JUnit - Using Assertion

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());
   }
}
```

Compile the Test case and Test Runner classes using javac.

```
C:\JUNIT_WORKSPACE>javac TestAssertions.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.

```
true
```

Annotation

Annotations are like meta-tags that you can add to your code, and apply them to methods or in class. These annotations in JUnit provide the following information about test methods –

which methods are going to run before and after test methods.

which methods run before and after all the methods, and.

which methods or classes will be ignored during the execution.

The following table provides a list of annotations and their meaning in JUnit −

Sr.No.	Annotation & Description
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 java class file named **JunitAnnotation.java** in C:\>JUNIT_WORKSPACE to test annotation.

```
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");
   }
}
```

Next, create a java class file named **TestRunner.java** in C:\>JUNIT_WORKSPACE to execute annotations.

```
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(JunitAnnotation.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.

```
C:\JUNIT_WORKSPACE>javac JunitAnnotation.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.

```
in before class
in before
in test
in after
in after class
true
```