



# KIỂM THỬ NÂNG CAO

## BÀI 2: KIỂM THỬ ĐƠN VỊ

- ◎ JUnit Test framework
- ◎ JUnit Annotations & API




## Phần I: JUnit Test framework

-  Cấu trúc, tổ chức junit framework

-  Test fixture, Setup và Teardown

## Phần II: JUnit Annotations & API

-  Các lớp hỗ trợ junit

-  Annotations & API



- ❑ Hỗ trợ tạo nhanh test case và test data
- ❑ org.junit package gồm nhiều class và interface (Test fixture, Assert, After, Before...)
- ❑ Test môi trường web không cần server



- ❑ Đánh giá đoạn code viết 2 test case bên dưới (tính rõ ràng phân định giữa các hàm? Có dễ chỉnh sửa khi cần? Thứ tự thực thi các logic?)

```
public class OutputFileTest {  
    private File output;  
    output = new File(...);  
    output.delete();  
    public void testFile1(){  
        //Code to verify Test Case 1  
    }  
    output.delete();  
    output = new File(...);  
    public void testFile2(){  
        //Code to verify Test Case 2  
    }  
    output.delete();  
}
```

- ❑ So sánh với đoạn code dùng Junit (sv tìm khác biệt?)

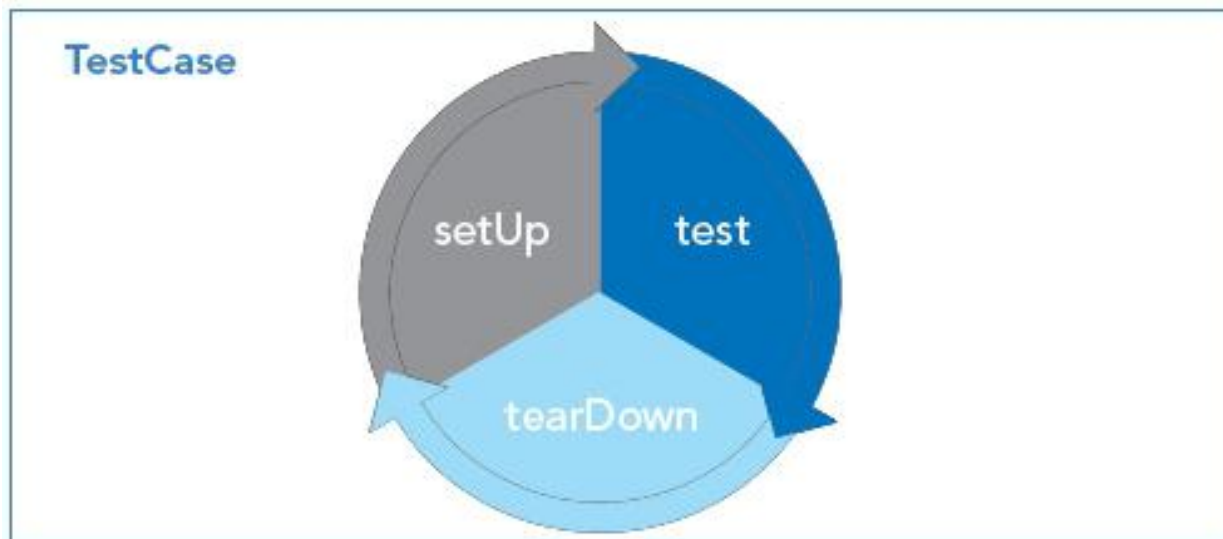
```
public class OutputFileTest
{
    private File output;
    @Before public void createOutputFile()
    {
        output = new File(...);
    }

    @After public void deleteOutputFile()
    {
        output.delete();
    }

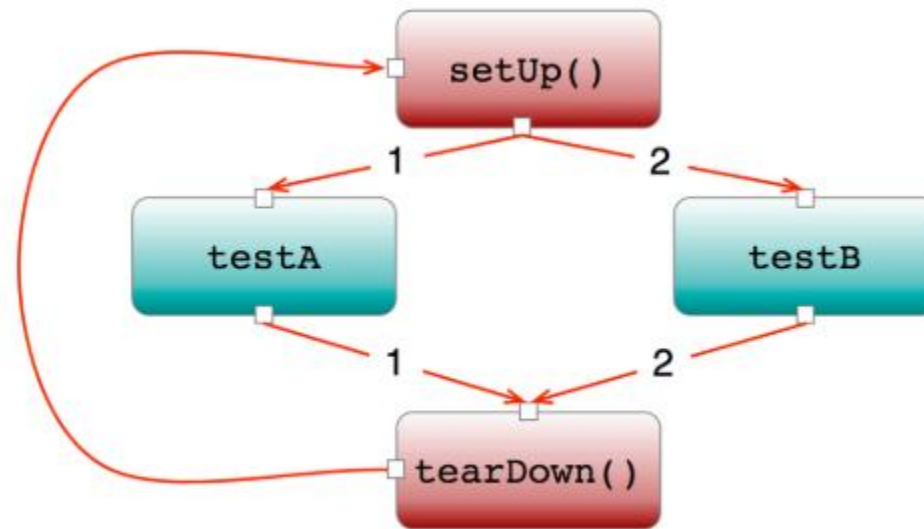
    @Test public void testFile1()
    {
        // code for test case objective
    }

    @Test public void testFile2()
    {
        // code for test case objective
    }
}
```

- ❑ Cấu trúc code của junit sử dụng một số annotations : @Before, @After...được gọi là Text fixture
- ❑ Text fixture quy định ngữ cảnh khi chạy test case thông qua khái niệm setup và teardown



- ❑ Một số phương thức cần được hoàn thành trước khi chạy test case, ví dụ tạo kết nối csdl
- ❑ Một số phương thức cần được hoàn thành sau khi chạy test case, ví dụ tắt kết nối csdl





❑ Cho biết thứ tự các phương thức thực thi ?

```
public class OutputFileTest
{
    private File output;
    @Before      public void createOutputFile()
    {
        output = new File(...);
    }

    @After public void deleteOutputFile()
    {
        output.delete();
    }

    @Test public void testFile1()
    {
        // code for test case objective
    }

    @Test public void testFile2()
    {
        // code for test case objective
    }
}
```

## □ Thứ tự các phương thức thực thi

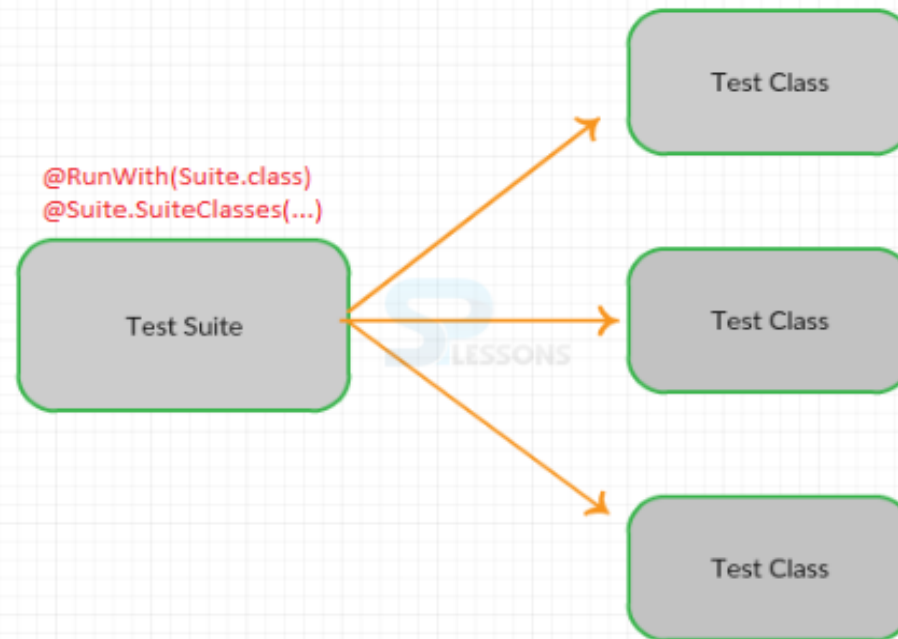
- ❖ `createOutputFile()`
- ❖ `testFile1()`
- ❖ `deleteOutputFile()`
- ❖ `createOutputFile()`
- ❖ `testFile2()`
- ❖ `deleteOutputFile()`



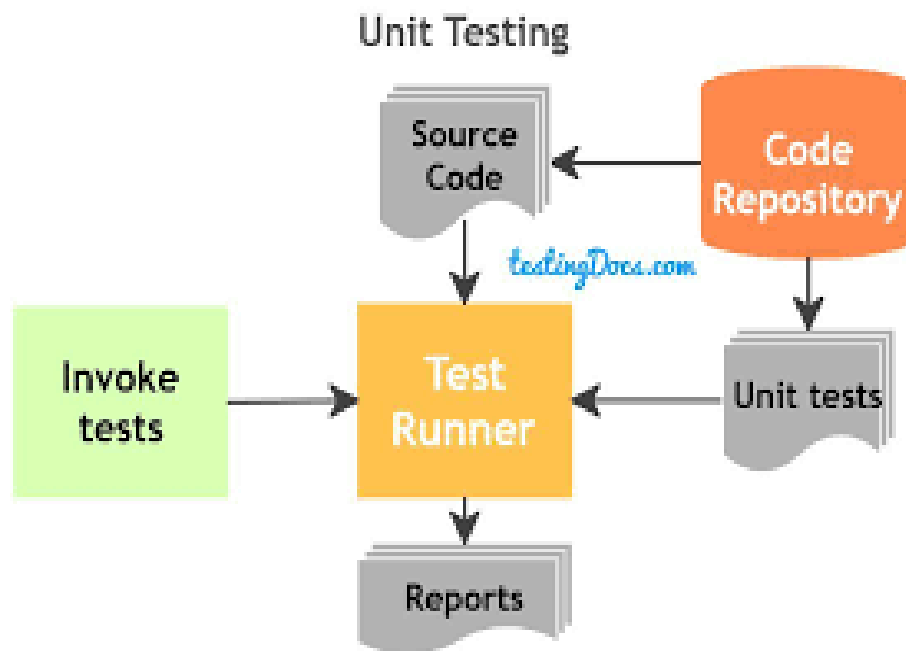
- ❑ @Before: Phương thức được đánh dấu annotation này sẽ được thực thi trước mỗi phương thức test.
- ❑ @After: Phương thức được đánh dấu annotation này sẽ được thực thi sau mỗi phương thức test.
- ❑ @BeforeClass: Phương thức được đánh dấu annotation này sẽ thực thi trước tất cả các phương thức test.
- ❑ @AfterClass: Phương thức được đánh dấu với annotation này sẽ được thực thi ngay sau khi tất cả các test đều được thực hiện.



- ❑ Test suites cho phép gom chạy cùng lúc nhiều test case có trình tự
- ❑ Để thực thi test suites, cần dùng các class:
  - ❖ `@RunWith(Suite.class)`
  - ❖ `@SuiteClasses(test1.class, test2.class.....)`  
or `@Suite.SuiteClasses ({test1.class, test2.class.....})`



- ❑ Junit cung cấp công cụ thực thi các test case là Junit Test Runner
- ❑ Cần sử dụng các thư viện `org.junit.runner.JUnitCore`, `org.junit.runner.Result`, `org.junit.runner.notification.Failure`;



## ❑ Tạo Test Runner Class

### ❖ Tạo lớp test tên MyFirstClassTest.java

```
package fpoly.JUnit;  
  
import static org.junit.Assert.*;  
  
import org.junit.Test;  
  
public class MyFirstClassTest {  
  
    @Test  
    public void myFirstMethod(){  
        String str= "Fpoly is working fine";  
  
        assertEquals("Fpoly is working fine",str);  
  
    }  
}
```

## ❑ Tạo Test Runner Class

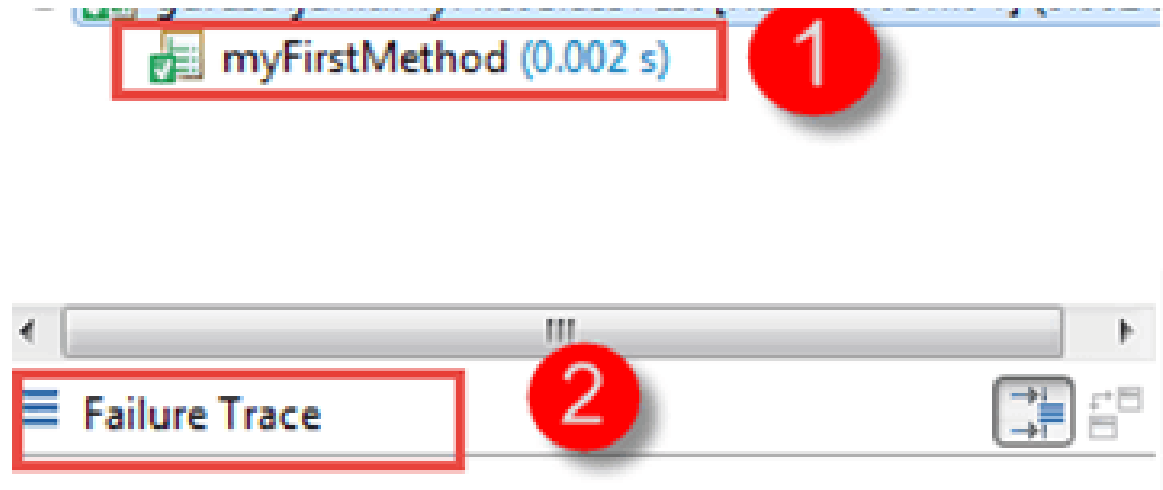
- ❖ Tạo lớp test runner thực thi test case tên TestRunner.java

```
package fpoly.JUnit;  
  
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(MyFirstClassTest.class);  
  
        for (Failure failure : result.getFailures()) {  
  
            System.out.println(failure.toString());  
  
        }  
        System.out.println("Result==" + result.wasSuccessful());  
  
    }  
}
```



## ❑ Tạo Test Runner Class

- ❖ Kết quả chạy test case thành công sẽ thông báo màu xanh, nếu thất bại là màu đỏ





# DEMO

- Tạo test suites với Setup và Teardown
- Chạy test suites





# **KIỂM THỬ NÂNG CAO**

## **BÀI 4: KIỂM THỬ ĐƠN VỊ (P2)**

- ❑ Junit Annotations là dạng đặc biệt được thêm vào code java làm cho cấu trúc rõ ràng hơn
- ❑ Variables, parameters, packages, methods và classes đều có thể dùng với annotation

## JUnit Annotations

**@BeforeClass**      **@Before**

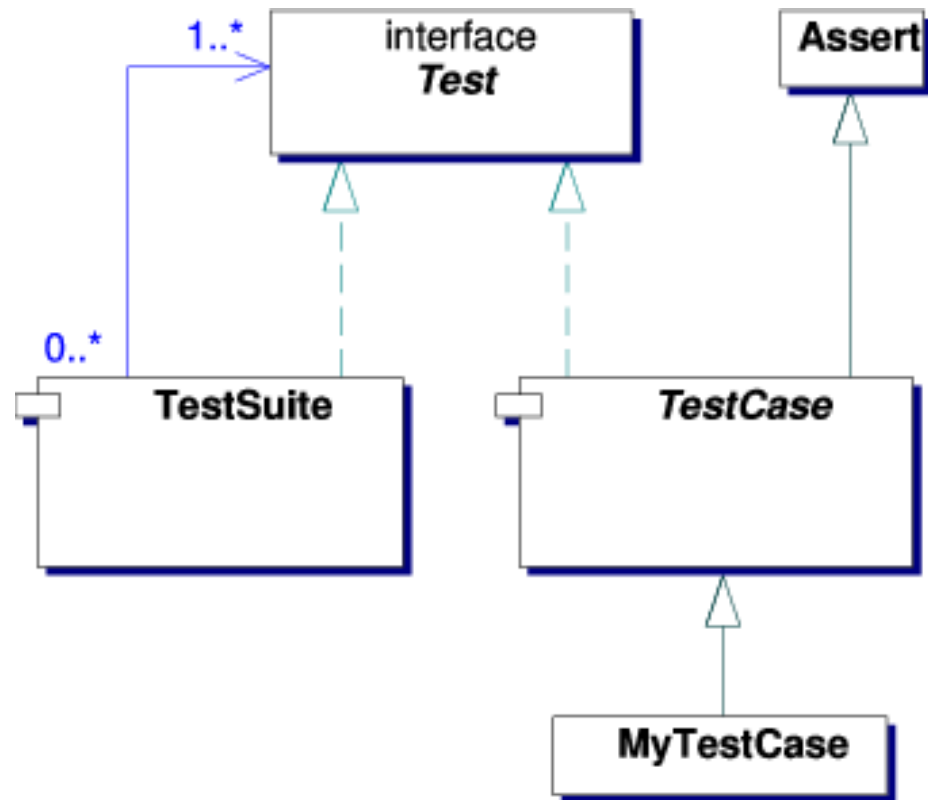
**@Test**

**@After**      **@AfterClass**

**@RunWith**      **@Parameters**

## ❑ Các class hữu ích cho việc viết test case

- ❖ JUnit Assert Class
- ❖ JUnit Test Cases Class
- ❖ JUnit TestResult Class
- ❖ JUnit Test Suite Class
- ❖ JUnit Annotations



## JUnit Assert Class

S.No.	Method	Description
1.	<code>void assertEquals(boolean expected, boolean actual)</code>	It checks whether two values are equals similar to equals method of Object class
2.	<code>void assertFalse(boolean condition)</code>	functionality is to check that a condition is false.
3.	<code>void assertNotNull(Object object)</code>	"assertNotNull" functionality is to check that an object is not null.
4.	<code>void assertNull(Object object)</code>	"assertNull" functionality is to check that an object is null.
5.	<code>void assertTrue(boolean condition)</code>	"assertTrue" functionality is to check that a condition is true.
6.	<code>void fail()</code>	If you want to throw any assertion error, you have fail() that always results in a fail verdict.
7.	<code>void assertSame([String message])</code>	"assertSame" functionality is to check that the two objects refer to the same object.
8.	<code>void assertNotSame([String message])</code>	"assertNotSame" functionality is to check that the two objects do not refer to the same object.

## JUnit Test Cases Class

S.No.	Method	Description
1.	int countTestCases()	This method is used to count how many number of test cases executed by <b>run(TestResult tr)</b> method.
2.	TestResult createResult()	This method is used to create a <b>TestResult</b> object.
3.	String getName()	This method returns a string which is nothing but a <b>TestCase</b> .
4.	TestResult run()	This method is used to execute a test which returns a <b>TestResult</b> object
5.	void run(TestResult result)	This method is used to execute a test having a <b>TestResult</b> object which doesn't returns anything.
6.	void setName(String name)	This method is used to set a name of a <b>TestCase</b> .
7.	void setUp()	This method is used to write resource association code. e.g. Create a database connection.
8.	void tearDown()	This method is used to write resource release code. e.g. Release database connection after performing transaction operation.

## JUnit TestResult Class

S.No.	Method	Description
1.	<code>void addError(Test test, Throwable t)</code>	This method is used if you require add an error to the test.
2.	<code>void addFailure(Test test, AssertionFailedError t)</code>	This method is used if you require add a failure to the list of failures.
3.	<code>void endTest(Test test)</code>	This method is used to notify that a test is performed(completed)
4.	<code>int errorCount()</code>	This method is used to get the error detected during test execution.
5.	<code>Enumeration&lt;TestFailure&gt; errors()</code>	This method simply returns a collection (Enumeration here) of errors.
6.	<code>int failureCount()</code>	This method is used to get the count of errors detected during test execution.
7.	<code>void run(TestCase test)</code>	This method is used to execute a test case.
8.	<code>int runCount()</code>	This method simply counts the executed test.
9.	<code>void startTest(Test test)</code>	This method is used to notify that a test is started.
10.	<code>void stop()</code>	This method is used to test run to be stopped.



## JUnit Test Suite Class

S.No.	Method	Description
1.	<code>void addTest(Test test)</code>	This method is used if you want to add a test to the suite.
2.	<code>void addTestSuite(Class&lt;? extends TestCase&gt; testClass)</code>	This method is used if you want to specify the class while adding a test to the suite.
3.	<code>int countTestCases()</code>	This method is used if you want to count the number of test cases.
4.	<code>String getName()</code>	This method is used to get the name of the test suite.
5.	<code>void run(TestResult result)</code>	This method is used to execute a test and collect test result in <b>TestResult</b> object.
6.	<code>void setName(String name)</code>	This method is used to set the name of <b>TestSuite</b> .
7.	<code>Test testAt(int index)</code>	This method is used if you want to return the test at given index.
8.	<code>int testCount()</code>	This method is used if you want to return a number of tests in the Suite.
9.	<code>static Test warning(String message)</code>	This method returns a test which will fail and log a warning message.

## JUnit Annotations

S.No.	Annotations	Description
1.	@Test	This annotation is a replacement of org.junit.TestCase which indicates that public void method to which it is attached can be executed as a test Case.
2.	@Before	This annotation is used if you want to execute some statement such as preconditions before each test case.
3.	@BeforeClass	This annotation is used if you want to execute some statements before all the test cases for e.g. test connection must be executed before all the test cases.
4.	@After	This annotation can be used if you want to execute some statements after each Test Case for e.g resetting variables, deleting temporary files ,variables, etc.

## JUnit Annotations

5. `@AfterClass`

This annotation can be used if you want to execute some statements after all test cases for e.g. Releasing resources after executing all test cases.

6. `@Ignores`

This annotation can be used if you want to ignore some statements during test execution for e.g. disabling some test cases during test execution.

7. `@Test(timeout=500)`

This annotation can be used if you want to set some timeout during test execution for e.g. if you are working under some SLA (Service level agreement), and tests need to be completed within some specified time.

8. `@Test(expected=IllegalArgumentException.class)`

This annotation can be used if you want to handle some exception during test execution. For, e.g., if you want to check whether a particular method is throwing specified exception or not.


# Tổng kết bài học

## Phần I: JUnit Test framework

-  Cấu trúc, tổ chức junit framework

-  Test fixture, Setup và Teardown

## Phần II: JUnit Annotations & API

-  Các lớp hỗ trợ junit

-  Annotations & API





**KẾT THÚC**