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.

he most important package in JUnit is **junit.framework**, which contains all the core classes. Some of the important classes are as follows −

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **Class Name** | **Functionality** |
| 1 | Assert | A set of assert methods. |
| 2 | TestCase | A test case defines the fixture to run multiple tests. |
| 3 | TestResult | A TestResult collects the results of executing a test case. |
| 4 | TestSuite | A TestSuite is a composite of tests. |

## Assert Class

Following is the declaration for **org.junit.Assert** class −

public class Assert extends java.lang.Object

This class provides a set of assertion methods useful for writing tests. Only failed assertions are recorded. Some of the important methods of Assert class are as follows −

|  |  |
| --- | --- |
| **Sr.No.** | **Methods & Description** |
| 1 | **void assertEquals(boolean expected, boolean actual)**  Checks that two primitives/objects are equal. |
| 2 | **void assertFalse(boolean condition)**  Checks that a condition is false. |
| 3 | **void assertNotNull(Object object)**  Checks that an object isn't null. |
| 4 | **void assertNull(Object object)**  Checks that an object is null. |
| 5 | **void assertTrue(boolean condition)**  Checks that a condition is true. |
| 6 | **void fail()**  Fails a test with no message. |

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;

}

}

**EmployeeDetails** class is used to −

* get/set the value of employee's name.
* get/set the value of employee's monthly salary.
* get/set the value of employee's age.

Create a file called **EmpBusinessLogic.java** in C:\>JUNIT\_WORKSPACE, which contains the business logic.

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;

}

}

**EmpBusinessLogic** class is used for calculating −

* the yearly salary of an employee.
* the appraisal amount of an employee.

Create a file called **TestEmployeeDetails.java** in C:\>JUNIT\_WORKSPACE, which contains the test cases to be tested.

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);

}

}

**TestEmployeeDetails** class is used for testing the methods of **EmpBusinessLogic** class. It

* tests the yearly salary of the employee.
* tests the appraisal amount of the employee.

Next, create a java class filed 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(TestEmployeeDetails.class);

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

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

}

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

}

}

Compile the test case and Test Runner classes using javac.

## Assertion

All the assertions are in the Assert class.

public class Assert extends java.lang.Object

This class provides a set of assertion methods, useful for writing tests. Only failed assertions are recorded. Some of the important methods of Assert class are as follows −

|  |  |
| --- | --- |
| **Sr.No.** | **Methods & Description** |
| 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'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. |

Let's use some of the above-mentioned methods in an example. Create a java class file named **TestAssertions.java** in C:\>JUNIT\_WORKSPACE.

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);

}

}

Next, 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 TestRunner2 {

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());

}

}

**TestSuite**

**Test suite** is used to bundle a few unit test cases and run them together. In JUnit, both **@RunWith** and **@Suite** annotations are used to run the suite tests. This chapter takes an example having two test classes, **TestJunit1** & **TestJunit2**, that run together using Test Suite.

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;

}

// add "Hi!" to the message

public String salutationMessage(){

message = "Hi!" + message;

System.out.println(message);

return message;

}

}

Create Test Case Classes

Create a java class file named **TestJunit1.java** in C:\>JUNIT\_WORKSPACE.

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());

}

}

Create a java class file named **TestJunit2.java** in C:\>JUNIT\_WORKSPACE.

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());

}

}

Create Test Suite Class

* Create a java class.
* Attach @RunWith(Suite.class) Annotation with the class.
* Add reference to JUnit test classes using @Suite.SuiteClasses annotation.

Create a java class file named **TestSuite.java** in C:\>JUNIT\_WORKSPACE to execute test case(s).

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

@RunWith(Suite.class)

@Suite.SuiteClasses({

TestJunit1.class,

TestJunit2.class

})

public class JunitTestSuite {

}

Create Test Runner Class

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(JunitTestSuite.class);

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

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

}

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

}

}

Compile all the java classes using javac.