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, ...)

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.ap ach e.l ogg ing.lo g4j.Lo gMa nager;`	
	import org.apache.logging.log4j.Logger;`	
*Class Variable	`static final Logger LOGGER = LogMan age r.g etL ogg er();`	
Write Logs	-	
	`LOGGER.tr ace ("Hello, I am a trace log")`;	
	`LOGGER.er ror ("Hello, I am an error log with an except ion ", new Except ion());`	
	`LOGGER.in fo( " Hello, I am an info log with 2 variables : first {} ; second {}", "I am the first variab le", "I am the second variab le");`	
	...	
Log4j2 Formats	-	
	Key/Value	
	YAML	.yaml / .yml
	JSON	.json / .jsn
	XML	.xml
Appender	`<ap pender type="F ile " name="F ILE " fileNa me= " tar get /te st.l og "> <layout type="P att - ern Lay out " patter n="% d{H H:m m:s s.SSS} [%t] %-5level %logge r{36} - %msg%n " /> </a ppe - nde r>`	
	`<ap pender type="C ons ole " name="C ONS OLE "> <layout type="P att ern Lay out " patter n="% - d{H H:m m:s s.SSS} [%t] %-5level %logge r{36} - %msg%n " /> </a ppe nde r>`	

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 Equ als (ex pected, actual) ;`
- `assert Not Equ als (ex pected, actual);`
- `assert Nul l(o bject);` `assert Not Nul l(o bject);`
- `assert Tru e(c ond ition);` `assert Fal se( con dit ion);`

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
- @BeforClass (**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 `Assert ion s.a sse rtT hat (ob jec tUn der Test)`

```
Assert ion s.a sse rtT hat (re sult) .conta ins Exa ctl yIn Any Ord erE lem ent sOf (ca rds);
```

Mockito

General

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

Mockito (cont)

Using Mockito in TestCase

- Init	`@RunWi th( Moc kit oJU nit Run ner.class)`
- Mock	`@Mock annotation above object to mock`
- Inject	`@InjectMock
- When	`Mockit o.w hen (my Moc ked Obj ect.my met hod())`
- thenThrow	`Mockit o.w hen ([...] ).t hen Thr ow( myE xce ption);`
- thenCallRealMethod	`Mockit o.w hen ([...] ).t hen Cal lRe alM eth od();`
- Verify	`Mockit o.v eri fy( myM ock edO bje ct, Ver ifi cat ion Mod e).m ym eth od( par - ams ,...);`

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
-	ncourage behaviors that help reduce the number or error and bugs
-	Find and fix bugs quicker
-	Deliver updates faster

Build

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