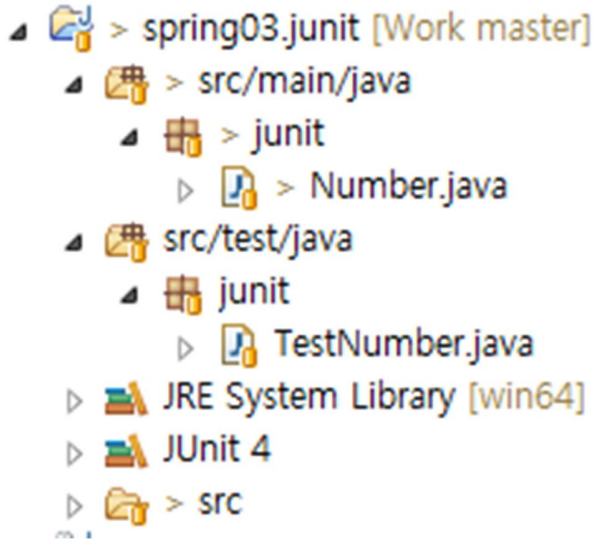
목차

1. JUnit 사용법

이클립스에는 JUnit 이 기본적으로 포함되어 있기 때문에 쉽게 테스트 케이스를 작성하고, 테스트를 해볼 수 있습니다.

아래 그림과 같이 Java 프로젝트를 생성하고 Number 라는 클래스를 작성합니다.



Number 클래스는 JUnit 을 통해 테스트를 실행하게 될 클래스이며, test 는 테스트 클래스들이 들어갈 패키지입니다.(패키지명은 자유롭게 지정할 수 있습니다.)

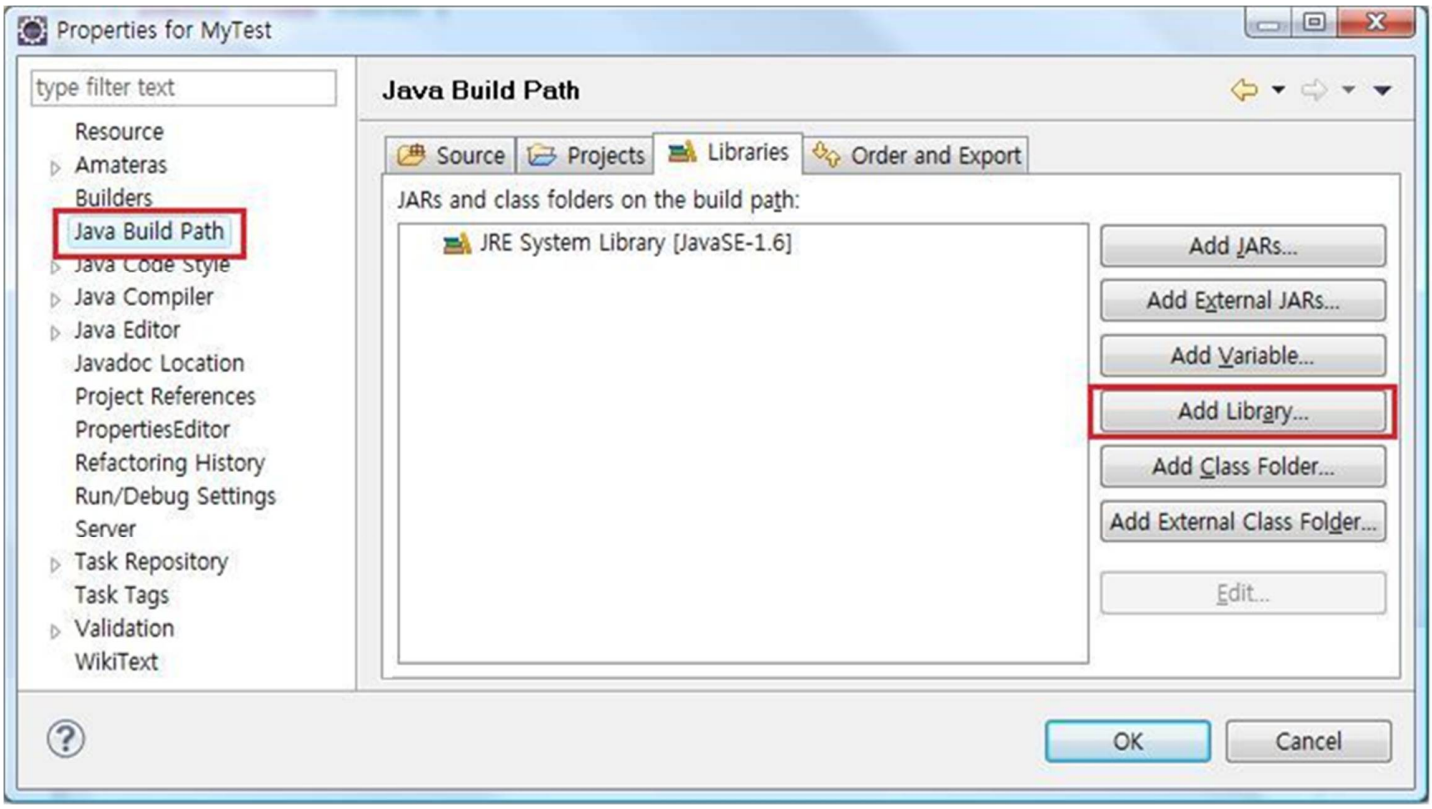
Number 클래스의 코드는 다음과 같습니다.

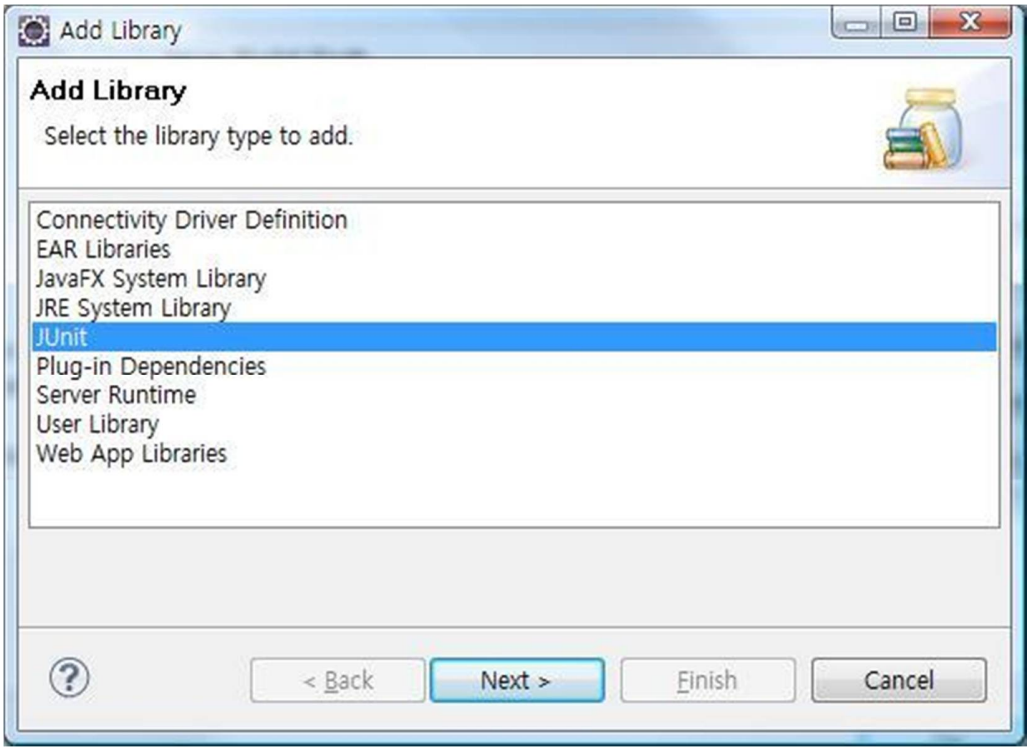
```
package junit;

public class Number {

    private int value;

    // getter & setter
    public int getValue() {
        return value;
    }
    public void setValue(int value) {
        this.value = value;
    }

    // constructor
    public Number() {
```

```
        this(0);
    }

    public Number(int value) {
        this.value = value;
    }

    // method
    public int add(int rhs) {
        return value += rhs;
    }

    public int minus(int rhs) {
        return value -= rhs;
    }

    public int multiply(int rhs) {
        return value *= rhs;
    }

    public int divide(int rhs) {
        return value /= rhs;
    }

}
```

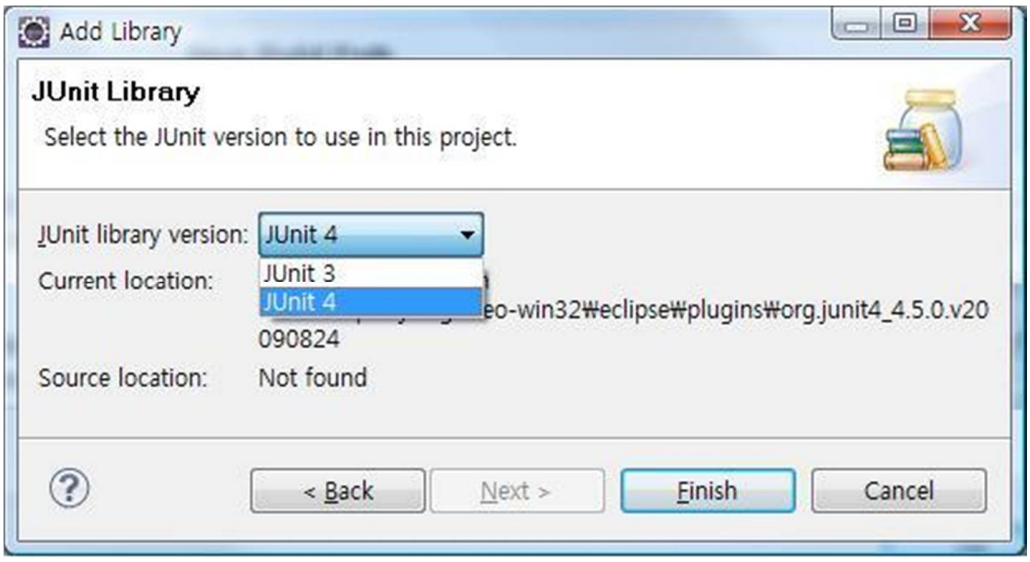
1. 프로젝트에 JUnit 추가

프로젝트의 Properties 를 열고 Java Build Path 에서 Add Library 버튼을 클릭해 JUnit 을 프로젝트에 추가할 수 있습니다.

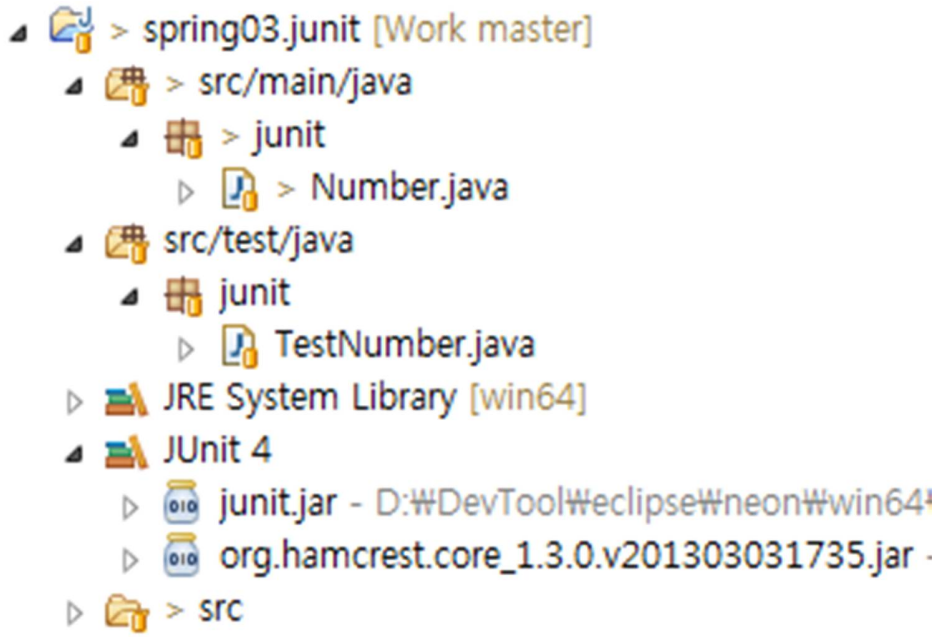




선택할 수 있는 JUnit 은 버전 3 과 버전 4 가 있는데 본 예제에서는 4 를 선택하도록 하겠습니다.



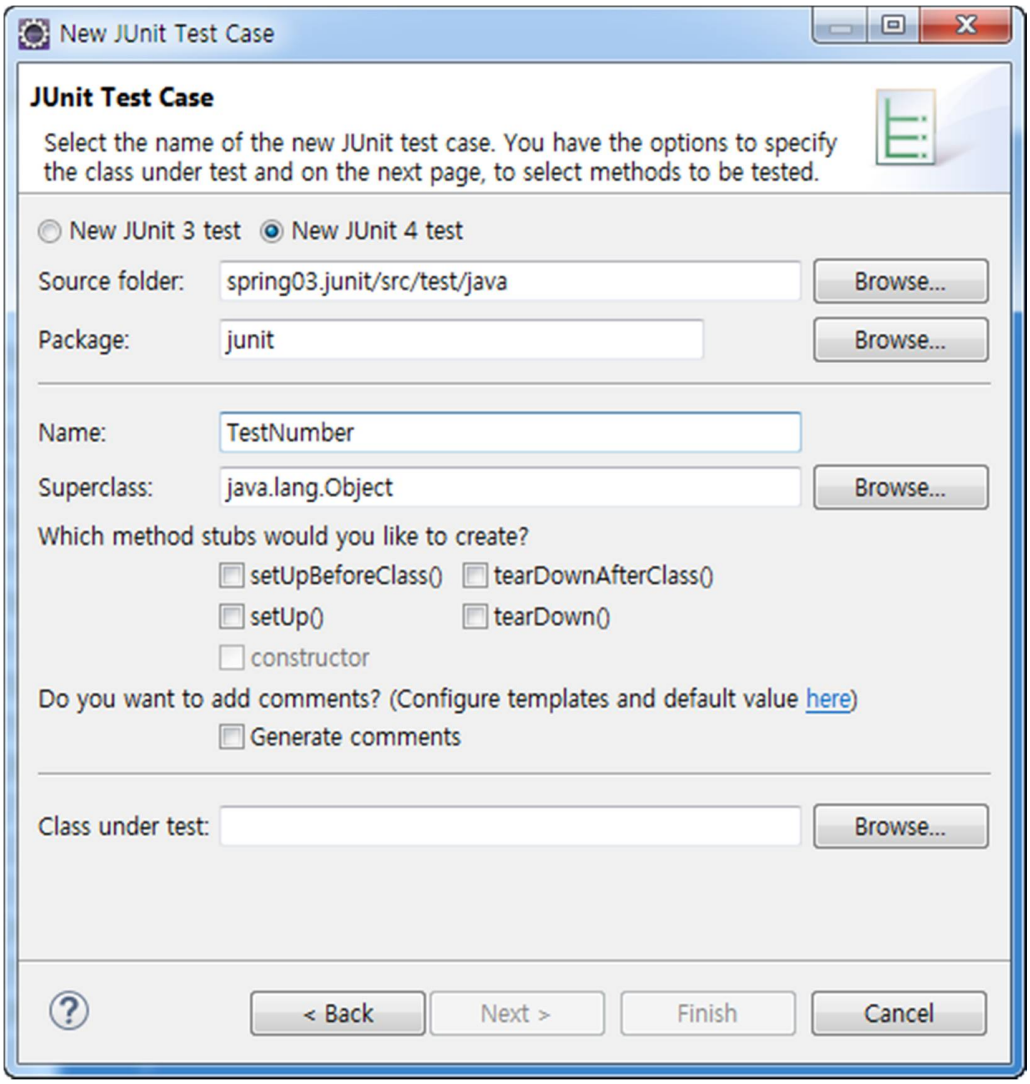
아래 그림과 같이 JUnit 라이브러리가 프로젝트에 추가된 것을 확인할 수 있습니다.



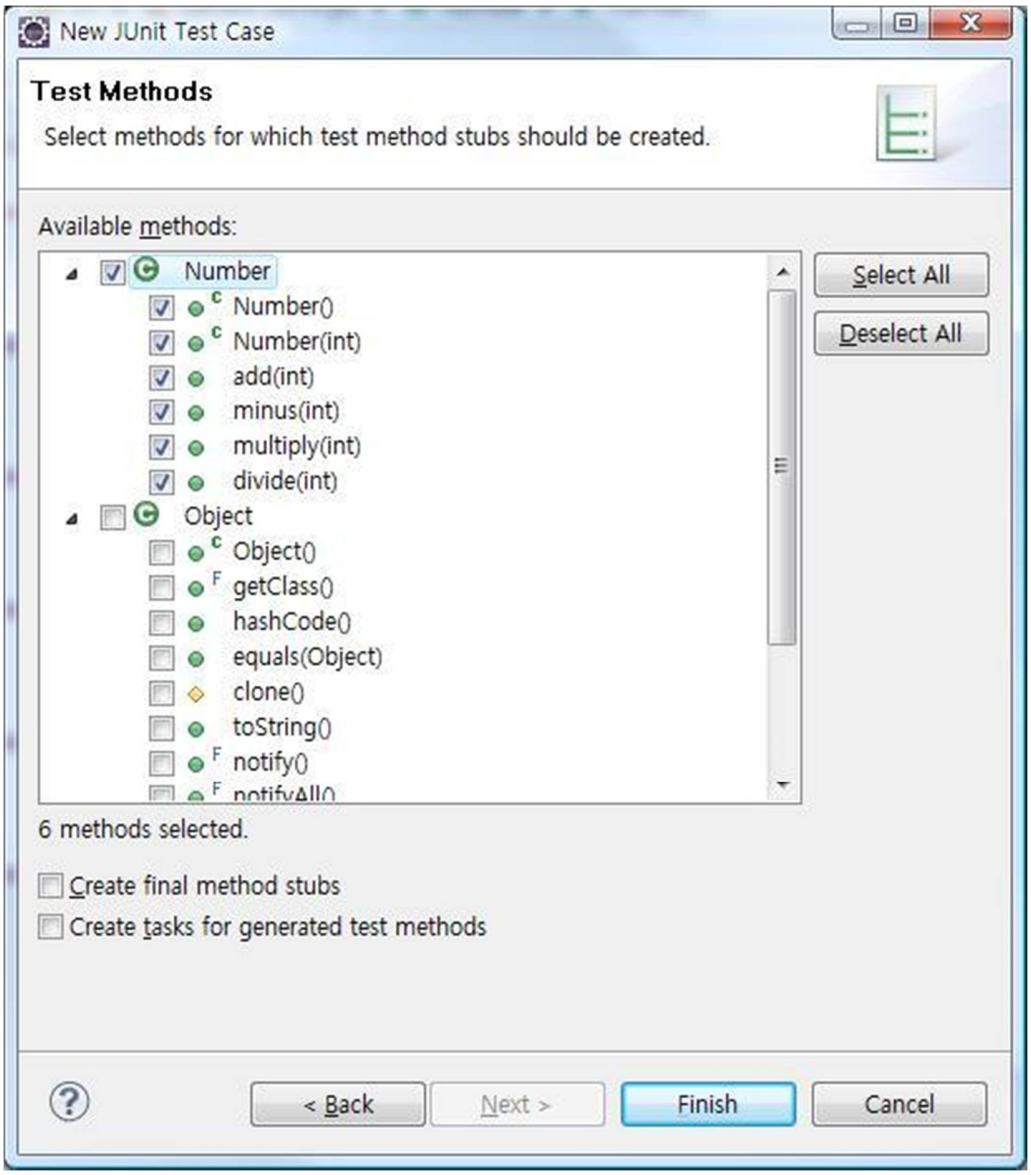
2. JUnit Test Case

Number 클래스에서 오른쪽 마우스를 클릭해 New - JUnit Test Case 를 클릭합니다.

아래 그림과 같이 Package 를 Number.test 로 지정하고 Next 를 클릭합니다.



아래 그림과 같이 테스트를 수행할 Number 클래스를 체크하고 Finish 버튼을 클릭하면 테스트 클래스가 생성됩니다.



테스트 코드는 다음과 같습니다.

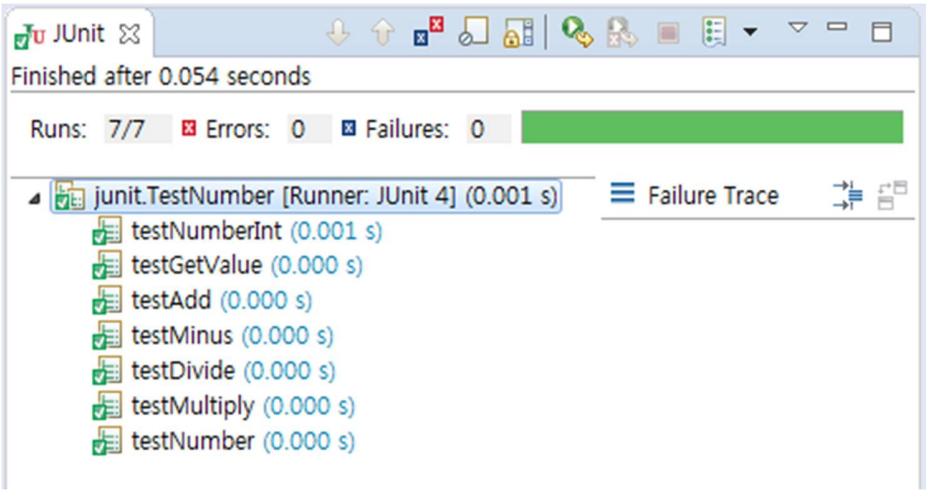
```
package junit;

mport static org.junit.Assert.*;

import org.junit.Test;

import junit.Number;

public class TestNumber {

    @Test
    public void testNumber() {
```

```
        Number num = new Number();
        assertEquals(0, num.getValue());
    }

    @Test
    public void testNumberInt() {
        Number num = new Number(10);
        assertEquals(10, num.getValue());
    }

    @Test
    public void testAdd() {
        Number num = new Number(10);
        assertEquals(20, num.add(10));
    }

    @Test
    public void testMinus() {
        Number num = new Number(10);
        assertEquals(5, num.minus(5));
    }

    @Test
    public void testMultiply() {
        Number num = new Number(5);
        assertEquals(25, num.multiply(5));
    }

    @Test
    public void testDivide() {
        Number num = new Number(20);
        assertEquals(2, num.divide(10));
    }

    @Test
    public void testGetValue() {
        Number num = new Number(20);
        assertEquals(20, num.getValue());
    }
}
```

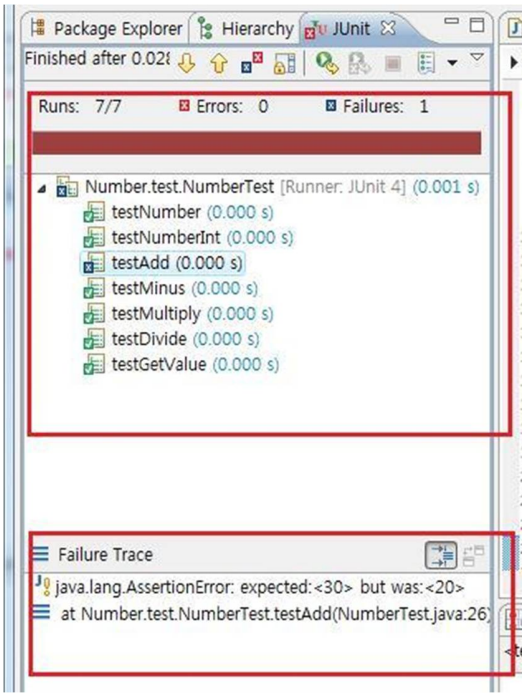
3. JUnit Test

테스트를 실행하기 위해 NumberTest 클래스에서 오른쪽 마우스를 클릭해 Run as - JUnit Test 를 클릭 합니다.

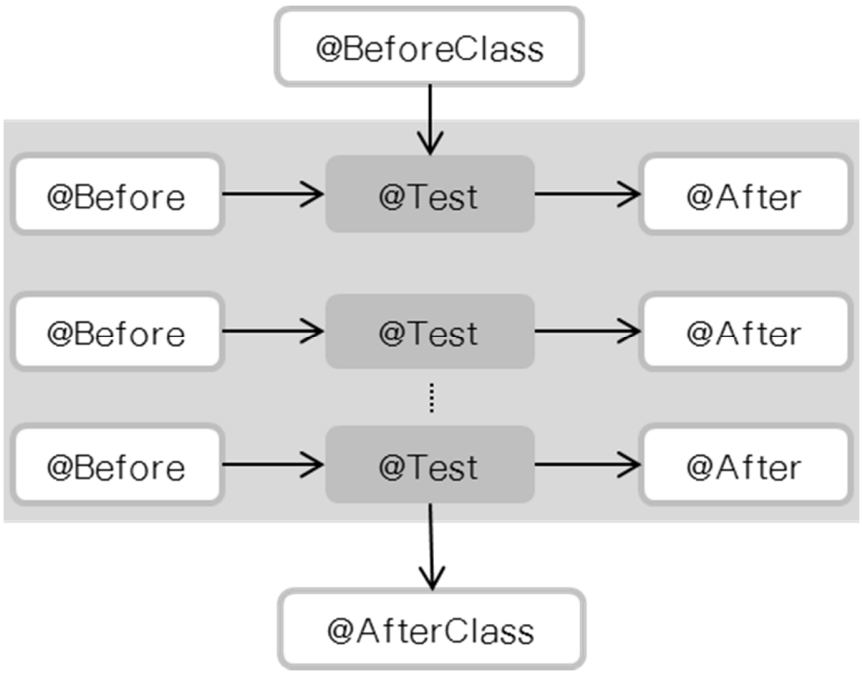
테스트가 실행되고 아래 그림과 같이 모든 테스트가 성공했음을 확인할 수 있습니다.



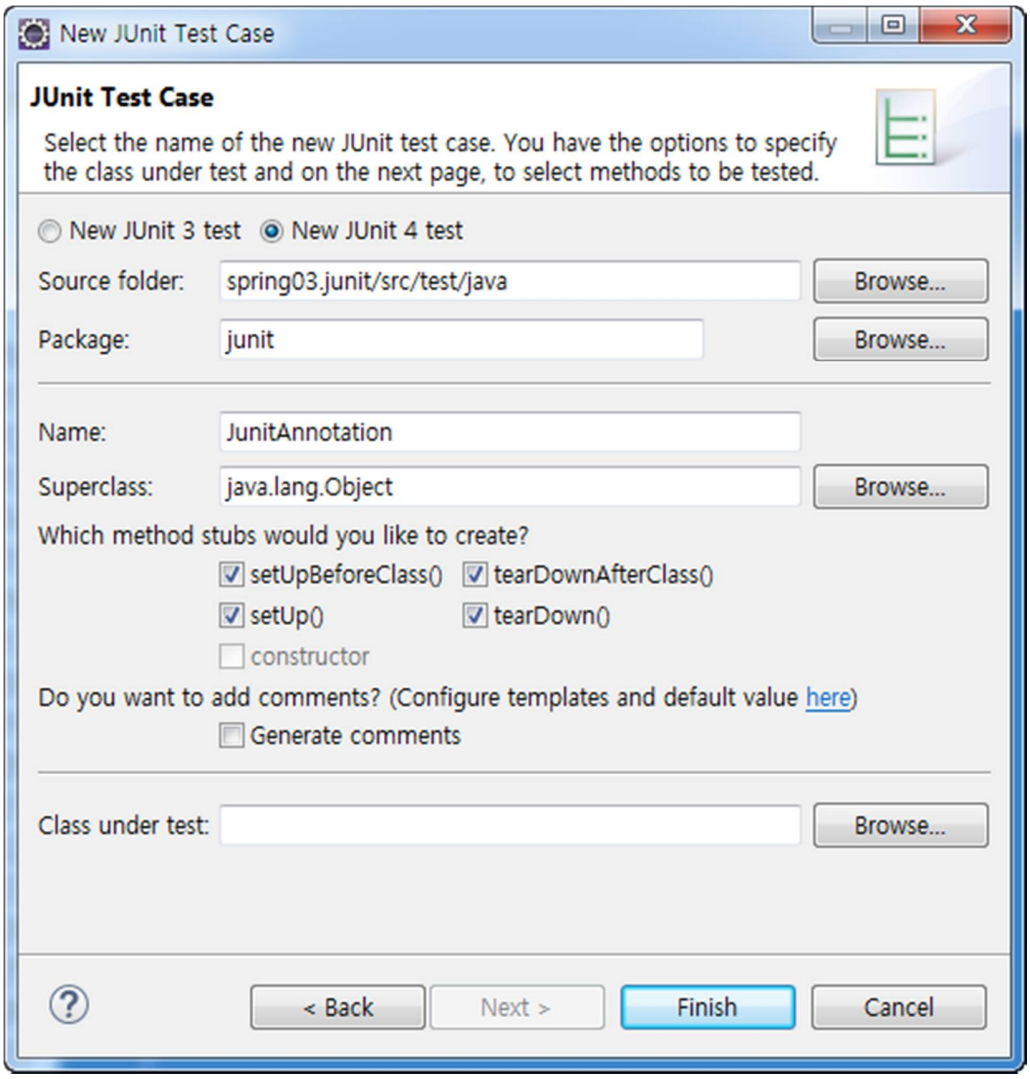
만약 테스트가 실패하면 아래 그림과 같이 실패한 테스트와 메세지를 확인할 수 잇습니다.



2. JUnit annotations



No.	Annotation	Description
1	@Test	The Test annotation tells JUnit that the public void method to which it is attached can be run as a test case.
2	@Before	Several tests need similar objects created before they can run. Annotating a public void method with @Before causes that method to be run before each Test method.
3	@After	If you allocate external resources in a Before method, you need to release them after the test runs. Annotating a public void method with @After causes that method to be run after the Test method.
4	@BeforeClass	Annotating a public static void method with @BeforeClass causes it to be run once before any of the test methods in the class.
5	@AfterClass	This will perform the method after all tests have finished. This can be used to perform clean-up activities.
6	@Ignore	The Ignore annotation is used to ignore the test and that test will not be executed.
7	@Rule	




```
package junit;

import static org.junit.Assert.*;

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 JunitAnnotation {

    @BeforeClass
    public static void beforeClass() {
       System.out.println("in before class");
```

```
    }

    @AfterClass
    public static void  afterClass() {
       System.out.println("in after class");
    }

    @Before
    public void before() {
       System.out.println("in before");
    }

    @After
    public void after() {
       System.out.println("in after");
    }

    @Test
    public void test() {
       System.out.println("in test");
    }

    @Ignore
    public void ignoreTest() {
       System.out.println("in ignore test");
    }
}
```

3. assert methods

assertEquals(a, b);	객체 A 와 B 가 일치함을 확인한다.
assertArrayEquals(a, b);	배열 A 와 B 가 일치함을 확인한다.
assertSame(a, b);	객체 A 와 B 가 같은 객임을 확인한다. assertEquals 메서드는 두 객체의 값이 같은가를 검사는데 반해 assertSame 메서드는 두 객체가 동일한가 즉 하나의 객인 가를 확인한다.
assertNotSame(a, b);	
assertTrue(a);	조건 A 가 참인가를 확인한다.
assertFalse(a);	조건 A 가 참인가를 확인한다.
assertNull(a);	객체 A 가 null 이 아님을 확인한다.
assertNotNull(a);	객체 A 가 null 이 아님을 확인한다.

- assertArrayEquals()
- assertEquals()
- assertTrue() + assertFalse()
- assertNull() + assertNotNull()
- assertSame() + assertNotSame()
- assertThat()

Throughout the rest of this text I will explain how these assert methods work, and show you examples of how to use them. The examples will test an imaginary class called MyUnit. The code for this class is not shown, but you don't really need the code in order to understand how to test it.

1. assertArrayEquals()

The assertArrayEquals() method will test whether two arrays are equal to each other. In other words, if the two arrays contain the same number of elements, and if all the elements in the array are equal to each other.

To check for element equality, the elements in the array are compared using their equals() method. More specifically, the elements of each array are compared one by one using their equals() method. That means, that it is not enough that the two arrays contain the same elements. They must also be present in the same order.

Here is an example:

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

```
public class MyUnitTest {

    @Test
    public void testGetTheStringArray() {
        MyUnit myUnit = new MyUnit();

        String[] expectedArray = {"one", "two", "three"};

        String[] resultArray =  myUnit.getTheStringArray();

        assertArrayEquals(expectedArray, resultArray);
    }
}
```

First the expected array is created. Second the myUnit.getTheStringArray() method is called, which is the method we want to test. Third, the result of the myUnit.getTheStringArray() method call is compared to the expected array.

If the arrays are equal, the assertArrayEquals() will proceed without errors. If the arrays are not equal, an exception will be thrown, and the test aborted. Any test code after the assertArrayEquals() will not be executed.

2. assertEquals()

The assertEquals() method compares two objects for equality, using their equals() method.

Here is a simple example:

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

public class MyUnitTest {

    @Test
    public void testConcatenate() {
        MyUnit myUnit = new MyUnit();

        String result = myUnit.concatenate("one", "two");

        assertEquals("onetwo", result);
    }
}
```

First the myUnit.concatenate() method is called, and the result is stored in the variable result.

Second, the result value is compared to the expected value "onetwo", using the assertEquals() method.

If the two objects are equal according to their implementation of their equals() method, the assertEquals()method will return normally. Otherwise the assertEquals() method will throw an exception, and the test will stop there.

This example compared to String objects, but the assertEquals() method can compare any two objects to each other. The assertEquals() method also come in versions which compare primitive types like int and float to each other.

3. assertTrue() + assertFalse()

The assertTrue() and assertFalse() methods tests a single variable to see if its value is either true, or false. Here is a simple example:

import org.junit.Test;

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

public class MyUnitTest {

    @Test
    public void testGetTheBoolean() {
        MyUnit myUnit = new MyUnit();

        assertTrue (myUnit.getTheBoolean());

        assertFalse(myUnit.getTheBoolean());
    }
}
```

As you can see, the method call to myUnit.getTheBollean() is inlined inside the assertTrue()assertFalse() calls.

If the getTheBoolean() method returns true, the assertTrue() method will return normally. Otherwise an exception will be thrown, and the test will stop there.

If the getTheBoolean() method returns false, the assertFalse() method will return normally. Otherwise an exception will be thrown, and the test will stop there.

Of course the above test will fail in either the assertTrue() or assertFalse() call, if the getTheBoolean()method returns the same value in both calls. One of the assertTrue() or assertFalse() calls will fail.

4. assertNull() + assertNotNull()

The assertNull() and assertNotNull() methods test a single variable to see if it is null or not null. Here is an example:

```
import org.junit.Test;
```

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

public class MyUnitTest {

    @Test
    public void testGetTheObject() {
        MyUnit myUnit = new MyUnit();

        assertNull(myUnit.getTheObject());

        assertNotNull(myUnit.getTheObject());
    }
}
```

The call to myUnit.getTheObject() is inlined in the assertNull() and assertNotNull() calls.
If the myUnit.getTheObject() returns null, the assertNull() method will return normally. If a non-null value is returned, the assertNull() method will throw an exception, and the test will be aborted here.
The assertNotNull() method works oppositely of the assertNull() method, throwing an exception if a null value is passed to it, and returning normally if a non-null value is passed to it.

5. assertSame() and assertNotSame()

The assertSame() and assertNotSame() methods tests if two object references point to the same object or not. It is not enough that the two objects pointed to are equals according to their equals() methods. It must be exactly the same object pointed to.
Here is a simple example:

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

public class MyUnitTest {

    @Test
    public void testGetTheSameObject() {
        MyUnit myUnit = new MyUnit();

        assertSame   (myUnit.getTheSameObject(),
                      myUnit.getTheSameObject());

        assertNotSame(myUnit.getTheSameObject(),
                      myUnit.getTheSameObject());
    }
}
```

The calls to myUnit.getTheSameObject() are inlined into the assertSame() and assertNotSame() method calls.

If the two references points to the same object, the assertSame() method will return normally. Otherwise an exception will be thrown and the test will stop here.

The assertNotSame() method works oppositely of the assertSame() method. If the two objects do not poin to the same object, the assertNotSame() method will return normally. Othersise an exception is thrown and the test stops here.

6. assertThat()

The assertThat() method compares an object to an org.hamcrest.Matcher to see if the given object matches whatever the Matcher requires it to match.

Matchers takes a bit longer to explain, so they are explained in their own text (the next text in this trail).

4. Parameterized tests

Junit 4 has introduced a new feature Parameterized tests.Parameterized tests allow developer to run the same test over and over again using different values. There are five steps, that you need to follow to create Parameterized tests.

- Annotate test class with @RunWith(Parameterized.class)
- Create a public static method annotated with @Parameters that returns a Collection of Objects (as Array) as test data set.
- Create a public constructor that takes in what is equivalent to one "row" of test data.
- Create an instance variable for each "column" of test data.
- Create your tests case(s) using the instance variables as the source of the test data.

The test case will be invoked once per each row of data. Let's see Parameterized tests in action.

Create a Class

- Create a java class to be tested say PrimeNumberChecker.java in **C:\ > JUNIT_WORKSPACE**.

```
public class PrimeNumberChecker {
   public Boolean validate(final Integer primeNumber) {
      for (int i = 2; i < (primeNumber / 2); i++) {
         if (primeNumber % i == 0) {
            return false;
         }
      }
      return true;
   }
}
```

Create Parameterized Test Case Class

- Create a java test class say PrimeNumberCheckerTest.java.

Create a java class file name PrimeNumberCheckerTest.java in **C:\ > JUNIT_WORKSPACE**

```
import java.util.Arrays;
import java.util.Collection;

import org.junit.Test;
import org.junit.Before;
import org.junit.runners.Parameterized;
```

```
import org.junit.runners.Parameterized.Parameters;
import org.junit.runner.RunWith;
import static org.junit.Assert.assertEquals;

@RunWith(Parameterized.class)
public class PrimeNumberCheckerTest {
   private Integer inputNumber;
   private Boolean expectedResult;
   private PrimeNumberChecker primeNumberChecker;

   @Before
   public void initialize() {
      primeNumberChecker = new PrimeNumberChecker();
   }

   // Each parameter should be placed as an argument here
   // Every time runner triggers, it will pass the arguments
   // from parameters we defined in primeNumbers() method
   public PrimeNumberCheckerTest(Integer inputNumber,
      Boolean expectedResult) {
      this.inputNumber = inputNumber;
      this.expectedResult = expectedResult;
   }

   @Parameterized.Parameters
   public static Collection primeNumbers() {
      return Arrays.asList(new Object[][] {
         { 2, true },
         { 6, false },
         { 19, true },
         { 22, false },
         { 23, true }
      });
   }

   // This test will run 4 times since we have 5 parameters defined
   @Test
   public void testPrimeNumberChecker() {
      System.out.println("Parameterized Number is : " + inputNumber);
      assertEquals(expectedResult,
      primeNumberChecker.validate(inputNumber));
   }
}
```

Create Test Runner Class

Create a java class file name TestRunner.java in C:₩ > JUNIT_WORKSPACE to execute Test case(s)

```
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(PrimeNumberCheckerTest.class);
      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

Verify the output.

```
Parameterized Number is : 2
Parameterized Number is : 6
Parameterized Number is : 19
Parameterized Number is : 22
Parameterized Number is : 23
true
```

5. Reference

http://lyb1495.tistory.com/73
http://www.tutorialspoint.com/junit/junit_parameterized_test.htm

http://tutorials.jenkov.com/java-unit-testing/asserts.html