What it is?

- Testing framework for Java
- Based on annotations
- Inspired by frameworks JUnit and NUnit, but with an introduction of new functionalities
- Separates source codes from tests, so there is no need to recompile code after changing tests suite
Features

• Running a test with different thread policies (one thread per method, one thread per class tests...)

• Testing if a code is multithread safe
Features

- Tests configurations (grouping of slow and fast tests into individual groups, which can be run separately)
- Parametrization and data-driven testing
- Plug-ins and tools for Eclipse, IDEA, Maven, Ant...
Features

- Provides a so-called dependent methods mechanism
- Dependent methods mechanism guarantees that certain groups of methods are always invoked before others and if these methods fail, dependent methods will be marked as skipped instead of failing.
Features

- TestNG can also create HTML reports
Features

- Suited to cover all categories of tests such as unit, end-to-end, integration…
Simple example

- **@BeforeClass**
  - Method will be executed after class creation but before all tests

- **@Test(groups = ("group name"))**
  - method will be executed only if chosen group of tests will be selected
  - method can be a member of multiple groups

- No need to extend a class or implement interfaces

- Important are annotations not method names (but both are important in JUnit)

```java
package example1;
import org.testng.annotations.*;

public class SimpleTest {
    @BeforeClass
    public void setUp() {
        // code that will be invoked when this test is instantiated
    }

    @Test(groups = {"fast"})
    public void aFastTest() {
        System.out.println("Fast test");
    }

    @Test(groups = {"slow"})
    public void aSlowTest() {
        System.out.println("Slow test");
    }
}
```
<table>
<thead>
<tr>
<th>Functionality - JUnit 4 vs TestNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Annotation Support</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>TestNG</td>
</tr>
<tr>
<td>JUnit 4</td>
</tr>
</tbody>
</table>
Basic annotations

@BeforeSuite
The annotated method will be run only once before all tests in this suite have run.

@AfterSuite
The annotated method will be run only once after all tests in this suite have run.

@BeforeClass
The annotated method will be run only once before the first test method in the current class is invoked.

@AfterClass
The annotated method will be run only once after all the test methods in the current class have run.

@BeforeTest
The annotated method will be run before any test method belonging to the classes inside the <test> tag is run.

@AfterTest
The annotated method will be run after all the test methods belonging to the classes inside the <test> tag have run.

@BeforeGroups
The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.

@AfterGroups
The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.
<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@BeforeMethod</td>
<td>The annotated method will be run before each test method.</td>
</tr>
<tr>
<td>@AfterMethod</td>
<td>The annotated method will be run after each test method.</td>
</tr>
<tr>
<td>@DataProvider</td>
<td>Marks a method as supplying data for a test method. The annotated method must return an Object[][][], where each Object[] can be assigned the parameter list of the test method. The @Test method that wants to receive data from thisDataProvider needs to use a dataProvider name equals to the name of this annotation.</td>
</tr>
<tr>
<td>@Factory</td>
<td>Marks a method as a factory that returns objects that will be used by TestNG as Test classes. The method must return Object[].</td>
</tr>
<tr>
<td>@Listeners</td>
<td>Defines listeners on a test class.</td>
</tr>
<tr>
<td>@Parameters</td>
<td>Describes how to pass parameters to a @Test method.</td>
</tr>
<tr>
<td>@Test</td>
<td>Marks a class or a method as a part of the test.</td>
</tr>
</tbody>
</table>
Invoking TestNG

- Three possibilities:
  - with a testng.xml file
  - with ant
  - directly from command line
Example class

/*
 * This class prints the given message on console.
 */

public class MessageUtil {

    private String message;

    // Constructor
    //@param message to be printed
    public MessageUtil(String message){
        this.message = message;
    }

    // prints the message
    public String printMessage(){
        System.out.println(message);
        return message;
    }
}

Example test case class

```java
import org.testng.Assert;
import org.testng.annotations.Test;

public class SampleTest {

    String message = "Hello World";
    MessageUtil messageUtil = new MessageUtil(message);

    @Test
    public void testPrintMessage() {
        Assert.assertEquals(message, messageUtil.printMessage());
    }
}
```
Example of testng.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<suite name="Sample test Suite">
  <test name="Sample test">
    <classes>
      <class name="SampleTest" />
    </classes>
  </test>
</suite>
```
Running tests

• Firstly, we need to compile sources using javac

  ● `C:\TestNG_WORKSPACE>javac MessageUtil.java SampleTest.java`

• Next, we will run the TestNG

  ● `C:\TestNG_WORKSPACE>java -cp "C:\TestNG_WORKSPACE" org.testng.TestNG testng.xml`
Running tests

- Previous call will give us this output

  Hello World

  Sample test Suite
  Total tests run: 1, Failures: 0, Skips: 0
Examples

- In JUnit we have annotations Before, After and Expected while TestNG uses more clear and understandable.

```java
@BeforeClass
public void oneTimeSetUp() {
    // one-time initialization code
    System.out.println("@BeforeClass - oneTimeSetUp");
}
```
Examples - exceptions

• You can test if an exception is being thrown (also found in JUnit)

```java
@Test(expectedExceptions = ArithmeticException.class)
public void divisionWithException() {
    int i = 1/0;
}
```
Examples - ignoring

- Some tests might be not yet prepared and therefore should be ignored
- In JUnit `@Ignore` annotation is used

```java
@Test(enabled=false)
public void divisionWithException() {
    System.out.println("Method is not ready yet");
}
```
Examples – time limit

- Method will be terminated after time specified in milliseconds (in this case 1000) and test will be marked as failed.

```java
@Test(timeOut = 1000)
public void infinity() {
    while (true);
}
```
Examples – suite test

@Test(groups="method1")
public void testingMethod1() {
    System.out.println("Method - testingMethod1()");
}

@Test(groups="method2")
public void testingMethod2() {
    System.out.println("Method - testingMethod2()");
}

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

• Found also in JUnit, but the approach is different here

• TestNG uses xml files to specify which test should be executed
Examples - parametrization

One way to set parameters is via annotations (@Parameters) and xml file.
Examples - parametrization

- We can also use `@DataProvider`
Examples – dependency

- In this case **method2** will be executed only if **method1** is successful otherwise **method2** will be skipped

```java
@Test
def testExample1()
    System.out.println("This is method 1");
}

@Test(dependsOnMethods="method1")
public void method2()
    System.out.println("This is method 2");
}

JUnit does not support this feature
Thank you for your attention
Sources

- http://www.mkyong.com/unittest/junit-4-vs-testng-comparison/
- http://www.tutorialspoint.com/testng/
- http://1.bp.blogspot.com/-jMRbusn1DHk/T19CEhXljGI/AAAAAAAAAD4/q62KEo2Khec/s1600/results.png