« Outils » de développement

Programmation Orientée Objet

Licence mention Informatique
Université Lille – FST - Informatique

javac et java

- JAVA est un langage compilé

compilateur = **javac**

`javac` *`SomeClasse`*`.java` ⟶ *`SomeClass`*`.class`

Exécution d'un programme : lancement de la JVM avec **java**

`java` *`SomeClass [args]`*

exécution de la méthode statique de `SomeClass`

```
public static void main(String[] args)
```

CLASSPATH

voir variable système `PATH`

- la variable d'environnement `CLASSPATH` est utilisée pour localiser toutes les classes nécessaires pour la compilation ou l'exécution.
- elle contient la liste des répertoires où chercher ces classes
  les classes fournies de base avec le *jdk* sont automatiquement trouvées
- par défaut elle est réduite au répertoire courant ("`.`").
- il est possible de spécifier un "classpath" propre à une exécution (ou à la compilation avec `javac`) :

  (Windows) : `java -classpath "lib;.;/truc/classes" SomeClass`
  (Linux) : `java -classpath lib:.:/truc/classes SomeClass`

Paquetages

~ bibliothèques JAVA

- regrouper les classes selon un critère (arbitraire) de cohésion :
  - dépendances entre elles (donc réutiliser ensemble)
  - cohérence fonctionnelle
  - ...
- un paquetage peut aussi être décomposé en « sous-paquetages »
- le nom complet de la classe `NomClasse` du sous-paquetage `souspackage` du package `nompackage` est :

  `nompackage.souspackage.NomClasse`

  notation UML : **`nompackage::souspackage::NomClasse`**

Utilisation de paquetages

- utiliser le nom complet :

```
new java.math.BigInteger("123");
```

- importer la classe : `import`
  - permet d'éviter la précision du nom de paquetage avant une classe (sauf si ambiguïté)
  - on peut importer tout un paquetage ou seulement une classe du paquetage.
  - la déclaration d'importation d'une classe se fait avant l'entête de déclaration de la classe.

```
import java.math.BigInteger;
public class AClass {
   ...  new BigInteger("123");
}
```

```
import java.math.*;
public class AClass {
   ...  new BigInteger("123");
}
```

l'importation `java.lang.*` est toujours réalisée

Création de paquetage

elle est implicite

- *déclaration* : première ligne de code du fichier source :

  `package` ***`nompackage`***;

  ou `package` ***`nompackage.souspackage`***;

- convention : nom de paquetage en minuscules
  - le paquetage regroupe toutes les classes qui le déclarent.
  - une classe ne peut appartenir qu'à un seul paquetage à la fois.

Assurer l'unicité des noms : utilisation des noms de domaine "renversés"

`fr.univ-lille.l2info.project`

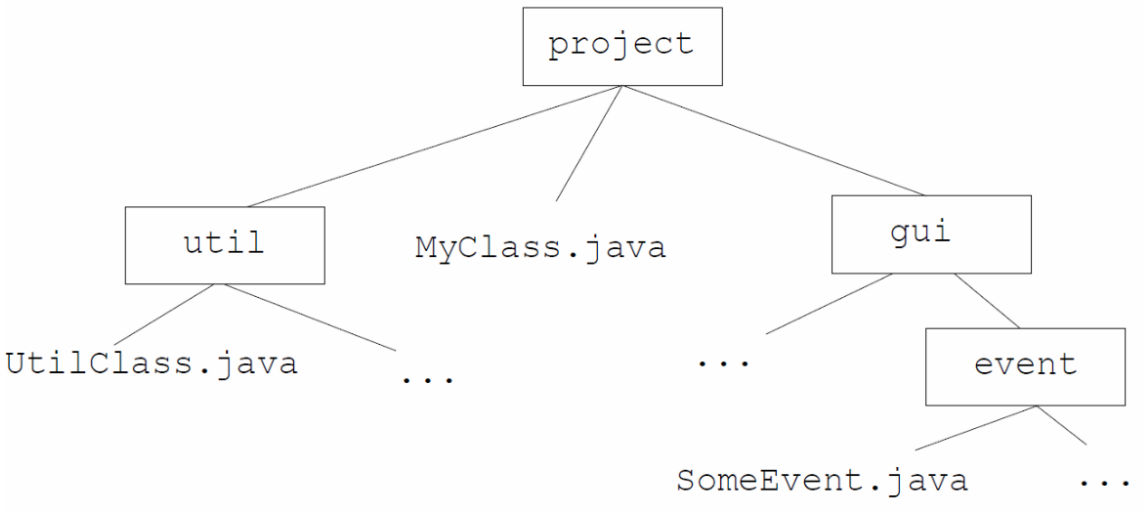
Correspondance avec la structure de répertoires

- à chaque paquetage doit correspondre un répertoire de même nom.
- les fichiers sources des classes du paquetage **doivent** être placés dans ce répertoire.
- chaque sous-paquetage est placé dans un sous-répertoire (de même nom).

Règle

Il faut toujours **toujours** créer un paquetage

- permettre une réutilisation sans craindre l'ambiguïté de nom.
- permettre la diffusion des classes, utilisation dans autres contextes.




```
project
project.util
project.gui
project.gui.event
```

à partir de la **racine des paquetages** :

`javac project/*.java` `javac project/util/*.java`

et les fichiers `.class` sont placées dans une hiérarchie de répertoires copiant celle des paquetages/sources

`java project.util.utilClass` *[args]*

et il faut que le répertoire racine du répertoire `nompackage` soit dans le `CLASSPATH`

javadoc

SomeClass.java ⟶ *SomeClass*.html

- Commentaires encadrés par `/** ... */`
- utilisation possible de tags HTML
- Tags spécifiques :
  - classe `@version`, `@author`, `@see`, `@since`
  - méthode `@param`, `@return`, `@exception`, `@see`, `@deprecated`
- conservation de l'arborescence des paquetages
- liens hypertextes "entre classes"

```
javadoc -sourcepath src src/bigproject/JavaDocExample.java -d docs
```

```
package bigproject;
/** description de la classe, sa responsabilité
* @author <a href=mailto:bilbo@theshire.me>Bilbo Baggins</a>
* @version 0.0.0.0.1
*/
public class JavaDocExample {
   /** documentation attribut */
   private int i;
   /** ...  */
   public void f(String s,Timoleon t) {}
   /** documentation sur la méthode avec <em>tags html</em>
   * sur plusieurs lignes aussi
   * @param o description rôle paramètre o
   * @return description valeur de retour
   * @exception IllegalArgumentException description cas exception
   * @see #f(String,Timoleon)
   */
   public String someMethod(Order o) throws IllegalArgumentException {
      return(o.getId());
   }
} // JavaDocExample
```

Archives : `jar`

voir outil système `tar`

- Regrouper dans une archive les fichiers d'un projet (compressés). Faciliter la distribution.
- syntaxe et paramètres similaires au `tar`

```
jar ctxu[vfm0M] [nom-jar] [nom-manifest] [-C rép] fichiers ...
```

- **c** **c**réation
- **x** e**x**traction
- **t** afficher "**t**able"
- **u** mettre à jour (**u**pdate)
- **v** "**v**erbose" : bavard
- **f** spécifier le nom du **f**ichier d'archives
- **m** inclure le **m**anifeste
- **etc.**

```
jar cf archive.jar Class1.class Class2.class
jar cvf archive.jar fr gnu
jar xf archive.jar
```

`jar cvfm archive.jar mymanifest -C classes` *pack1*

ou

`jar cvfe archive.jar pack1.MyMain -C classes` *pack1*

- manifest : fichier dans `META-INF/MANIFEST.MF`
  - `jar` "exécutable" :
    `Main-Class:` `classname` (sans `.class`)
    puis `java -jar archive.jar`

Utilisation des classes contenues dans une archive sans extraction :

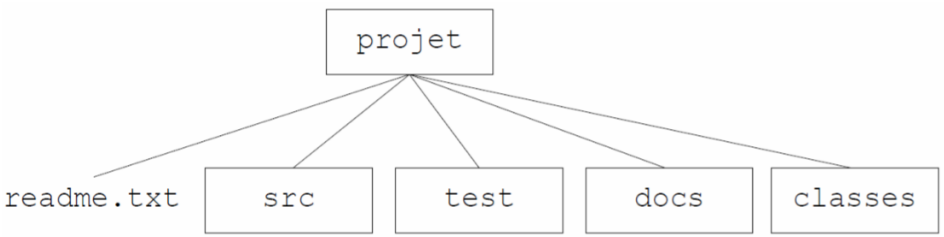
- mettre le fichier jar dans le `CLASSPATH`.

`export CLASSPATH=$CLASSPATH:/home/java/jars/`*paquetage*`.jar`

ou `java -classpath $CLASSPATH:/home/java/jars/`*paquetage*`.jar ...`

organisation les fichiers

Pour chaque projet, créer l'arborescence :



`projet` répertoire racine

`src` racine de l'arborescence des paquetages avec sources `.java`

`test` les tests qui valident le code

`docs` la javadoc générée

`classes` les `.class` générés

\+ ··· (bibliothèques, images, etc.)

```
.../projet> javac -sourcepath src -d classes src/package1/*.java
.../projet> javadoc -sourcepath src -d docs package1
.../projet> jar cvfm project.jar themanifest -C classes .
```