MỤC TIÊU:

Kết thúc bài thực hành này bạn có khả năng

- ✓ Hiểu được các khái niệm Test Fixture,tổ chức cấu trúc test case với test fixture
- ✓ Hiểu được các khái niệm Test Suites, tạo và thực thi test suites
- ✓ Vận dụng linh hoạt các annotation, class trong junit

PHẦN I

Bài 1 (2 điểm)

Sinh viên thực hiện xây dựng 1 chương trình kiểm thử ứng dụng có 2 phương thức là tính giai thừa một số nguyên không âm và tính tổng 2 số nguyên, có sử dụng test fixture

- ➢ Tạo class có tên MathFunc, xây dưng 2 phương thức factorial và plus:

```
public class MathFunc {
    int calls;

    public int getCalls() {
        return calls;
    }

    public long factorial(int number) {
        calls++;

        if (number < 0)
            throw new IllegalArgumentException();

        long result = 1;
        if (number > 1) {
            for (int i = 1; i <= number; i++)
                result = result * i;
        }

        return result;
    }

    public long plus(int num1, int num2) {
        calls++;
        return num1 + num2;
    }
}
```

- Tạo junit test case, tạo class có tên MathFuncTest chứa các test case cần thiết để test 2 phương thức bên trên.
- Sử dụng " @Before và @After " quy định phương thức chạy trước và sau các test case.

```
@Before
public void init() { math = new MathFunc(); }
@After
public void tearDown() { math = null; }
```

- Dùng annotation @Ignore để bỏ qua một test case

```
@Ignore
@Test
public void todo() {
    assertTrue(math.plus(1, 1) == 3);
}
```

- Class MathFuncTest được code như sau:

```
public class MathFuncTest {
    private MathFunc math;

    @Before
    public void init() { math = new MathFunc(); }
    @After
    public void tearDown() { math = null; }

    @Test
    public void calls() {
        assertEquals(0, math.getCalls());

        math.factorial(1);
        assertEquals(1, math.getCalls());

        math.factorial(1);
        assertEquals(2, math.getCalls());
    }
```

```
    @Test
    public void factorial() {
        assertTrue(math.factorial(0) == 1);
        assertTrue(math.factorial(1) == 1);
        assertTrue(math.factorial(5) == 120);
    }

    @Test(expected = IllegalArgumentException.class)
    public void factorialNegative() {
        math.factorial(-1);
    }

    @Ignore
    @Test
    public void todo() {
        assertTrue(math.plus(1, 1) == 3);
    }
}
```

- Sinh viên chú ý :
  - Phương thức public void calls() kiểm tra so sanh giá trị của biến "calls"
  - Phương thức public void factorial() kiểm tra phương thức tính giai thừa
  - Phương thức factorialNegative() kiểm tra trường hợp số âm
  - Phương thức public void todo() kiểm tra phướng thức tính tổng
- Tạo class thực thi có chưa main như sau:

```
public static void main(String[] args) throws Exception {
    JUnitCore runner = new JUnitCore();
    Result result = runner.run(MathFuncTest.class);
    System.out.println("run tests: " + result.getRunCount());
    System.out.println("failed tests: " + result.getFailureCount());
    System.out.println("ignored tests: " + result.getIgnoreCount());
    System.out.println("success: " + result.wasSuccessful());
}
```

- Chạy kiểm tra kết quả:

```
run tests: 3
failed tests: 0
ignored tests: 1
success: true
```

Bài 2 (3 điểm)

Tạo JUnit Test Suite sử dụng @RunWith @Suite. Giả sử có 2 lớp SuiteTest1.java và SuiteTest2.java, mỗi class chứa những test case khác nhau.

- Tạo class SuiteTest1.java với mục đích xuất ra một chuỗi ký tự

```
package fpoly.junit;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class SuiteTest1 {

    public String message = "Fpoly";


    JUnitMessage junitMessage = new JUnitMessage(message);


    @Test(expected = ArithmeticException.class)

    public void testJUnitMessage() {

        System.out.println("Junit Message is printing ");

        junitMessage.printMessage();

    }
```

```
    @Test
    public void testJUnitHiMessage() {
        message = "Hi!" + message;

        System.out.println("Junit Hi Message is printing ");

        assertEquals(message, junitMessage.printHiMessage());

        System.out.println("Suite Test 2 is successful " + message);

    }
}
```

- Tương tự tạo class SuiteTest2.java

```
package fpoly.junit;

import org.junit.Assert;
import org.junit.Test;

public class SuiteTest2 {


    @Test
    public void createAndSetName() {


        String expected = "Y";
        String actual = "Y";

        Assert.assertEquals(expected, actual);

        System.out.println("Suite Test 1 is successful " + actual);

    }

}
```

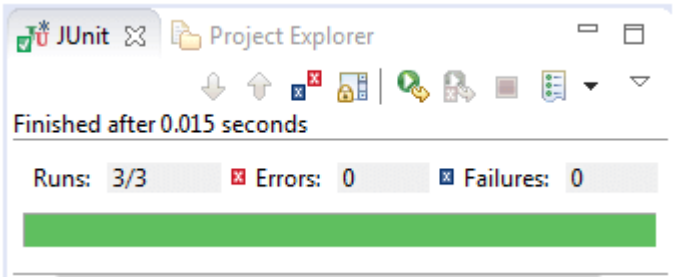
- Tạo class JunitTest.java để thực thi 2 lớp Suitest bên trên

```
package fpoly.junit;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
  SuiteTest1.class,
  SuiteTest2.class,
})

public class JunitTest {
                        // This class remains empty, it is used only as a hold
er for the above annotations
}
```

Chạy test và kiểm tra kết quả



Bài 3 (3 điểm)

Xây dựng lớp kiểm thử với JUnit Annotations: @Before , @BeforeClass, @After, @AfterClass, @Test, @Ignore…

- Tạo lớp kiểm thử JunitAnnotationsExample.java

```
package fpoly.junit;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;

import java.util.ArrayList;

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 JunitAnnotationsExample {

    private ArrayList<String> list;

    @BeforeClass
    public static void m1() {
        System.out.println("Using @BeforeClass , executed before all test cases ");

    }
```

```
    @Before
    public void m2() {
        list = new ArrayList<String>();
        System.out.println("Using @Before annotations ,executed before each test cases ");

    }
```

```
    @AfterClass
    public static void m3() {
        System.out.println("Using @AfterClass ,executed after all test cases");

    }

    @After
    public void m4() {
        list.clear();
        System.out.println("Using @After ,executed after each test cases");

    }

    @Test
    public void m5() {
        list.add("test");
        assertFalse(list.isEmpty());
        assertEquals(1, list.size());
    }

    @Ignore
    public void m6() {
        System.out.println("Using @Ignore , this execution is ignored");

    }
```

```
    @Test(timeout = 10)
    public void m7() {
        System.out.println("Using @Test(timeout),it can be used to enforce timeout in JUnit4 test c
ase");
    }

    @Test(expected = NoSuchMethodException.class)
    public void m8() {
        System.out.println("Using @Test(expected) ,it will check for specified exception during its
execution");

    }

}
```

- Tạo lớp TestRunner.java thực thi đoạn test case bên trên

```
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(JunitAnnotationsExample.class);

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

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

      }
      System.out.println("Result=="+result.wasSuccessful());

   }
}
```

Chạy và kiểm tra kết quả

Bài 4 (2 điểm)

Giảng viên cho thêm