

Maven

groupId	
artifactId	
version	
scope	(mandatory)
mvm clean	Clean up project
mvm test	Execute unit tests
mvm package	Package compiled code in target directory
mvm install	Install products in the local repository
mvm deploy	Upload products in the remote repository

Configuration

Project Home	pom.xml and following directory
src/main/java	Deliverable Java source code for the project.
src/main/ressources	Deliverable resources for the project, such as properties files.
src/main/webapp	Specific web code source
src/test/java	Testing Java source code for the project.
src/test/ressources	Resources necessary for testing.
target	Compiled files and project archive

Logs

Usage

- Investigation to find the source of an anomaly
- Detect suspicious behavior (to be alert before users)
- Monitor the use of the software (with Elasticsearch for example)

Log Levels

-
- FATAL Unexpected events that prevent the application from running.
- ERROR Unexpected events with impact for the user but does not prevent the application from running.
- WARN Unexpected events without impact for the user
- INFO Expected events with high added value (for examples : user actions, long treatments status, ...)



By [tomp11](https://tomp11.com)
cheatography.com/tomp11/

Published 18th November, 2022.
 Last updated 17th November, 2022.
 Page 1 of 5.

Sponsored by [Readable.com](https://readable.com)
 Measure your website readability!
<https://readable.com>

Logs (cont)

DEBUG	Information about main methods with parameters and result
TRACE	All other informations

Good practices

- Add timestamp to the logs
- Use a specific format for logs
- Show stacktrace to an unexpected exception

Bad practices

-
- Show personal data (only id)
- Show password
- Show stacktrace to an expected exception

Use Loggers

Imports

```
-
import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.LogManager;
```

*Class

```
static final Logger LOGGER = LogManager.getLogger();
```

Variable

Write

```
-
```

Logs

```
LOGGER.trace("Hello, I am a trace log");
LOGGER.error("Hello, I am an error log with an exception ", new Exception());
LOGGER.info(" Hello, I am an info log with 2 variables : first {} ; second {}", "I am the first variable", "I am the second variable");
...
```

Log4j2

Formats

Key/Value	
YAML	.yaml / .yml
JSON	.json / .json
XML	.xml

Appender

```
<appender type="File" name="FILE" fileName="target/test.log"> <layout type="PatternLayout" pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" /> </appender>
<appender type="Console" name="CONSOLE"> <layout type="PatternLayout" pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" /> </appender>
```



Test

Test Goals

- To detect issues before production
- To ensure the proper working of all the functionalities
- To be sure that the system is reliable

Testing Levels

- | | | |
|----------------------------------|--|---------------------|
| - Unit Testing | To check each component (unit of source code) individually. | Tout seul OK |
| - Integration Testing | To check the interconnection between the different components. | Chacun tout seul OK |
| - System Testing | In a complete and integrated system to verify its compliance with specified requirements.. | 1000x |
| - Operational acceptance testing | To verify that the product conforms to the specification | Utilisable |

Write Test and Conventions

- | | |
|---------|---|
| - GIVEN | Describe the initial context of the system. |
| - WHEN | Describe an event/action. |
| - THEN | Describe an expected outcome/result. |

- Test is a **public** method in test case

- Test Have **@Test** annotation

- Test return **void**

- `shouldReturnTrue()`

- `shouldReturnTrue().shouldThrowNotFoundException()`

- `shouldThrowNotFoundException()`

Result of Test

- | | |
|------------|---|
| - Success | All assertions are correct |
| - Faillure | At least one assertion is incorrect |
| - Error | An unexpected exception has been thrown |



JUnit

Assertions

- `assert Equals (expected, actual) ;`
- `assert Not Equals (expected, actual);`
- `assert Null (object)` `assert Not Null (object);`
- `assert True (condition)` `assert False (condition);`

Exceptions

- `fail() ;` Used to fail a test when an expected exception has not been thrown :
- expected attribute in `@Test` annotation Used to catch an expected exception :

JUnit Tools

- `@Before` annotates a method that will be executed before each test
- `@After` annotates a method that will be executed after each test
- `@BeforeClass (static)` annotates a method that will be executed once before all the tests of a test case.
- `@AfterClass (static)` annotates a method that will be executed once after all the tests of a test case.

AssertJ

AssertJ is to be used in addition to JUnit

The change is only for the assertion

Assertion begins `Assertion.assertThat (objectUnderTest)`

`Assertion.assertThat (result) .containsExactlyInAnyOrderElementsOf (cards);`

Mockito

General

- Framework to mock java object
- Framework to mock java object
- Can check calls to methods



Mockito (cont)

Using Mockito in TestCase

- Init	<code>@RunWith(MockitoJUnitRunner.class)</code>
- Mock	<code>@Mock</code> annotation above object to mock
- Inject	<code>@InjectMocks</code>
- When	<code>Mockito.when(myMockedObject.myMethod())</code>
- thenThrow	<code>Mockito.when([...]).thenReturn(myException);</code>
- thenCallRealMethod	<code>Mockito.when([...]).thenReturn(callRealMethod());</code>
- Verify	<code>Mockito.verify(myMockedObject, VerificationMode.with(parameters, ...));</code>

Coverage

Measure to describe the proportion of code executed during tests

Higher the percentage, higher the code is safe

Jacoco in Maven for Coverage

Continuous Integration

Definition *Continuous integration is a set of practices used in software engineering to verify at each source code change that the result of the changes does not produce regression in the developed application*

Goals

- Free developers from recurring tasks
- encourage behaviors that help reduce the number of errors and bugs
- Find and fix bugs quicker
- Deliver updates faster

Build

- Step 1 **Compile** Ensure code actually compiles on every platform
- Step 2 **Test** Verify that the features run as expected. Check that there is no regression
- Step 3 **Deploy** Generate a new resource. Upload the resource on the remote repository

