Gestion de version, paquetage

Arnaud Labourel arnaud.labourel@univ-amu.fr

29 mars 2022

Section 1

Gestion de version

Gestion de version

Principe

- Le code d'un projet est stocké dans un serveur.
- Les développeurs soumettent des modifications avec des commentaires à chaque fois.
- Le serveur conserve l'historique des mises à jour

Pourquoi la gestion de version ?

- Pour travailler de manière harmonieuse en équipe sans se marcher dessus
- Pour revenir en arrière en cas de problèmes (chaque modification correspond à une version du code sur laquelle il est possible de revenir)
- Possibilité de faire valider le code (via des tests) par le serveur et de rendre le déploiement automatique

Git

- Logiciel de gestion de version le plus populaire
- Serveur gratuit : github
- Version libre de logiciel serveur : gitlab
- Gestion de version décentralisée : la gestion de version se fait aussi en local

Utilisation de git

- Via l'IDE : VCS (Version Control Systems) dans le menu d'IntelliJ
- En ligne de commande : commande git

Git (source: Randall Munroe (xkcd))

Fonctionnement basique de git

- À la première utilisation, on crée une copie locale du dépôt git (`clone` ou `init`)
- À chaque commit :
  - on récupère la version courante du dépôt sur le serveur (`pull`)
  - On ajoute les fichiers à modifier (`add`)
  - On finalise le commit en donnant un message résumant les modifications (`commit`)
- Après un ou plusieurs commits, on met à jour la version distante avec nos modifications (`push`)

Exemples de commandes git

`git clone adresse_projet`

⇒ Clone un projet en local depuis un serveur

`git add nom_de_fichier`

⇒ Ajoute un fichier à la prochaine mise à jour.

`git commit -m"commentaire"`

⇒ Fait une mise à jour en local

`git pull`

⇒ Récupère les mises à jour du serveur en local

`git push`

⇒ Pousse les mises à jour locales sur le serveur

Gestion de version

- Importance d'avoir une gestion de version de son code
- Commandes de base : `clone`, `add`, `commit`, `push`, `pull`, ...

Git avancé (dans vos prochains cours)

- Notion de branche : on "fork" le dépôt pour créer une branche séparée sur laquelle on va travailler et on demande ensuite de l'intégrer dans la branche principale une fois les modifications effectuées.
- CI/CD (Intégration Continue/Déploiement (ou livraison) continue) : automatisation des tests et du déploiement (par exemple sur les serveurs qui utilise le code) ou de la livraison (publication du code compilé sur un serveur) à chaque push sur la branche principale.

Section 2

Structure d'un projet et paquetages

Structure d'un projet

En Java, un projet peut être découpé en paquetages (`package`).

Les paquetages permettent de :

- associer des classes afin de mieux organiser le code
- de créer des parties indépendantes réutilisables
- d'avoir plusieurs classes qui possèdent le même nom (du moment qu'elles ne sont pas dans le même paquetage)

Un paquetage (`package`) :

- est une collection de classes
- peut contenir des sous-paquetages

Lors de l'exécution...

Java utilise l'arborescence de fichier pour retrouver les fichiers `.class`

- Une classe (ou une interface) correspond à un fichier `.class`
- Un dossier correspond à un paquetage

Les `.class` du paquetage `com.univ_amu` doivent :

- être dans le sous-dossier com/univ_amu
- le dossier com doit être à la racine d'un des dossiers du `ClassPath`

Le `ClassPath` inclut :

- le répertoire courant
- les dossiers de la variable d'environnement `CLASSPATH`
- des archives JAR
- des dossiers précisés sur le ligne de commande de java (`-classpath path`)

Lors de la compilation... (1/2)

Le mot-clé `package` permet de préciser le paquetage des classes ou interfaces définies dans le fichier :

```
package com.univ_amu;
public class MyClass { /* ... */ }
```

Java utilise l'arborescence pour retrouver le code des classes ou interfaces :

- Une classe (ou une interface) `MyClass` est cherchée dans le fichier `MyClass.java`
- Le fichier `MyClass.java` est cherché dans le dossier associé au paquetage de `MyClass`

Lors de la compilation... (2/2)

```java
package com.univ_amu;
public class MyClass { /* ... */ }
```

Dans l'exemple précédent, il est donc conseillé que le fichier :

- se nomme `MyClass.java`
- se trouve dans le dossier `com/univ_amu` (Par défaut, la compilation crée `MyClass.class` dans `com/univ_amu`)

Utilisation d'une classe à partir d'un autre paquetage

Accessibilité :

- Seules les classes publiques sont utilisable à partir d'un autre paquetage
- Un fichier ne peut contenir qu'une seule classe publique

On peut désigner une classe qui se trouve dans un autre paquetage :

```
package com.my_project;
    public class Main {
    public static void main(String[] args) {
        com.univ_amu.MyClass myClass =
            new com.univ_amu.MyClass();
    }
}
```

Importer une classe

Vous pouvez également importer une classe :

```
package com.my_project;
import com.univ_amu.MyClass;

public class Main {
    public static void main(String[] args) {
        MyClass myClass = new MyClass();
    }
}
```

Deux classes de deux paquetages différents peuvent avoir le même nom :

- Exemple : `java.util.List` et `java.awt.List`
- Attention de choisir le bon import

Utiliser une classe d'un autre paquetage sans import

En fait, il est possible d'utiliser des classes d'un autre paquetage sans import.

Il suffit de mettre le chemin complet à chaque utilisation de la classe.

```
public class AppPackage {
  static public void main(String[] args){
    java.util.List<Integer> list1 =
      new java.util.ArrayList<>();
    java.awt.List list2 =
      new java.awt.List();
  }
}
```

⇒ On peut même utiliser dans ce cas deux classes ayant le même nom mais des paquetages différents.

Importer un paquetage

Vous pouvez également importer toutes les classes d'un paquetage :

```
package com.my_project;
import com.univ_amu.*;

public class Main {
    public static void main(String[] args) {
        MyClass myClass = new MyClass();
    }
}
```

Remarques :

- Les classes des sous-paquetages ne sont pas importées
- Le paquetage `java.lang` est importé par défaut

Importer des membres statiques

Depuis Java 5, il est possible d'importer directement des méthodes ou attributs de classes (`static`).

La syntaxe est la suivante :

```
import static my_package.my_class.myStaticMember;
```

Exemple :

```
import static java.lang.Math.PI;
import static java.lang.Math.pow;
```

Exemple sans import statique

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
    System.out.println(
      "A circle with a diameter of 5 cm has");
    System.out.println("a circumference of "
                       + (Math.PI * 5) + " cm");
    System.out.println("and an area of "
            + (Math.PI * Math.pow(2.5, 2))
            + " sq. cm");
  }
}
```

Exemple d'import statique

```
import static java.lang.Math.PI;
import static java.lang.Math.pow;
import static java.lang.System.out;

public class HelloWorld {
    public static void main(String[] args) {
        out.println("Hello World!");
        out.println(
            "A circle with a diameter of 5 cm has");
        out.println("a circumference of "
                + (PI * 5) + " cm");
        out.println("and an area of "
            + (PI * pow(2.5, 2)) + " sq. cm");
    }
}
```

Un exemple

Le fichier com/univ_amu/HelloWorld.java :

```
package com.univ_amu;
    public class HelloWorld {
    public static void main(String[] arg) {
        System.out.println("Hello world ! ");
    }
}
```

```
$ javac com/univ_amu/HelloWorld.java
$ ls com/univ_amu
HelloWorld.java HelloWorld.class
$ java com.univ_amu.HelloWorld
Hello world !
```

Quelques remarques

Nommage des paquetages :

- Les noms de paquetages sont écrits en minuscules
- Pour éviter les collisions, on utilise le nom du domaine à l'envers ⇒ `com.google.gson`, `com.oracle.jdbc`
- Si le nom n'est pas valide, on utilise des *underscores* : ⇒ `com.univ_amu`

Fichier JAR (Java Archive) :

- est une archive ZIP pour distribuer un ensemble de classes Java
- contient un *manifest* (qui peut préciser la classe qui contient le main)
- peut également faire partie du `ClassPath`
- peut être généré en ligne de commande (`jar`) ou avec un IDE