TP Outil de développement logiciel : build #2

Rendre une archive par TP sur Celene ; un rendu par étudiant

Consignes générales

— Tout ce qui est nécessaire au TP est installé dans la VM. Il y a un compte "sti" mot de passe "sti"
— Vous rendez vos rapports pour le contrôle continu sur Celene, avant minuit le jour du TP, sous forme d'archive contenant un rapport en pdf ainsi que tout code utile

But de ce TP

— Comprendre le fonctionnement de Maven.
— Etre capable d'utiliser Maven en ligne de commande.
— Etre capable d'utiliser maven au sein d'Eclipse.

Les mots-clefs liés à ce TP dont on aura pas le temps de parler

Mais que vous devriez creuser par vous-même à l'occasion : make, autoconf, cmake, ant, maven, rake, nix, guix, buildr, gradle, msbuild, nant, sbt, waf

1 Du pain sur la planche : Ant

Vous trouvez vraiment que make, doxygen et CMake sont des outils un petit pau datés. Vous vous dites que Java + javadoc + Ant serait une super bonne alternative ! Vous vous décidez donc à reprendre tout de zéro et recoder le tout en Java.Vous avez mis tout ça au propre dans un répertoire de la VM, sous votre compte : `~/TP_Arithmetics/Java`

— Importez le projet dans Eclipse
— Générez un script Ant pour le projet
— Copiez votre projet dans un répertoire, et exécutez Ant à la main
— Générez la javadoc.

Question Quelles sont les cibles crées par défaut dans le build.xml ?

Question Donnez la ligne de commande pour exécuter Ant et son résultat dans votre rapport.

2 Prologue au TP de Maven

vous êtes donc passionné(e) de chiffres et avez commencé à développer votre propre projet dédié aux questions de ppcm. Après avoir erré en quête d'outils plus efficace pour gérer votre documentation, vous avez fini par utiliser Ant.

Mais votre quête n'est pas finie : vous voudriez pour votre projet Java plus de structure imposée par votre builder, une gestion des dépendances : vous vous tournez naturellement vers maven.

Maven est un outil de build : de ce côté là il ressemble à Ant ou CMake (make étant dépendant de l'OS, contrairement à CMake ou Ant). Il offre en plus une gestion de dépendances basée sur des dépôts (*repositories*) plus ou moins centralisés. On décrit donc ces dépendances en fonction de ses dépôts.

De plus, si maven fonctionne aussi en exprimant les étapes de la construction (de binaire, de doc et autre) sous forme de document en xml, il est plus proche de la phlcmilosophie de CMake car la construction se fait en se basant sur des primitives et pas des cibles.

L'essentiel des choses se passe au niveau de Maven à la racine du projet dans un fichier appelé POM.xml (pour Project Object Model). Les POM.xml suivent des *archétypes* qui sont des organisations classiques pour un certain nombre de projets habituels (Spring, Struts par exemple).

Si Ant se base sur des cibles, Maven se base sur des buts (*goals*) qui font référence au cycle de vie standard d'un logiciel. Les principaux sont :

— compile
— test
— package
— install
— deploy

C'est un ordre strict : deploy ne peut être réalisé que si install l'est, package ne peut être réalisé que si test l'est, etc, etc.

Eclipse propose un plugin pour le support de maven. Attention, c'est une installation dans l'environnement Eclipse. Il y a aussi dans la VM maven qui est installé.

Question Ouvrez Eclipse et créez un projet maven d'archétype *maven-archetype-quickstart*. Le groupid n'a pas grande importance ici (c'est un identificant de groupe de projet). Quelle est la structure de répertoire créé par défaut ?

Question Quelles sont les dépendances présentes dans le POM ?

Vous aimez les maths, mais vous n'êtes pas seul(e) ! D'autre gens de la fondation Apache aiment les maths aussi et ont créé une bibliothèque appelée commons-math qui implémente pas mal de choses standard. C'est cette dépendance qu'il faut rajouter dans votre fichier POM :

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-math3</artifactId>
    <version>3.6</version>
</dependency>
```

Question Rajoutez cette dépendance dans votre POM. Que se passe-t-il dans l'interface d'Eclipse ?

Question Incluez dans votre App.java l'affichage de la décomposition en facteur premier d'un nombre que vous choisirez au hasard (voir la doc ici). Donnez le code. Lancez votre programme.

Question Il est temps de sortir d'Apache. Copiez votre projet dans un nouveau répertoire. Tentez de lancer en ligne de commande mvn pour construire votre programme, et commentez ce que vous voyez dans la console.

Question Lancez votre petit programme à l'aide de la commande `mvn exec:java -Dexec.mainClass="classpath.App"` où $classpath$ est le chemin d'accès à votre programme et incluez la sortie console dans votre rapport.

3 Jouons un peu avec le POM

Il est plus pratique d'avoir un jar exécutable contenant un MANIFEST indiquant quelle est la classe main à lancer. Le code suivant doit être rajouté dans le POM pour pouvoir générer le MANIFEST :

```
  <plugins>
   <plugin>
<artifactId>maven-assembly-plugin</artifactId>
<version>3.0.0</version>
<configuration>
<descriptorRefs>
<descriptorRef>jar-with-dependencies</descriptorRef>
</descriptorRefs>
<archive>
<manifest>
<mainClass>classpath.App</mainClass>
</manifest>
</archive>
</configuration>
```

```
<executions>
<execution>
<id>make-assembly</id>
<phase>package</phase>
<goals>
<goal>single</goal>
</goals>
</execution>
</executions>
</plugin>
</plugins>
```

La question est de savoir où dans le document il faut ajouter cet appel à un plugin de maven qui gère le manifest. Chaque phase est décrite plus précisément dans un tag correspondant : *install, deploy, build, compile, test, package*.

Question Trouvez dans quel tag ajouter ce bout de code. De plus il faut que vous modifiez la classe par défaut à lancer suivant le nom que vous avez donné à votre application. Indiquez la phase concernée dans votre rapport.