TP 1 - ANT

Configuration

Vérifiez que ant est bien installé sur votre machine. Ouvrez une shell et écrivez:

ant -h

Vous devriez voir l'aide en ligne de *ant* avec les différentes options.

Pour installer et utiliser ant chez vous:
*Installez le package java (**jdk**)*
https://www.oracle.com/fr/java/technologies/javase/javase-jdk8-downloads.html
Vérifiez que Java est bien installé: si vous faites echo $JAVA_HOME vous devriez voir le chemin d'accès à l'interprète Java.

Installer ANT depuis: https://ant.apache.org/bindownload.cgi (1.10.9 release)
*Vérifiez si la variable ANT_HOME existe avec **echo $ANT_HOME**. Si ANT_HOME n'existe pas, modifiez votre fichier .bashrc ou .profile avec*
`export ANT_HOME="chemin du binaire ant"`
*vous pouvez trouver le chemin du binaire ant avec la commande **which ant**.*

Pour les utilisateurs Windows:
vérifiez que java et ant sont sur le path
https://www.java.com/fr/download/help/path.html

Le projet "Banque"

On va implémenter un simple programme qui simule des comptes bancaires et l'accès à ces comptes. Ils nous faudra une classe Banque et une classe Compte.
Créer un dossier "projbanque" avec un sous-répertoire "src"::

projbanque
|
+-src

a) Ecrire un fichier Banque.java qui contient l'implémentation d'une classe Banque avec les méthodes suivantes:

- public void ouvrirCompte(String nom, String password):
  - Ce méthode prend le nom de l'utilisateur et un mot de passe, s'il existe déjà, il soulève une exception, sinon, il crée un Compte avec 0€ et met le compte dans

une table Hash (Hashtable ou HashMap) qui associe le nom de l'utilisateur à son compte.

- public Compte verifie(String nom, String password):
    - Si le compte correspondant (c'est à dire, avec le même nom utilisateur et mot de passe) existe, il retourne le compte correspondant.
    - Si le mot de passe est incorrecte, il affiche un erreur et il sorte du programme
- public int fermerCompte(String nom, String password):
    - Si le compte correspondant existe, le compte est retiré et renvoie le solde disponible
- public void deposer(int somme, Compte c):
    - Dépose la somme sur le compte c
- public int retirer(int somme, Compte c):
    - Rétire la somme du compte c; renvoie le solde après l'opération
- public int solde(Compte c):
    - donne le solde disponible sur le compte c

b) Ecrire un fichier Compte.java qui contient l'implémentation d'une classe Compte avec les informations suivantes:

- le nom du propriétaire du compte
- un mot de passe
- le solde du compte

c) Finalement, écrire un dernier fichier (TestBanque.java) qui contient l'implémentation d'une classe qui contient le main et permet d'interagir avec la banque et ses comptes.

Sauvegarder tous les fichiers dans le dossier **src** précédemment créé.

Compilation du projet avec ANT

La structure d'un buildfile ANT est:

```
<project name="nom_du_projet" default="cible_par_defaut"
basedir="repertoire_par_defaut">

     <target name="nom_cible" depends="cible_nécessaires">
     ...tâches...
     </target>
     ...cibles...
</project>
```

Ecrivez maintenant votre fichier **build.xml** pour le projet. Le fichier doit être contenu dans le répertoire *projbanque*.

1.1 Le build.xml aura pour nom de projet “projbanque”, avec une cible nommée “compile” qui contient une seule tâche[1] qui fait la compilation des sources contenues dans le dossier “src”.

Une tâche de compilation est spécifiée de la façon suivante; `includes`, `excludes` et `classpath` sont des attributs facultatifs.

```
<javac srcdir="répertoire des sources"
         destdir="répertoire où sauvegarder les classes compilées"
         includes="fichiers à inclure, séparés par des , "
         excludes="fichiers à exclure"
         classpath="classpath des libs utilisées par le projet"
  />
```

1.2 Compilez maintenant votre projet avec la commande **ant**. Quel est le résultat? Modifiez le buildfile pour lancer automatiquement la cible “compile”.

1.3 Ajoutez dans la cible “compile” une tâche **echo** pour écrire à l’écran le message “Compilation en cours…” : <echo message="Compilation en cours..." />
et afficher “compilation finie” après la compilation.

1.4 Rajoutez au buildfile une cible “prepare” qui crée un répertoire nommé **bin**
La tâche pour créer un répertoire est **mkdir**:
`<mkdir dir="repertoire" />`

1.5 Modifiez la cible “compile” pour sauvegarder le résultat de la compilation dans le répertoire **bin.** La cible “compile” doit dépendre de la cible “prepare”. Lancez ant et vérifiez le résultat.

1.6 Créez une cible “clean” pour effacer les classes dans le répertoire bin. Pour effacer la tâche est **delete**:

`<delete file="/lib/ant.jar"/>` efface le fichier /lib/ant.jar.

`<delete dir="lib"/>` efface le répertoire lib, son contenu et tous ses sous-répertoires (ça correspond à rm -rf).

Lancez **ant clean** pour nettoyer le répertoire.

1.7 Créez une propriété **source** et une propriété **classes** pour le nom du répertoire src et le nom du répertoire bin.

---

[1] Une référence pour les tâches disponibles dans ANT: http://www.jmdoudoux.fr/java/dej/chap-ant.htm

Une propriété est créé comme suit:
`<property name="nom" value="valeur" />`

1.8 Modifiez les autres cibles pour utiliser les propriétés au lieu des noms des répertoires (une propriété est utilisée comme ${nom_propriété}. Testez votre buildfile.

2. Créez un autre buildfile nommé **build1.xml**. Vous pouvez utiliser ant avec un buildfile spécifique en exécutant `ant -buildfile` *`nom_fichier_build`*
Ajoutez une cible **build** a votre projet qui crée un jar du projet dans un répertoire **build** en utilisant la commande jar.
Exemple: `<jar destfile="build/app.jar" basedir="bin/classes"/>`
Cette nouvelle cible doit évidemment être dépendante de la cible *compile*. Le nom du fichier jar et le nom du répertoire cible doivent être facilement paramétrables. N'oubliez pas de mettre à jour la cible *clean* pour qu'elle supprime le fichier .jar en plus des classes.

2.1 Ajoutez une condition **if** pour la compilation du .jar : si l'option lib est selectionnée, alors il faut compiler le .jar uniquement avec le code de Banque.java et Compte.java, en cas contraire, le .jar contiendra aussi le code de TestBanque.java

Rappel: le **if** s'obtient de la forme suivante:
```
<target name="setupProduction" if="${production}"> … </target>
<target name="setupDevelopment" unless="${production}"> … </target>
```

Testez votre solution avec la commande **`ant -Dlib=true`**

2.2 Ajoutez une cible pour lancer le programme TestBanque. La tâche pour lancer un programme est la tâche **java**:

```
<java classname=”nom_classe” />
```

Vérifiez que la cible dépend des cibles nécessaires pour la compilation.

2.3 Modifiez le buildfile pour demander à l'utilisateur le nom du dossier à créer pour sauvegarder les fichiers .class résultat de la compilation.

La tâche **input** permet l'interaction avec l'utilisateur:

```
<input
    message="Please enter db-username:"
    addproperty="db.user"
/>
```

2.4 Modifiez le buildfile pour utiliser un fichier externe (“`build.properties`”) pour stocker les propriétés

2.5 Vérifiez l’existence des fichiers .java avant de compiler. Pour vérifier l’existence d’un fichier vous pouvez utiliser la tâche **available**:

```
<target name="check_jar">
      <available file="${file}" property="found"/>
      <antcall target="check_message"/>
</target>

<target name="check_message" unless="found">
       <echo message="Could not find ${file}"/>
</target>
```

Graphes de dépendance

3. Dessiner le graphe de dépendance du buildfile obtenu.

(optionnel) 3.1 Téléchargez le buildfile du projet dbcp de Apache:
https://github.com/apache/commons-dbcp/blob/master/test-jar.xml

et écrire son graphe de dépendance

(optionnel) 3.2 Modifier le build.xml et/ou les fichiers des propriétés pour compiler et installer dans un répertoire appelé *dbcp* dans votre répertoire personnel