TP4 avec maven et traitement des fichiers en Java

Tester votre outil maven

```
mvn --version
```

- Quel est l'intérêt d'un outil tel que maven ?
- Quelles alternatives existe-t-il ?

Créer un projet maven de base

```
mvn archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart
```

Ceci donnera une structure du type (avec package essai, artifact : mud)

```
mud
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── essai
    │           └── App.java
    └── test
        └── java
            └── essai
                └── AppTest.java
```

Mettez votre projet MUD sous cette forme

Initialisation maven avec un package précis

```
mvn archetype:generate -DgroupId=fr.univ_orleans.iut45.mud -DartifactId=mud
-DinteractiveMode=false
```

Structure obtenue :

```
mud
├── pom.xml
└── src
    ├── main
    │   └── java
```

```
│       └── fr
│           └── univ_orleans
│               └── iut45
│                   └── mud
│                       └── App.java
└── test
    └── java
        └── fr
            └── univ_orleans
                └── iut45
                    └── mud
                        └── AppTest.java
```

Mise en place du projet dans l'arborescence

- Déplacer les fichiers pour obtenir l'arborescence suivante :

```
├── pom.xml
├── src
│   ├── main
│   │   └── java
│   │       └── fr
│   │           └── univ_orleans
│   │               └── iut45
│   │                   └── mud
│   │                       ├── App.java
│   │                       ├── Box.java
│   │                       └── Thing.java
│   └── test
│       ├── java
│       │   └── fr
│       │       └── univ_orleans
│       │           └── iut45
│       │               └── mud
│       │                   └── TestBoxes.java
│       └── resources
│           ├── test1.json
│           ├── test2.json
│           ├── test3.json
│           └── test4.json
```

Les fichiers json sont ici utilisés dans les tests donc placés dans le dossiers resources de l'arborescence *src/test*.

- Créer les packages correspondants dans les fichiers

En ajoutant la ligne :

```
package fr.univ_orleans.iut45.mud;
```

au début des fichiers Thing.java, Box.java et TestBoxes.java.

Essayez les commandes `mvn clean`, `mvn test` et `mvn compile`. Que se passe-t-il ?

Ajout de règles à mvn pour la lib gson

GSON, ajouter une *dependency* gson après celle de junit.

```
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.10.1</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>
```

Puis ajouter la règle properties dans pom.xml après dependencies.

```
 <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
 </properties>
```

Il faut aussi adapter le code dans TestBoxes.java pour pouvoir accéder aux fichiers de Test placés dans test/resources/ :

```
@Test
    public void testBoxFromJSONSimple(){
        // use the file test1.json in ressources folder
        String path =
TestBoxes.class.getClassLoader().getResource("test1.json").getFile();
        Box b = Box.fromJSON(path);
        assertEquals(b.capacity(), -1);
        assertTrue(b.isOpen());
    }
```

Vous devriez maintenant pouvoir exécuter `mvn test` !

Autres cibles : package et exec:java

Pour packager, complétons la classe principale App.java, par exemple :

```
package fr.univ_orleans.iut45.mud;

/**
 * main App
 *
 */
public class App
{
    public static void main( String[] args )
    {
        Box b = new Box();
        Thing truc = new Thing("truc");
        Thing machin = new Thing("machin");
        b.add(truc);
        b.add(machin);
        b.actionLook();
    }
}
```

Puis configurer le plugin maven-shade-plugin pour packager votre App dans un fichier jar directement exécutable avec `mvn package`.

```
<build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.4.1</version>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
            <configuration>
              <transformers>
                <transformer

implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTra
nsformer">
                  <mainClass>fr.univ_orleans.iut45.mud.App</mainClass>
                </transformer>
              </transformers>
            </configuration>
          </execution>
```

```
          </executions>
        </plugin>
      </plugins>
```

Où est placé le jar obtenu ? Comment l'utiliser ?

puis ajoutez le plugin maven-exec-plugin pour executer directement l'App :

```
    <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>exec-maven-plugin</artifactId>
          <version>1.6.0</version>
          <configuration>
          <mainClass>fr.univ_orleans.iut45.mud.App</mainClass>
          </configuration>
    </plugin>
```

Tapez `mvn exec:java` pour cette nouvelle cible.

Au total, quelles cibles mvn sont-elles maintenant disponibles ?

Ce n'est pas la peine de versionner le dossier *target* !