JUnit tests



Object Oriented Programming

https://softeng.polito.it/courses/09CBI



Version 4.3.1 © Marco Torchiano, 2021









JUnit

- JUnit is a testing framework for Java programs
 - Written by Kent Beck and Erich Gamma
- It is a framework with unit-testing functionalities
- Integrated in Eclipse development Environment



http://www.junit.org

Unit Testing

- Unit testing is particularly important when software requirements change frequently
 - Code often has to be refactored to incorporate the changes
 - Unit testing helps ensure that the refactored code continues to work

JUnit Framework

- JUnit helps the programmer:
 - Define and execute tests and test suites
 - Formalize requirements and clarify architecture
 - Write and debug code
 - Integrate code and always be ready to release a working version

History

- 1997 on the plane to OOPSLA97 Kent Beck and Erich Gamma wrote JUnit
- Junit.org August 2000
- Junit 3.8.1 September 2002
- Junit 4.0 February 2006
 - ◆ Latest release: 4.12 Dec 2012
- Junit 5.0 September 2017

What JUnit does

- For each test (method) JUnit
 - calls pre-test fixture

method executed before executing any other method test

- Intended to acquire resources and create any objects that may be needed for testing
- calls the test method
 - Checks the output of the element under test
- calls post-test fixtures
 - Intended to release resources and remove any objects created that is no longer needed

Test method

- A test method doesn't return a result void
- It performs operations on the code under test and checks the results
- Checks are performed using a set of assert*() methods
- The JUnit framework detects the anomalies and reports them

Standard assert* () methods

```
assertTrue (boolean test)
assertFalse(boolean test)
assertEquals (expected, actual)
assertSame(Object exp, Object actual)
assertNotSame(Object exp, Object act)
assertNull(Object object)
assertNotNull(Object object)
fail()
```

Standard assert* () methods

For a condition

assertTrue (condition)

- If the tested condition is
 - true => proceeds with execution
 - false => aborts the test method execution,
 prints out the optional message

Standard assert*()

For objects, int, long, byte:

```
assertEquals( expected, actual)
```

- EX. assertEquals(2 , unoStack.size());
- For floating point values:

```
assertEquals (expected, actual, err)
```

EX. assertEquals(1.0, Math.cos(3.14),
 0.01);

Standard assert* w/message

 All the assertion methods may take an optional addition message argument, e.g.

```
static void assertTrue(
JUnit 3 / 4
       String message, boolean test)
     static void assertTrue(
         boolean test, String message )
     static void assertTrue(
                                  JUnit 5
         boolean test,
       Supplier<String> messageSupplier)
```

Testing exceptions

- In presence of exceptions two main cases shall be checked:
 - a normal behavior is expected, therefore no exception should be thrown
 - In this case the tests fails if that exception is raised
 - an anomaly is expected, therefore an exception should be thrown
 - In this case the tests fails if NO exception is detected

Fixtures

- Portions of test cases that are cloned
 - For initializing or releasing resources
- They are called fixtures
 - Pre-test (set up)
 - Post-test (tear down)
- Collected in separate methods
 - Avoid duplication
 - Shorten test cases
 - Automatically executed before tests

Sample class: Stack

```
public class Stack {
private int[] stack; private int next = 0;
public Stack() { this(10);}
public Stack(int size) { stack = new int[size];}
public boolean isEmpty() { return next==0; }
public boolean push(int i) {
  if(next==stack.length) return false;
  stack[next++] = i;
  return true; }
public int pop() { return stack[--next]; }}
```

JUnit 3 SYNTAX

Test a Stack

```
public class StackTest extends TestCase {
 testSomething
   Stack aStack = new Stack();
   assertTrue("Should be empty at first!",
             aStack.isEmpty());
   aStack.push(10);
   assertTrue ("Should not be empty!",
               !aStack.isEmpty());
   aStack.push(-4);
   assertEquals(-4, aStack.pop());
   assertEquals(10, aStack.pop());
                      One or more assertions
```

to check results

Test a Stack

```
public void testStackEmpty() {
  Stack aStack = new Stack();
  assertTrue ("Stack should be empty!",
              aStack.isEmpty());
  aStack.push(10);
  assertFalse ("Stack should not be empty!",
               aStack.isEmpty());
public void testStackOperations() {
  Stack aStack = new Stack();
  aStack.push(10);
  aStack.push(-4);
  assertEquals(-4, aStack.pop());
  assertEquals(10, aStack.pop());
```

Test case execution

- Running a test case
 - Executes all methods
 - public
 - Returning void
 - Name starting with "test"
 - With no arguments ()
 - Ignores the rest of methods
- The class can contain helper methods provided they
 - are not public or
 - do not start with "test"

Creating a test class

- Define a subclass of TestCase
- Override the setUp() method to initialize object(s) under test.
- Override the tearDown() method to release object(s) under test.
- Define one or more public testxxx() methods that exercise the object(s) under test and assert expected results.

Pre-test fixture

- Override setUp() to initialize the variables, and objects
 - Implements a initialization fixture
- Since setUp() is your code, you can modify it any way you like (such as creating new objects in it)
 - Typically it initializes instance attributes that are later used by test methods

Post-test fixture

- In most cases, the tearDown() method doesn't need to do anything
 - The next time you run setUp(), your objects will be replaced, and the old objects will be available for garbage collection
 - Like the finally clause in a try-catch—finally statement, tearDown() is where you would release system resources (such as streams)

Expected exception test

```
try{
    // e.g. method invoked with "wrong" args
    obj.method(null);
    fail("Method didn't detected anomaly");
}catch(PossibleException e) {
    assertTrue(true); // OK
}
```

Unexpected exception test

```
public void testSomething()
    throws PossibleException {
    // e.g. method invoked with right args
    obj.method("Right Argument");
}
```

```
Exception → Error

Runs: 2/2 
Errors: 1 
Failures: 0
```

Unexpected exception test

```
try{
    // e.g. method invoked with right args
    obj.method("Right Argument");
    assertTrue(true); // OK
}catch(PossibleException e) {
    fail("Method should not raise except.");
}
```

```
Exception → Failure

Runs: 2/2 
Errors: 0 
Failures: 1
```

TestSuite

- Allow running a group of related tests
- To do so, group your test methods in a class which extends TestSuite

```
public class AllTests extends TestSuite {
  public static TestSuite suite() {
    TestSuite suite = new TestSuite();
    suite.addTestSuite(StackTester.class);
    suite.addTestSuite(AnotherTester.class);
}
```

Junit 4 SYNTAX

JUnit 4



- Makes use of java annotations
 - Less constraints on names
 - Easier to read/write
- Backward compatible with JUnit 3
- Assertions
 - * assert*() methods
 - * assertThat() method
 - To use the Hamcrest matchers

Test a Stack (JUnit4)

Any class



```
public class TestStack {
                                         @Test annotation
 @Test
 public void testStack() {
                                            you distinguish them with the @Test annotation
    Stack aStack = new Stack();
    assertTrue ("Stack should be empty",
                                             //checking if the return value of the method is true
                   aStack.isEmpty());
   aStack.push(10);
    assertFalse ("Stack should not be empty!",
                   aStack.isEmpty());
                                           //checking if the return value of the method is false
    aStack.push(-4);
   assertEquals(-4, aStack.pop());
    assertEquals(10, aStack.pop());
                                        //checking if the return value of the method is equal to
                                   One or more assertions
                                   to check results
```

Running a test case



- The JUnit runner executes all methods
 - ◆ Annotated with "@Test"
 - public
 - Returning void
 - With no arguments ()
 - Ignores the rest of the class
- The class may contain any helper methods provided they are
 - Not annotated
 - Not public

The pre-test fixture



- Annotate a method with @Before to make it a pre-test fixture:
 - It is executed before each test method is run
 - It is intended to initialize the objects that will be used by test methods
- There is no limit to the setup you can do in a pre-test method
- Helps reducing duplication of code

The post-test fixture



- Annotate a method with @After to make it a post-test fixture
 - It is executed after each test method is run
 - It is intended to release system resources (such as streams)
- In most cases, a post-test fixture is not required
 - Before the next test is executed the pre-test fixture is run again so attribute will be reinitialized

Expected exception test



```
@Test(expected=PossibleException.class)
public void testSomething()
         throws PossibleException {
        // e.g. method invoked with "wrong" args
        obj.method("Wrong Argument");
}
```

Unexpected exception test



```
@Test
public void testSomething()
        throws PossibleException {
   // e.g. method invoked with right args
   obj.method("Right Argument");
                                   Failute: the test can be executed but it is not passed
                                   Error: during the execution there is a problem that prevent the test to run
                                 Exception \rightarrow Error
                 Runs: 2/2 Errors:

☑ Failures: 0
```

TestSuite



• Allows running a group of related tests as a single batch:

in the specified order

```
@RunWith(Suite.class)
@SuiteClasses({
    TestStack.class, AnotherTest.class
})
public class AllTests { }
```

JUnit 4 Annotations



- @Test
 - Marks test methods
- @Before and @After
 - Mark pre and post fixtures
- Test suites require:
- @RunWith(Suite.class)
- @Suite.SuiteClasses({ ... })

JUnit 4 Imports



- All classes are in package org.junit
- Assertions are made available with
 - import static org.junit.Assert.*;
- Annotations must be imported as
 - * import org.junit.After;
 - import org.junit.Before;
 - * import org.junit.Test;
- Suites require:
 - import org.junit.runners.Suite;
 - import org.junit.runners.Suite. SuiteClasses;

typically only: import static org.junit.Assert.*; import org.junit.Test;



JUnit 5



- Uses Java annotations but different from 4:
 - ◆ @Before/@After → @BeforeEach/@AfterEach
 - ◆ @BeforeClass/@AfterClass → @BeforeAll/@AfterAll
- Test methods not necessarily public
- Java8 Lambda support, extensions, parameterized tests, etc.
- Suites
 - @RunWith(JUnitPlatform.class)
 - * @SelectClasses({ ... })
 - * @SelectPackages({ ... })

Test a Stack (JUnit5)

Any class

```
public class TestStack {
                                  @Test annotation
 @Test
 public void testStack() {
                                         Message as
   Stack aStack = new Stack();
                                         last argument
   assertTrue(aStack.isEmpty(),
              "Stack should be empty");
   aStack.push(10);
   assertFalse(aStack.isEmpty(),
              "Stack should not be empty!");
   aStack.push(-4);
   assertEquals(-4, aStack.pop());
   assertEquals(10, aStack.pop());
                           One or more assertions
                           to check results
```

6

Expected exception test

```
@Test
public void testSomething() {
    // e.g. method invoked with "wrong" args
    assertThrows (PossibleException.class, () -> {
        obj.method("Wrong Argument")
    });
}
```

6

Unexpected exception test

```
@Test
public void testSomething()
    throws PossibleException {
    // e.g. method invoked with right args
    obj.method("Right Argument");
}
```

```
Exception → Error

Runs: 2/2 ☐ Errors: 1 ☐ Failures: 0
```

TestSuite



- Indicate the JUnitPlatform runner
- Select classes with SelectClasses

```
@RunWith(JUnitPlatform.class)
@SelectClasses({
   TestStack.class, AnotherTest.class
})
public class AllTests { }
```

JUnit 5 Imports



- All in package org.junit.jupiter.api
- Assertions are made available with
 - * import static
 org.junit.jupiter.api.Assertions.*;
- Annotations have to be imported as
 - import org.junit.jupiter.api.AfterEach;
 - import org.junit.jupiter.api.BeforeEach;
 - import org.junit.jupiter.api.Test;
- Suites require:
 - import org.junit.platform.runner.JUnitPlatform;
 - import org.junit.platform.suite.api.SelectClasses;

ASSERTION STYLES

Assertions Styles

- Different ways of writing assertions
 - Standard JUnit assert methods
 - Hamcrest matchers
 - AssertJ fluent assertions

Hamcrest matchers

A single assert method:

```
assertThat(value, matcher)
```

 Accepts the actual value and a matcher, e.g.

```
assertThat(res, is(equalTo(expect)));
```

- This is equivalento to:
 - assertEquals(expect, res);
- More readable, better message

Hamcrest matchers - Object

- equalTo()
 - test object equality using Object.equals
- instanceOf()
 - test type
- notNullValue(), nullValue()
 - test for null
- sameInstance()
 - test object identity

Hamcrest matchers - Numbers

- closeTo()
 - test floating point values are close to a given value
- greaterThan(),
 greaterThanOrEqualTo(),
 lessThan(), lessThanOrEqualTo()
 - test ordering

Hamcrest matchers - Logical

allOf()

 matches if all matchers match, short circuits (like Java &&)

anyOf()

 matches if any matchers match, short circuits (like Java ||)

not()

 matches if the wrapped matcher doesn't match and vice versa

Hamcrest matchers - String

- equalToIgnoringCase()
 - test string equality ignoring case
- equalToIgnoringWhiteSpace()
 - test string equality ignoring differences in runs of whitespace
- containsString(), endsWith(), startsWith()
 - test string matching

Hamcrest matchers - Collections

- array()
 - * test an array's elements against an array
- hasItemInArray()
 - test an array contains an element
- hasItem(), hasItems()
 - test a Collection contains elements
- hasEntry(), hasKey(), hasValue()
 - test a Map contains an entry, key or value

Hamcrest imports

Required imports

```
import static
org.hamcrest.MatcherAssert.assertThat;import static org.hamcrest.Matchers.*;
```

 The jar with only core matchers is included in Eclipse when using the Java4 library

AssertJ Fluent

A single assert builder method:

assertThat(actual)

- Accepts the actual value
- Returns an Assert object the provides building methods
- Method as () can be used to define a contextual message in case of failure
- More readable, better messages

AssertJ Fluent example

```
assertThat(res).
  as("Checking return value").
  isNotNull().
  isEqualTo(expected);
```

• Equivalent to:

Using AssertJ

- Download latest jar from Maven
 - https://search.maven.org/remotecontent ?filepath=org/assertj/assertjcore/3.12.2/assertj-core-3.12.2.jar
- Include the jar in the classpath
- Import the static definitions:

```
import static org.assertj.core.api.Assertions.*;
```

AssertJ – Object

- isEqualTo()
 - test object equality using Object.equals
- isInstanceOf()
 - test type
- isNotNull(), isNull()
 - test for null
- isSameAs()
 - test object identity

AssertJ – Object

- returns()
 - test a value is returned by Function
- hasFieldOrPropertyWithValue()
 - Retrieves value (using reflection) looking for:
 - Getter: field → getField()
 - Field: attribute in class
- extracting()
 - Applies extractor Function and produces an object or a list that becomes the actual value

Checking method return

Equivalent assertions:

```
assertThat(new Counter())
.isNotNull()
.hasFieldOrPropertyWithValue("value", 0)
.returns(0, Counter::getValue)
.extracting(c -> c.getValue())
          .isEqualTo(0);
```

```
fava.lang.AssertionError: [Initial counter state]
Chec
          Expecting
            <TestExampleAsserJ$Counter@3339ad8e>
           to have a property or a field named <"value"> with value
            <0>
           but value was:
  asser
            <1>
   .isNotNull()
   .hasFieldOrPropertyWithValue("value", 0)
   .returns (O
                org.junit.ComparisonFailure: [Initial counter state]
                expected:<[0]> but was:<[1]>
   .extracting(c
                            getValue())
                 .isEqualTo(0);
```

AssertJ – Numbers

- isCloseTo()
 - test fp value is close to a given value
- isGreaterThan(),
 isGreaterThanOrEqualTo(),
 isLessThan(),
 isLssThanOrEqualTo()
 - test ordering
- isBetween()
 - test actual value is in range

AssertJ – Strings

- startsWith(),
 endsWith()
 - test endings
- contains(),
 doesNotContain()
 - test content contains a substring
- isEqualToIgnoringCase()
 - test content

AssertJ - Containers

- hasSize(), hasSizeLessThan(), hasSizeGreaterThan(),...
 - test collection size
- isSorted()
 - test if list is sorted
- contains(), containsExactly(),
 - test that elements are present
- extracting()
 - Map each element to a list
- filteredOn()
 - Filters the element of the list

AssertJ - Maps

- containsKeys(), containsEntry()
 - test if list is sorted
- contains()
 - checks entries, that can be defined with entry()
- extractingFromEntries()
 - Map each entry to a list or a list of tuples that can be matched with a contains of tuple()

Maps example

```
assertThat(turnout).as("Turnout map").
contains (
  entry("PIEMONTE", 67.45),
  entry("SICILIA", 42.88),
  entry("UMBRIA", 70.50)
```

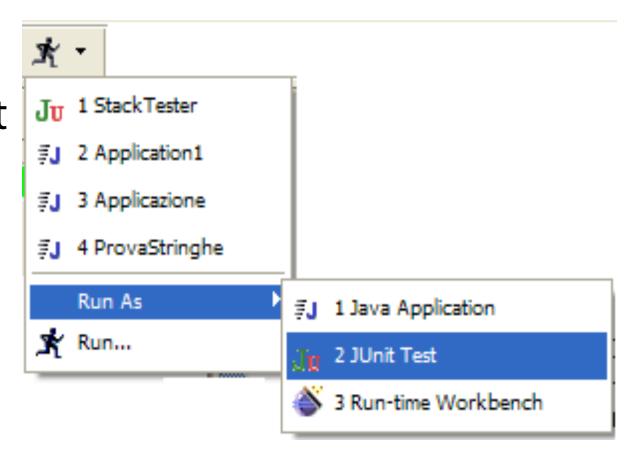
AssertJ – Exceptions

- assertThatThrownBy()
 - Create an assert on the thrown exception
- hasMessage()
 - Checks the message of exception

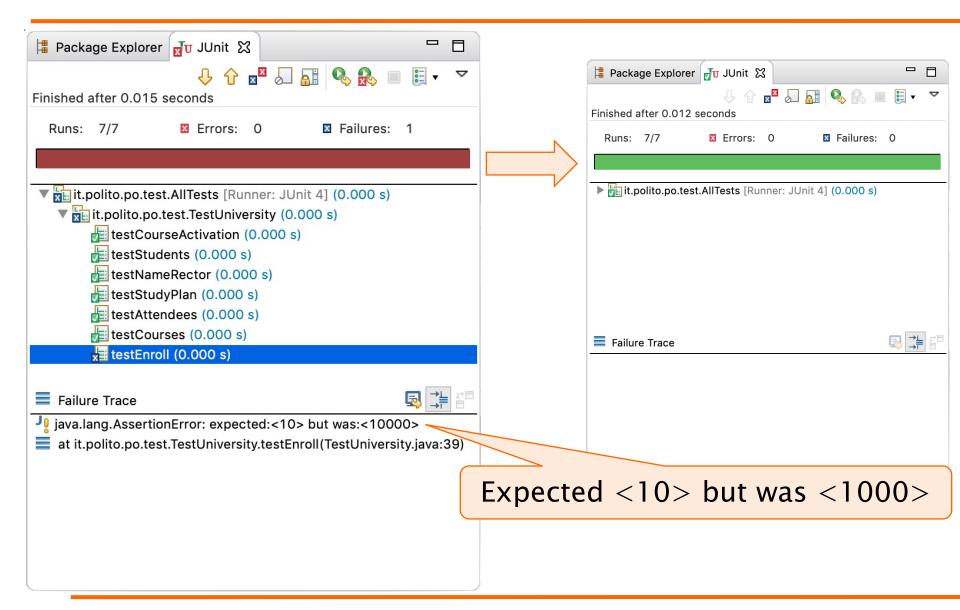
ECLIPSE JUNIT PLUG-IN

Junit in Eclipse – Run as JUnit Test

- Run
- Run As...
- Junit Test



Red Bar - Green Bar



Failures vs. Errors

Failure

- An assert*() method found the condition it has to check is not verified
- The program produced an output, but it is not the expected one

Error

- During the execution of the tests an error was found (e.g. NullPointerException)
- The program could not produce any output due to an error

...use JUnit

Keep the bar green to keep the code clean...



JUnit in Eclipse - Path Setup

- When creating a new test case
 - Eclipse suggests adding the JUnit library
- When importing a test (e.g. in a jar),
 the library must be added explicitly
 - open project's property window
 - Select java build path
 - Then Libraries
 - Select Junit, the suitable version

Organizing Tests in Eclipse

- Separate source folder (test)
 - Place tests in a separate source folder
 - Allows clear separation
 - Add JUnit library to the project
- Separate project
 - Place tests inside a separate project
 - Refer to the primary project
 - Add JUnit library to the new project
 - No test libraries are added to your primary project

USING JUNIT

Test-Driven Development

- Specify part of a feature yet to be coded
- Run the test and see it fail (red bar)
- Write code until the test pass (green bar)
- Repeat until whole feature implemented
- Refactor while maintaining the bar green

Keep your code clean: keep the bar green

Bug reproduction

- When a bug is reported
- Specify the expected correct outcome
- See the test fail
 - I.e. reproduce the bug
- Modify the code and adjust it until the bug-reproducing tests pass.
- Check for regressions
 - With the existing test suites

Guidelines

- Test should be written before code
- Test everything that can break
- Run tests as often as possible

Whenever you are tempted to type something into a print statement or a debugger expression write it as a test

- M.Fowler

Limitations of unit testing

- JUnit is designed to call methods and compare the results they return against expected results
 - This ignores:
 - Programs that do work in response to GUI commands
 - Methods that are used primary to produce output

Limitations of unit testing...

- Heavy use of JUnit encourages a "functional" style, where most methods are called to compute a value, rather than to have side effects
 - This can actually be a good thing
 - Methods that just return results, without side effects (such as printing), are simpler, more general, and easier to reuse

Summary: elements of JUnit

- assert*()
 - Assertion functions
- Test cases
 - Are implemented by methods in test classes
- TestSuite
 - Class containing a sequence of TestCase

References

- K.Beck, E.Gamma. Test
 Infected: Programmers Love Writing Tests
 - http://members.pingnet.ch/gamma/junit.htm
- Junit home page
 - https://junit.org
- Hamcrest matchers
 - http://hamcrest.org/JavaHamcrest/
- AssertJ Fluent assertions
 - http://joel-costigliola.github.io/assertj/index.html