Automatisation des tests et Reporting (Ant)

1 La tâche Ant JUnit

Ant Junit est une tâche dont le rôle principal est d'exécuter un ou plusieurs tests Junit puis générer les rapports résultats sous différents formats text, xml ou html.

Exemple de tâche junit avec le nom de la classe de test et son classpath :

```
<?xml version="1.0" encoding="UTF-8" ?>
<project default="test" basedir=".">
....
<target name="test">
 <junit>
  <classpath refid="test.classpath"/>
  <test name="monPackage.nomMaClasseTest"/>
 </junit>
</target>
</project>
```

2 Structure des répertoires du code adaptée aux tests

Il est recommandé de séparer le code du test d'un projet (build/test) du code de production (build/src). De plus on va configurer le build de Ant de telle sorte à :

- Compiler les classes de tests dans **build/test/classes** et celle du code vers **build/classes**
- Générer les fichiers des rapports dans **build/test/reports.**

Remarque : on peut créer la structure du projet à la main ou ajouter une cible pour initialiser cette structure des répertoires de manière automatique par Ant (Voir la dernière section).

On va donc ajouter des cibles Ant dans le build du projet comme suit :

1. Cible pour initialiser la structure des répertoires des tests (optionnel)
2. Cible pour la compilation
3. Cible pour l'exécution des tests
4. Cible pour la génération des rapports

Pour éviter les chemins codés en dur et donc faciliter des modification de la structure (ex : placer les rapports sur un serveur web), on va utiliser des paramètres (balise property du build) comme dans l'exemple suivant. Les valeurs des propriétés doivent être modifiées en fonction de la structure adoptée pour votre projet. Vous pouvez rajouter d'autres paramètres en cas de besoin.

```
<property name="test.dir" location="${build.dir}/test"/>
<property name="test.data.dir" location="${test.dir}/data"/>
<property name="test.reports.dir" location="${test.dir}/reports"/>
```


Cibles pour la compilation

Pour compiler on a besoin de définir les classpath (balise path du build) comme suit :
compile.classpath définit le classpath pour le code de production.

Le classpath de compilation pour le code de production est :

```
<path id="compile.classpath">
 <pathelement location="src"/>
</path>
```

Le classpath nécessaire pour compiler et exécuter les tests est différent de celui du code de production. compile.classpath sera ré-utilisé pour définir le classpath de compilation et exécution des tests comme suit:

```
<path id="test.classpath">
 <path refid="compile.classpath"/>
 <pathelement location="${build.dir}/classes"/>
 <pathelement location="${build.dir}/test"/>
</path>
```

La cible compile utilise compile.classpath :

```
<target name="compile" depends="init">
 <javac destdir="${build.classes.dir}"
       debug="${build.debug}"
       includeAntRuntime="yes"
       srcdir="${src.dir}">
  <classpath refid="compile.classpath"/>
 </javac>
</target>
```

La cible test-compile utilise test.classpath et test.dir:

```
<target name="test-compile" depends="compile,test-init">
 <javac destdir="${test.dir}"
       srcdir="test">
  <classpath refid="test.classpath"/>
 </javac>
</target>
```

Cible pour l'exécution

```
<junit>
 <classpath refid="test.classpath"/>
 <test name="monPackage.nomMaClasseTest"/>
</junit>
```

Après exécution du build.xml on obtient :

```
[junit] TEST monPackage.nomMaClasse FAILED
BUILD SUCCESSFUL
```

Deux problèmes :

- Manque d'informations sur le test qui a échoué
- Le build affiche une réussite malgré l'échec du test

Pour corriger cela il suffit de rajouter deux attributs à la tâche junit (donc dans la balise <junit>) :

haltonfailure="true" et printsummary="true"

On obtient alors l'affichage amélioré suivant :

```
    [junit] Running nomMonPackage.NomMaClasse
     [junit] Tests run: 1, Failures: 1, Errors: 0, Time elapsed: 0.01 sec
BUILD FAILED
```

3 Récupération des résultats des tests

La tâche Junit collecte les résultats des tests en utilisant les formatters suivants :

<formatter> type	Description
brief	Provides details of test failures in text format.
plain	Provides details of test failures and statistics of each test run in text format.
xml	Provides an extensive amount of detail in XML format including Ant's properties at the time of testing, system out, and system error output of each test case.

Les balises <formatter> peuvent être placées :

- directement sous <junit>
- sous <test>
- sous <batchtest>

Les formatters écrivent les résultats dans :

- des fichiers dans le répertoire spécifié par l'élément <test>, ou
- des fichiers dans le répertoire spécifié par l'élément <batchtest>, ou
- la console Ant si usefile="false"

L'affichage par printsummary de junit est désactivé pour ne pas avoir des duplicats avec l'affichage des formatters.

Exemple :

```
<target name="test" depends="test-compile">
<junit printsummary="false" haltonfailure="true">
  <classpath refid="test.classpath"/>
  <formatter type="brief" usefile="false"/>
  <test name="nomMonPackage.NomMaClasse"/>
</junit>
</target>
```

Ce qui donne le résultat textuel suivant :

```
    [junit] Testsuite: nomMonPackage.NomMaClasse
```



```
    [junit] Tests run: 1, Failures: 1, Errors: 0, Time elapsed: 0.01 sec
    [junit]
    [junit] Testcase: testDoc(nomMonPackage.NomMaClasse):FAILED
    [junit] Title expected:<Test Title> but was:<null>
    [junit] junit.framework.AssertionFailedError:
            Title expected:<Test Title> but was:<null>
    [junit] at nomMonPackage.NomMaClasse.testDoc(NomMaClasse.java:20)
    [junit]
    [junit]

BUILD FAILED
```

4 XML Formatter

Pour sauvegarder les résultat sous format XML :

```
<target name="test" depends="test-compile">
 <junit printsummary="false" haltonfailure="true">
  <classpath refid="test.classpath"/>
  <formatter type="brief" usefile="false"/>
  <formatter type="xml"/>
  <test todir="${test.data.dir}"
       name="nomMonPackage.nomMaClasse"/>
 </junit>
</target>
```

5 Exécuter plusieurs tests grace à la balise batchtest

On peut mettre plusieurs éléments <test> mais cela peut rapidement prendre beaucoup de temps, d'où l'alternative qui consiste à utiliser batchtest et y placez des éléments fileset

Exemple :

```
<target name="test" depends="test-compile">
 <junit printsummary="true" haltonfailure="true">
  <classpath refid="test.classpath"/>
  <formatter type="brief" usefile="false"/>
  <formatter type="xml"/>
  <batchtest todir="${test.data.dir}">
   <fileset dir="${test.dir}" includes="**/*Test.class"/>
  </batchtest>
 </junit>
</target>
```

Remarque: si des classes non Junit ou abstraites sont passées à <junit> dans l'attribut include vous obtiendrez une erreur.

6 Générer les rapports de résultats des tests

Pour générer les rapports, il faut :

1. Écrire les résultats des tests en XML
2. Placer la tâche <junit-report> après la tâche <junit>. Cette tâche rassemble les fichiers XML générés pour chacun des éléments <test> ou <batchtest> en un fichier XML unique nommé

par défaut TESTS-TestSuites.xml. Ensuite elle exécute une transformation XSL sur ce fichier pour générer les rapports.

Exemple :

```
<junitreport todir="${test.data.dir}">
 <fileset dir="${test.data.dir}">
   <include name="TEST-*.xml"/>
 </fileset>
 <report format="frames" todir="$
{test.reports.dir}"/>
</junitreport>
```

7 Générer les rapports en cas d'échec de tests

Si <junit> arrête le build en cas d'échec de cas de test via *haltonfailure* alors le build échouera et ne créera pas les rapports. Pour éviter cela :

1. Désactiver haltonfailure
2. Spécifier les attributs failure-Property et errorProperty
3. Générer les rapports avant la mise en échec du build : pour ce faire rajouter un élément conditionnel <fail> après <junit-report>

```
<target name="test" depends="test-compile">
 <junit printsummary="false"
      errorProperty="test.failed"
      failureProperty="test.failed">
   <classpath refid="test.classpath"/>
   <formatter type="brief" usefile="false"/>
   <formatter type="xml"/>
   <batchtest todir="${test.data.dir}">
     <fileset dir="${test.dir}" includes="**/*Test.class"/>
   </batchtest>
 </junit>

 <junitreport todir="${test.data.dir}">
   <fileset dir="${test.data.dir}">
     <include name="TEST-*.xml"/>
   </fileset>
   <report format="frames"
     todir="${test.reports.dir}"/>
 </junitreport>
 <fail message="Tests failed. Check log and/or reports."
      if="test.failed"/>
</target>
```

Haltonfailure a été enlevée

ErrorProperty et failureProperty ont été rajoutées

Tâche <fail> conditionnelle

8 Exécuter un seul cas de test en ligne de commande

Pour traiter un problème particulier vous aurez besoin d'isoler un test à exécuter comme suit :

1. utiliser : les clauses if/unless sur <test> et <batchtest>
2. utiliser : ant test -Dtestcase=<fully qualified classname>

Exemple :

```
<junit printsummary="false"
      errorProperty="test.failed"
      failureProperty="test.failed">
  <classpath refid="test.classpath"/>
  <formatter type="brief" usefile="false"/>
  <formatter type="xml"/>
  <test name="${testcase}" todir="${test.data.dir}" if="testcase"/>
  <batchtest todir="${test.data.dir}" unless="testcase">
    <fileset dir="${test.dir}" includes="**/*Test.class"/>
  </batchtest>
</junit>
```

ant test -Dtestcase=nomMonPackage.NomMaClasseTest

9 Initialiser l'environnement des tests

Avant d'exécuter <junit> on a besoin de :

1. Créer les répertoires où
   - Les cas de tests seront compilés,
   - Les données résultats seront écrites,
   - Les rapports seront générés.
2. Placer les ressources externes utilisées par les tests dans le classpath.
3. Supprimer les fichiers de données et rapports générés précédemment.

La cible test-init est donc :

```
<target name="test-init">
 <mkdir dir="${test.dir}"/>
 <delete dir="${test.data.dir}"/>
 <delete dir="${test.reports.dir}"/>
 <mkdir dir="${test.data.dir}"/>
 <mkdir dir="${test.reports.dir}"/>
</target>
```

De même la cible init est :

```
<target name="init">
   <mkdir dir="build/classes" />
</target>
```