

Day 7 Exercises

Exercise 1

Time: 10 mins.

1. Open Spring tool suite (Eclipse) and open our player application
2. Go to test source code (src/test/java) and expand the package with the test file.
3. Open the test
4. Run the test.

Exercise 2

Time: 30 mins.

Creating tests

1. Make the test fail.

Note: Look that we included an import static line on the code, import static is not common but is very usefull when you don't want to write (in this case) `Assertion.fail` ("reason to fail") too many times you can try to use this solution for any static method you want to use.

Answer

```
package com.beisbolicos.playerStats;

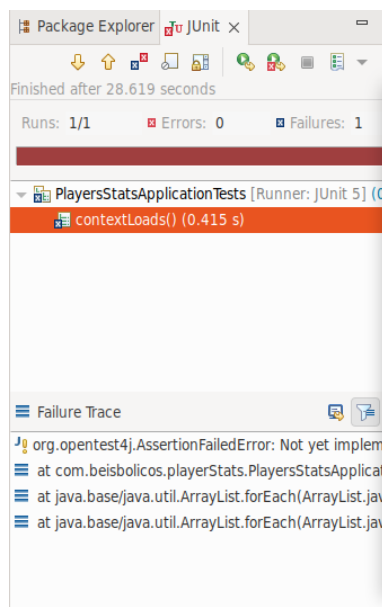
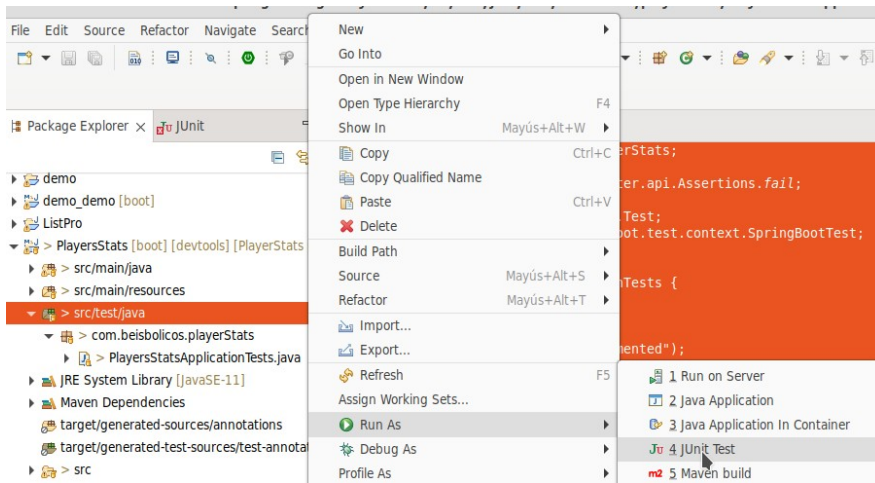
import static org.junit.jupiter.api.Assertions.fail;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class PlayersStatsApplicationTests {

    @Test
    void contextLoads() {
        fail("Not yet implemented");
    }

}
```



Exercise 3

Time 15 minutes.

Given the class:

package main.java;

```
public class MySimpleMath {
    /**
     * A simple method that takes an input and returns
     * "positive" or "negative" depending on the input number
     */
    public String checkSign(int number) {
        if (number >= 0) {
            return "positive";
        } else {
            return "negative";
        }
    }
}
```

```
}
```

create a testing class with two methods, one to check numbers as positives and other to check numbers as negative.

Execute testing.

Answer

Create a new “Maven project” on Eclipse IDE, set the MySimpleMath class on it.

Create the following test class:

```
package test.main;

import org.junit.*;
import main.java.MySimpleMath;

public class MySimpleMathTest {

    @Test
    public void testCheckSignShouldReturnPositive() {
        MySimpleMath sm = new MySimpleMath();
        Assert.assertEquals("positive", sm.checkSign(5));
        Assert.assertEquals("positive", sm.checkSign(0));
    }

    @Test
    public void testCheckSignShouldReturnNegative() {
        MySimpleMath sm = new MySimpleMath();
        Assert.assertEquals("negative", sm.checkSign(-5));
    }

}
```

Execute tests

Exercise 4

Time 15 mins

Add a division method to the MySimpleMath class:

```
/**
 * Returns the division of numerator by the denominator.
 * If the denominator is zero, it throws an Exception
 */
public double divide(int num, int denom) {
    if(denom == 0) {
        throw new ArithmeticException("Cannot divide by zero");
    } else {
        return num/(double)denom;
    }
}
```

```
}
```

Now add testing components for division. Include a part to validate an ArithmeticException.

Answer:

```
package com.beisbolicos.testing;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

import org.junit.jupiter.api.Test;

public class AppTest {
    @Test
    public void testCheckSignShouldReturnPositive() {
        MySimpleMath sm = new MySimpleMath();
        assertEquals("positive", sm.checkSign(5));
        assertEquals("positive", sm.checkSign(0));
    }

    @Test
    public void testCheckSignShouldReturnNegative() {
        MySimpleMath sm = new MySimpleMath();
        assertEquals("negative", sm.checkSign(-5));
    }

    @Test
    public void testDivisionShouldReturnPositiveQuotient() {
        MySimpleMath sm = new MySimpleMath();
        assertEquals(2.0, sm.divide(10, 5), 0);
        assertEquals(0.0, sm.divide(0, 5), 0);
    }

    @Test
    public void testDivisionShouldReturnNegativeQuotient() {
        MySimpleMath sm = new MySimpleMath();
        assertEquals(-2.0, sm.divide(10, -5), 0);
    }

    @Test
    public void testDivisionShouldThrowArithmeticException() {
        MySimpleMath sm = new MySimpleMath();
        assertThrows(ArithmeticException.class, () -> sm.divide(5, 0));
    }
}
```

Exercise 5

Create a class for multiplying arrays and found the minium

```
package com.beisbolicos.testing;

public class MySimpleArrayOperations {

    public int findMin(int[] array) {
```

```

        if(!(array.length > 0)) {
            throw new IllegalArgumentException("Input array is empty");
        }

        int min = Integer.MAX_VALUE;
        for(int i=0; i<array.length; i++) {
            if(array[i] <= min)
                min = array[i];
        }

        return min;
    }

    public void multiply(int[] array, int factor) {
        if(!(array.length > 0)) {
            throw new IllegalArgumentException("Input array is empty");
        }

        for( int i=0; i<array.length; i++ ) {
            array[i] = array[i] * factor;
        }
    }
}

```

Now test the class with data prepared before the test executions

Answer

```

package com.beisbolicos.testing;

import static org.junit.jupiter.api.Assertions.assertArrayEquals;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

public class MySimpleArrayOperationsTest {
    private static MySimpleArrayOperations msao = new MySimpleArrayOperations();
    private static int[] array;

    @BeforeAll
    public static void initInstanceVariables() {
        //System.out.println(this.getClass().getName() + " --> initializing
fields");
        msao = new MySimpleArrayOperations();
        array = new int[] {10, 2, 3, 10, 1, 0, 2, 3, 16, 0, 2};
    }

    @Test
    public void testFindMin() {
        assertEquals(0, msao.findMin(array));
        assertNotEquals(10, msao.findMin(array));
    }

    @Test

```

```

    public void testFindMinShouldThrowException() {
        assertThrows(IllegalArgumentException.class, ()->msao.findMin(new int[]
    {}));
    }

    @Test
    public void testMultiply() {
        msao.multiply(array, 10);
        assertEquals(new int[]{100, 20, 30, 100, 10, 0, 20, 30, 160, 0, 20},
array);
    }

    @Test
    public void testMultiplyShouldThrowException() {
        assertThrows(IllegalArgumentException.class, ()->msao.multiply(new int[]
    {}, 0)); //method call with dummy arguments
    }

}

```

Exercise 6

In a new project generate a calculator service like:

```

public interface CalculatorService {
    public double add(double input1, double input2);
    public double subtract(double input1, double input2);
    public double multiply(double input1, double input2);
    public double divide(double input1, double input2);
}

```

Then implement it:

```

public class MathApplication {
    private CalculatorService calcService;

    public void setCalculatorService(CalculatorService calcService){
        this.calcService = calcService;
    }

    public double add(double input1, double input2){
        //return calcService.add(input1, input2);
        return input1 + input2;
    }

    public double subtract(double input1, double input2){
        return calcService.subtract(input1, input2);
    }

    public double multiply(double input1, double input2){
        return calcService.multiply(input1, input2);
    }

    public double divide(double input1, double input2){
        return calcService.divide(input1, input2);
    }
}

```

```
}  
}
```

Create mocks to add and verify and test the service

Answer

```
import static org.mockito.Mockito.verify;  
import static org.mockito.Mockito.when;  
  
import org.junit.Assert;  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.runners.MockitoJUnitRunner;  
  
// @RunWith attaches a runner with the test class to initialize the test data  
@RunWith(MockitoJUnitRunner.class)  
public class MathApplicationTester {  
  
    // @InjectMocks annotation is used to create and inject the mock object  
    @InjectMocks  
    MathApplication mathApplication = new MathApplication();  
  
    // @Mock annotation is used to create the mock object to be injected  
    @Mock  
    CalculatorService calcService;  
  
    @Test  
    public void testAdd(){  
        //add the behavior of calc service to add two numbers  
        when(calcService.add(10.0,20.0)).thenReturn(30.00);  
  
        //test the add functionality  
        Assert.assertEquals(calcService.add(10.0, 20.0),30.0,0);  
  
        //verify the behavior  
        verify(calcService).add(10.0, 20.0);  
    }  
}
```