



# KIỂM THỬ NÂNG CAO

## BÀI 3: KIỂM THU ĐƠN VỊ (P3)

## Nội dung bài học

- JUnit Exception Test
- JUnit ErrorCollector
- JUnit Parameterized Test




## Phần I: JUnit Exception Test

 Khái niệm ngoại lệ trong Junit

 Kỹ thuật bắt ngoại lệ với junit

## Phần II: JUnit ErrorCollector

 Khái niệm JUnit ErrorCollector

 Kỹ thuật xử lý, quản lý error trong junit

## Phần III:JUnit Parameterized Test

 Khái niệm Parameters trong Junit

 Kỹ thuật dùng Parameters



- ❑ Tình huống phương thức bị rơi vào ngoại lệ hoặc phương thức không trả giá trị về.
- ❑ Junit hỗ trợ theo dõi các ngoại lệ trong quá trình unit test
- ❑ Sử dụng tham số "expected" và phương thức "fail()"



- ❑ @Test, optional 'expected' attribute

```
@Test(expected = ArithmeticException.class)
```

- ❑ Try-catch và fail()

```
try {  
    new ArrayList<>().get(0);  
    fail();  
} catch (IndexOutOfBoundsException e) {  
    assertThat(e.getMessage(), is("Index: 0, Size: 0"));  
}
```

- ❑ @Rule ExpectedException

```
@Rule  
public ExpectedException thrown = ExpectedException.none();
```

- ❑ Ví dụ dùng @Test, optional 'expected' attribute
  - ❖ Tạo class tên divzero

```
public class divzero{  
    public static int divide(int input1, int input2 ) throws Exception {  
        if (input2 == 0) {  
            throw new ArithmeticException("divide by zero");  
        }  
        return input1/input2;  
    }  
}
```

- ❑ Ví dụ dùng @Test, optional 'expected' attribute
  - ❖ Trường hợp input2 = 0 thì xảy ra exception chứ không trả về kết quả.
  - ❖ Tạo expect exception :

```
@Test(expected = ArithmeticException.class)
public void testMathUtils1() throws Exception {
    MathUtils.divide(100, 0);
}
```

- ❖ MathUtils.divide(100, 0); không xảy ra exception ArithmeticException thì tức là test case fail.

- ❑ Ví dụ dùng Try-catch và always fail()
  - ❖ Hỗ trợ thông báo rõ nghĩa khi có ngoại lệ

```
@Test
public void testtrycach() throws Exception {
    try {
        MathUtils.divide(100, 0);
        fail("Not throw exception");
    } catch (Exception e) {
        assertThat(e, instanceof(ArithmeticException.class));
        assertEquals(e.getMessage(), "divide by zero");
    }
}
```

- ❖ Test case thất bại khi lệnh "fail("Not throw exception");" thực thi.



- ❑ Ví dụ dùng Try-catch và always fail()
  - ❖ Nếu không xảy ra exception thì lệnh fail() không được thực thi tức là test case pass.

```
@Test
public void testtrycach() throws Exception {
    try {
        MathUtils.divide(1, 1);
    } catch (Exception e) {
        fail("throw exception");
    }
}
```

## ❑ Ví dụ dùng ExpectedException Rule

- ❖ Hỗ trợ xác định loại exception và message exception

```
@Rule
public ExpectedException thrown = ExpectedException.none();

@Test
public void shouldTestExceptionMessage() throws Exception{
    thrown.expect(ArithmeticException.class);
    thrown.expectMessage("divide by zero");
    MathUtils.divide(1, 0);
}
```

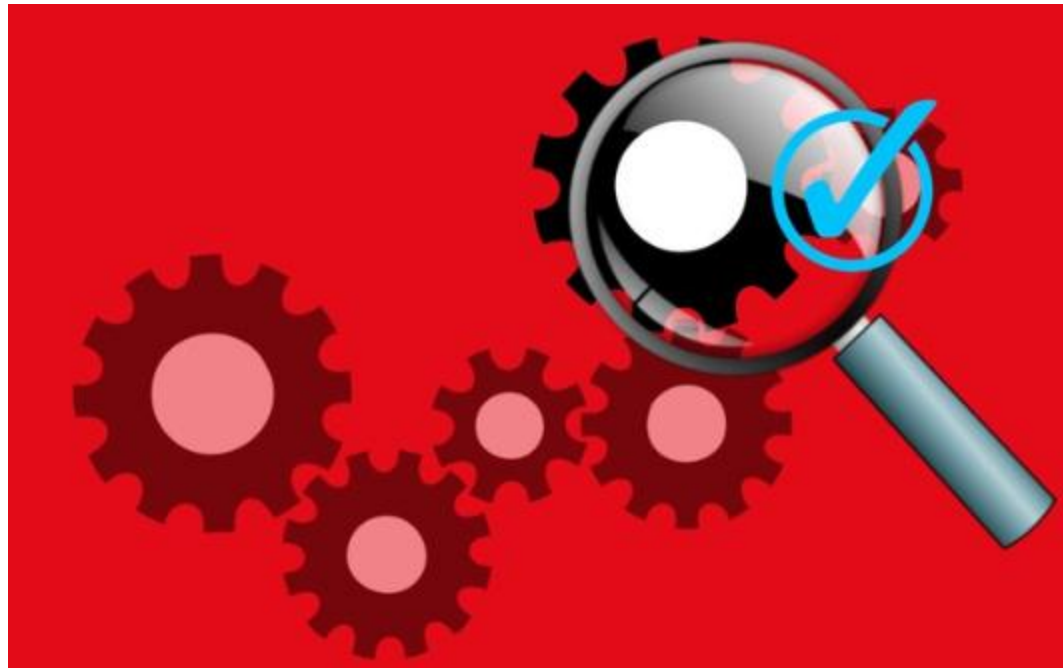


# DEMO

- Demo junit với Expected Exception



- ❑ Thông thường quá trình test gặp sự cố thì sẽ dừng, fix xong thì chạy lại test.
- ❑ Junit cho phép quá trình test diễn ra tiếp tục nhưng đồng thời thu thập các error, đưa ra báo cáo sau khi hoàn thành test



## ❑ Tính năng của ErrorCollector

- ❖ Cho phép kịch bản test tiếp tục chạy khi gặp sự cố
- ❖ Junit sử dụng @Rule annotation tạo đối tượng ErrorCollector
- ❖ Dùng Throwable thu thập error

### ▶ The ErrorCollector Rule

- ▶ Report on multiple error conditions in a single test

```
public class ErrorCollectorTest {  
  
    @Rule  
    public ErrorCollector collector = new ErrorCollector();  
  
    @Test  
    public void testSomething() {  
        collector.addError(new Throwable("first thing went wrong"));  
        collector.addError(new Throwable("second thing went wrong"));  
        String result = doStuff();  
        collector.checkThat(result, not(containsString("Oh no, not again")));  
    }  
  
    private String doStuff() {  
        return "Oh no, not again";  
    }  
}
```

Two things went wrong here

Check using Hamcrest matchers



- ❑ Sử dụng @rule để viết thêm hoặc thay đổi hành vi phương thức test

```
@Rule  
public ErrorCollector collector= new ErrorCollector();
```

- ❑ Ví dụ sử dụng ErrorCollector
  - ❖ Tạo lớp ErrorCollectorExample.java để gom error bằng cách dùng @rule và addError(throwable)

```
package fpoly.junit;

import org.junit.Assert;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ErrorCollector;

public class ErrorCollectorExample {
    @Rule
    public ErrorCollector collector = new ErrorCollector();

    @Test
    public void example() {
        collector.addError(new Throwable("There is an error in first line"));

        collector.addError(new Throwable("There is an error in second line"));

        System.out.println("Hello");
        try {
            Assert.assertTrue("A " == "B");
        } catch (Throwable t) {
            collector.addError(t);
        }
        System.out.println("World!!!!");
    }
}
```

## ❑ Tạo lớp TestRunner.java thực thi :

```
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(ErrorCollectorExample.class);

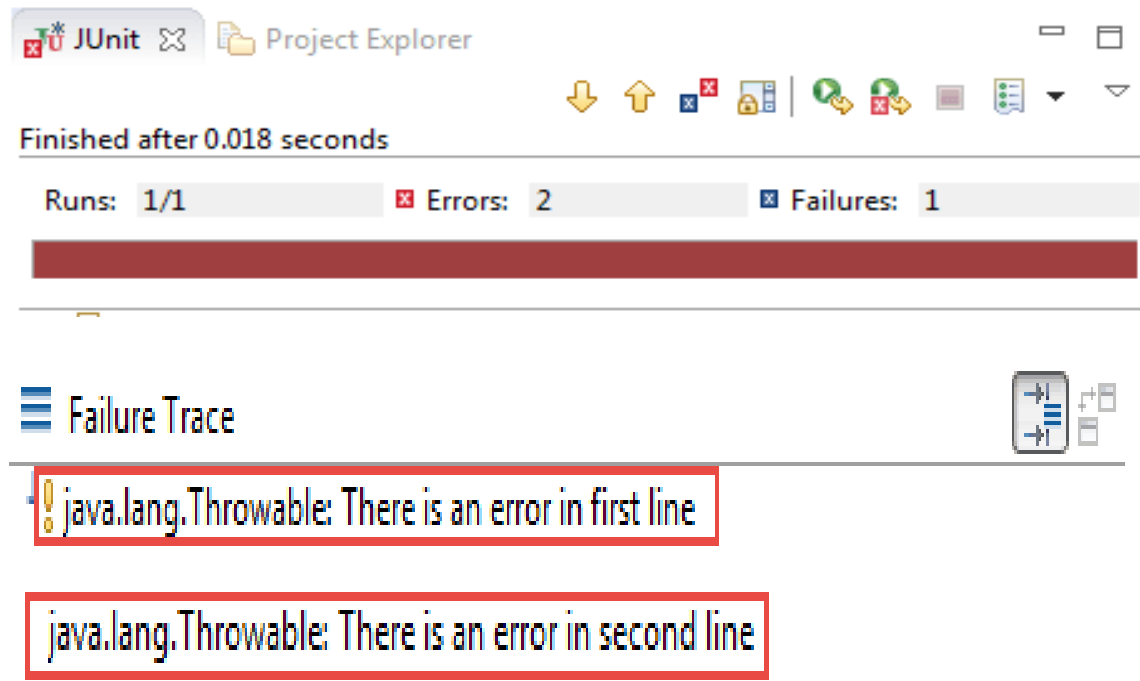
        for (Failure failure : result.getFailures()) {

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

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

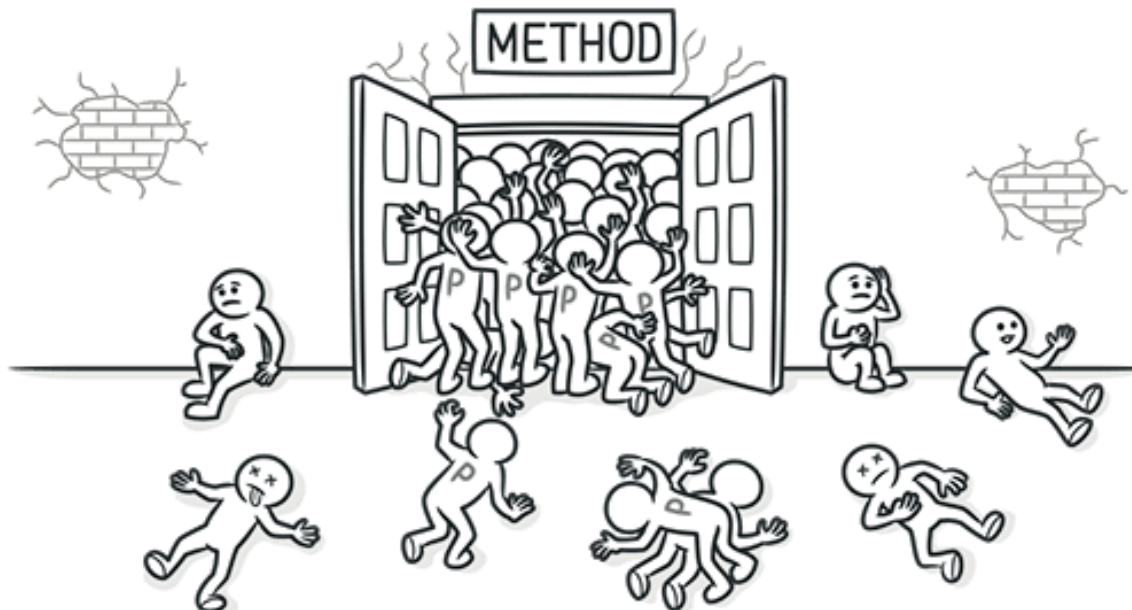


- ❑ Dễ dàng nhận thấy "A" không thể giống "B" nên kết quả có test case fail





- ❑ Thực thi cùng một test case với nhiều giá trị khác nhau
- ❑ Truy vấn dữ liệu từ nhiều nguồn
- ❑ Dùng Constructor và @Parameter



## ❑ Ví dụ tạo Parameterized JUnit test

### ❖ Tạo class để tính tổng 2 số nguyên

```
1 package junitTutorial;  
2  
3 public class Airthematic {  
4     public int sum(int a,int b){  
5         return a+b;  
6     }  
7  
8 }
```

### ❖ Tạo lớp chứa "Parameterized Test" để test

```
11 @RunWith(Parameterized.class)  
12 public class AirthematicTest {  
13     private int firstNumber;  
14     private int secondNumber;  
15     private int expectedResult;  
16     private Airthematic airthematic;  
17 }
```

## ❑ Ví dụ tạo Parameterized JUnit test

❖ @RunWith annotation xác định thực thi lớp test

```
11 @RunWith(Parameterized.class)
12 public class AirthematicTest {
13     private int firstNumber;
14     private int secondNumber;
15     private int expectedResult;
16     private Airthematic airthematic;
17
```

## ❑ Ví dụ tạo Parameterized JUnit test

❖ Tạo constructor khởi tạo giá trị

```
17  
18 public AirthematicTest(int firstNumber, int secondNumber, int expectedResult) {  
19     super();  
20     this.firstNumber = firstNumber;  
21     this.secondNumber = secondNumber;  
22     this.expectedResult = expectedResult;  
23 }  
24
```

## ❑ Ví dụ tạo Parameterized JUnit test

- ❖ Sử dụng phương thức static để gán giá trị và trả giá trị về
- ❖ Sử dụng mảng 2 chiều và @Parameters annotation

```
30 @Parameterized.Parameters
31 public static Collection input() {
32     return Arrays.asList(new Object[][] { { 1, 2, 3 }, { 11, 22, 33 },
33         { 111, 222, 333 }, { 10, 9, 19 }, { 100, 9, 109 } });
34 }
35
```

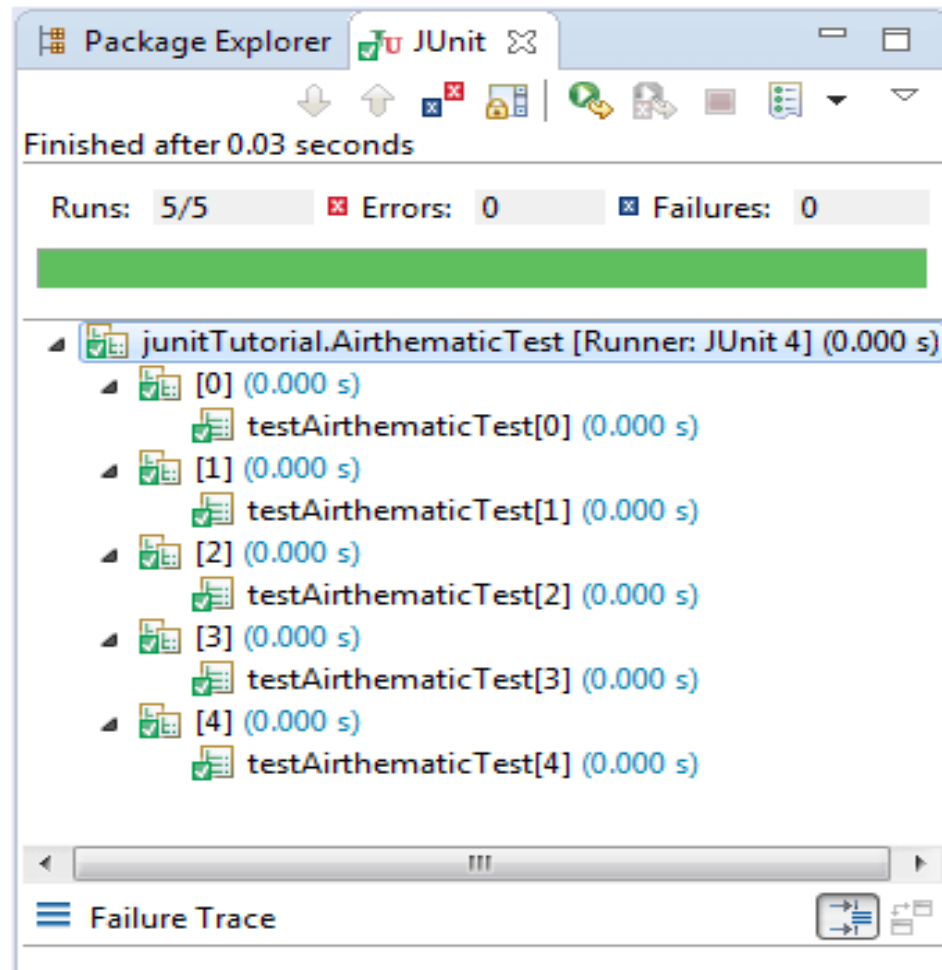
## ❑ Ví dụ tạo Parameterized JUnit test

❖ Tạo lớp test runner để thực thi lớp parameterized test

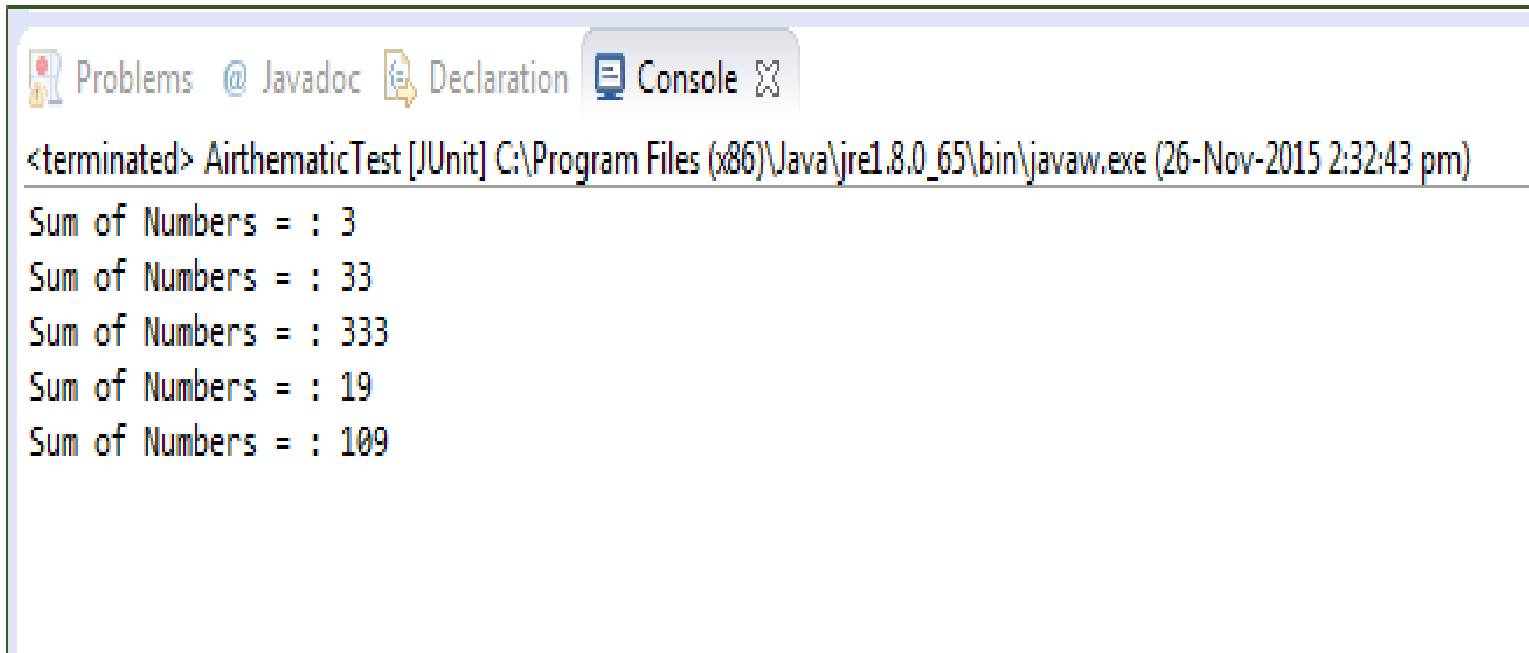
```
1 package junitTutorial;
2
3 import org.junit.runner.JUnitCore;
4 import org.junit.runner.Result;
5 import org.junit.runner.notification.Failure;
6
7 public class Test {
8     public static void main(String[] args) {
9         Result result = JUnitCore.runClasses(AirthematicTest.class);
10        for (Failure failure : result.getFailures()) {
11            System.out.println(failure.toString());
12        }
13        System.out.println(result.wasSuccessful());
14    }
15 }
```



- ❑ Ví dụ tạo Parameterized JUnit test
  - ❖ Kết quả chạy test class



- ❑ Ví dụ tạo Parameterized JUnit test
  - ❖ Kết quả hiển thị trên output



```
<terminated> AirthematicTest [JUnit] C:\Program Files (x86)\Java\jre1.8.0_65\bin\javaw.exe (26-Nov-2015 2:32:43 pm)
Sum of Numbers = : 3
Sum of Numbers = : 33
Sum of Numbers = : 333
Sum of Numbers = : 19
Sum of Numbers = : 109
```



# DEMO

- Demo junit với Parameterized



# Tổng kết bài học


## Phần I: JUnit Exception Test

 Khái niệm ngoại lệ trong Junit

 Kỹ thuật bắt ngoại lệ với junit

## Phần II: JUnit ErrorCollector

 Khái niệm JUnit ErrorCollector

 Kỹ thuật xử lý, quản lý error trong junit

## Phần III: JUnit Parameterized Test

 Khái niệm Parameters trong Junit

 Kỹ thuật dùng Parameters





**KẾT THÚC**