IN201 : 1 – Classes et objets

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

Résumé

Le but de ce TP est de manipuler les notions de classe et d'objet au travers de la classe `Equation` vue en cours et d'un exemple simple, la classe `Point`.

1 Objectifs

Les objectifs de ce TP sont les suivants :
– manipuler les outils du JDK avec la classe `Equation` ;
– à partir de la conception d'une classe `Point`, implanter cette classe ;
– comprendre les attributs et les méthodes ;
– comprendre l'utilisation du constructeur ;
– documenter une classe.

2 Conseils pratiques (à respecter !)

2.1 Organisation des répertoires

Pour chaque TP, vous créerez un répertoire TP*n* où *n* est le numéro du TP. À l'intérieur de ce répertoire, vous créerez un répertoire `src` qui contiendra vos sources Java, un répertoire `classes` qui contiendra les *bytecodes* de vos classes et un répertoire `doc` qui contiendra la documentation `Javadoc` de vos classes.
Rappel : pour créer un répertoire sous UNIX, la commande est `mkdir nomRepertoire`.

2.2 Édition des sources

Vous utiliserez pour éditer vos sources votre éditeur de texte habituel. Il est préférable d'un choisir un qui puisse indenter automatiquement votre code et le colorier syntaxiquement. Si vous utilisez Emacs [1], vous pouvez utiliser JDEE [2] pour avoir un environnement de développement complet. Attention, nous conseillons l'utilisation d'Emacs pour les personnes sachant déjà s'en servir. On peut également utiliser gedit (commande `gedit` dans un shell) pour avoir un éditeur de texte minimal supportant la coloration syntaxique.
Vous avez à votre disposition sur le site des conventions de code. Essayez de les respecter. En particulier :
– les noms de classes commencent par une majuscule ;
– les noms d'attributs et de méthodes commencent par une minuscule ;
– les noms de variables commencent par une minuscule ;
– en cas de nom composé, on ne sépare pas les mots, mais on commence chaque mot par une majuscule (ex : `methodeCalcul`) ;
– on place un espace avant et après un opérande.

2.3 Compilation et exécution

Les deux opérations principales que vous effectuerez sont la compilation des sources et l'« exécution » de vos programmes. Il est donc préférable d'avoir deux terminaux ouverts, un pour la compilation, un autre pour l'exécution.
Le terminal servant à la compilation sera positionné dans le répertoire `src`. Pour compiler une classe Java, on utilisera la commande suivante :

```
javac -d ../classes -cp .:../classes Classe.java
```

L'option `-d ../classes` signale au compilateur de placer les fichiers *bytecode* dans le répertoire `classes` que vous aurez créé précédemment.
L'option `-cp .:../classes` signale au compilateur que les fichiers *bytecode* dont il peut avoir besoin sont soit dans le répertoire courant, soit dans le répertoire `classes` précédemment créé (nous reparlerons de cette notion la prochaine séance).
Le terminal servant à l'exécution sera positionné dans le répertoire `classes`. La commande d'interprétation d'une classe sera :

```
java Classe
```

où `Classe` est le nom de la classe à interpréter.

2.4 Génération de la documentation

Pour générer la documentation de vos classes via l'utilitaire `Javadoc`, il suffit de vous placer dans le répertoire `src` et d'utiliser la commande :

```
javadoc -d ../doc -version -author -private *.java
```

L'option `-d ../doc` signale à `Javadoc` de placer la documentation dans le répertoire `doc`. L'option **`-private`** permet de spécifier que l'on veut également générer la documentation des composantes privées de la classe. Bien évidemment, on ne générera plus de la documentation javadoc pour les composantes privées d'une classe dans la suite du cours, car cette documentation est à usage externe.

2.5 Envoi de votre BE à votre PC(wo)man

Vous enverrez à chaque séance votre TP à votre PC(wo)man. Cela lui permettra de vous faire remarquer rapidement si vous faites des erreurs importantes.
Il vous faut dans un premier temps archiver votre répertoire. Nous utiliserons pour cela le format JAR qui est le format des archives utilisées pour Java. Pour le TP1, placez-vous dans le répertoire TP1, puis exécutez la commande suivante :

```
jar cvf TP1_NomBinome1_NomBinome2.jar src/*
```

Vous enverrez alors le fichier `TP1_NomBinome1_NomBinome2.jar` à votre PC(wo)man en précisant dans comme sujet de mail « [IN201] TP1 ». Ceci nous permet de récupérer facilement vos TPs. Si vous ne respectez pas cette procédure, vos mails risquent de se perdre parmi les mails quotidiens de votre PC(wo)man.

Pour vérifier que votre archive contient bien les fichiers sources `.java`, vous pouvez exécuter la commande suivante :

```
jar tvf TP1_NomBinome1_NomBinome2.jar
```

La liste des fichiers contenus dans l'archive devrait apparaître à l'écran.

2.6 Documentation Java disponible

La documentation `Javadoc` des API des classes Java est disponible aux URL suivantes :
– localement sur [3] ;
– de façon générale sur [4] (choisir la version 6.0).

3 Manipulation de la classe `Equation`

Le site `http://www.tofgarion.net/lectures/IN201` servira tout au long du cours. Sur ce site vous trouverez les sujets de chaque séance et les corrigés des TPs.
Récupérer sur le site l'archive `.jar` contenant les fichiers source `Equation.java` et `ResolutionEquation.java`. Décompresser l'archive via la commande :

```
jar xvf TP1.jar
```

et placer les sources dans le répertoire `src` précédemment créé.
– compiler les deux classes en utilisant le compilateur `javac` ;
– générer la documentation HTML en utilisant `javadoc` (utiliser l'option **`-private`**) ;
– « exécuter » la classe `ResolutionEquation` en utilisant la machine virtuelle `java`.

4 Conception et implantation d'une classe `Point`

On cherche à concevoir une classe `Point` qui représente un point dans le plan (au sens géométrique). On souhaite pouvoir réaliser le « programme » suivant :
– création d'un point p_1 de coordonnées cartésiennes $(1, 0)$;
– affichage des coordonnées du point p_1 ;
– translation du point p_1 en utilisant le vecteur $(5, 0)$. On pourra également translater le point en utilisant d'autres vecteurs ;
– affichage des coordonnées du point p_1 ;
– création d'un point p_2 de coordonnées cartésiennes $(-1, -5.5)$;
– calcul et affichage de la distance entre p_1 et p_2.

1. comment représenter l'état d'un point ? En déduire les attributs de la classe `Point` ;
2. de quoi a-t-on besoin pour construire un point ? Quelle est la signature du ou des constructeurs de la classe ?
3. les opérations (ou les méthodes) de la classe représentent les services qui peuvent être rendus par un objet de type `Point`. Utiliser le « programme » fourni pour déterminer les opérations de la classe `Point`. On se placera dans le cadre d'une conception d'une classe `Point` générique (pensez au méthodes : faut-il des paramètres, par exemple pour translater ?) ;
4. quelle est la visibilité des attributs et des méthodes ?
5. écrire le diagramme UML de la classe `Point` ;
6. implanter la classe `Point` en Java. On « traduira » dans un premier temps le diagramme UML réalisé, puis on documentera la classe, ses attributs et ses méthodes et enfin, on écrira le code des méthodes. On testera les méthodes **au fur et à mesure de leur écriture**.
   Comme vous aurez à utiliser des méthodes mathématiques, comme par exemple le calcul d'une racine carrée ou l'élévation d'un nombre à une puissance, **vous consulterez la documentation javadoc de la classe `Math`** (cf. adresse de la documentation javadoc de l'API Java sur le site). Attention, les méthodes de la classe sont *statiques*, on les appelle donc en utilisant le nom de la classe `Math` et pas un objet instance de `Math`.
7. écrire une classe `TestPoint` qui implante le programme décrit de façon informelle plus haut ;
8. (facultatif) : ajouter les opérations de comparaisons
   L'objectif de cette question est de définir des méthodes de comparaison sur les points.
   1. écrire une méthode `lt` qui indique si un point est strictement inférieur à un autre. Pour comparer deux points, on commencera par comparer les abscisses des points puis leurs ordonnées.
   2. **en utilisant** la méthode `lt` coder les méthodes `ge` (supérieur ou égal), `gt` (supérieur strict), `le` (inférieur ou égal).

Références

[1] *GNU Emacs*. http://www.gnu.org/software/emacs/emacs.html. URL : http://www.gnu.org/software/emacs/emacs.html.

[2] *JDEE : Java Development Environment for Emacs*. http://jdee.sunsite.dk/. URL : http://jdee.sunsite.dk/.

[3] *Java API specifications - local SUPAERO*. file:///opt/javadoc/api.

[4] *Java API specifications*. http://download.oracle.com/javase/6/docs/api/index.html. URL : http://download.oracle.com/javase/6/docs/api/index.html.