SoftServe | Empowering your Business through Software Development

Java. Unit and Integration Testing

IT Academy

22/07/2014

Agenda

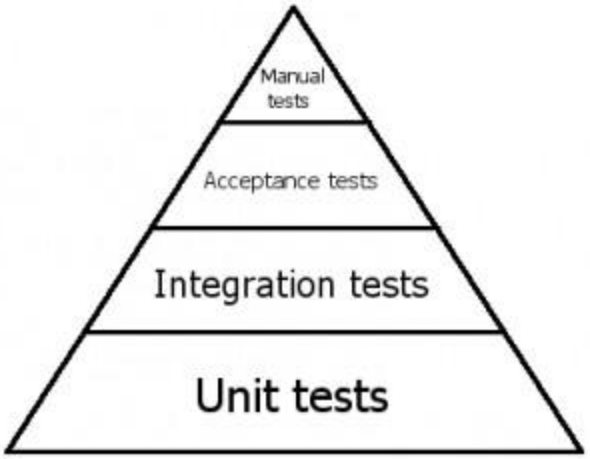
- What is Testing
- Testing Types
- Unit and Integration Testing
- JUnit
- Emma Coverage
- TestNG
- Parallel Tests Running
- Case studies

What is Testing

- **Software testing** is the process of program execution in order to find bugs.
- **Software testing** is the process used to measure the quality of developed computer software. Usually, quality is constrained to such topics as:
  - correctness, completeness, security;
- but can also include more technical requirements such as:
  - capability, reliability, efficiency, portability, maintainability, compatibility, usability, etc.

Testing Types. Unit Testing

- **Unit testing** is a procedure used to validate that individual units of source code are working properly.
- The goal of unit testing is to isolate each part of the program and show that the individual parts are correct.

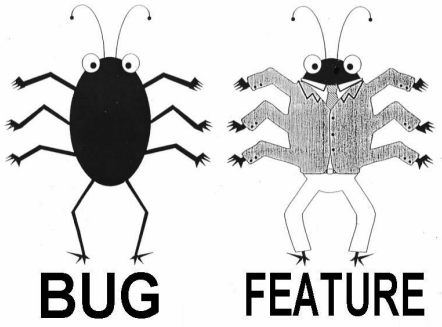


Testing Types. Unit Testing

- Unit testing – the **process** of programming, allowing you to test the correctness of the individual modules of the **program source**.
- The idea – to write tests for each **non-trivial** function or method.
- This allows you to:
  - **Check** whether an another code change to errors in the field of tested already the program;
  - **Easy** the detection and elimination of such errors.
- The purpose of unit testing – to isolate certain parts of the program and show that **individually** these pieces are **functional**.

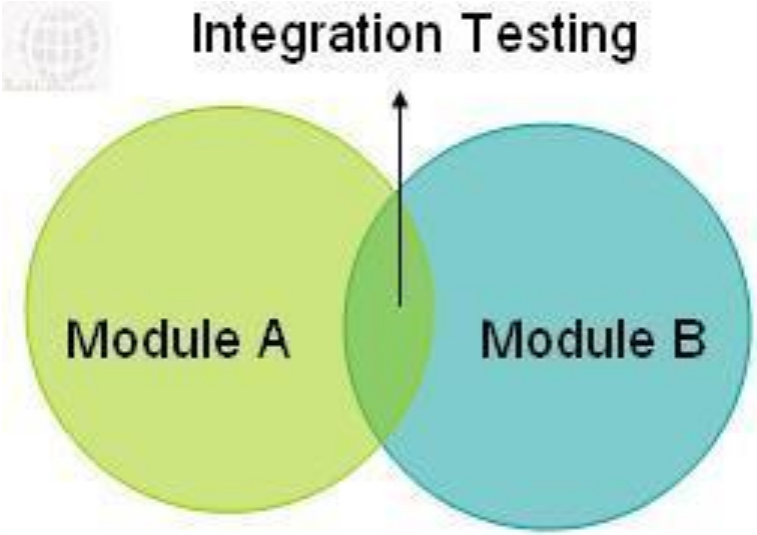
Testing Types. Unit Testing

- **Task**: Implement functionality to calculate *speed=distance/time* where distance and time values will be manually entered.
- **Unit Testing Procedure**: Test the implemented functionality with the different *distance* and *time* values.
- **Defect**: Crash when entered value *time=0*.



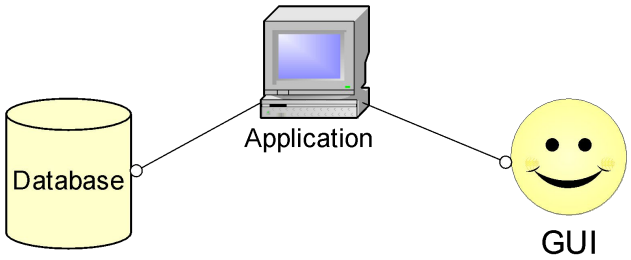
Integration Testing

- **Integration testing** is the phase of software testing in which individual software modules are combined and tested as a group.
- This is testing two or more modules or functions together with the intent of finding interface defects between the modules or functions.



Integration Testing

- **Task**: Database scripts, application main code and GUI components were developed by different programmers. There is need to test the possibility to use these 3 parts as one system.
- **Integration Testing Procedure**: Combine 3 parts into one system and verify interfaces (check interaction with database and GUI).
- **Defect**: “Materials” button action returns not list of books but list of available courses.



Unit / Integration Testing

```
public class One {
    private String text;
    public One() {
        // Code
    }
    // Functionality
    public int calc() {
        int result = 0;
        // Code
        return result;
    }
    // get, set, etc.
}
```

Unit / Integration Testing

```
public class Two {
    private One one;
    public Two(One one) {
        this.one = one;
    }
    public String resume() {
        // Functionality
        return one.getText() + "_" + one.calc().toString();
     }
    // Code
}
```

Unit / Integration Testing

```
public class Appl {
    public static void main(String[] args) {
        One one = new One();
        one.setText("data“);
        Two two = new Two(one);
        two.resume();
        // etc
    }
}
```

Unit / Integration Testing

```
public class TwoTest {
    @Test
    public void TestResume() {
        One one = new One();
        one.setText("Integration Test“);
        Two two = new Two(one);
        // TODO Initialize to an appropriate value
        String expected = "Integration Test_...";    // Result ???
        String actual;
        actual = two.resume();
        Assert.AreEqual(expected, actual);
} }
```

Unit / Integration Testing

- A method **stub** or simply stub in software development is a piece of code used to stand in for some other programming functionality.
- A stub may simulate the behavior of existing code or be a temporary substitute for yet-to-be-developed code.

```
public interface IOne {
    void setText(String text);
    String getText();
    int calc();
}
```

Unit / Integration Testing

```
public class One implements IOne {
    private String text;
    public One() {
        // Code
    }
    // Functionality
    public int calc() {
        int result = 0;
        // Code
        return result;
    }
    // get, set, etc.
}
```

Unit / Integration Testing

```
public class Two {
    private IOne one;
    public Two(IOne one) {
        this.one = one;
    }
    public String resume() {
        // Functionality
        return one.getText() + "_" + one.calc().toString();
     }
    // Code
}
```

Unit / Integration Testing

- **Stub** – a piece of code used to stand in for some other programming functionality.

```
public class StubOne extends IOne {
    private String text;
    public StubOne() {
    }
    public void setText(String text) {
    }
    public String getText() {
        return ""
    }
    public int Calc() {
        return 0;
    }
}
```

Unit / Integration Testing

```
public class TwoTest {
    @Test
    public void TestResume() {
        IOne one = new StubOne();
        one.setText(" Unit Test“);
        Two two = new Two(one);
        // TODO Initialize to an appropriate value
        String expected = " Unit Test_...";    // Result !!!
        String actual;
        actual = two.resume();
        Assert.AreEqual(expected, actual);
} }
```

Junit 3

```
import junit.framework.TestCase;
public class AddJavaTest extends TestCase {
   protected void setUp() throws Exception {
      // create object  }
   protected void tearDown() throws Exception {
      // to free resources }
    public AddJavaTest(String name){
       super (name); }
     public void testSimpleAddition( ){
       assertTrue(expect == actual);
}   }
```

- **deprecated**

JUnit

`Assert` class contains many different overloaded methods.

http://www.junit.org/ **https**://github.com/kentbeck/junit/wiki

`assertEquals(long expected, long actual)`

Asserts that two longs are equal.

`assertTrue(boolean condition)`

Asserts that a condition is true.

`assertFalse(boolean condition)`

Asserts that a condition is false.

`assertNotNull(java.lang.Object object)`

Asserts that an object isn't null.

`assertNull(java.lang.Object object)`

Asserts that an object is null.

JUnit

```
assertSame(java.lang.Object expected,
           java.lang.Object actual)
```

Asserts that two objects refer to the same object.

```
assertNotSame(java.lang.Object unexpected,
           java.lang.Object actual)
```

Asserts that two objects do not refer to the same object.

```
assertArrayEquals(byte[] expecteds,
           byte[] actuals)
```

Asserts that two byte arrays are equal.

```
fail()
```

Fails a test with no message.

```
fail(java.lang.String message)
```

Fails a test with the given message.

Junit 4

```
import org.junit.Test;
import org.junit.Assert;
public class MathTest {
    @Test
    public void testEquals( ) {
        Assert.assertEquals(4, 2 + 2);
        Assert.assertTrue(4 == 2 + 2);
    }
    @Test
    public void testNotEquals( ) {
        Assert.assertFalse(5 == 2 + 2);
} }
```

JUnit

- To integrate tests, you can use a class `TestSuite`.

```
public static void main(String[] args) {
   TestRunner runner = new TestRunner();
   TestSuite suite = new TestSuite();
   suite.addTest(new TestClass("testEquals"));
   suite.addTest(new TestClass("testNotEquals"));
   runner.doRun(suite);  }
public static Test suite() {          // without main() and runner.
   TestSuite suite = new TestSuite();
   suite.addTest(new TestClass("testEquals"));
   suite.addTest(new TestClass("testNotEquals"));
   return suite;  }
```

JUnit

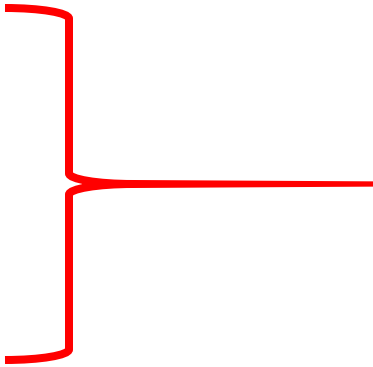
```
package com.softserve.edu;
public class Rectangle implements IRectangle {
    private double height;
    private double width;
    public Rectangle(double height, double width) {
        this.height = height;
        this.width = width;
    }
    public double getHeight() {
        return height;
    }
```

JUnit

```
    public void setHeight(double height) {
        this.height = height;
    }
    public double getWidth() {
        return width;
    }
    public void setWidth(double width) {
        this.width = width;
    }
    public double Perimeter() {
        return 2 * (height + width);
}   }
```

JUnit

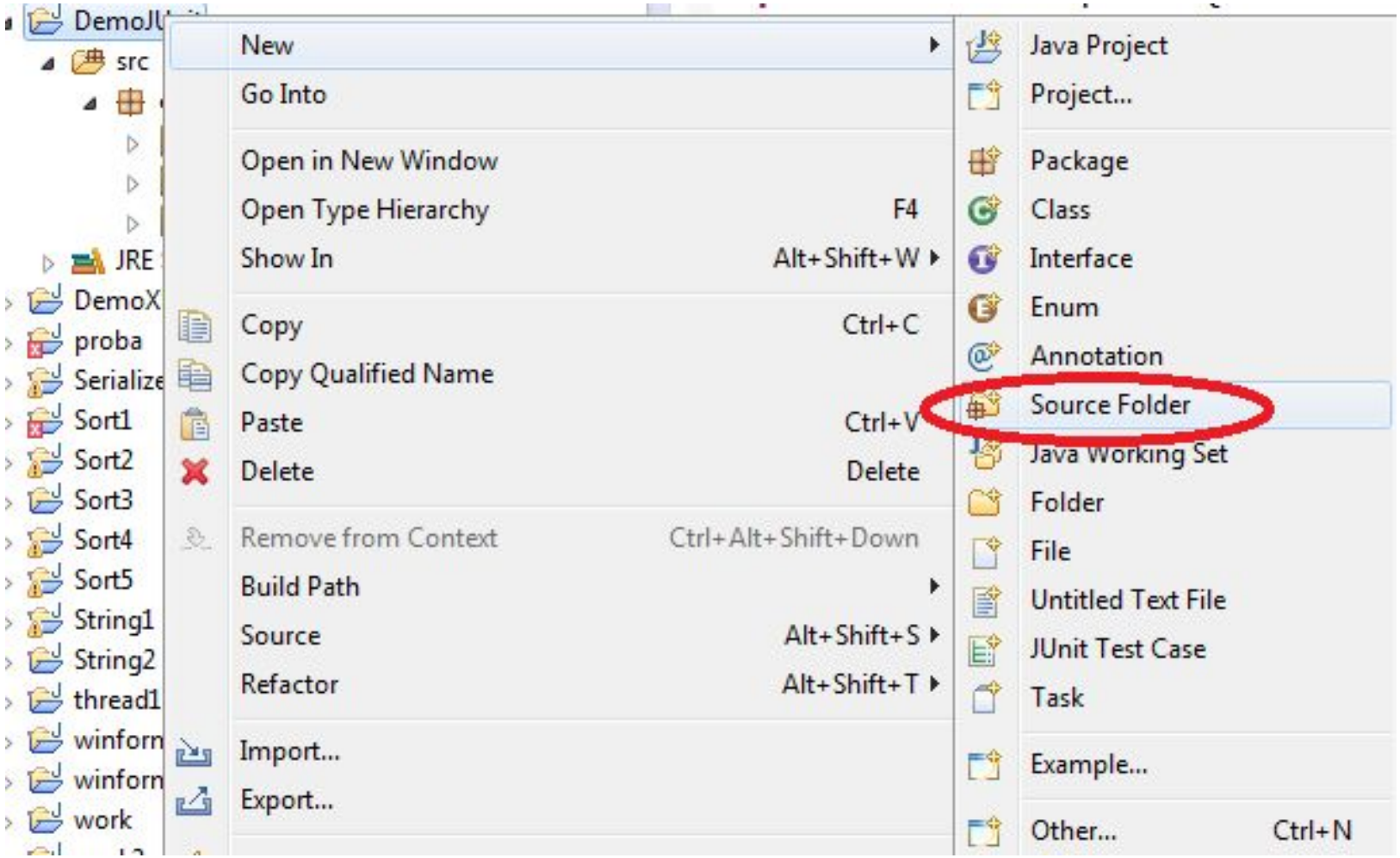
```
package com.softserve.edu;
public class Square {
   private IRectangle rectangle;

   public Square(double width) {
      rectangle = new Rectangle(width, width);
   }
   // For testing
   public void setSquare(IRectangle rectangle) {
      this.rectangle = rectangle;
   }
```



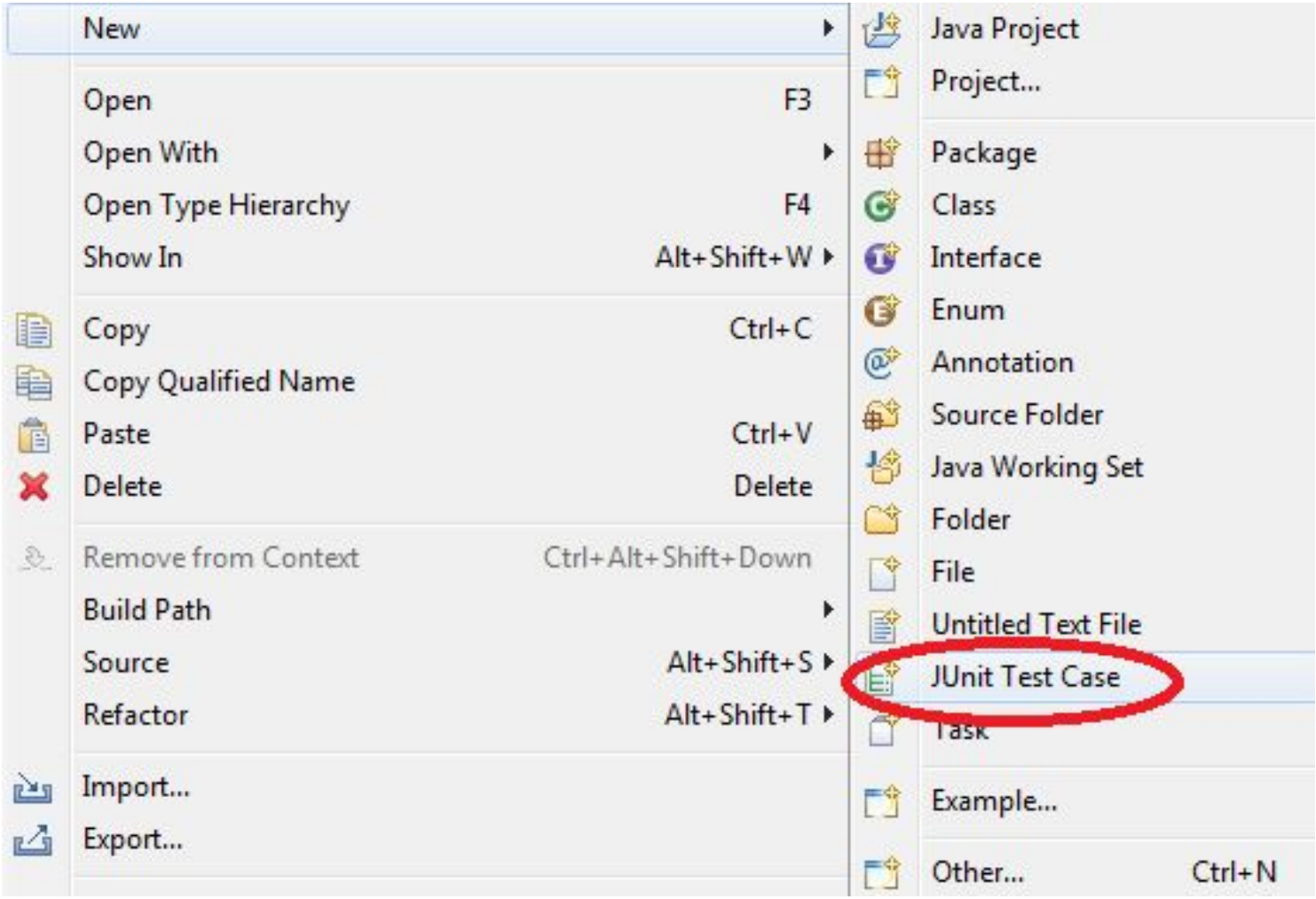
JUnit

```
    public double Perimeter(int multiply) {
        double p = 1;
        for (int i = 0; i < multiply; i++) {
            p = p * rectangle.Perimeter();
        }
        return p;
}   }
package com.softserve.edu;
public interface IRectangle {
    public double Perimeter();
}
```

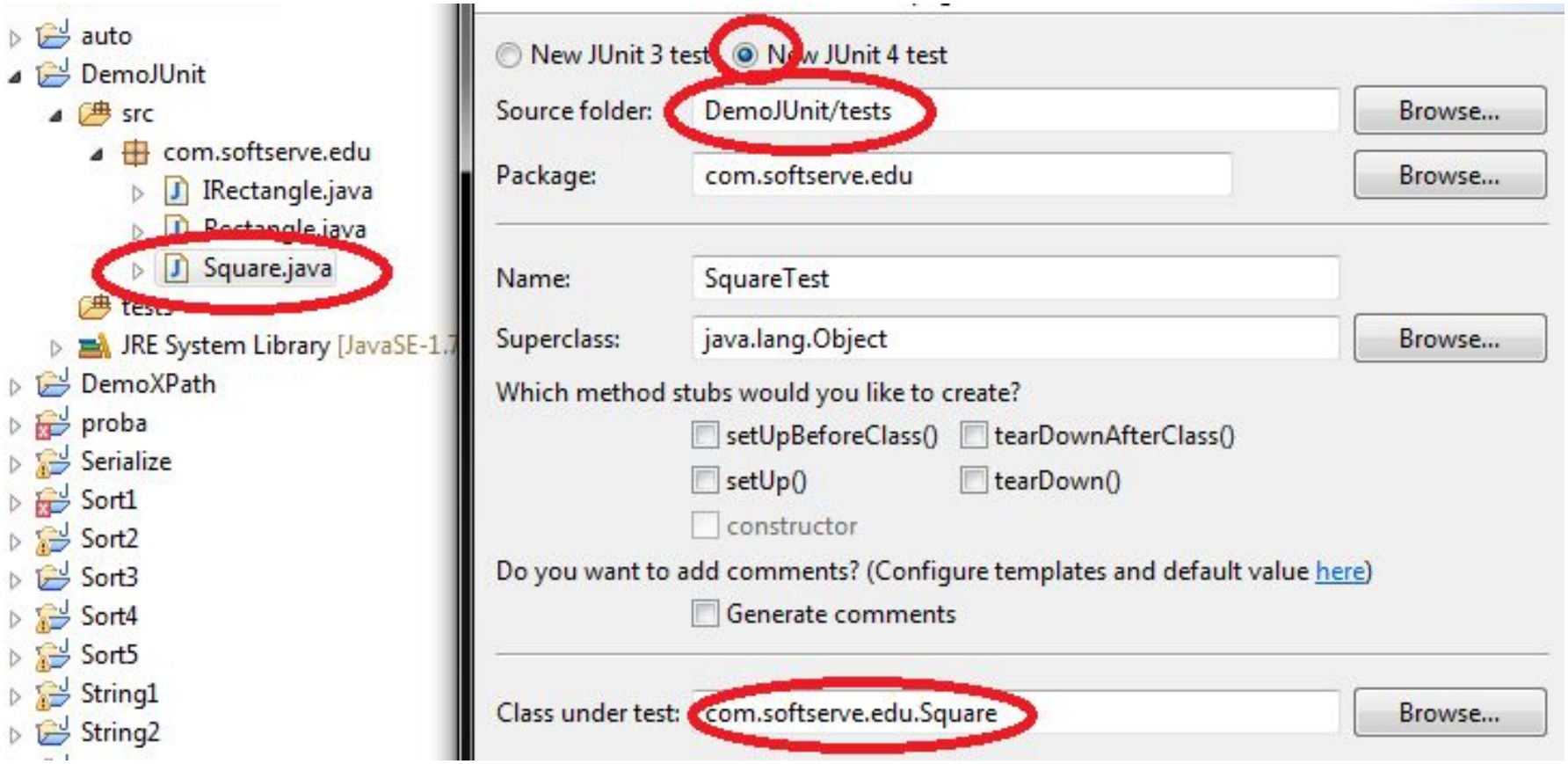
JUnit


DemoJU
src
JRE
DemoX
proba
Serialize
Sort1
Sort2
Sort3
Sort4
Sort5
String1
String2
thread1
winform
winform
work
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Java Project
Project...
Package
Class
Interface
Enum
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Task
Example...
Other... Ctrl+N

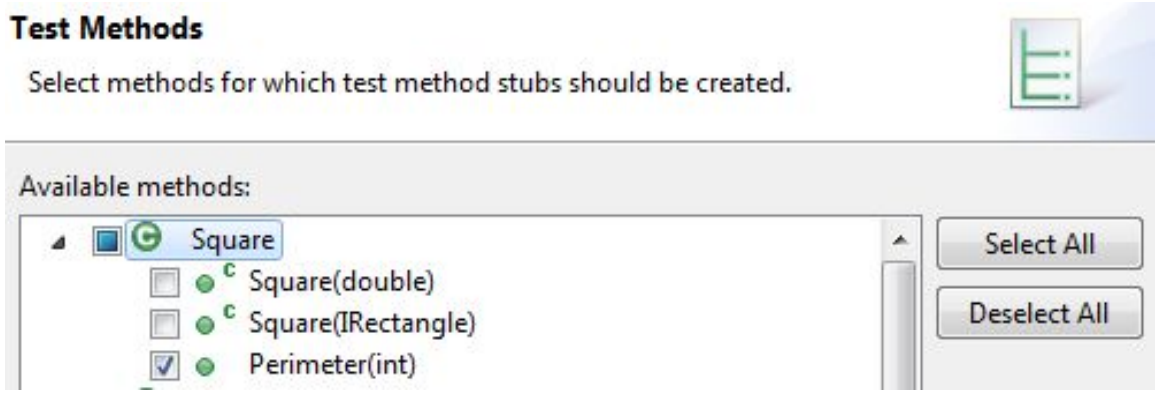
JUnit

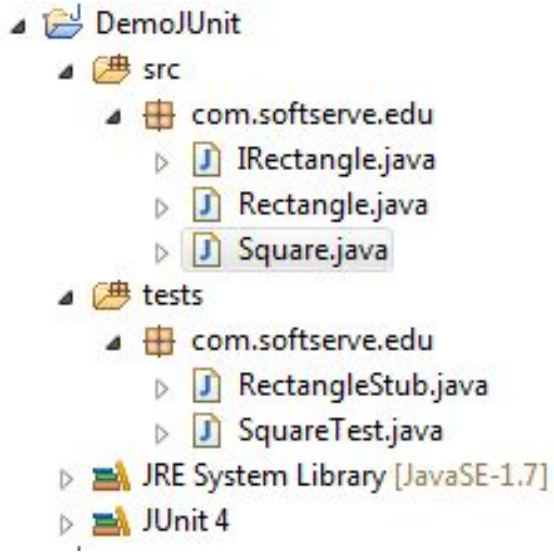

New
Open F3
Open With
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Java Project
Project...
Package
Class
Interface
Enum
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Task
Example...
Other... Ctrl+N

JUnit


auto
DemoJUnit
src
com.softserve.edu
IRectangle.java
Square.java
JRE System Library [JavaSE-1.7
DemoXPath
proba
Serialize
Sort1
Sort2
Sort3
Sort4
Sort5
String1
String2
New JUnit 3 test
New JUnit 4 test
Source folder:
DemoJUnit/tests
Browse...
Package:
com.softserve.edu
Browse...
Name:
SquareTest
Superclass:
java.lang.Object
Browse...
Which method stubs would you like to create?
setUpBeforeClass()
tearDownAfterClass()
setUp()
tearDown()
constructor
Do you want to add comments? (Configure templates and default value here)
Generate comments
Class under test:
com.softserve.edu.Square
Browse...

JUnit


Test Methods
Select methods for which test method stubs should be created.
Available methods:
Square
Square(double)
Square(IRectangle)
Perimeter(int)
Select All
Deselect All


DemoJUnit
src
com.softserve.edu
IRectangle.java
Rectangle.java
Square.java
tests
com.softserve.edu
RectangleStub.java
SquareTest.java
JRE System Library [JavaSE-1.7]
JUnit 4

JUnit

```
package com.softserve.edu;
import org.junit.Assert;
import org.junit.Test;
public class SquareTest {
   @Test
   public void testPerimeter() {
      Square square = new Square(2);
      double expected = 64;          // Result ???
      double actual;
      actual = square.Perimeter(2);  // Method with error
      Assert.assertEquals(expected, actual);
      //fail("Not yet implemented");    }
```

JUnit

```
    @Test
    public void testPerimeter2() {
        Square square = new Square(2);
        square.setSquare(new RectangleStub())
        double expected = 4;
        double actual;
        actual = target.Perimeter(2);
        Assert.assertEquals(expected, actual);
        //fail("Not yet implemented");
    }
}
```

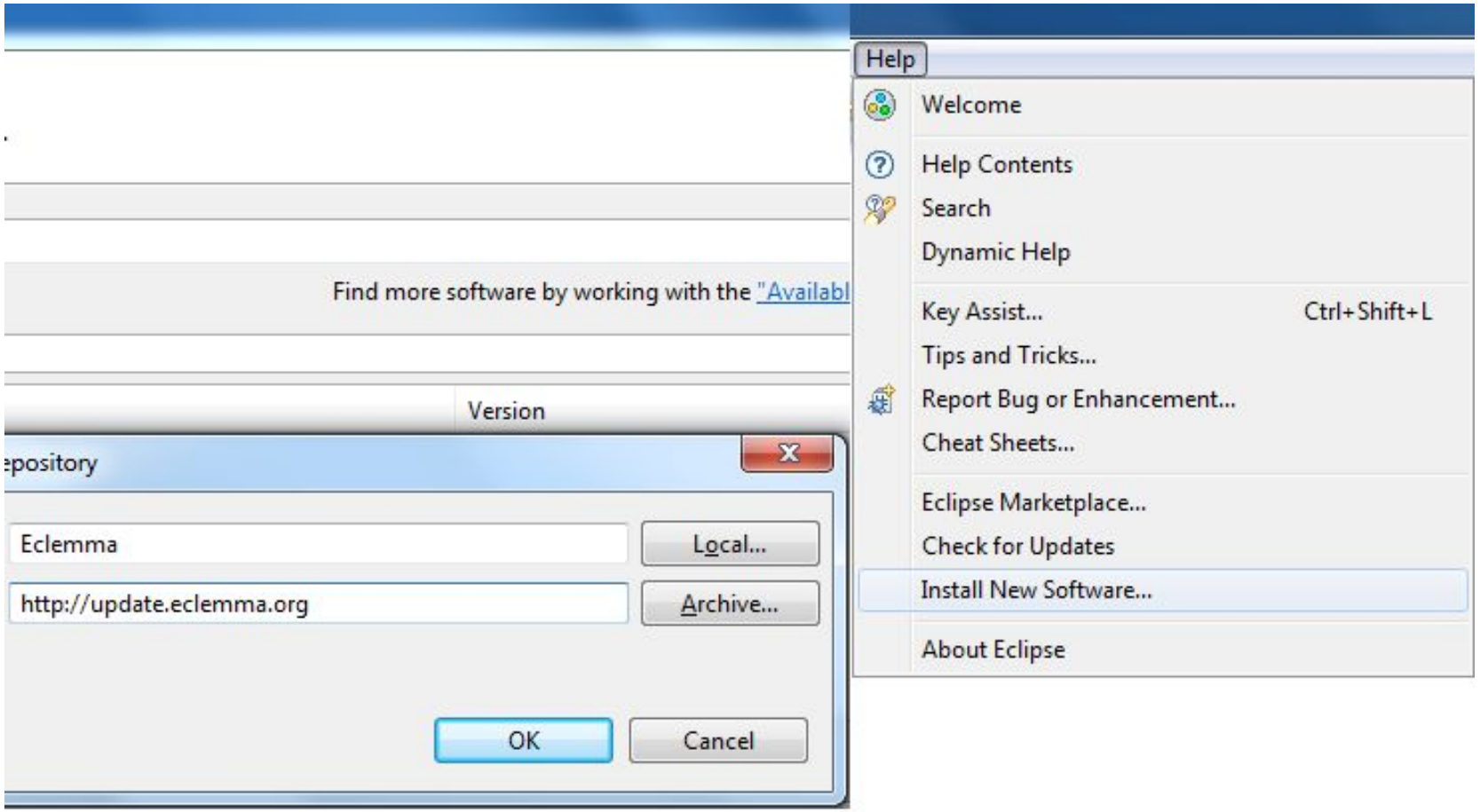
JUnit

- **Stub** – a piece of code used to stand in for some other programming functionality.

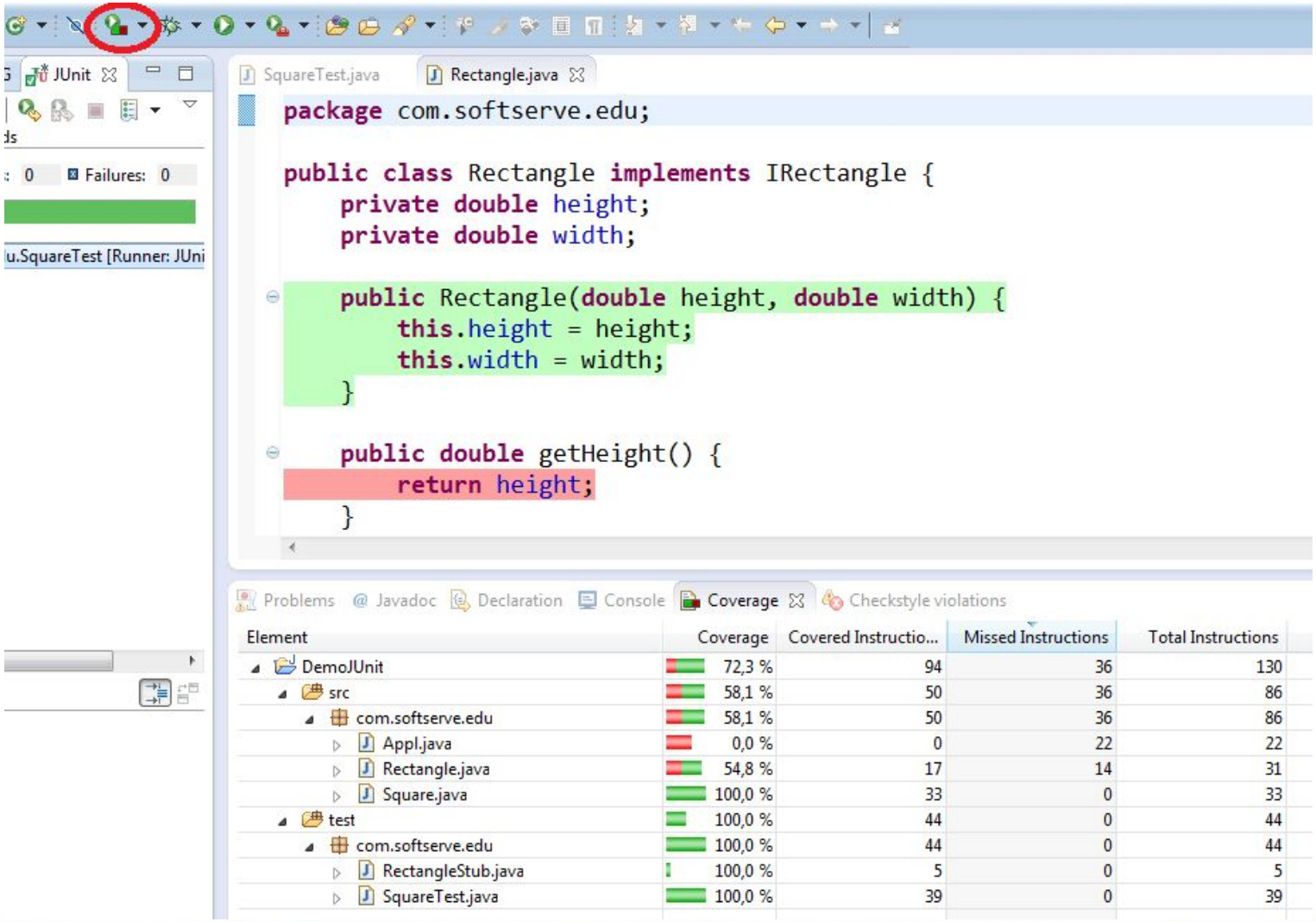
```
package com.softserve.edu;

public class RectangleStub implements IRectangle {
    public double Perimeter() {
        return 2;
    }
}
```

Emma Coverage

- **EclEmma** – Java Code Coverage for Eclipse.



Emma Coverage



Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
DemoJUnit	72,3 %	94	36	130
src	58,1 %	50	36	86
com.softserve.edu	58,1 %	50	36	86
Appl.java	0,0 %	0	22	22
Rectangle.java	54,8 %	17	14	31
Square.java	100,0 %	33	0	33
test	100,0 %	44	0	44
com.softserve.edu	100,0 %	44	0	44
RectangleStub.java	100,0 %	5	0	5
SquareTest.java	100,0 %	39	0	39

TestNG

- JUnit 4 and TestNG are both very popular unit test framework in Java.
- TestNG (**Next Generation**) is a testing framework which inspired by JUnit and NUnit, but introducing many new innovative functionality like dependency testing, grouping concept to make testing more powerful and easier to do. It is designed to cover all categories of tests: unit, functional, end-to-end, integration, etc.

Functionality - JUnit 4 vs TestNG

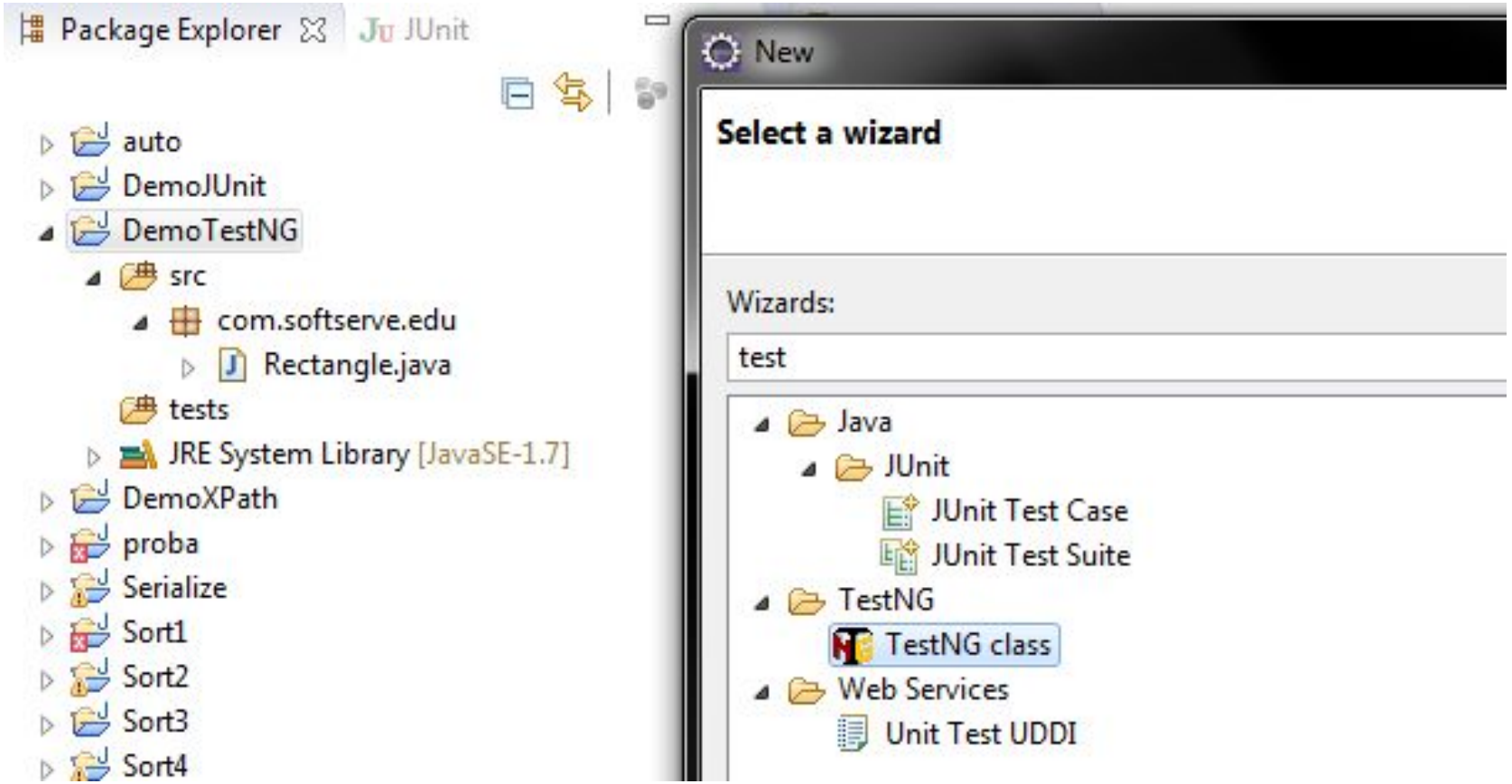
	Annotation Support	Exception Test	Ignore Test	Timeout Test	Suite Test	Group Test	Parameterized (primitive value)	Parameterized (object)	Dependency Test
TestNG	✓	✓	✓	✓	✓	✓	✓	✓	✓
JUnit 4	✓	✓	✓	✓	✓	✗	✓	✗	✗

TestNG

- Setting up TestNG with Eclipse.
- Click Help –> Install New Software
- Type http://beust.com/eclipse in the "Work with" edit box and click ‘Add’ button.
- Click Next and click on the radio button "I accept the terms of the license agreement“.
- After the installation, it will ask for restart of Eclipse. Then restart the Eclipse.
- Once the Eclipse is restarted, we can see the TestNG icons & menu items as in the below figures.

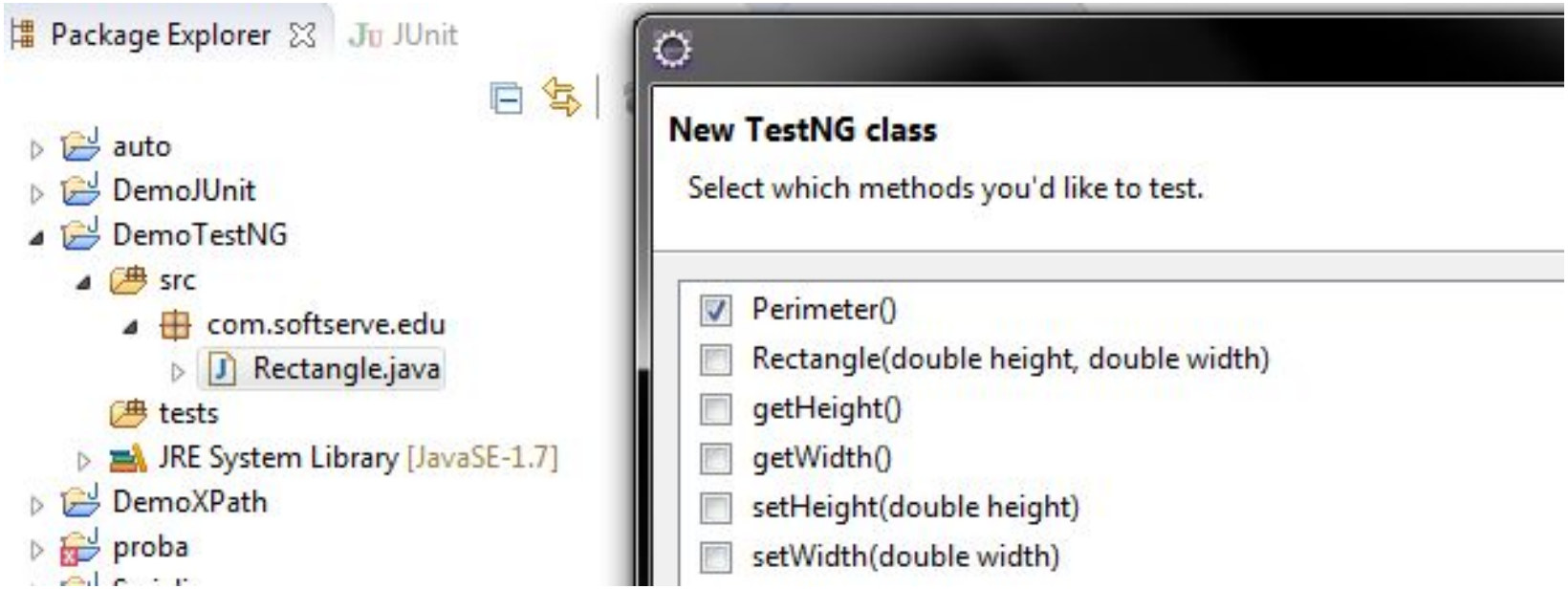
TestNG

- To create a new TestNG class, select the menu `File/New/TestNG`



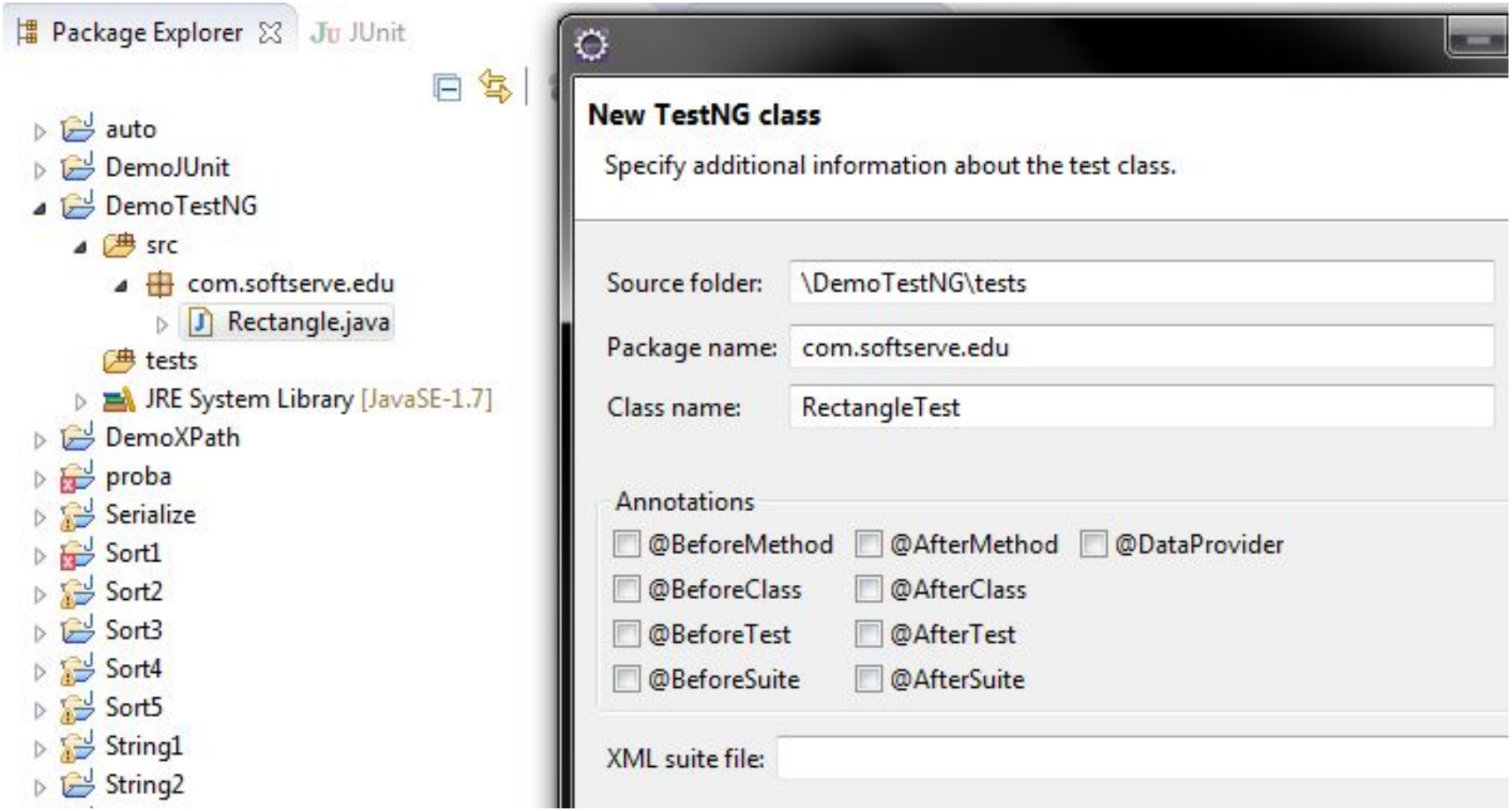
TestNG

- The first page of the wizard will show you a list of all the public methods of that class and it will give you the option to select the ones you want to test. Each method you select on this page will be included in the new TestNG class with a default implementation.



TestNG

- The next page lets you specify where that file will be created



TestNG

```
package com.softserve.edu;
import org.testng.AssertJUnit;
import org.testng.annotations.Test;
public class RectangleTest {
      @Test
       public void Perimeter() {
       Rectangle rectangle = new Rectangle(2, 3);
       double expected = 10;
       double actual;
       actual = rectangle.Perimeter();
       System.out.println(actual);
       AssertJUnit.assertEquals(expected, actual);
       // throw new RuntimeException("Test not implemented");
}     }
```

TestNG

- **Hierarchy** of tests
- All tests belong to a sequence of tests (suite), include a number of classes, each consisting of several test methods. Classes and test methods can belong to a group.

```
+- suite/
   +- test0/
   |  +- class0/
   |  |  +- method0(integration group)/
   |  |  +- method1(functional group)/
   |  |  +- method2/
   |  +- class1
   |     +- method3(optional group)/
   +- test1/
      +- class3(optional group, integration group)/
         +- method4/
```

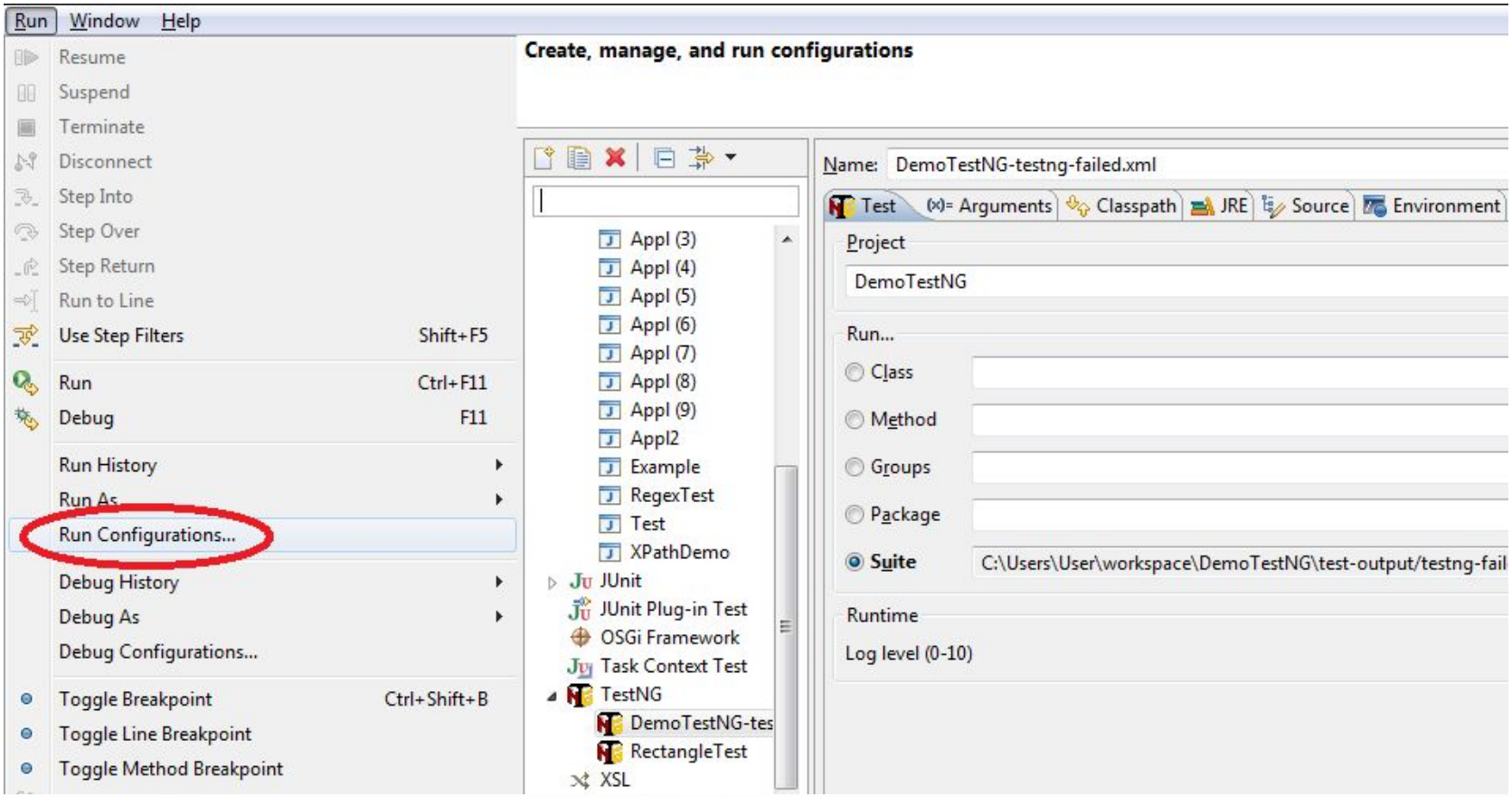
TestNG

- Every available before and after configurators. Started it all in the order

```
+- before suite/
   +- before group/
      +- before test/
         +- before class/
            +- before method/
               +- test/
            +- after method/
            ...
         +- after class/
         ...
      +- after test/
      ...
   +- after group/
   ...
+- after suite/
```

TestNG

- You can create a TestNG Launch Configuration. Select the Run / Debug menu and create a new TestNG configuration.

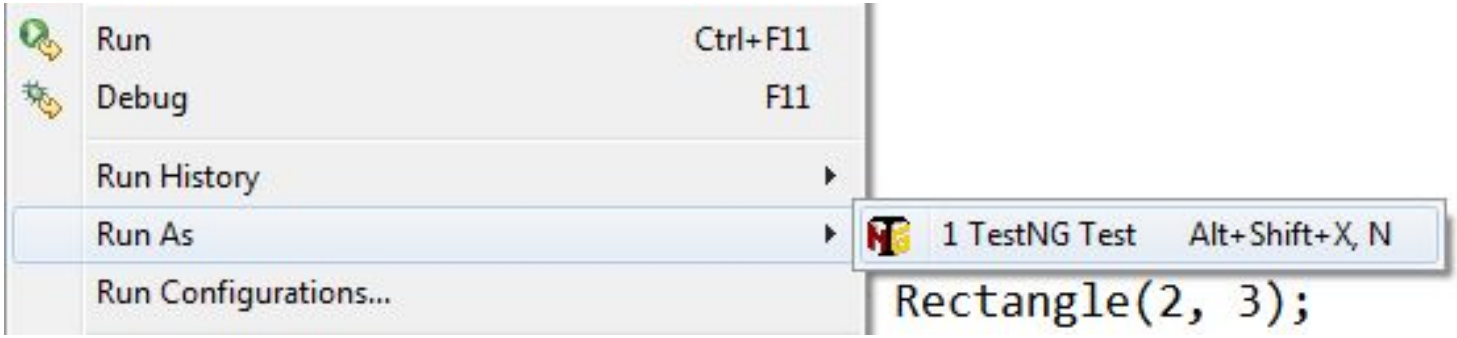


TestNG

- You should change the name of this configuration.
- Then you choose to launch your TestNG tests in the following ways
- From a class file
    - Make sure the box near Class is checked and then pick a class from your project.
    - This list only contains classes that contain TestNG annotations
- From groups
    - If you only want to launch one or several groups, you can type them in the text field or pick them from a list

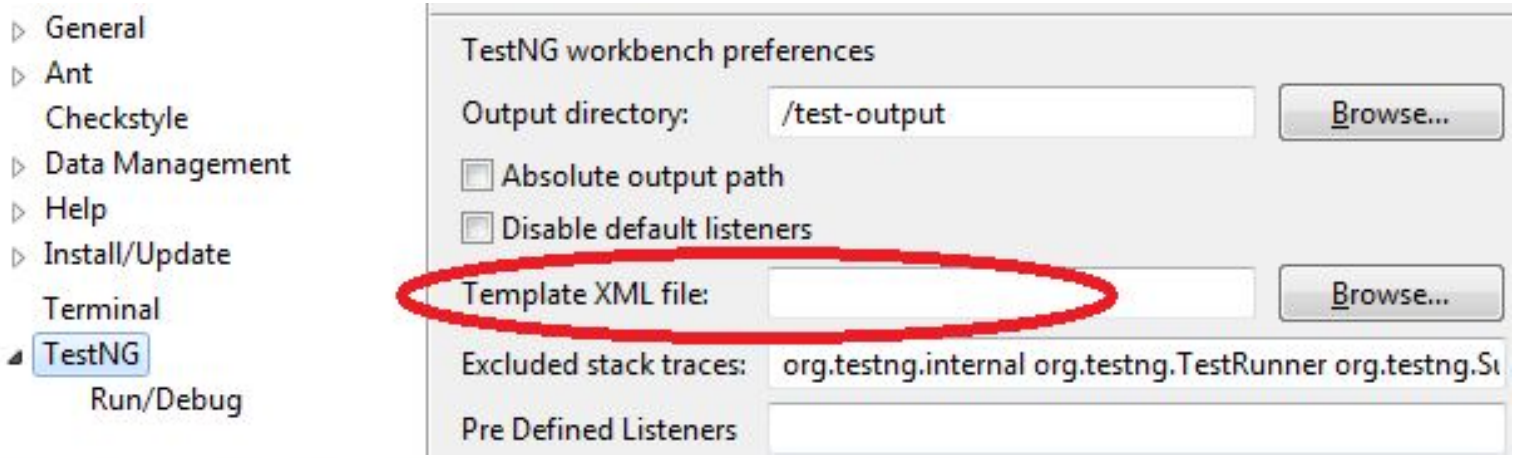
TestNG

- From a definition file
  - You can select a suite definition from your project.
  - It doesn't have to be named **testng.xml**, the plug-in will automatically identify all the applicable TestNG XML files in your project
- From a method
  - This launch isn't accomplished from the Launch dialog but directly from your Outline view



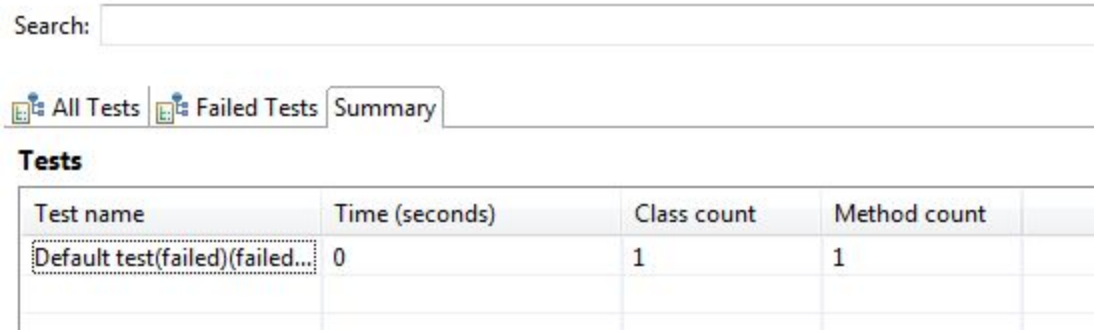
TestNG

- You start tests in many different ways: from an XML file, from a method, a class, etc.
- You can configure the settings for all the launches that are not done from an XML file.
- In order to give you access to the most flexibility, TestNG lets you specify an XML suite file for all these launches, which you can find in the Preferences menu.



TestNG

- The Summary tab gives you statistics on your test run, such as the timings, the test names, the number of methods and classes, etc.



- You can easily convert JUnit 3 and JUnit 4 tests to TestNG.

TestNG

- **Basic usage**

```
import java.util.*;
import org.testng.Assert;
import org.testng.annotations.*;
public class TestNGTest1 {
   private Collection collection;
   @BeforeClass
   public void oneTimeSetUp() {
      System.out.println("@BeforeClass - oneTimeSetUp");    }
   @AfterClass
   public void oneTimeTearDown() {
      // one-time cleanup code
      System.out.println("@AfterClass - oneTimeTearDown");    }
```

TestNG

```
@BeforeMethod
 public void setUp() {
    collection = new ArrayList();
    System.out.println("@BeforeMethod – setUp");
 }
 @AfterMethod
 public void tearDown() {
    collection.clear();
    System.out.println("@AfterMethod – tearDown");
 }
```

TestNG

```
    @Test
    public void testEmptyCollection() {
        Assert.assertEquals(collection.isEmpty(),true);
        System.out.println("@Test - testEmptyCollection");
    }
    @Test
    public void testOneItemCollection() {
        collection.add("itemA");
        Assert.assertEquals(collection.size(),1);
        System.out.println("@Test - testOneItemCollection");
    }
}
```

TestNG

- **Result**
- @BeforeClass – oneTimeSetUp
- @BeforeMethod – setUp
- @Test – testEmptyCollection
- @AfterMethod – tearDown
- @BeforeMethod – setUp
- @Test – testOneItemCollection
- @AfterMethod – tearDown
- @AfterClass – oneTimeTearDown

TestNG

- **Expected Exception Test**
- The `divisionWithException()` method will throw an ArithmeticException Exception, since this is an expected exception, so the unit test will pass.

```
import org.testng.annotations.*;
public class TestNGTest2 {
   @Test(expectedExceptions = ArithmeticException.class)
   public void divisionWithException() {
     int i = 1/0;
   }
}
```

TestNG

- **Ignore Test**
- TestNG will not test the `divisionWithException()` method

```
import org.testng.annotations.*;
public class TestNGTest3 {
    @Test(enabled=false)
    public void divisionWithException() {
      System.out.println("Method is not ready yet");
    }
}
```

TestNG

- **Time Test**
- The “Time Test” means if an unit test takes longer than the specified number of milliseconds to run, the test will terminated and mark as failed

```
import org.testng.annotations.*;
public class TestNGTest4 {
    @Test(timeOut = 1000)
    public void infinity() {
        while (true);
    }
}
```

TestNG

- **Suite Test**. The “Suite Test” means bundle a few unit test cases and run it together.
- In TestNG, XML file is use to define the suite test.

```
<!DOCTYPE suite SYSTEM
    "http://beust.com/testng/testng-1.0.dtd" >
<suite name="My test suite">
  <test name="testing">
    <classes>
      <class name="TestNGTest1" />
      <class name="TestNGTest2" />
    </classes>
  </test>
</suite>
```

- "TestNGTest1" and "TestNGTest2" will execute together

TestNG

- TestNG provides a "**Grouping**" feature to bundle few methods as a single unit for testing

```
import org.testng.annotations.*;
public class TestNGTest5 {
   @Test(groups="method1")
   public void testingMethod1() {
     System.out.println("Method - testingMethod1()");
   }
   @Test(groups="method2")
   public void testingMethod2() {
      System.out.println("Method - testingMethod2()");
   }
```

TestNG

```
    @Test(groups="method1")
    public void testingMethod1_1() {
        System.out.println("Method – testingMethod1_1()");
    }

    @Test(groups="method4")
    public void testingMethod4() {
        System.out.println("Method – testingMethod4()");
    }
}
// Every method is tie to a group.
```

TestNG

- You can execute the unit test with group “method1” only

```
<!DOCTYPE suite SYSTEM
     "http://beust.com/testng/testng-1.0.dtd" >
<suite name="My test suite">
  <test name="testing">
      <groups>
      <run>
        <include name="method1"/>
      </run>
    </groups>
    <classes>
       <class name="TestNGTest5" />
    </classes>
  </test>
</suite>
```

Result

Method - testingMethod1_1()
Method - testingMethod1()

TestNG

- **Parameterized Test**
- In TestNG, **XML** file or "**@DataProvider**" is used to provide vary parameter for unit testing.
- Declare "**@Parameters**" annotation in method which needs parameter testing, the parametric data will be provide by TestNG's XML configuration files. By doing this, you can reuse a **single test case** with different data sets easily.

```
import org.testng.annotations.*;
public class TestNGTest6 {
    @Test
    @Parameters(value="number")
    public void parameterIntTest(int number) {
      System.out.println("Parameterized Number is: " +
                                  number);
}   }
```

TestNG

```
<!DOCTYPE suite SYSTEM
                    "http://beust.com/testng/testng-1.0.dtd" >
<suite name="My test suite">
  <test name="testing">
    <parameter name="number" value="2"/>
    <classes>
      <class name="TestNGTest6" />
    </classes>
  </test>
</suite>
```

Result: Parameterized Number is : 2

TestNG

- Test cases occasionally may require complex data types, which **can't be represented** as a String or a primitive value in XML file. TestNG handles this scenario with **@DataProvider** annotation, which facilitates the mapping of complex parameter types to a test method.

```
import org.testng.annotations.*;
public class TestNGTest7 {
   @Test(dataProvider = "Data-Provider-Function")
   public void parameterIntTest(Class clzz, String[] number) {
      System.out.println("Parameterized Number is: " +
                                 number[0]);
      System.out.println("Parameterized Number is: " +
                                  number[1]);
   }
```

TestNG

```
// This function will provide the parameter data
   @DataProvider(name = "Data-Provider-Function")
   public Object[][] parameterIntTestProvider() {
      return new Object[][]{
        {Vector.class, new String[] {"java.util.AbstractList",
                                     "java.util.AbstractCollection"}},
        {String.class, new String[] {"1", "2"}},
        {Integer.class, new String[] {"3", "4"}}
        };
   }
}
```

```
Parameterized Number is : java.util.AbstractList
Parameterized Number is : java.util.AbstractCollection
Parameterized Number is : 1
Parameterized Number is : 2
Parameterized Number is : 1
Parameterized Number is : 2
```

TestNG

```
public class TestNGTest8 {
    private int number;
    private String msg;
    public void setNumber(int number){
        this.number = number;   }
    public int getNumber(){
        return this.number;   }
    public void setMsg(String msg){
        this.msg = msg;   }
    public String getMsg(){
        return this.msg;  }
}
```

TestNG

```
import org.testng.annotations.*;
public class TestNGTest9 {
   @Test(dataProvider = "Data-Provider-Function")
   public void parameterIntTest(TestNGTest8 clzz) {
      System.out.println("Number is: " + clzz.getMsg());
      System.out.println("Number is: " + clzz.getNumber());   }
  @DataProvider(name = "Data-Provider-Function")
   public Object[][] parameterIntTestProvider() {
       TestNGTest8 obj = new TestNGTest8();
       obj.setMsg("Hello");
       obj.setNumber(123);
       return new Object[][]{  { obj }  };
}  }
```

TestNG

- **Testing example**. **Converting** the character to ASCII or vice verse.

```
package com.softserve.edu;
public class CharUtils
{
    public static int CharToASCII(final char character){
        return (int)character;
    }
    public static char ASCIIToChar(final int ascii){
        return (char)ascii;
    }
}
```

TestNG

```
import org.testng.Assert;
import org.testng.annotations.*;
public class CharUtilsTest {
    @DataProvider
    public Object[][] ValidDataProvider() {
        return new Object[][]{
                { 'A', 65 },  { 'a', 97 },
                { 'B', 66 },  { 'b', 98 },
                { 'C', 67 },  { 'c', 99 },
                { 'D', 68 },  { 'd', 100 },
                { 'Z', 90 },   { 'z', 122 },
                { '1', 49 },   { '9', 57 },  };
    }
```

TestNG

```
    @Test(dataProvider = "ValidDataProvider")
    public void CharToASCIITest(final char character,
                                final int ascii) {
        int result = CharUtils.CharToASCII(character);
        Assert.assertEquals(result, ascii);     }
    @Test(dataProvider = "ValidDataProvider")
    public void ASCIIToCharTest(final char character,
                                final int ascii) {
        char result = CharUtils.ASCIIToChar(ascii);
        Assert.assertEquals(result, character);
}   }
```

TestNG

- **Dependency Test**. TestNG uses “dependOnMethods“ to implement the dependency testing. If the dependent method fails, all the subsequent test methods will be skipped, not marked as failed.
- The “**method2()**” will execute only if “**method1()**” is run successfully, else “**method2()**” will skip

```
import org.testng.annotations.*;
public class TestNGTest10 {
    @Test
    public void method1() {
      System.out.println("This is method 1");   }
    @Test(dependsOnMethods={"method1"})
    public void method2() {
       System.out.println("This is method 2");   }   }
```

TestNG. Parallel Tests Running

- **Multithreading**. Tests were performed simultaneously on multiple threads.

```
public class ConcurrencyTest1 extends Assert {
 private Map<String, String> data;
 @BeforeClass
 void setUp() throws Exception {
  data = new HashMap<String, String>();
 }
 @AfterClass
 void tearDown() throws Exception {
  data = null;
 }
```

TestNG. Parallel Tests Running

```
@Test(threadPoolSize = 30,
  invocationCount = 100,
  invocationTimeOut = 10000)
  public void testMapOperations() throws Exception {
   data.put("1", "111");     data.put("2", "111");
   data.put("3", "111");     data.put("4", "111");
   data.put("5", "111");     data.put("6", "111");
   data.put("7", "111");
   for (Map.Entry<String, String> entry : data.entrySet()) {
     System.out.println(entry);     }
   data.clear();
  }
```

TestNG

```
@Test(singleThreaded = true,  // run in a single thread
    invocationCount = 100,
    invocationTimeOut = 10000)
  public void testMapOperationsSafe() throws Exception {
    data.put("1", "111");     data.put("2", "111");
    data.put("3", "111");     data.put("4", "111");
    data.put("5", "111");     data.put("6", "111");
    data.put("7", "111");
    for (Map.Entry<String, String> entry : data.entrySet()) {
      System.out.println(entry);     }
    data.clear();   }
}
```

TestNG

```
public class ConcurrencyTest2 extends Assert {
  // some staff here
  @DataProvider(parallel = true)
  public Object[][] concurrencyData() {
    return new Object[][] {
      {"1", "2"},      {"3", "4"},
      {"5", "6"},      {"7", "8"},
      {"9", "10"},     {"11", "12"},
      {"13", "14"},    {"15", "16"},
      {"17", "18"},    {"19", "20"},
    };
  }
```

TestNG

- Set parallel option at the date of the provider in true.
- Tests for each data set to be launched in a separate thread.

```
@Test(dataProvider = "concurrencyData")
 public void testParallelData(String first, String second) {
   final Thread thread = Thread.currentThread();
   System.out.printf("#%d %s: %s : %s", thread.getId(),
                          thread.getName(), first, second);
   System.out.println();
 }
}
```