Playtika

AGENDA

1. What is unit testing?
2. What is unit testing framework?
3. TestNG overview: setup and usage

WHAT IS UNIT TESTING?

Unit testing – method of testing when tested application splitted on small separated pieces (units), that tests independently

Goal of unit testing: make sure that **every individual part** of application works as expected

Unit of testing can have different explanations depending on programming approach

- **Procedural** approach means testing of modules or separated procedures/functions
- **Object-oriented** approach – testing of interface (class), but individual methods also can be tested

ADVANTAGES

1. Allows to detect problem early
2. Makes easier code changing
3. Simplifies integration (integration testing, bottom-up testing)
4. Live documentation
5. Design

LIMITATIONS AND DISADVANTAGES

1. Unit testing can’t help to catch all kind of errors
2. Unit testing should be performed with other testing types
3. Unit testing != integration testing
4. Time-consuming
5. It’s hard to create realistic and useful test

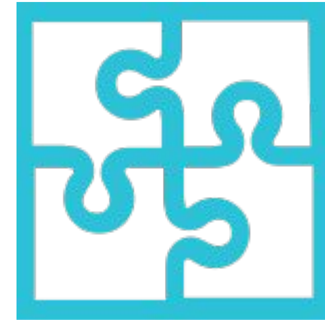
SIGNS OF GOOD UNIT TEST

- Able to be fully **automated**
- Has full control over all the pieces running (Use mocks or stubs to achieve this isolation when needed)
- Can be run in any **order** if part of many other tests
- Runs in **memory** (no DB or File access, for example)
- **Consistently** returns the same result (You always run the same test, so no random numbers, for example. save those for integration or range tests)
- Runs **fast**
- Tests a **single logical concept** in the system
- **Readable**
- **Maintainable**
- **Trustworthy** (when you see its result, you don't need to debug the code just to be sure)

WHAT IS A TESTING FRAMEWORK?

A test framework is a software tool for writing and running unit-tests that provides reusable test functionality which:

- Enables automatic execution for regression tests
- Is standardized
- Easy to use
- Test report generation



TYPICAL UNIT TESTING FRAMEWORKS COMPONENTS

1. Test runner
2. Test case
3. Test fixtures (preconditions)
4. Test suites
5. Execution
6. Test result formatter
7. Assertions

UNIT TESTING STAGES (EXECUTION WORKFLOW)

1

SET UP

Creates an instance of the object to be tested, referred to as SUT (System Under Test)

2

EXECUTION

Invoking SUT methods. Saving outcome results to local variable

3

VERIFY

Verifying outcome results. Comparing actual with expected

Note there is at least two approaches:

one-assert-per-test and *single-concept-per-test*

4

TEAR DOWN

Cleanup persistent changes that can affect workflow of following tests

Playtika

UNIT TESTING FRAMEWORKS FOR JAVA

JUnit

JUNIT AND TESTNG FEATURES

Feature	JUnit	TestNG
Annotation Support	Y	Y
Exception Test	Y	Y
Ignore Test	Y	Y
Timeout Test	Y	Y
Suite Test	Y (Java class)	Y (XML)
Group Test	N (@Category as alt.)	Y
Parameterized (primitive)	Y	Y
Parameterized (object)	N	Y
Dependency Test	N (Only sorting by name)	Y

TEST EXECUTION CONTROL

Feature	JUnit	TestNG
Before suite	N/A	@BeforeSuite
Before class	@BeforeClass (only static)	@BeforeClass
Before group	N/A	@BeforeGroup
Before method	@Before	@BeforeMethod
Test	@Test	@Test
After method	@After	@AfterMethod
After group	N/A	@AfterGroup
After class	@AfterClass (only static)	@AfterClass
After suite	N/A	@AfterSuite

EXAMPLE OF TEST CLASS

```java
public class TestNgExample {
    Object object = new Object();

    @Test
    public void test1() {
        System.out.println("I'am object: " + object.toString());
    }

    @Test
    public void test2() {
        System.out.println("I'am object: " + object.toString());
    }
}
```

SUITES

```xml
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="My cool suite with tests">

  <test name="TestNgExample">
    <parameter name="first-name" value="Jhon"></parameter>
    <classes>
      <class name="com.epam.tat.test.TestNgExample"></class>
    </classes>
  </test>

  <test name="Divider">
    <classes>
      <class name="com.epam.tat.test.DividerTest"></class>
    </classes>
  </test>
</suite>
```

INHERITANCE

```java
public class ConfigurationTest {
   @BeforeClass(description = "Before class (invokes once per class instance)")
   public void setUp() {
      System.out.println("Set some configuration for class");
   }

   @AfterClass(description = "After class (invokes once per class instance)")
   public void tearDown() {
      System.out.println("Return configuration back after all test methods");
   }
}
```

```java
public class TestNgExample extends ConfigurationTest {
   @Parameters({"first-name"})
   @Test
   public void testPrintFirstName(@Optional(value = "Bill") String firstName) {
      System.out.println("I got from parameters name: " + firstName);
   }
}
```

COMMON ATTRIBUTES FOR @TEST, @BEFORE*, @AFTER*

- enabled
- groups
- dependsOnGroups
- dependsOnMethods
- alwaysRun
- inheritGroups
- description
- timeOut

SPECIFIC ATTRIBUTES FOR @TEST

- dataProvider
- dataProviderClass
- invocationCount
- invocationTimeout
- threadPoolSize
- expectedExceptions
- expectedExceptionsRegExp
- singleThreaded
- skipFailedInvocations
- priority

GROUPS

We can group test methods by functionality using specific attribute. Methods can have dependencies on particular groups

```java
@Test(dependsOnMethods = "testPrintObject", groups = "first")
public void testPrintObject2() {
   System.out.println("I'am object: " + object.toString());
}
```

```xml
<test name="TestNgExample">
   <parameter name="first-name" value="Jhon"/>
   <groups>
      <run>
         <include name="first"/>
      </run>
   </groups>
   <classes>
      <class name="com.epam.tat.test.TestNgExample"></class>
   </classes>
</test>
```

EXCLUDE / INCLUDE

We can control which test methods should be run using exclude and include options

```
<test name="TestNgExample">
   <parameter name="first-name" value="Jhon"/>
   <classes>
      <class
name="com.epam.tat.test.TestNgExample">
         <methods>
            <include name="testPrintObject"/>
         </methods>
      </class>
   </classes>
</test>
```

```
<test name="TestNgExample">
   <parameter name="first-name" value="Jhon"/>

   <groups>
      <run>
         <include name="first"/>
      </run>
   </groups>

   <classes>
      <class
name="com.epam.tat.test.TestNgExample">
      </class>
   </classes>
</test>
```

DEPENDENCIES AND PRIORITIES

Dependencies and priorities for methods allow to control execution order

Note that:

- @Test methods can depend only on other @Test
- Configuration methods can depend on other configurations e.g. @After* and @Before* methods
- Order for methods using priority attribute counts like this: $order \sim \frac{1}{priority}$

```
@Test
public void testPrintObject() {
   System.out.println("I'am object: " + object.toString());
}

@Test(dependsOnMethods = "testPrintObject")
public void testPrintObject2() {
   System.out.println("I'am object: " + object.toString());
}
```

PARAMETRIZATION

```
@Parameters({"first-name"})
@Test
public void testPrintFirstName(@Optional(value = "Bill") String firstName) {
   System.out.println("I got from parameters name: " + firstName);
}
```

```
<test name="TestNgExample">
    <parameter name="first-name" value="Jhon"></parameter>
    <classes>
        <class name="com.epam.tat.test.TestNgExample"></class>
    </classes>
</test>
```

```
▼ OK TestNgExample
   ▼ OK <no name>
        OK testPrintFirstName[Jhon]
        OK testPrintObject
        OK testPrintObject2
```

PARAMETRIZATION

```java
@Test(dataProvider = "dataProviderForDiv")
public void testDivDataFromDataProvider(double a, double b, double expectedResult) throws Exception {
   double result = div(a, b);
   Assert.assertEquals(result, expectedResult, "Invalid result of division, expected: " + expectedResult);
}

@DataProvider(name = "dataProviderForDiv")
public Object[][] dataProvider() {
   return new Object[][] {
         {3, 2, 1.5},
         {0, 3, 0.0},
         {0, 3, 0.0},
         {3, 2, 1.5}};
}
```

```
▼ OK Default Suite                                        14ms
  ▼ OK unit-testing                                       14ms
    ▼ OK DividerTest                                      14ms
        OK testDivDataFromDataProvider[3, 2, 1.5]          3ms
        OK testDivDataFromDataProvider[0, 3, 0.0] (1)      0ms
        OK testDivDataFromDataProvider[0, 3, 0.0] (2)      0ms
        OK testDivDataFromDataProvider[3, 2, 1.5] (3)      0ms
```

FACTORIES

```java
public class TestNgExampleFactory {
    private static final int COUNT = 3;

    @Factory
    public Object[] createInstances() {
        Object[] tests = new Object[COUNT];
        for (int i = 0; i < COUNT; i++) {
            tests[i] = new TestNgExample("custom-" + i);

        }
        return tests;
    }
}
```

```java
public class TestNgExample extends ConfigurationTest {
    private String arg;

    @Factory(dataProvider = "dp")
    public TestNgExample(String arg) {
        this.arg = arg;
    }

    @Test
    public void testParameterFromArgument() {
        System.out.println("arg: " + arg);
    }

    @DataProvider(name = "dp")
    public Object[][] dataProvider() {
        return new Object[][] {{"custom-a"},{"custom-b"}};
    }
}
```

ASSERTS

Assert - Assertion tool class. Presents assertion methods with a more natural parameter order. The order is always **actualValue**, **expectedValue** [, message].

Possible variants:

1. assertEquals
2. assertNotEquals
3. assertTrue
4. assertFalse
5. assertSame
6. assertNotSame
7. assertNull
8. assertNotNull

```
@Test(description = "test division for 3 by 2 equals 1.5", enabled = false)
public void testDivThreeByTwo() throws Exception {
   double result = div(3, 2); // 1.5
   Assert.assertEquals(result, 1.5, "Invalid result of division");
}
```

CUSTOM RUNNER

BENEFITS

- Ability to run tests outside IDE
- Possibility to add CLI-parser for parameters that will be applied to tested system or test runner
- Flexible configuration: custom listeners, suites configuration, parallel execution, etc

CUSTOM RUNNER

```
public class TestRunner {
  public static void main(String[] args) {
    TestListenerAdapter tla = new TestListenerAdapter();
    TestNG tng = new TestNG();

    XmlSuite suite = new XmlSuite();
    suite.setName("TmpSuite");
    List<String> files = new ArrayList<>();
    files.addAll(new ArrayList<String>() {{
      add("./src/test/resources/testng.xml");
    }});
    suite.setSuiteFiles(files);

    List<XmlSuite> suites = new ArrayList<XmlSuite>();
    suites.add(suite);
    tng.setXmlSuites(suites);
    tng.run();
  }
}
```

PARALLEL EXECUTION: TYPES

- tests
- methods
- suites
- classes
- false

PARALLEL EXECUTION: CONFIGURATION VIA XML

```java
public class ParallelTest {
   @Test
   public void testParallel1() {
      checkTime();
      sleep(2);
   }

   @Test
   public void testParallel2() {
      checkTime();
      sleep(2);
   }

   private void checkTime() {
      System.out.println("Current time: " + new Date(System.currentTimeMillis()));
   }
}
```

```xml
<suite name="My parallel suite" parallel="methods" thread-count="2">
    <test name="ParallelTest">
        <classes>
            <class name="com.epam.tat.test.ParallelTest"></class>
        </classes>
    </test>
</suite>
```

```
My parallel suite        4s 32ms
  ParallelTest           4s 32ms
    <no name>            4s 32ms
      testParallel1      2s 16ms
      testParallel2      2s 16ms
```

```
Current time: Tue Nov 17 10:57:06 BRT 2015
Current time: Tue Nov 17 10:57:06 BRT 2015
```

PARALLEL EXECUTION: CONFIGURATION VIA JAVA

```
XmlSuite suite = new XmlSuite();
suite.setName("TmpSuite");

suite.setParallel(XmlSuite.ParallelMode.METHODS);
suite.setThreadCount(2);

List<XmlSuite> suites = new ArrayList<>();
suites.add(suite);
tng.setXmlSuites(suites);
```

LISTENERS

```
public class MyTestListener implements IInvokedMethodListener {

   @Override
   public void beforeInvocation(IInvokedMethod method, ITestResult testResult) {
      System.out.println("method started: " + method.getTestMethod().getMethodName());
   }

   @Override
   public void afterInvocation(IInvokedMethod method, ITestResult testResult) {
      System.out.println("method finished [" + testResult.getStatus() + "]: " +
            method.getTestMethod().getMethodName() + "\n");
   }
}
```

```
TestNG tng = new TestNG();
tng.addListener(new MyTestListener());
```

```
method started: testPrintObject
I'am object: java.lang.Object@48e8e8bf
method finished [1]: testPrintObject

===============================================
My cool suite with tests
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

THANK YOU FOR ATTENDING!

Additional materials for self-study:

- http://testng.org/doc/documentation-main.html
- http://www.tutorialspoint.com/testng/index.htm
- http://seleniumcamp.com/talk/testng-vs-junit-5-battle/