JUNIT – JUnit is a OpenSource Unit Testing FW for Java.
It is useful for Java Developers to write and run repeatable tests.
Erich Gamma and Kent Beck initially develop it. It is an instance of xUnit archi.
It is used for Unit Testing of a Small Chunk of Code.

Note: Developers who are following test-driven methodology must write and execute unit test first before any code. types of Testing ① Manual Testing
② Automated Testing.

features of Junit → 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 provides immediate feedback. No need to manually execute a report 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 green – test is running smoothly.
red – test fails.

Latest version of Junit is 5
19-Nov-2017

org.junit

Junit FW provides the following important features.
① Fixtures
② Test Suites
③ Test runners
④ Junit classes.

① Fixtures :- Test Fixtures is a content where a Test Case runs.
Includes → Objects @ resources that are available for any test case.
→ Activities required that makes these objects/resources available.
→ these activities are ① allocation (Setup).
② de-allocation (tear down).

Setup and Teardown :- There are some repeated tasks that must be done prior to each test case Ex:- Create a DB Connection.
→ Likewise at the end of each test case, there may be some repeated task Ex to clean up once test execution is over.
→ Junit Provides annotations that help in setup and teardown. It ensures that resources are released, and the test system is in a ready state for next test case.

Setup	Teardown
@Before : RUn before @Test, public void.	@After : Run after @Test; public void.
@BeforeClass: RUn once before any of the test methods in the class p s v	@AfterClass : Run once after all the tests in the class have been run p s v

Once-only Setup

Once-only-setup are useful for Starting Servers, Opening Communications. It's time-consuming to close and re-open resources for each test.

Ex: @BeforeClass p s v Method_Name(){
    // class setup code here
}

~~If Exception occurs~~

@After annotated methods are run even if any exceptions are thrown in the test case or in the case of assertion failure

Once-only-tear down

@AfterClass:-

It is useful for stopping Servers, Closing Communication lines, etc

@AfterClass P S V Method_Name(){
    // class cleanup code here.
}

~~If Exception occurs~~

@Before annotated methods Execution will be exit from the process.

Is it possible to write multiple ~~@After~~ @Before / @After annotated method?

Yes we can write multiple @Before / @After annotation methods

Order of Execution → It depends on the method Signature it will be in the Descending order.

Junit Test Suites

If we want to execute multiple tests in a Specified order, it can be done by Combining all the tests in one place. This place is called as the test suites.

→ @RunWith(Suite.class)

→ @SuiteClasses(test1.class, test2.class, ...) or

→ @Suite.SuiteClasses({test1.class, test2.class.....})

Ex:

```
P class Test1{

  @Test
  P v getTest1(){
     s.o.p("Test1");
}
```

```
P class Test2{

  @Test
  P v getTest2(){
     s.o.p("Test2");
}
```

TestSuiteclass

```
@RunWith(Suite.class)
@Suite.SuiteClasses({Test1.class,
                     Test2.class})
P class TestSuite{
// This class remains empty, it is used only
   as a holder for the above annotation.
}
```

Junit Test Runner

:- Junit provides a tool for execution of your test cases. TestRunner is used for executing the test cases.

→ JunitCore class is used to execute these tests.

org.junit.runner

Interface → Describable.

Classes → Description
JUnitCore
Request
Result
Runner.

AnnotationTypes → RunWith

→ A method called runClasses provided by org.junit.runner.JunitCore, is used to run one (or) several test classes.

Ex

```
static Result runClasses(java.lang.Class<?>...
                                    classes)
```

→ Result object (org.junit.runner.Result), which is used to access Information about the tests.

org.junit.runner
Class Result
extends java.lang.Object.

Methods

```
RunListner createListner();
int getFailureCount();
java.util.List<Failure> getFailures()
int getIgnoreCount();
int getRunCount();
```

```
long getRunTime()
boolean wasSuccessful ()
```

Ex :-

basic way to use to run Junit Test Runner

```
public class TestE
  P s v main(Str args[]){
Result rs = JunitCore.runClasses(Test1.class);
    for(Failure failure: rs.getFailures()){
        S.O.P(failure.toString());
    }
}
```

org.junit.runner

JunitCore

Methods :- void addListner(RunListner listner)

```
public class JUnitCore
extends java.lang.Object
```

`JUnitCore` is a facade for running tests. It supports running JUnit 4 tests, JUnit 3.8.x tests, and mixtures. To run tests from the command line, run `java org.junit.runner.JUnitCore TestClass1 TestClass2 ....` For one-shot test runs, use the static method `runClasses(Class[])`. If you want to add special listeners, create an instance of `JUnitCore` first and use it to run the tests.

See Also:

Result, RunListener, Request

Constructor Summary

`JUnitCore()` Create a new `JUnitCore` to run tests.

Method Summary

`void`	`addListener(RunListener listener)` Add a listener to be notified as the tests run.
`java.lang.String`	`getVersion()`
`static void`	`main(java.lang.String... args)` Run the tests contained in the classes named in the `args`.
`void`	`removeListener(RunListener listener)` Remove a listener.
`Result`	`run(java.lang.Class<?>... classes)` Run all the tests in `classes`.
`Result`	`run(Request request)` Run all the tests contained in `request`.
`Result`	`run(Runner runner)` Do not use.
`Result`	`run(junit.framework.Test test)` Run all the tests contained in JUnit 3.8.x `test`.
`static Result`	`runClasses(java.lang.Class<?>... classes)` Run the tests contained in `classes`.
`Result`	`runMain(org.junit.internal.JUnitSystem system, java.lang.String... args)` Do not use.
`static void`	`runMainAndExit(org.junit.internal.JUnitSystem system, java.lang.String... args)` Do not use.

Methods inherited from class java.lang.Object

nipulat
ficatio

org.junit.runner

Class Result

```
java.lang.Object
  └org.junit.runner.Result
```

```
public class Result
extends java.lang.Object
```

A `Result` collects and summarizes information from running multiple tests. Since tests are expected to run correctly, successful tests are only noted in the count of tests that ran.

Constructor Summary

`Result()`

Method Summary

`RunListener`	`createListener()` Internal use only.
`int`	`getFailureCount()`
`java.util.List<Failure>`	`getFailures()`
`int`	`getIgnoreCount()`
`int`	`getRunCount()`
`long`	`getRunTime()`
`boolean`	`wasSuccessful()`

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Result

```
public Result()
```

JUnit Test Classes

org.junit

Class

Assert →

Assume

Test.None

Assert → public class Assert extends java.lang.Object.

→ A set of assertion methods useful for writing tests.

→ In this class all the methods are static.

→ They read better if they are refferd through static import.

```
import static org.junit.Assert.*;
```