### Junit & Mockito

### JUnit is a testing framework for Java

Use: It is a simple framework to write repeatable tests

* A test case is a program written in Java
* JUnit is linked as a JAR at compile-time
* Integrate JUnit in your project (with Maven)

**With Maven**

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

...

</dependencies>

* Without Maven: add **junit.jar** on your classpath

**Junit:key concepts**

* JUnit is based on **Java annotations**
* Java annotations are a form of metadata, provide data about a program that is not part of the program itself.
* Java annotations have several uses:
  + Information for the compiler
  + Compile-time and deployment-time processing
  + Runtime processing

**Junit most used annotations**

* @org.junit.Test
* @org.junit.BeforeClass
* @org.junit.Before
* @org.junit.AfterClass
* @org.junit.After

Importorg.junit.\*;

publicclasstestClass1{

@BeforeClass

public static void setUpClass() throws Exception {

// Code executed before the first test method

}

@Before

public void setUp() throws Exception {

// Code executed before each test

}

@AfterClass

public static void tearDownClass() throws Exception {

// Code executed after the last test method

}

@After

public void tearDown() throws Exception {

// Code executed after each test

}

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

**Junit assertions**

JUnit provides **assertion methods** for all primitive types and Objects and arrays

* In these methods the **expected value** is compared with the

### actual value.

* The parameter order is:
  + Optional: a string that is output on failure
  + expected value
  + actual value

**Junit assertions**

### import static org.junit.Assert.\*;

assertEquals("failure - strings not equal", "text",

### "text");

assertFalse("failure - should be false", false); assertSame("should be same", number, number); assertArrayEquals("failure - byte arrays not same", expected, actual);

**Mockito**

**Introduction to Mockito**

* **What is Mockito?**

**Mockito** is a Java framework allowing the creation of **mock objects** in automated unit tests

* A **mock object** is a dummy implementation for an interface or a class in which you define the output of certain method calls.

**Why Mocking**

* Some “real” objects required in Unit tests are really complex to instantiate and/or configure
* Sometimes, only interfaces exist, implementations are not even coded.
* If you use Mockito in tests you typically:
  + Mock away external dependencies and insert the mocks into the code under test
  + Execute the code under test
  + Validate that the code executed correctly

**How to use Mockito**

* Integrate Mockito in your project with Maven
  + With Maven

<dependencies>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-all</artifactId>

<version>1.10.19</version>

<scope>test</scope>

</dependency>

...

</dependencies>

* + Without Maven: add Mockito JARs on your classpath

**Mocking a class**

**Import static org.mockito.**Mockitoto.\*; **import static org.junit.**Assert.\*;

**@Test**

**public void test1() {**

//create Mock

**MyClass test = mock(MyClass.class);**

// define return value for method getUniqueId()

**when(test.getUniqueId()).thenReturn(43);**

// use mock in test....

**assertEquals(test.getUniqueId(), 43);**

**}**

**Argument matchers :** Mockito verifiesargument values by using an equals() method

* When flexibility is required then you should use **argument matchers**

//stubbing using anyInt() argument matcher

**when(mockedList.get(anyInt())).thenReturn("element");**

//verify using an argument matcher

**verify(mockedList).get(anyInt());**

* Other argument matchers: anyString(), anyObject(), anyVararg(), …

**Note:** If you are using argument matchers, all arguments have to be provided by matchers

**Mockito:spy**

With Mockito you can **spy** a real class. When you use the spy then the real methods are called (unless a method was stubbed) **List<String> list = new LinkedList<>();**

**List<String> spy = spy(list);**

//optionally, you can stub out some methods:

**when(spy.size()).thenReturn(100);**

//using the spy calls \*real\* methods

**spy.add("one");**

**spy.add("two");**

//prints "one" - the first element of a list

**System.out.println(spy.get(0));**

//size() method was stubbed - 100 is printed

**System.out.println(spy.size());**

//optionally, you can verify

**verify(spy).add("one");**

**verify(spy).add("two");**