IN201 : 2 – Tests unitaires

Auteur : Christophe Garion <garion@isae.fr>
Public : SUPAERO 2A
Date :

Résumé

Le but de ce TP est de construire des tests unitaires avec JUnit et de commencer à utiliser Subversion.

1 Objectifs

Les objectifs de ce TP sont les suivants :
– utiliser une classe grâce à sa documentation `javadoc` ;
– utiliser des méthodes statiques ;
– utiliser des objets dans une classe ;
– écrire des tests unitaires avec JUnit ;
– utiliser le logiciel de gestion de version Subversion.

2 Préparation de l'utilisation de Subversion

À partir de cette séance, nous allons utiliser Subversion pour gérer le code source de vos TP en version. Chaque binôme possède son propre dépôt, qui n'est accessible que par lui-même. Ce dépôt est accessible depuis l'extérieur de l'ISAE.
L'URL de votre dépôt est `https://eduforge.isae.fr/svn/sup-2a-in201-GPE-bNUM` où :
– `GPE` est le nom de votre groupe de PC en minuscule
– `NUM` est votre numéro de binôme (votre PC(wo)man les possède)
Par exemple, l'URL du dépôt du binôme 9 du groupe S est `https://eduforge.isae.fr/svn/sup-2a-in201-s-b9`.
Pour connaître votre numéro de binôme, connectez-vous sur `https://eduforge.isae.fr`. Identifiez-vous et allez ensuite sur « Aller à un projet » (à droite). Vous verrez alors votre projet avec votre numéro de binôme. Si vous allez ensuite dans votre projet, en cliquant sur l'onglet « Dépôt », vous pourrez vérifier que les fichiers que vous ajoutez depuis votre copie locale sont bien ajoutés dans le dépôt.

Placez-vous dans le répertoire où vous mettrez tous vos TPs (créez-en un si besoin). Tapez ensuite la commande suivante

```
svn checkout URL .
```

où `URL` est l'adresse de votre dépôt (n'oubliez pas le « . » à la fin de la commande).
Votre répertoire est maintenant configuré pour interagir avec votre dépôt. Vous pouvez créer un répertoire `TP2` dedans et travailler comme d'habitude dans `TP2`.

Si vous souhaitez ajouter un fichier dans votre archive, il faut auparavant que le répertoire le contenant appartienne déjà à l'archive. Pour cela, il suffit d'ajouter (`svn add`) le répertoire.
Attention, le comportement par défaut de Subversion lors de l'ajout d'un répertoire est d'ajouter également le contenu du répertoire (y compris les sous-répertoires). Pour éviter cela, utilisez `svn add -N` (*non-recursive*) lorsque vous voulez ajouter un répertoire.

Remarque importante : n'oubliez pas d'ajouter vos fichiers sources et de *commiter* vos changements quand vous aurez fini votre TP !
Remarque importante (bis) : n'ajoutez que vos fichiers sources dans le dépôt (pas les *bytecodes* en particulier !).

Remarque très importante : maintenant que votre répertoire est géré en version, ne déplacez plus ou ne renommez plus des fichiers « à la main ». Subversion conserve des informations importantes dans un répertoire caché dans chacun de vos répertoires, si vous déplacez vos répertoires, ces informations ne seront plus à jour !

3 Présentation informelle

Un segment au sens mathématique peut être défini par ses deux extrémités. Comme toute figure mathématique du plan, on peut le translater. On peut également calculer sa longueur, car ses deux extrémités définissent ses « limites ». Comme

pour toute représentation d'un concept, il existe des moyens simples de le représenter sous forme textuelle, par exemple en utilisant ses deux extrémités.

4 Conception de la classe `Segment`

On cherche à concevoir et implanter une classe `Segment`. Pour cela, on dispose d'une classe `Point`, dont vous pouvez récupérer le bytecode sur le site sous la forme d'une archive (la documentation `javadoc` de la classe est bien sûr disponible).

1. quels sont les attributs modélisant l'état d'un objet de type `Segment` ? Quelle est leur visibilité ?
2. doit-on écrire un constructeur par défaut pour la classe `Segment` ? Si non, quels sont les paramètres que doit prendre ce constructeur ?
3. les services rendus par un objet de type `Segment` vont être modélisés par des opérations (ou méthodes) publiques. On souhaite pouvoir effectuer les opérations suivantes sur un segment :
   – le translater ;
   – l'afficher ;
   – renvoyer une chaîne de caractères le représentant ;
   – renvoyer sa longueur.
   Écrire le diagramme UML de la classe `Segment`.

5 Implantation de la classe `Segment`

5.1 Préparation

La classe `Point` vous est fournie sur le site sous la forme d'une archive JAR. Cette archive contient le *bytecode* de la classe `Point`. Pour pouvoir l'utiliser, il faudra utiliser l'option `-cp` de `javac` et `java`.
Par exemple, en utilisant `-cp .:~/TP_JAVA/TP2/classes/:~/TP_JAVA/TP2/TP2.jar`, on précisera à `javac` ou `java` qu'ils peuvent trouver les classes compilées dont ils ont besoin :
– soit dans le répertoire courant « `.` »
– soit dans le répertoire `TP_JAVA/TP2/classes/` contenue dans votre répertoire personnel
– soit dans l'archive `TP_JAVA/TP2/TP2.jar` contenue dans votre répertoire personnel
Une autre solution (à préférer) est de positionner la variable d'environnement `CLASSPATH`[1] dans chaque terminal de la façon suivante :

```
export CLASSPATH=$CLASSPATH:/cheminVersJAR/TP2.jar
```

Attention, l'utilisation de l'option `-cp` est prioritaire par rapport au contenu de la variable `CLASSPATH`. Il faut donc, si vous souhaitez utiliser à la fois `CLASSPATH` et `-cp` ajouter la valeur de `CLASSPATH` dans `-cp` :

```
-cp .:../classes:$CLASSPATH
```

Nous vous conseillons d'ouvrir deux terminaux :
– un pour compiler les sources ;
– un pour exécuter les tests unitaires de JUnit.
Dans chaque terminal, on positionnera le `CLASSPATH` comme présenté ci-dessus. On pourra séparer les sources des classes « métier » (`Segment` en fait) des sources des classes de tests via deux répertoires séparés (`src` et `tests` par exemple).

Une fois que vous aurez implanté le squelette de votre classe `Segment`, vérifiez que vous pouvez correctement la compiler. Quand c'est le cas, copiez les commandes `export CLASSPATH=...` et `javac -cp ...` dans un fichier texte pour pouvoir les réutiliser par la suite.

1. `CLASSPATH` est une variable d'environnement permettant aux utilitaires `javac` et `java` de savoir où chercher les fichiers *bytecode* dont ils ont besoin.

5.2 Travail d'implantation

Implanter et documenter maintenant la classe `Segment` en Java en utilisant la classe `Point` fournie. Il faudra bien sûr faire appel le plus possible aux méthodes disponibles dans `Point`.
En parallèle du développement de la classe `Segment`, développer **au fur et à mesure** une classe `SegmentTest` permettant de tester la classe `Segment` en utilisant JUnit (vous disposez d'un squelette de classe sous l'onglet « Ressources » du site).
La chaîne de caractères représentant un segment sera de la forme `[extremite1;extremite2]`.
Si vous développez une application utilisant `Segment` (autre qu'une classe de test JUnit), vous pourrez utiliser la classe `Console` disponible sur le site sous l'onglet « Ressources » pour utiliser des entrées clavier. Si vous souhaitez l'utiliser, ajoutez la ligne `import console.*;` au début du fichier Java concerné.