Méthodologie de développement

Automatisation de code

1 Utilisation de Ant

1.1 Introduction

Apache *Ant* (*Another Neat Tool*) est un outil d'aide à la compilation entièrement écrit en *Java*. Contrairement à `make`, *Ant* est indépendant de la plate-forme sur laquelle il est exécuté et utilise du *XML* plutôt que des commandes Shell. Chaque tâche est exécutée par un objet *Java* ; il est donc possible de créer soit-même de nouvelles tâches pour un projet en particulier.

1.2 Installation

La dernière version de *Ant* est disponible en téléchargement à l'adresse suivante : `https://www.apache.org/dist/ant/binaries/`. Les fichiers ayant l'extension `.zip` sont destinés à *Windows*, et ceux aux extensions `.tar.gz`, `.tar.bz2` aux systèmes *Unix*.

- Décompresser le fichier téléchargé dans un répertoire,
- Créer ou mettre à jour les variables d'environnement suivantes :
  - `JAVA_HOME` indique le dossier dans lequel votre *JDK* est installé,
  - `ANT_HOME` indique le répertoire dans lequel vous avez décompressé *Ant*,
  - `PATH` doit être mis à jour en ajoutant `%ANT_HOME%/bin` pour *Windows* ou `${ANT_HOME}/bin` pour les systèmes *Unix*.
- Depuis le répertoire `ANT_HOME`, exécuter la commande `ant -f fetch.xml -Ddest=system` pour récupérer les bibliothèques dont les commandes de base de *Ant* dépendent.

Il vous est désormais possible d'invoquer *Ant* pour compiler un projet **sans utiliser NetBeans**.

Remarque : Les projets NetBeans créés avec Ant possèdent déjà un fichier `build.xml` que vous pouvez regarder. Il fait référence à un fichier `nbproject/build-impl.xml` qui est automatiquement généré par NetBeans en fonction des paramètres de votre projet et qui contient les cibles principales.

Ne l'éditez pas.
Vous pouvez en revanche pour la suite du TP modifier le fichier `build.xml` *d'un projet NetBeans Ant ou en créer un autre.*

Remarque : Pour voir l'utilisation et la liste des paramètres possibles de la commande ant, tapez *ant -help* dans un terminal

1.3 Notions de base et fonctionnement

Comme dit dans l'introduction, *Ant* travaille avec des fichiers *XML*. Par convention, le fichier de construction d'un projet géré par *Ant* s'appelle `build.xml`. Il est possible d'utiliser un autre nom de fichier, mais dans ce cas il faudra le préciser lors de l'invocation de *Ant* sur votre projet. Un fichier de *build* est une suite de cibles, chacune composée de tâches.

Chaque fichier de *build* doit commencer par la ligne suivante, indiquant qu'il s'agit d'un fichier contenant du *XML* et que l'encodage des caractères utilisé est l'*UTF-8* :

```
<?xml version="1.0" encoding="UTF-8"?>
```

Vous allez construire petit à petit votre fichier `build.xml` pour compiler le projet Java de votre choix. Vérifiez que votre fichier est correct à chaque étape avant de passer à la suivante en exécutant *ant*.

1.3.1 Invocation de *Ant*

Par convention, le fichier de *build* se nomme `build.xml`. En invoquant la commande `ant` à l'endroit où se trouve votre fichier de *build*, *Ant* sera lancé sur la cible par défaut de votre projet.

Si votre fichier ne se nomme pas `build.xml`, vous devrez invoquer *Ant* de la manière suivante : `ant -buildfile myBuild.xml` ou `ant -f myBuild.xml`. Pour invoquer une autre cible que celle par défaut, il suffit de donner le nom de cette cible à *Ant* lors de l'invocation : `ant myTarget`.

1.3.2 Concepts de base

L'élément `project` L'élément `project` est la racine de tout fichier de *build*. Il possède trois attributs :

- `name` est le nom du projet,
- `default` est la cible utilisée par défaut lorsque aucune cible n'est spécifiée,
- `basedir` est le répertoire de base utilisé pour les opérations sur les chemins.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="testAnt" default="build" basedir=".">
       <description>This is a simple test project.</description>
</project>
```

L'élément `target` Chaque projet définit au moins une cible. Une cible est un ensemble de tâches que l'on souhaite exécuter. Elle peut posséder une description, une condition d'exécution basée sur la présence ou non d'une certaine propriété ainsi qu'une liste de dépendances. Les dépendances sont d'autres cibles qui doivent être exécutées avant la cible elle-même, mais chaque cible n'est exécutée qu'une seule fois

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="testAnt" default="build" basedir=".">
        <description>This is a simple test project.</description>

        <target name="checkDependencies"
                description="Check if dependencies are satisfied">
        </target>

        <target name="build" description="Build the application"
                depends="checkDependencies" if="src.available">
        </target>
</project>
```

Remarque : Dans l'exemple précédent, la cible par défaut est `build` et le répertoire de base (`basedir`) est le dossier courant.

L'élément `property` Les propriétés sont globales au projet. Chaque propriété a un nom **sensible à la casse** et une valeur. Elles peuvent être affectées à l'intérieur ou à l'extérieur des cibles. La valeur d'une propriété ne peut pas être modifiée : une propriété n'est pas une variable ! La valeur d'une propriété est accessible lorsque son nom est enfermé par `${` et `}`. Il existe plusieurs manières de les définir, dont les plus courantes sont listées ci-après :

- `<property name="foo.dist"value="dist"/>` initialise la propriété `foo.dist` à la valeur `dist`,
- `<property name="src.location"location="src"/>` initialise la propriété `src.location` avec le chemin absolu correspond à la valeur de l'attribut `location` ; si `location` est un chemin absolu, la valeur est inchangée, sinon le chemin absolu généré se base sur le `basedir` du projet,
- `<property file="project.properties"/>` lit et crée un ensemble de propriétés à partir du fichier pointé par l'attribut `file` (le fichier utilise la même syntaxe que les propriétés *Java* `java.util.Properties`),

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="testAnt" default="build" basedir=".">
        <description>This is a simple test project.</description>

        <target name="checkDependencies"
                description="Check if dependencies are satisfied">
            <property name="src.location" location="src"/>
            <property name="classes.location" location="build/classes"/>
        </target>

        <target name="build" description="Build the application"
                depends="checkDependencies" if="src.available">
        </target>
</project>
```

Remarque : la condition `src.available` de la cible `build` n'est pas encore définie et le sera par la suite.

L'élément `echo` L'élément `echo` permet d'afficher du texte sur la sortie standard lors de l'exécution de *Ant* sur votre projet, via l'attribut `message`.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="testAnt" default="build" basedir=".">
        <description>This is a simple test project.</description>

        <target name="checkDependencies"
                description="Check if dependencies are satisfied">
            <property name="src.location" location="src"/>
            <property name="classes.location" location="build/classes"/>

            <echo message="Source directory is ${src.location}"/>
            <echo message="Classes directory is ${classes.location}"/>
        </target>

        <target name="build" description="Build the application"
                depends="checkDependencies" if="src.available">
            <echo message="Building application..."/>
            <echo message="Done!"/>
        </target>
</project>
```

L'élément `condition` L'élément `condition` permet de créer une propriété avec une certaine valeur si la condition exprimée dans l'élément est vraie.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="testAnt" default="build" basedir=".">
        <description>This is a simple test project.</description>

        <target name="checkDependencies"
                description="Check if dependencies are satisfied">
                <property name="src.location" location="src"/>
                <property name="classes.location" location="build/classes"/>

                <echo message="Source directory is ${src.location}"/>
                <echo message="Classes directory is ${classes.location}"/>

            <condition property="src.available">
                <and>
                        <available file="${src.location}" type="dir"/>
                        <available file="${classes.location}" type="dir"/>
                </and>
            </condition>
        </target>
```

```
        <target name="build" description="Build the application"
                depends="checkDependencies" if="src.available">
            <echo message="Building application..."/>
            <echo message="Done!"/>
        </target>
</project>
```

L'élément `javac` L'élément `javac` permet de compiler un ensemble de sources *Java* pour obtenir des fichiers `.class` en sortie. Il possède 46 attributs différents [1], mais un seul d'entre eux est obligatoire, `srcdir`, qui indique à quel endroit sont les sources à compiler.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="testAnt" default="build" basedir=".">
        <description>This is a simple test project.</description>

        <target name="checkDependencies"
description="Check if dependencies are satisfied">
<property name="src.location" location="src"/>
<property name="classes.location" location="build/classes"/>

<echo message="Source directory is ${src.location}"/>
<echo message="Classes directory is ${classes.location}"/>

<condition property="src.available">
<and>
<available file="${src.location}" type="dir"/>
<available file="${classes.location}" type="dir"/>
</and>
</condition>
</target>

        <target name="build" description="Build the application"
                depends="checkDependencies" if="src.available">
            <echo message="Building application..."/>

            <javac srcdir="${src.location}" destdir="${classes.location}"
                includeAntRuntime="false"/>

            <echo message="Done!"/>
        </target>
</project>
```

1. Vous trouverez la liste des attributs possibles dans la documentation de ant `http://ant.apache.org/manual/` (Ant Task>List of Tasks>Javac).

1.3.3 Pour aller plus loin

Il existe des dizaines de tâches *Ant*, dont les principales sont : Ant, AntCall, AntStructure, AntVersion, Apply/ExecOn, Apt, Available, Basename, BuildNumber, BUnzip2, BZip2, Checksum, Chmod, Concat, Condition, Supported conditions, Copy, Copydir, Copyfile, Cvs, CvsChangeLog, CvsVersion, CVSPass, CvsTagDiff, Defaultexcludes, Delete, Deltree, Dependset, Diagnostics, Dirname, Ear, Echo, EchoXML, Exec, Fail, Filter, FixCRLF, GenKey, Get, GUnzip, GZip, Import, Input, Jar, Java, Javac, Javadoc/Javadoc2, Length, LoadFile, LoadProperties, LoadResource, MakeURL, Mail, MacroDef, Manifest, ManifestClassPath, Mkdir, Move, Nice, Parallel, Patch, PathConvert, PreSetDef, Property, Record, Rename, Replace, ResourceCount, Retry, Rmic, Sequential, SignJar, Sleep, Sql, Subant, Sync, Tar, Taskdef, Tempfile, Touch, Truncate, TStamp, Typedef, Unjar, Untar, Unwar, Unzip, Uptodate, Waitfor, WhichResource, War, XmlProperty, XSLT/Style, Zip.

Il est donc impossible de les présenter ici de manière exhaustive. C'est pourquoi vous devrez durant ces travaux pratiques consulter la documentation de *Ant* à l'adresse suivante : `http://ant.apache.org/manual/`. Toutes les tâches dont vous aurez besoin y sont décrites et illustrées par de nombreux exemples.

1.4 Ajout de cibles à votre projet avec *Ant*

L'objectif de la suite du TP est de continuer à gérer la compilation d'un projet avec *Ant*, toujours en se passant de l'IDE *Netbeans*. En vous basant sur votre projet en cours, vous devrez compléter votre fichier `build.xml`. L'invocation de *Ant* devra (en plus de la compilation déjà faite) :

- compilez les sources de votre projet pour obtenir un *jar*
  `http://ant.apache.org/manual/Tasks/jar.html`,
- générez la *javadoc* de votre projet à partir de vos sources,
  `http://ant.apache.org/manual/Tasks/javadoc.html` (vous trouverez aussi un exemple ici `https://www.jmdoudoux.fr/java/dej/chap-ant.htm`),
- lancez les tests unitaires de votre projet
  `https://ant.apache.org/manual/Tasks/junitlauncher.html` et en particulier inspirez-vous de `https://github.com/junit-team/junit5-samples/tree/r5.5.2/junit5-jupiter-starter-ant`,
- lancer `sonar-scanner` sur votre projet pour vérifier la qualité de votre code (déjà fait au tp précédent)
  `https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-ant/`.

2 Maven

Il est possible de faire exactement la même chose avec un autre outil d'automatisation : `Maven` `http://maven.apache.org/what-is-maven.html`.

Téléchargez `http://maven.apache.org/download.cgi` et installez `http://maven.apache.org/install.html` `maven` puis suivez le tutoriel suivant `http://maven.apache.org/guides/getting-started/index.html` pour créer un projet maven (sans NetBeans) et faire les mêmes actions que dans votre script *Ant*. Vous pouvez éventuellement commencer par ce tutoriel plus court `http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html`.

3 Jenkins

3.1 Introduction

Jenkins est un outil open source d'intégration continue. Écrit en Java, Jenkins fonctionne dans un conteneur de servlets tel qu'Apache Tomcat, ou en mode autonome avec son propre serveur Web embarqué.

Il s'interface avec des systèmes de gestion de versions tels que CVS, Git et Subversion, et exécute des projets basés sur Apache Ant et Apache Maven aussi bien que des scripts arbitraires en shell Unix ou batch Windows.

Les générations de projets peuvent être amorcées par différents moyens, tels que des mécanismes de planification similaires au cron, des systèmes de dépendances entre générations, ou par des requêtes sur certaines URL spécifiques.

source : `https://fr.wikipedia.org/wiki/Jenkins_(logiciel)`

3.2 Installation

Commencez par télécharger la version de Jenkins Long-Term Support qui correspond à votre système : `https://jenkins.io/download/`.

Ensuite, pour l'installation, vous suivrez la documentation suivante : `https://jenkins.io/doc/pipeline/tour/getting-started/`. *Inutile d'installer Docker.*

Après avoir téléchargé, installé et exécuté Jenkins, l'assistant de configuration post-installation commence. Vous trouverez des explications supplémentaires sur la marche à suivre ici : `https://jenkins.io/doc/book/installing/#setup-wizard`. Vous installerez les plug-ins recommandés par la communauté.

3.3 Configuration

Une fois votre installation locale terminée, vous devez configurer Jenkins via le menu "Administrer Jenkins".

Dans le menu "Configurer le système" vous trouverez la position du répertoire Home où Jenkins va créer et/ou copier ses fichiers.

Dans "Configuration globale des outils" renseignez la partie JDK avec le chemin d'accès à votre JAVA_HOME et la partie *Ant* avec les information concernant votre installation de *Ant* (ou installez le automatiquement d'Apache).

3.4 Travail demandé - création d'un item de build avec Ant

Créez une version plus simple du build.xml précédent ne faisant que la génération de la documentation (par exemple). Vous pourrez rajouter d'autres commandes ultérieurement.

Nous allons créer un item Jenkins pour votre projet. Dans "Nouveau Item" choisissez "Construire un projet free-style". Précisez la "Gestion de code source" selon votre système de gestion de version. Précisez aussi "Ce qui déclenche le build", ici se sera par exemple "Scrutation de l'outil de gestion de version" qui vous planifierez toutes les 10 min (voir la documentation).

Dans la partie "Build" vous ajouterez comme étape au Build "Appeler Ant" en précisant la version de Ant utilisée. Si votre fichier build.xml de Ant n'est pas dans la racine de votre projet, vous préciserez son chemin d'accès à partir de la racine de votre projet dans les paramètres avancés dans la partie "Fichier de Build".

Vous pouvez ensuite lancer votre build qui va utiliser votre script ant et qui sera lancé toutes les 10 min.

Vous trouverez des explications supplémentaires sur les tâches de build ici :
`https://jenkins-le-guide-complet.github.io/html/chapter-build-jobs.html`.