

JUnit

Unit Testing

Unit tests

- A *unit test* is code that tests the smallest unit of functionality.
- The percentage of code which is tested by unit tests is typically called *test coverage*.
- A unit test targets a small unit of code, e.g., a method or a class, (local tests).

Integration, functional, acceptance, performance

- Integration tests build on unit tests by combining the units of code and testing the resulting combination.
- Functional tests usually check a particular feature for correctness by comparing the results for a given input against the specification.
- An acceptance test is a particular type of functional test.
- Performance tests are used to benchmark software components in a repeatable way.

Using JUnit

- *JUnit* in version 4.x is a test framework which uses annotations to identify methods that specify a test.
- Typically these test methods are contained in a class called a *Test class*.

test method in a class

The following code shows a JUnit test method

.

```
@Test
public void testMultiply() {

    // MyClass is tested
    MyClass tester = new MyClass();

    // check if multiply(10,5) returns 50
    assertEquals("10 x 5 must be 50", 50, tester.multiply(10, 5));
}
```

test creation

- JUnit assumes that all test methods can be executed in an arbitrary order. Therefore tests should not depend on other tests.
- To write a test with JUnit you annotate a method with the *@org.junit.Test* annotation and use a method provided by JUnit to check the expected result of the code execution versus the actual result.

Annotation	Description
<code>@Test</code> public void method()	The <code>@Test</code> annotation identifies a method as a test method.
<code>@Test (expected = Exception.class)</code>	Fails if the method does not throw the named exception.
<code>@Test(timeout=100)</code>	Fails if the method takes longer than 100 milliseconds.
<code>@Before</code> public void method()	This method is executed before each test. It is used to prepare the test environment (e.g., read input data, initialize the class).
<code>@After</code> public void method()	This method is executed after each test. It is used to cleanup the test environment (e.g., delete temporary data, restore defaults). It can also save memory by cleaning up expensive memory structures.
<code>@BeforeClass</code> public static void method()	This method is executed once, before the start of all tests. It is used to perform time intensive activities, for example, to connect to a database. Methods marked with this annotation need to be defined as <code>static</code> to work with JUnit.
<code>@AfterClass</code> public static void method()	This method is executed once, after all tests have been finished. It is used to perform clean-up activities, for example, to disconnect from a database. Methods annotated with this annotation need to be defined as <code>static</code> to work with JUnit.
<code>@Ignore</code>	Ignores the test method. This is useful when the underlying code has been changed and the test case has not yet been adapted. Or if the execution time of this test is too long to be included.

Assert Statements

- JUnit provides static methods in the Assert class to test for certain conditions.
- These *assertion methods* typically start with assert and allow you to specify the error message, the expected and the actual result.
- An *assertion method* compares the actual value returned by a test to the expected value, and throws an AssertionError if the comparison test fails.

Statement	Description
fail(String)	Let the method fail. Might be used to check that a certain part of the code is not reached or to have a failing test before the test code is implemented. The String parameter is optional.
assertTrue([message], boolean condition)	Checks that the boolean condition is true.
assertFalse([message], boolean condition)	Checks that the boolean condition is false.
assertEquals([String message], expected, actual)	Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays.
assertEquals([String message], expected, actual, tolerance)	Test that float or double values match. The tolerance is the number of decimals which must be the same.
assertNull([message], object)	Checks that the object is null.
assertNotNull([message], object)	Checks that the object is not null.
assertSame([String], expected, actual)	Checks that both variables refer to the same object.
assertNotSame([String], expected, actual)	Checks that both variables refer to different objects.

JUnit Test Suite

If you have several test classes, you can combine them into a *test suite*. Running a test suite will execute all test classes in that suite in the specified order.

```
package com.orasi.intro;

import org.junit.runner.RunWith;

@RunWith(Suite.class)
@SuiteClasses({ TestPerson.class, TestCar.class })
public class AllTests {

}
```

Installing JUnit

- Eclipse allows you to use the version of JUnit which is integrated in Eclipse.
- If you want to control the used JUnit library explicitly, download JUnit4.x.jar from the JUnit website.

Creating Test Suite

To create a test suite in Eclipse, you select the test classes which should be included into this in the *Package Explorer* view

- right-click on them and select *New* → *Other...* → *JUnit* → *JUnit Test Suite*.



New JUnit Test Suite



JUnit Test Suite

Create a new JUnit Test Suite class for a package



☐ New JUnit 3 suite ☒ New JUnit 4 suite

Source folder: Intro/src



Browse...

Package: com.orasi.intro

Browse...

Name: AllTests

Test classes to include in suite:

- ☒  CarTest
- ☒  PersonTest

Select All

Deselect All

2 classes selected



< Back

Next >

Finish

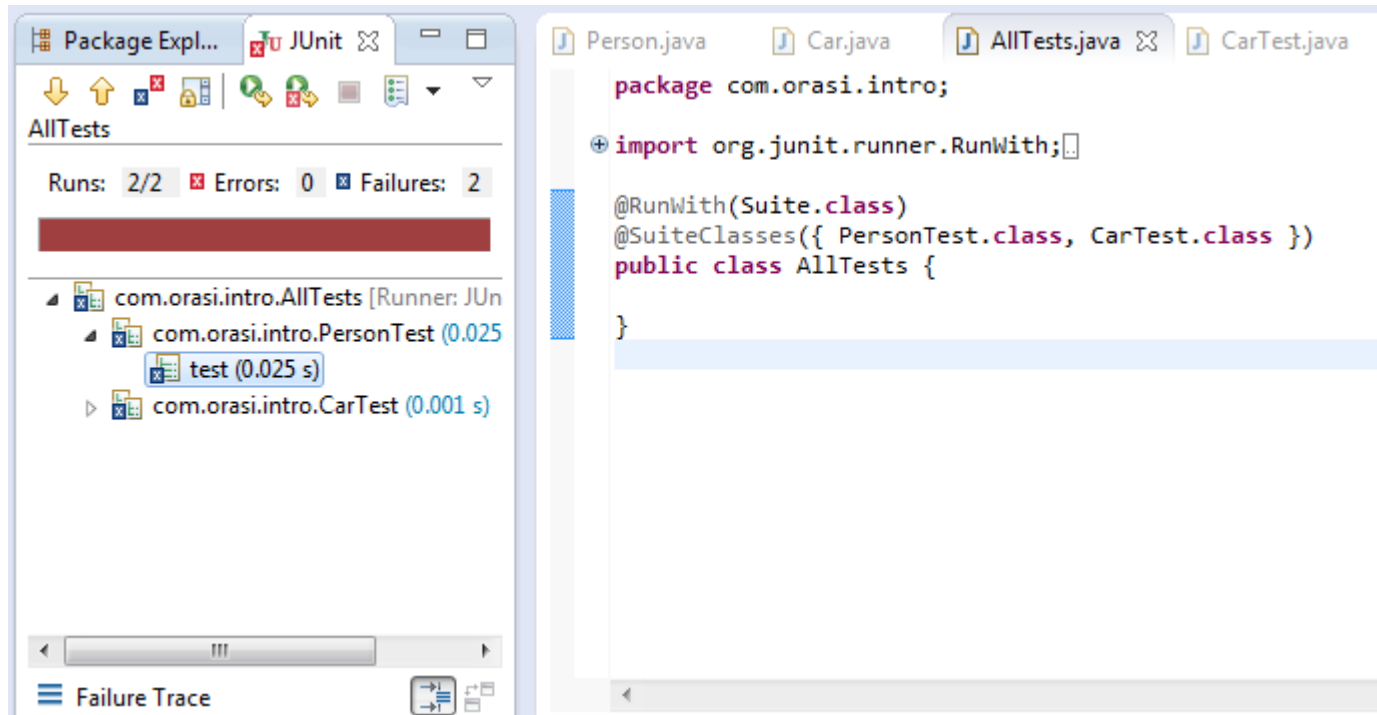
Cancel

Creating JUnit tests

- You can write the JUnit tests manually, but Eclipse supports the creation of JUnit tests via wizards.
- For example, to create a JUnit test class for an existing class, right-click on your new class, select this class in the *Package Explorer* view, right-click on it and select *New → JUnit Test Case*.

Running JUnit Tests

- To run a test, select the class which contains the tests, right-click on it and select *Run-as* → *JUnit Test*. This starts JUnit and executes all test methods in this class.
- Eclipse provides the **Alt+Shift+X, ,T** shortcut to run the test in the selected class. If you position the cursor on one method name, this shortcut runs only the selected test method.
- To see the result of an JUnit test, Eclipse uses the *JUnit* view which shows the results of the tests. You can also select individual unit tests in this view , right-click on them and select *Run* to execute



By default this view shows all tests. You can also configure, that it only shows failing tests or that the view is only activated upon failing tests.

Eclipse JUnit Content Assist

- JUnit uses static methods and Eclipse cannot always create the corresponding static import statements automatically.
- You can make the JUnit test methods available via the *Content Assists*. *Content Assists* is a functionality in Eclipse which allows the developer to get context sensitive code completion in an editor upon user request.
- Open the Preferences via *Window* → *Preferences* and select *Java* → *Editor* → *Content Assist* → *Favorites*.
- Use the *New Type* button to add the org.junit.Assert type. This makes, for example, the assertTrue, assertFalse and assertEquals methods directly available in the *Content Assists*.

Let's make some tests!

Create a new source folder *test*.

- right-click on your project, select *Properties* and choose *Java Build Path*
- Select the *Source* tab.
- Press the *Add Folder* button.
- Click the *Create New Folder* button.

Add this method to Person

```
public int getYearsToRetirement(int retirementAge) {  
    if (retirementAge > 100) {  
        throw new IllegalArgumentException("Hopefully you can retire by 100!");  
    }  
    return retirementAge + this.age;  
}
```

Create a JUnit test

- Right-click on your Person class in the *Package Explorer* view
- Select *New* → *JUnit Test Case*.
- In the following wizard ensure that the *New JUnit 4 test* flag is selected
- Set the source folder to *test*, so that your test class gets created in this folder.
- Check `setUpBeforeClass` and `tearDownAfterClass`
- Press the *Next* button and select the methods that you want to test.
 - Select `getYearsToRetirement`
- If the JUnit library is not part of the classpath of your project, Eclipse will prompt you to add it.

```

import static org.junit.Assert.*;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;

public class PersonTest {

    @BeforeClass
    public static void testSetup() {
    }

    @AfterClass
    public static void testCleanup() {
        // Teardown for data used by the unit tests
    }

    @Test(expected = IllegalArgumentException.class)
    public void testExceptionIsThrown() {
        Person tester = new Person();
        tester.getYearsToRetirement(65);
    }

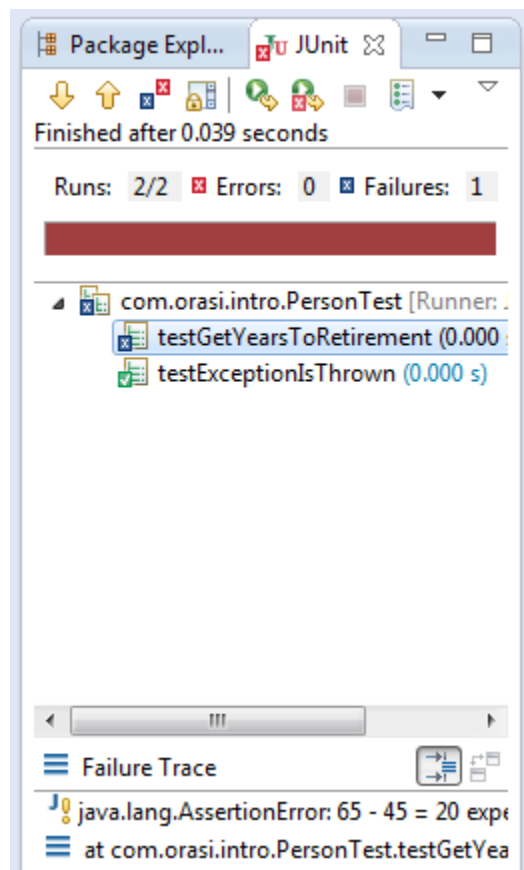
    @Test
    public void testGetYearsToRetirement() {
        Person tester = new Person();
        tester.setAge(45);
        assertEquals("65 - 45 = 20", 20, tester.getYearsToRetirement(65));
    }
}

```

Create test with this code. Adding other @Test and filling out stubbed @Test

Run your test

- Right-click on your new test class
- Select *Run-As* → *JUnit Test*
- The result of the tests will be displayed in the JUnit view
- One test should be succesful and one test should show an error.
- This error is indicated by a red bar.



Fix the bugs

- The test is failing, because our Person class is currently not working correctly.
- It does an addition instead of subtraction.
- Also need a value over 100 to test Exception
- Fix the bugs and re-run the test to get a

End