuccessful());

}

}

Output:

True

Annotation:  
Annotations are like meta-tags that you can add to your code, and apply them to methods or in class.

1.@Test

The Test annotation tells JUnit that the public void method to which it is attached can be run as a test case.

2.@Before

Several tests need similar objects created before they can run. Annotating a public void method with @Before causes that method to be run before each Test method.

3.@After

If you allocate external resources in a Before method, you need to release them after the test runs. Annotating a public void method with @After causes that method to be run after the Test method.

4.@BeforeClass

Annotating a public static void method with @BeforeClass causes it to be run once before any of the test methods in the class.

5.@AfterClass

This will perform the method after all tests have finished. This can be used to perform clean-up activities.

6.@Ignore

The Ignore annotation is used to ignore the test and that test will not be

executed.

Create a class JunitAnnotation.

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Ignore;

import org.junit.Test;

public class JunitAnnotation {

// execute before class

@BeforeClass

public static void beforeClass() {

System.out.println("in before class");

}

// execute after class

@AfterClass

public static void afterClass() {

System.out.println("in after class");

}

// execute before test

@Before

public void before() {

System.out.println("in before");

}

// execute after test

@After

public void after() {

System.out.println("in after");

}

// test case

@Test

public void test() {

System.out.println("in test");

}

// test case ignore and will not execute

@Ignore

public void ignoreTest() {

System.out.println("in ignore test");

}

}

Create test runner class.

import org.junit.runner.JUnitCore;

import org.junit.runner.Result;

import org.junit.runner.notification.Failure;

public class TestRunner5 {

public static void main(String[] args) {

Result result = JUnitCore.runClasses(JunitAnnotation.class);

for (Failure failure : result.getFailures()) {

System.out.println(failure.toString());

}

System.out.println(result.wasSuccessful());

}

}

Output:

in before class

in before

in test

in after

in after class

true

Create Test Case Classes:  
create TestJunit1 class;

import org.junit.Test;

import org.junit.Ignore;

import static org.junit.Assert.assertEquals;

public class TestJunit1 {

String message = &quot;Robert&quot;;

MessageUtil messageUtil = new MessageUtil(message);

@Test

public void testPrintMessage() {

System.out.println(&quot;Inside testPrintMessage()&quot;);

assertEquals(message,

messageUtil.printMessage());

}

}

And TestJunit2

import org.junit.Test;

import org.junit.Ignore;

import static org.junit.Assert.assertEquals;

public class TestJunit2 {

String message = &quot;Robert&quot;;

MessageUtil messageUtil = new MessageUtil(message);

@Test

public void testSalutationMessage() {

System.out.println(&quot;Inside

testSalutationMessage()&quot;);

message = &quot;Hi!&quot; + &quot;Robert&quot;;

assertEquals(message,messageUtil.salutationMessage());

}

}

Create Test Suite Class

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(JunitTestSuite.class);

for (Failure failure : result.getFailures()) {

System.out.println(failure.toString());

}

System.out.println(result.wasSuccessful());

}

}

Compile all the java classes and running testrunner we will get output as

Follows:

Inside testPrintMessage()

Robert

Inside testSalutationMessage()

Hi Robert

True

Create a Class:  
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;

}

// add &quot;Hi!&quot; to the message

public String salutationMessage(){

message = &quot;Hi!&quot; + message;

System.out.println(message);

return message;

}

}

Create Test Case Class:  
Create a java class file named TestJunit.java

import org.junit.Test;

import org.junit.Ignore;

import static org.junit.Assert.assertEquals;

public class TestJunit {

String message = &quot;Robert&quot;;

MessageUtil messageUtil = new MessageUtil(message);

@Ignore

@Test

public void testPrintMessage() {

System.out.println(&quot;Inside testPrintMessage()&quot;);

message = &quot;Robert&quot;;

assertEquals(message,messageUtil.printMessage());

}

@Test

public void testSalutationMessage() {

System.out.println(&quot;Inside

testSalutationMessage()&quot;);

message = &quot;Hi!&quot; + &quot;Robert&quot;;

assertEquals(message,messageUtil.salutationMessage());

}

}

Create Test Runner Class:  
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 Now run the Test Runner,

Output:

Inside testSalutationMessage()

Hi!Robert

True

package junitTesting;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class TestJunit2 {

String message = "Robert";

MessageUtil messageUtil = new MessageUtil(message);

@Test

public void testSalutationMessage() {

System.out.println("InsidetestSalutationMessage");

message = "Hi" + "Robert";

assertEquals(message,messageUtil.salutationMessage());

}

}

Compile the above file.

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());

}

}

Run the above file and we will get

Output: true

Create MessageUtil.java file

**public class MessageUtil {**

**private String message;**

**//Constructor**

**//@param message to be printed**

**public MessageUtil(String message){**

**this.message = message;**

**}**

**// prints the message**

**public void printMessage(){**

**System.out.println(message);**

**while(true);**

**}**

**// add &quot;Hi!&quot; to the message**

**public String salutationMessage(){**

**message = &quot;Hi!&quot; + message;**

**System.out.println(message);**

**return message;**

**}**

**}**

Then Create Test Case Class

import org.junit.Test;

import org.junit.Ignore;

import static org.junit.Assert.assertEquals;

public class TestJunit {

String message = &quot;Robert&quot;;

MessageUtil messageUtil = new MessageUtil(message);

@Test(timeout = 1000)

public void testPrintMessage() {

System.out.println(&quot;Inside testPrintMessage()&quot;);

messageUtil.printMessage();

}

@Test

public void testSalutationMessage() {

System.out.println(&quot;Inside

testSalutationMessage()&quot;);

message = &quot;Hi!&quot; + &quot;Robert&quot;;

assertEquals(message,messageUtil.salutationMessage());

}

}

Then Create Test Runner Class,

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 and Now run the Test Runner

Output:

Inside testPrintMessage()

Robert

Inside testSalutationMessage()

Hi!Robert

testPrintMessage(TestJunit): test timed out after 1000

milliseconds

false