MAVEN

2019-2020

ESPRIT thouraya.louati@esprit.tn UP JavaEE / .NET (Bureau E204)

Séance 2 (Partie 1)

- Séquence 2: Mise en place de l'environnement de développement et déploiement d'un premier projet maven.
  - Charge : 3 heures
    - Installation et mise en place des différents outils de travail.
    - Mise en place d'un premier projet MAVEN.
    - Présentation de l'architecture du projet maven, le fichier pom.xml et les plugins Maven.
    - **TP1:** Projet avec Maven (JAR)
    - **TP2:** Projet Web avec Maven (WAR)


Apache
maven

Gestion de bibliothèques avec Maven

- Maven
  - Définition
    - Maven est un outil permettant de **gérer les dépendances**, la **compilation**, le "**packaging**" et bien d'autres choses. Il s'utilise au moyen de fichier "**pom**" décrivant votre projet (fichier xml).
    - Maven est un outil open source (développé par la fondation Apache), écrit en Java largement distribué dans la communauté Java.
  - Outil de build: C'est un outil de construction de projets (build).
- Gestion des dépendances
- Possède de nombreux plugins
  - Maven possède un grand nombre de plugins offrant de nouvelles fonctionnalités.

Organisation classique

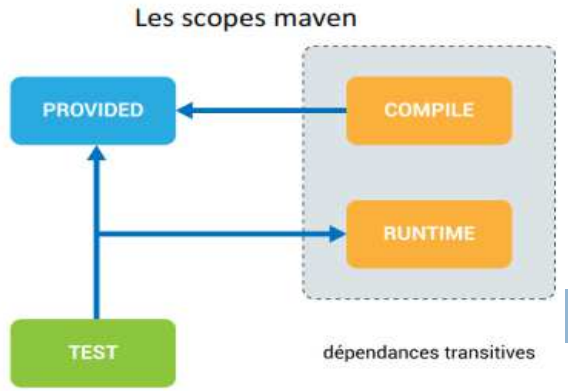

```
/src
   /main
      /java
      /resources
      /webapp
   /test
      /java
      /resources
/target
pom.xml
```

- Organisation normée des répertoires
  - Un projet Maven est un projet structuré comme suit:
    - Répertoire de sources découpé en sous-répertoires
      - Sources Java, ressources nécessaires à l'exécution ou à la compilation du projet, les fichiers «properties» (connexion à la BD, etc.), un répertoire web app (tout ce qui est web: les images, les css, etc.).
      - Un répertoire «test»: un ensemble de sources Java utile à l'exécution des tests unitaires.
    - Un fichier pom.xml qui définit les propriétés du projet et les dépendances nécessaires pour compiler et exécuter le projet.
    - Un répertoire «target» dans lequel l'ensemble des fichiers Java seront compilés avant d'être packagés ou exécutés.

Build

- Ensemble de tâche pour compiler ou préparer une librairie
  - Compile: compiler les sources Java qui sont à (src\main\java ou src\test\java).
  - Clean: effacer le contenu du répertoire «target» et permettre de faire une archive de votre projet plus légère
  - Test: exécuter les test unitaires
  - Package: archiver les classes du projet sous un format défini «jar» ou «war»
- Tâches en dépendances
  - package>>test>>compile

Dépendances



- Un projet Maven définit un ensemble de dépendances vers une ou plusieurs librairies qui sont définies selon:
  - groupe, artefact, version
  - Ces dépendances sont utiles à différentes phases de l'application (compilation & exécution).

- Scope: définit l'utilité de la dépendance.
  - Les Scopes sont interdépendants et en interaction entre les dépendances transitives des différents projets.
    Exemple: pour pouvoir compiler un projet, on a besoin de toues les dépendances définit dans le scope compile et scope «provided».

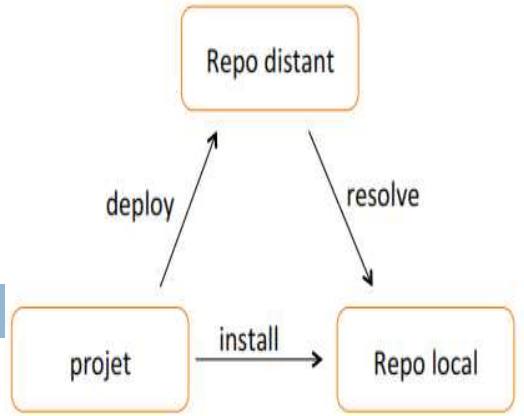
Dépendances

- Une dépendance «provided » est une dépendance fournie par le système vers lequel vous visez le déploiement de votre application.
  - Exemple: La librairie 'servlet api' est une librairie qui est fournie par Tomcat que vous n'avez pas besoin de distribuer avec votre projet web.
  - Pour l'exécution de votre programme, l'ensemble des dépendances noté dans le scope «runtime» seront nécessaires ainsi que l'ensemble des dépendances fourni en «provided».
  - Enfin pour l'exécution de vos tests unitaires ce seront les dépendances du scope «provided» qui seront utiles ainsi que les dépendances du scope «runtime».

Repository

- Les dépendances sont distribués à travers différentes unités de stockage logiques des dépendances
- **Repo «central»** https://mvnrepository.com/
  - Dépôt « officiel »
  - Héberge des projets open sources
- **Repo «tier»**
  - Autre dépôt géré par une organisation
- **Repo «local»** dans lequel l'ensemble des dépendances sont téléchargées lorsque vous utilisez un projet Maven.
  - Cache de dépendances utilisées
  - Librairies par groupe (~/.m2/repository)

Opérations sur les repository



- Un ensemble de tâches ou de commandes sont nécessaires à la manipulation des repository.
  - "**resolve**" permet de télécharger l'ensemble de dépendances de votre projet qu'elle soit directe ou transitive vers votre repository local.
  - "**install**" permet de packager votre application et de la mettre à disposition dans votre repository local pour d'autres projets que vous auriez sur votre ordinateur.
  - "**deploy**" permet d'envoyer votre librairie, votre projet vers un repository distant.

Squelette d'un pom


```
<project xmlns="http://maven.apache.org/...0.0"
  xmlns:xsi="http://www.w3.org2001/XMI...ma-instance"
  xsi:schemaLocation="http://maven.a...he.org/POM/4.0.0
 http://maven.apahe.org/xsd/maven-...0.
  <modelVersion>4.0.0</modelVersion>
  <groupId>my.project</groupId>
  <artifactId>test</artifactId>
  <version>1.0-SNAPSHOT</versio
  <packaging>jar</packaging>

  <properties>
    <maven.compiler.source>1.8</maven.compile
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Le groupe : sont le nom de votre projet, le nom de votre librairie et sa version.

Le format de distribution, le packaging en anglais, qui définira la manière dont seront archivés vos classes Java pour leur distribution.

La version dans laquelle doit utiliser Maven pour compiler vos sources Java.

Un ensemble de dépendance vers d'autres librairies, les librairies tierces, qui définiront à chaque fois leur groupe, le nom de leur artefact, la version et le scope pour lequel cette librairie nécessaire pour votre projet.

Définition de votre projet

- Définir à minima:
  - Groupe (~package)
  - Artefact (nom de la librairie)
  - Version
  - Packaging (jar par défaut)
- Puis…
  - Propriétés interprétées par les tâches
  - Maven ou plugins (ex: version java)
  - Dépendances

Moteur de cherche des dépendances de Maven :
http://mvnrepository.com

Résumé

- Maven est un outil
  - De gestion de build et de dépendances qui facilite la vie du développeur dans la gestion de projets.
  - Adapté au développement de projets java
  - Facilite le travail en équipe et la maintenance

Installation de Maven

- Maven en tant que Plugin : Un plugin Maven est installé par défaut avec STS.

- Maven en stand-alone :
  - Télécharger Maven : Site http://maven.apache.org , apache-maven-xxx.zip
  - Décompresser l'archive dans le dossier de votre choix, par exemple : C:\Products\apache-maven-3.5.0

Installation de Maven

- Variables d'environnement :

Nom	Description	Exemples
JAVA_HOME	Répertoire racine du JDK	C:\Products\Java\jdk1.8.0_121
M2_HOME	Répertoire racine de Maven	C:\Products\apache-maven-3.5.0
PATH	Chemin d'accès vers les principaux exécutables du système	PATH=%PATH%;%JAVA_HOME%\bin;%M2_HOME%\bin

Premières commandes de Maven

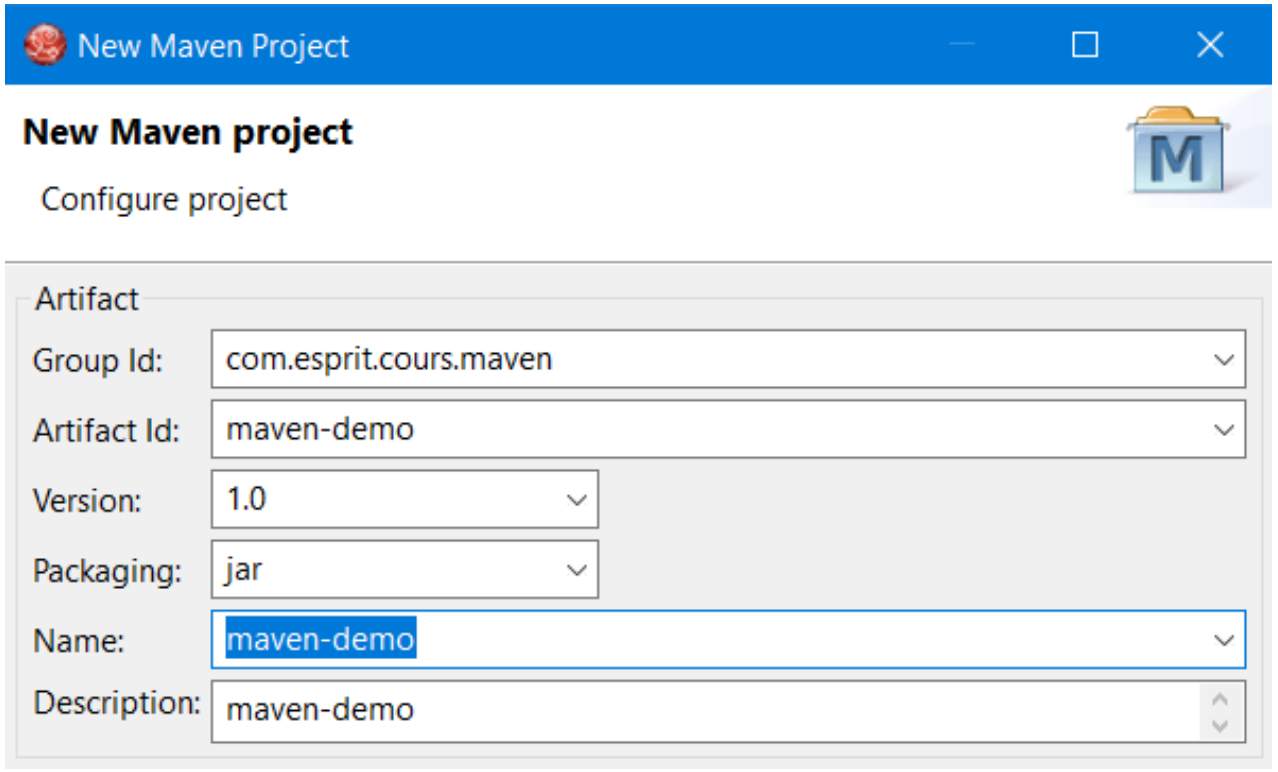
- Maven est-il présent et quelle version ?

```
C:\Users\ASUS>mvn -version
Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T12:57:37+01:00)
Maven home: C:\Users\ASUS\Desktop\Esprit\Ressources JEE\Semaine 1\Outils - JEE\Maven\apache-maven-3.3.3\bin\..
Java version: 1.8.0_60, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_60\jre
Default locale: fr_FR, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "dos"
```

But du cours

- Maitriser l'arborescence standard du code et Créer un projet MAVEN simple via le plugin Maven de ses ressources
- Maîtriser les différents buts (Goals) du cycle de vie d'un projet Maven (la compilation, le test, le packaging d'une application, …)
- Installer une application dans un Repository local
- Gérer les dépendances (bibliothèques) d'un projet donné
- Exécuter les tests unitaires automatiquement

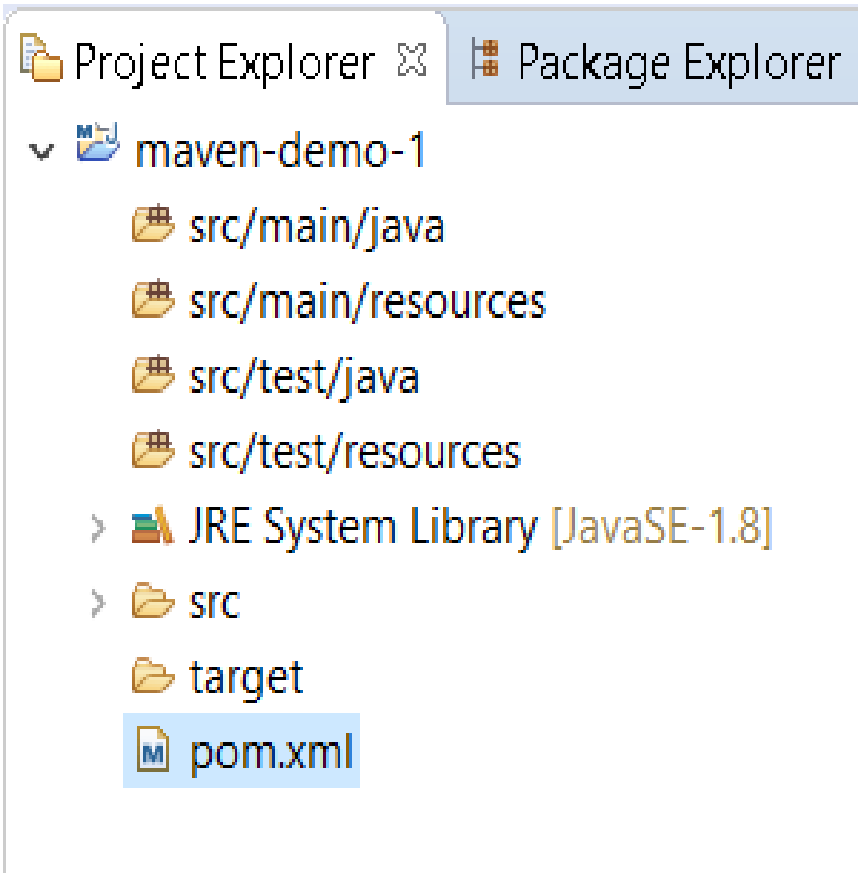
Création d'un projet Maven



Balise du pom.xml

- **pom.xml :** Project Model Object
- **project :** Balise racine de tous les fichiers pom.xml.
- **modelVersion :** Version de POM utilisée.
- **groupId :** Identifier un groupe qui a créé le projet. Ex: org.apache.
- **artifactId :** Nom unique utilisé pour nommer l'artifacts à construire.
- **packaging :** Type de packaging du projet (ex. : JAR, WAR, EAR...).
- **version :** Version de l'artifact généré par le projet.
- **name :** Nom du projet.
- **description :** Description du projet.
- **dependencies :** balise permettant de gérer les dépendances.
- **archetype :** Template de Projet.

Arborescence standard



Première commandes

- Mettez vous sur la racine du projet maven, en ligne de commande et lancer la commande : **mvn eclipse:eclipse l'équivalent de l'update en cas d'échange de projets**
- En ligne de commande lancer : mvn clean install
- Changer la version de Java utilisée dans le projet à Java 8 car maven par défaut pointe sur la version 5

```
<properties>
    <maven.compiler.target>1.8</maven.compiler.target>
    <maven.compiler.source>1.8</maven.compiler.source>
</properties>
```

Arborescence standard

- **pom.xml :** le fichier de configuratiosource n du projet
- **/src :** code source et fichiers principaux
- **/src/main/java :** code source java
- **/src/main/resources :** fichiers de ressources (images, fichiers config...)
- **/src/main/webapp :** webapp du projet
- **/src/test :** fichiers de test
- **/src/test/java :** code source Java de test
- **/src/test/resources :** fichiers de ressources de test
- **/target/site :** informations sur le projet et/ou les rapports générés suites aux traitement effectués
- **/target :** fichiers résultat, les binaires (du code et des tests), les packages générés et les résultats des tests

Buts

- **mvn compile :** Créer les .class
- **mvn test :** Jouer les tests unitaires
- **mvn package :** Creation du livrable dans target.
- **mvn install :** Copie du livrable dans le Repository local : ~\.m2\repository\...
- **mvn deploy :** Copie du livrable sur le repository distant
- **mvn clean :** Supprime le contenu du dossier target.

Test

- mvn **install -Dmaven.test.skip**=true
- mvn **install -Dmaven.test.skip=false**
- Il est possible de configurer le plugin Maven de Eclipse pour sauter les tests.

Test

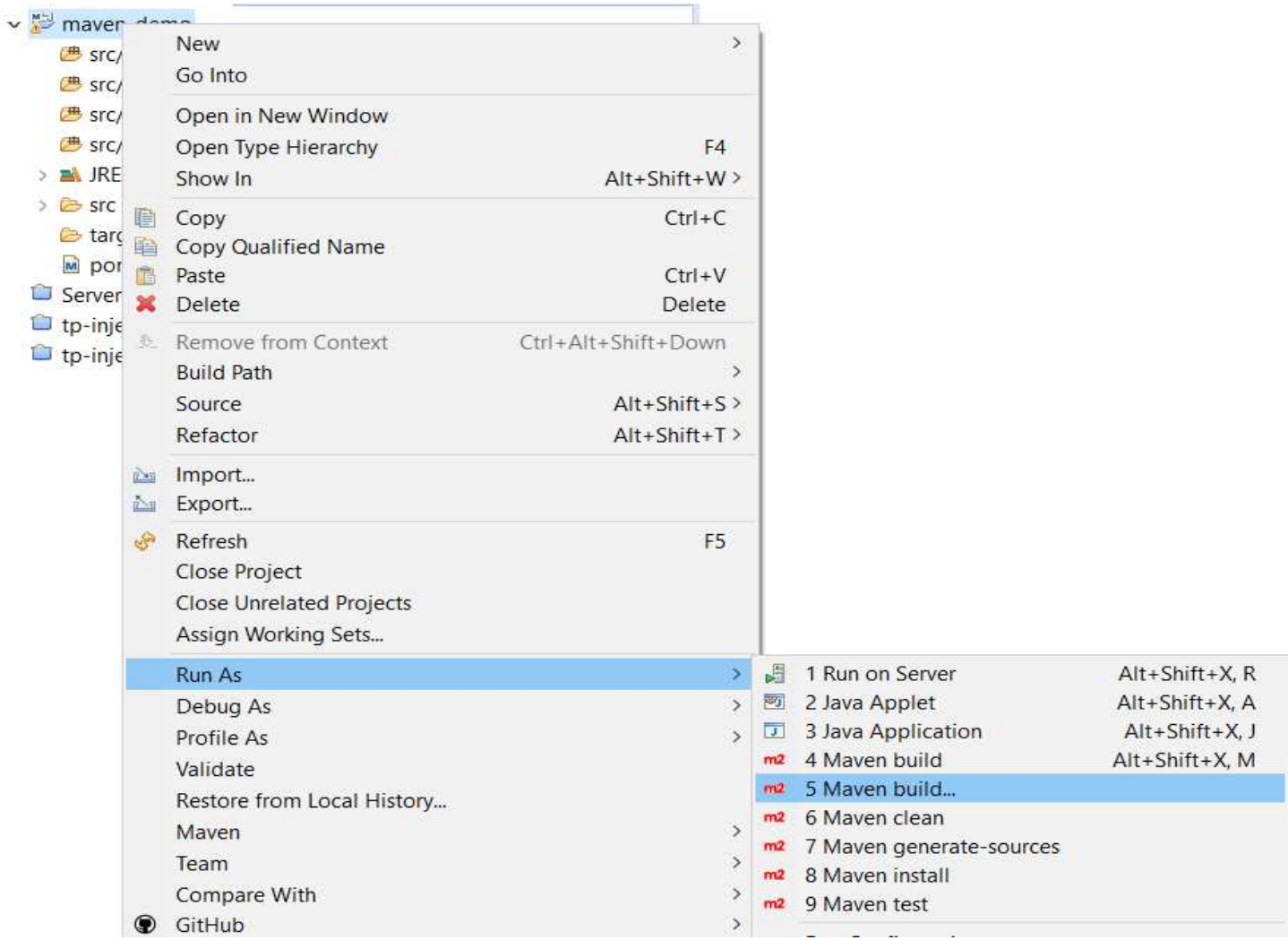
- mvn **install -Dmaven.test.skip**=true
- mvn **install -Dmaven.test.skip=false**
- Il est possible de configurer le plugin Maven de Eclipse pour sauter les tests.

But

- Emplacement du livrable :
  {emplacement Repository}/groupId/artifactId/version
  Exemple: C:\Users\ASUS\.m2\repository\org\json\json\20160810
- Nom du package (jar en général) : {artifactId}-{version}.{package}

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

Exécution sous Eclipse



TP1: Projet avec Maven (JAR)

- Créer un Projet **Maven** :
  - Simple : sans archetype, type Jar, groupId : com.esprit.cours.maven
  - artefactId /nom/description : maven-demo-2
  - package : com.esprit.cours.maven
- Mettre à jour le pom.xml pour utiliser Java 1.8 et Ajouter dans le pom.xml les dépendances JSON et HTTPCLIENT 5 voir dépendances page suivante).
- Créer le package : tn.esprit
- Créer la Classe : CallRestWebService (Voir Code Source slides suivants).
- Créer le livrable avec Maven
- Exécuter la méthode main.

TP1: Projet avec Maven (JAR)


```
package tn.esprit;
import java.io.IOException;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;
import org.json.JSONObject;



public class CallRestWebService {
public static final String endpoint = "http://ip-api.com/json";
public static void main(String[] args) {
HttpClient client = new DefaultHttpClient();
HttpGet request = new HttpGet(endpoint);
String ip = "not found";
```

TP1: Projet avec Maven (JAR)


```
try {
HttpResponse response = client.execute(request);
String jsonResponse = EntityUtils.toString(response.getEntity());
System.out.println("Response as String : " + jsonResponse);
JSONObject responseObj = new JSONObject(jsonResponse);

ip = responseObj.getString("query");
System.out.println("ip : " + ip);

} catch (IOException e) { e.printStackTrace(); }
}
}
```

TP1: Projet avec Maven (JAR)

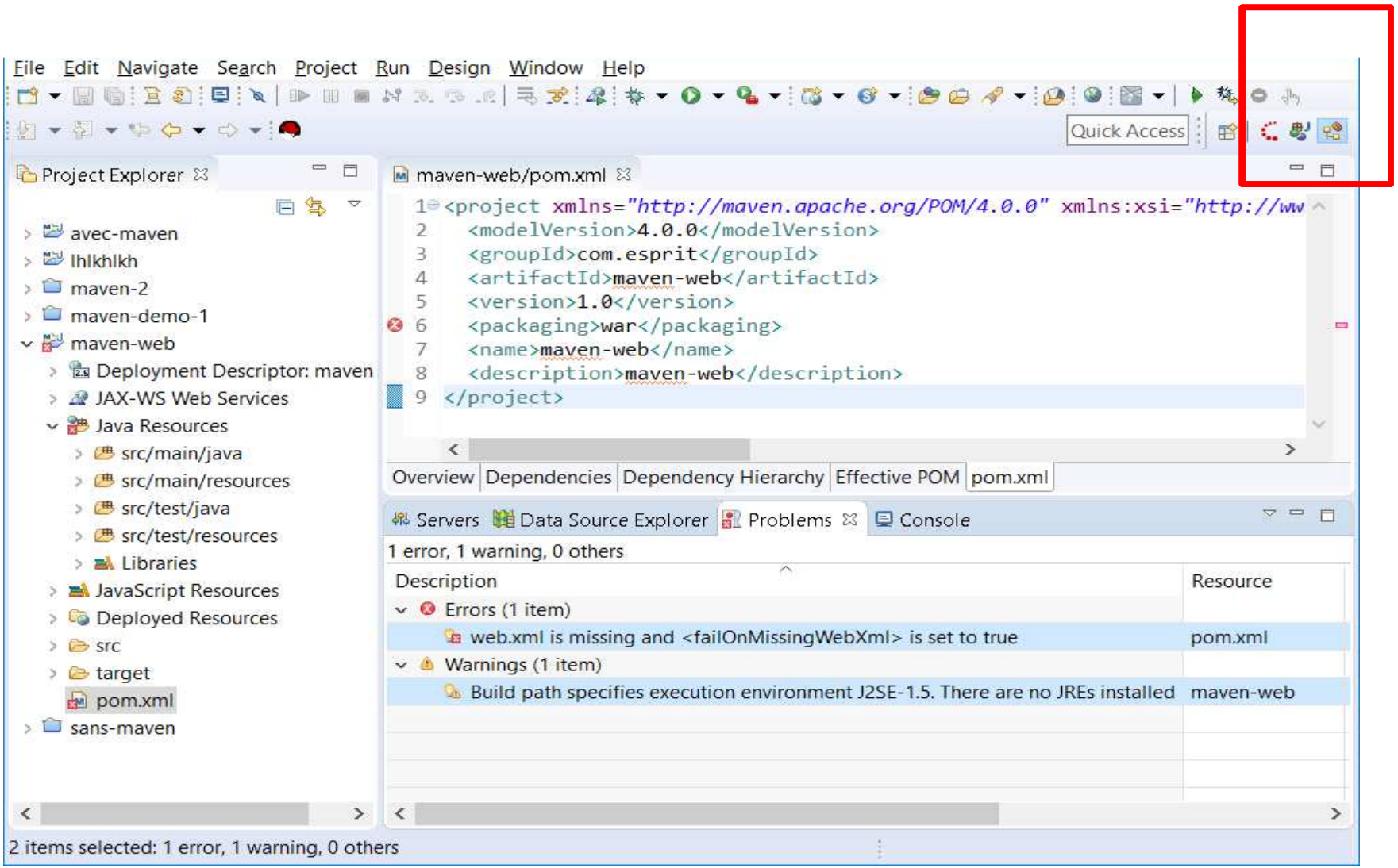

```
<project …>
    …
    <properties>
        <maven.compiler.target>1.8</maven.compiler.target>
        <maven.compiler.source>1.8</maven.compiler.source>
    </properties>

    <dependencies>
        <dependency>
        <groupId>org.json</groupId>
        <artifactId>json</artifactId>
        <version>20160810</version>
        </dependency>
        <dependency>
        <groupId>org.apache.httpcomponents</groupId>
        <artifactId>httpclient</artifactId>
        <version>4.1.1</version>
        </dependency>
    </dependencies>
</project>
```

TP2: Projet Web avec Maven (WAR)

- Créer un nouveau projet de type Maven, simple (sans archetype)
- Projet de type **WAR**

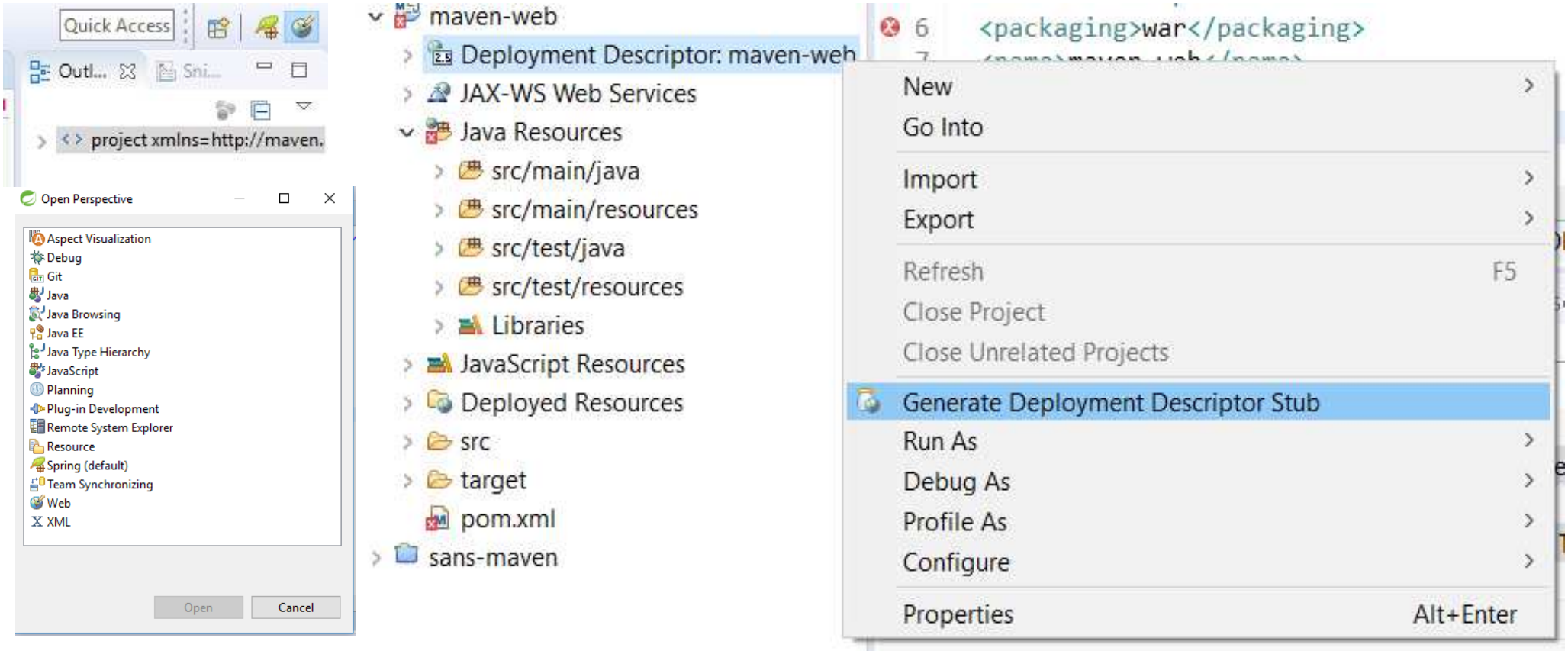
TP2: Projet Web avec Maven (WAR)

- Corriger l'erreur et le warning ci-dessous :



TP2: Projet Web avec Maven (WAR)

- Correction de l'erreur « web.xml missing », tout projet web doit contenir un fichier web.xml, cliquer sur « Generate … » pour le générer :



TP2: Projet Web avec Maven (WAR)

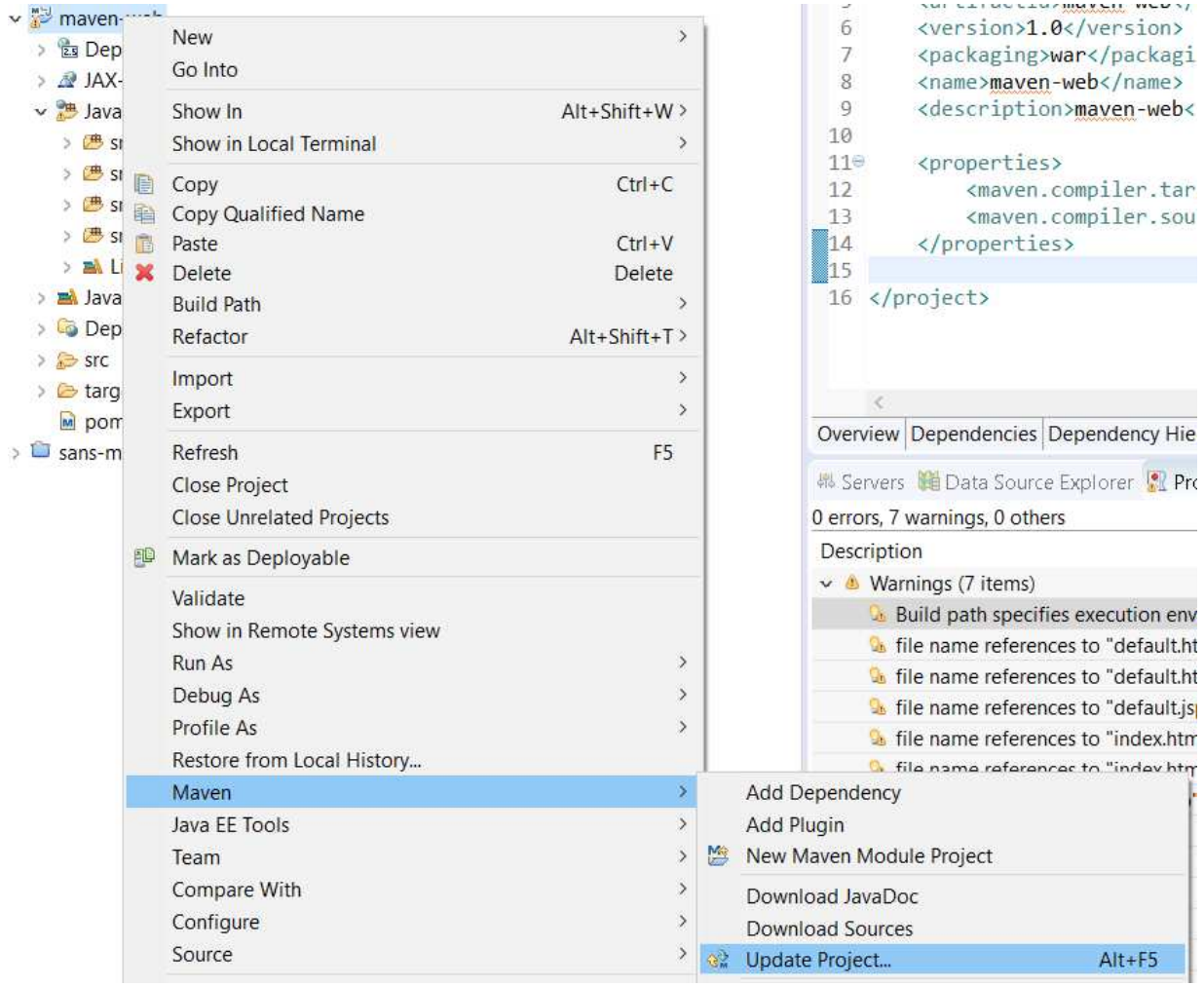
- Correction du warning « Java 1.5 », Pointer sur Java 8 :
- Ajouter dans pom.xml les propriétés suivantes :

```
<properties>
   <maven.compiler.target>1.8</maven.compiler.target>
   <maven.compiler.source>1.8</maven.compiler.source>
</properties>
```

- Puis, Faites un Maven Update.

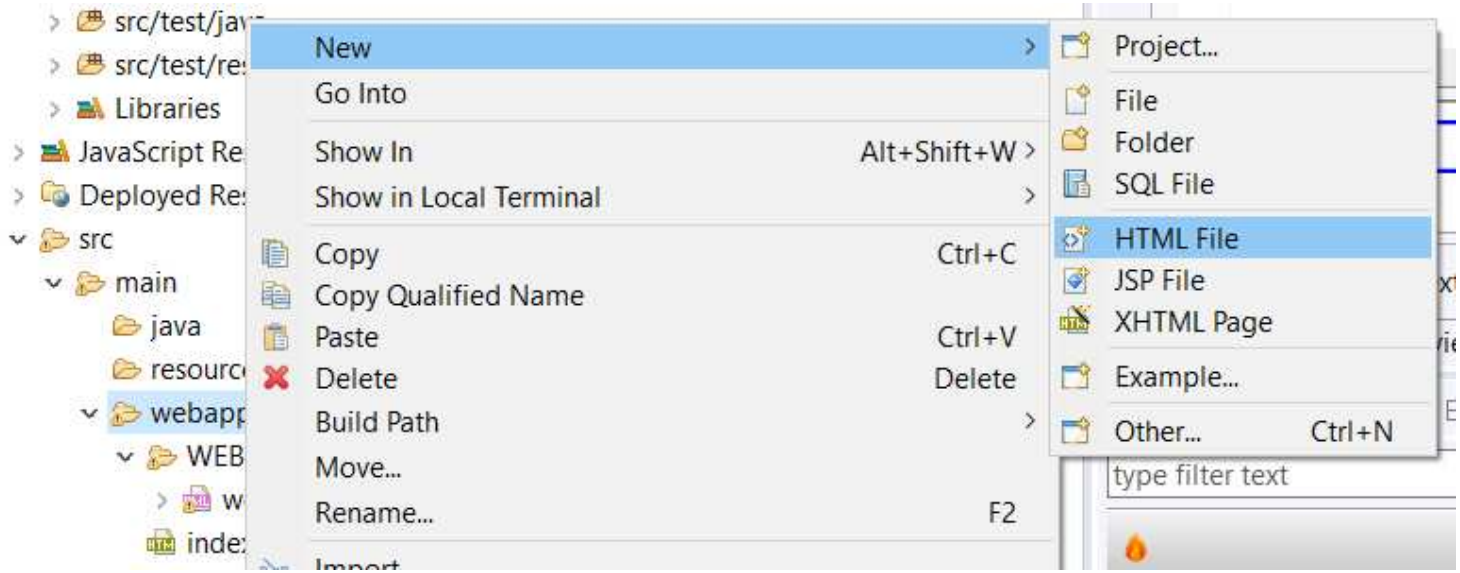
TP2: Projet Web avec Maven (WAR)

- Faites un Maven Update.



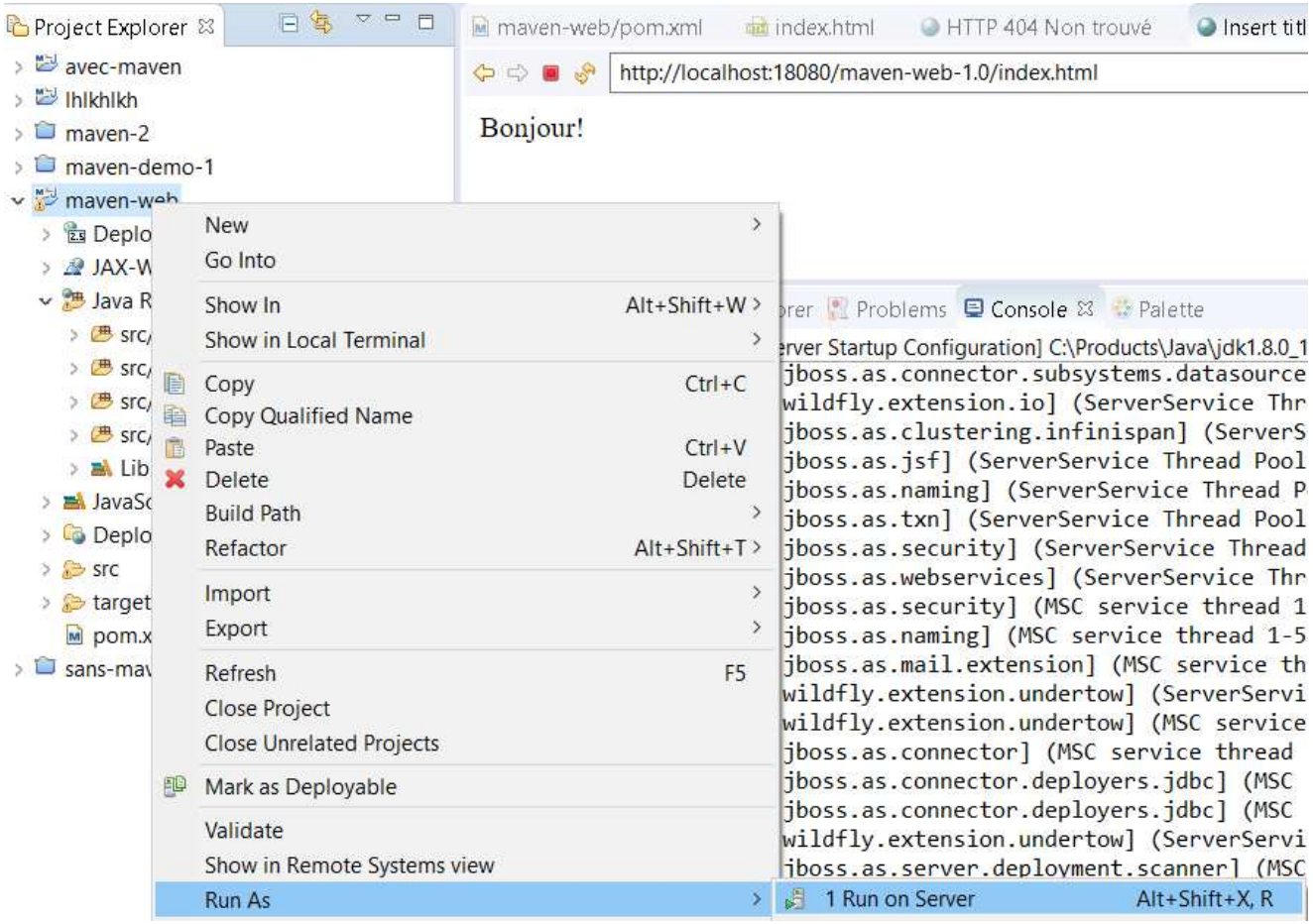
TP2: Projet Web avec Maven (WAR)

- Ajouter une page web basique index.html :



TP2: Projet Web avec Maven (WAR)

- Déployer l'application sur Tomcat, et lancer l'URL : http://localhost:8080/maven-demo-2/



MAVEN

2019-2020

ESPRIT thouraya.louati@esprit.tn UP JavaEE / .NET (Bureau E204)