Développement informatique avancé orienté application - TP

Etoffons nos classes!

Virginie Van den Schrieck

Ce TP se base sur les classes réalisées pendant le TP1. Si vous n'avez pas fini ce dernier, mettez-vous en ordre au plus vite, et soyez sûrs de rattraper votre retard avant la prochaine séance !

1 Constructeurs

Devoir passer par une méthode main pour initialiser les variables d'instance d'un objet avec des valeurs arbitraires n'est pas une bonne solution. A la place, on utilise des méthodes spéciales : les constructeurs, qui servent à la création des objets.

Un constructeur est une méthode portant le nom de la classe, et pouvant prendre différents types de paramètres. Son exécution résulte en la création d'un objet dont l'état a été initialisé en fonction des paramètres transmis. Un constructeur n'a pas de valeur de retour, et il est appelé via le mot-clé `new`. Lors de la création d'un objet, les étapes effectuées par la JVM sont :

- La JVM réserve de la mémoire pour le nouvel objet
- La zone mémoire est nettoyée de tout ce qu'elle contenait (champs du nouvel objet mis à 0, null ou false)
- Le constructeur invoqué via le mot-clé new est exécuté

Jusqu'ici, nous avons créé des objets en utilisant le mot-clé new, mais sans définir de constructeur. Cela était possible parce que lorsqu'aucun constructeur n'est défini dans une classe, la JVM utilise le constructeur par défaut : Il s'agit d'un constructeur sans paramètre, qui n'exécute aucune instruction autre que la création de l'objet en mémoire. Ainsi, dans notre classe Calculatrice créée plus tôt, Java considère implicitement le constructeur suivant :

```java
/**
 * Constructeur vide
 */
public Calculatrice(){

}
```

Listing 1 – Constructeur par défaut

Si nous souhaitons pouvoir créer une calculatrice avec une valeur spécifique, nous pouvons ajouter un autre constructeur avec un paramètre.

```java
/**
 * Constructeur initialisant la calculatrice
 * avec la valeur indiquée.
 * @param value : valeur initiale pour la calculatrice
 */
public Calculatrice(double value){
    this.valeurCourante = value;
}
```

Listing 2 – Constructeur avec paramètre

Lorsqu'il existe au moins un constructeur explicite, Java ne crée plus de constructeur par défaut. Le constructeur sans paramètre n'existe alors que s'il est explicitement défini. Ce constructeur sans paramètre peut être utilisé pour initialiser les variables d'instance dans un constructeur plutôt que de le faire lors de la déclaration de la variable. Ainsi, le constructeur sans paramètre pourrait être défini comme ceci :

```java
/**
 * Constructeur initialisant la calculatrice avec la valeur 0.
 */
public Calculatrice(){
    this.valeurCourante = 0;
}
```

Listing 3 – Constructeur sans paramètre initilisant les variables d'instance

Nous vous conseillons de toujours initialiser vos variables d'instance dans les constructeurs plutôt que dans les déclarations.

La figure 1 montre le diagramme UML de la classe Calculatrice avec ses constructeurs.

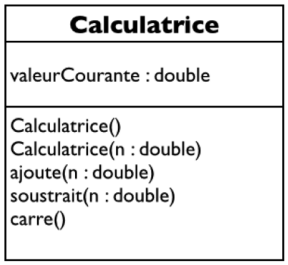

FIGURE 1 – Diagramme UML de la classe Calculatrice avec ses constructeurs

1.1 Exercice

Pour toutes les classes du TP1, ajoutez les constructeurs ad-hoc, complétez et regénérez la javadoc et mettez à jour les diagrammes UML.

2 Accesseurs et mutateurs

Comme nous l'avons vu dans les exemples précédentes, pour accéder aux valeurs des attributs d'un objet, nous avons jusqu'ici deux possibilités :

- A l'intérieur des méthodes de la classe, nous pouvons utiliser directement le nom de l'attribut (ex : valeurCourante dans la classe Calculatrice)
- A l'extérieur des méthodes de la classe, il nous faut partir d'une référence vers un objet de la classe et utiliser l'opérateur `.` (ex : `maCalc.valeurCourante`). Notez qu'il est également possible d'utiliser l'opérateur `.` à l'intérieur de la classe, en l'appliquant au mot-clé `this`, qui est en fait une référence vers l'objet courant.

Néanmoins, il n'est pas conseillé d'utiliser cet opérateur pour accéder directement aux valeurs des variables d'instance des objets. Imaginer par exemple une classe CompteBanquaire. Avec ce mode d'accès aux variables d'instance, n'importe qui peut modifier l'état interne des objets CompteBanquaire, à savoir typiquement le solde. Il nous faut donc un moyen de contrôler l'accès aux données. C'est le principe de l'encapsulation, qui est mis en place à travers deux mécanismes :

1. La visibilité des variables d'instance : Il est possible de contrôler l'accès aux variables d'instance via les mot-clés `public`, `private` et `protected`. Ces modificateurs de visibilités peuvent également être appliqués aux classes et aux méthodes.
2. L'ajout de méthodes spéciales, appelées accesseurs et mutateurs (ou plus familièrement, getters et setters). Ces méthodes commencent par convention par `get` et `set`. Les Getters servent à renvoyer les valeurs des variables d'instance (ex : `getValeurCourante()` dans le cas de

notre calculatrice), après un éventuel contrôle d'accès. Les Setters servent à modifier les variables d'instance. Avant la modification, ils appliqueront une vérification pour s'assurer que le changement ne nuit pas à l'intégrité de l'objet. Par exemple, dans la classe CompteBanquaire, la méthode `setSolde()` peut n'autoriser de changement de solde qu'à la condition que le solde ne passe pas en négatif.

Ces changements se reflètent aussi dans les diagrammes UML. Les champs et méthodes `private` seront précédés par un −, les public par un +, et les protected par un #. Le diagramme de la classe `Calculatrice` devient alors :

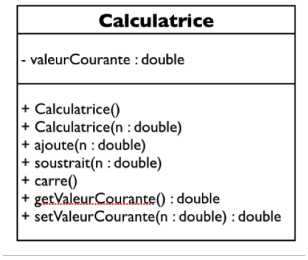

FIGURE 2 – Diagramme UML de la classe Calculatrice avec ses constructeurs

Encore une fois, l'IDE Eclipse nous facilite le travail, grâce à sa fonctionnalité de génération automatique des Getters et Setters. Vous pouvez soit choisir de les générer lors de la création de la classe, soit les ajouter en cours de route. Pour ça, faites un clic droit sur le nom de la classe, sélectionnez *Source*, puis *Generate Getters and Setters*. Vous pouvez indiquer les accesseurs à générer.

Dans un premier temps, nous vous demandons de systématiquement mettre vos variables en visibilité private , et de générer les getters et les setters adéquats.

2.1 Exercices

2.1.1 Mise en oeuvre de l'encapsulation dans Calculatrice

Reprenez la classe Calculatrice. Vérifiez que la variable d'instance valeurCourante est bien en visibilité public. Vous allez à présent créer une instance de cette classe et modifier la valeur de la variable d'instance valeurCourante depuis deux endroits différents. Vérifiez à chaque fois que la modification est bien effective, à l'aide par exemple d'une impression à la console.

— Depuis la méthode main de la classe `Calculatrice`
— Depuis la méthode main d'une nouvelle classe que vous appellerez `UtilisationCalculatrice`

Changez ensuite la visibilité de `valeurCourante` en `private`. Exécutez à nouveau les deux classes pour vérifier si elles sont toujours capables d'effectuer la modification de la variable d'instance. Que constatez-vous ?

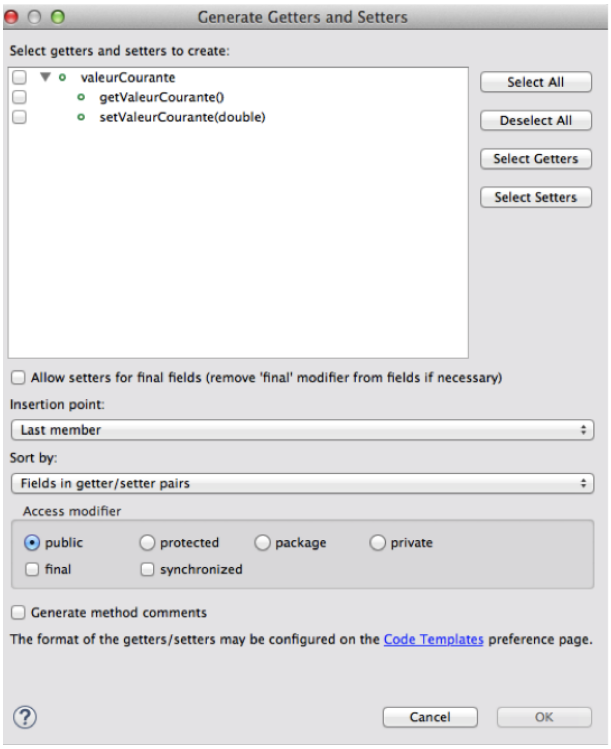

FIGURE 3 – Génération automatique des constructeurs dans Eclipse

Ajoutez enfin des Getters et des Setters à la classe `Calculatrice`. Modifiez la classe `UtilisationCalculatrice` pour qu'elle utilise le setter à la place de l'accès direct à la variable d'instance. Que constatez-vous ?

2.1.2 Mise à jour des autres classes

Reprenez à présent les classes que vous avez écrites dans le cadre du TP1. Pour chacune, demandez à Eclipse de générer automatiquement les Getters et les Setters. Vérifiez dans chaque cas s'il ne faut pas ajouter du code au Setter pour vérifier que la modification demandée est conforme (ex : valeurs ne pouvant être négatives, contraintes sur les dates de naissance, etc.).

3 Méthode toString()

Lors du premier TP et dans le cadre du cours théorique, nous avons vu qu'il est possible d'imprimer des informations à la console. Cela permet notamment d'aller consulter le contenu de certaines variables.

Dans le cas d'objet, on souhaite souvent imprimer un résumé de son état (variables d'instance). Plutôt que d'imprimer autant de `System.out.println()`

que de variables d'instance, il existe un mécanisme interne à Java pour générer une représentation textuelle de l'objet. Il s'agit de la méthode `toString()`.

Cette méthode ne prend pas de paramètres, mais renvoie un String contenant cette fameuse représentation textuelle. Par exemple, dans le cas de la calculatrice, la méthode `toString()` pourrait être :

```
/**
 * @return une représentation textuelle
 * de la valeur actuelle de la calculatrice
 */
public String toString(){
    return "Valeur affichée : " + valeurCourante
}
```

Listing 4 – Méthode toString() de Calculatrice

En soi, cette méthode n'a rien d'extraordinaire : Elle accède juste à la variable d'instance de l'objet pour en faire un String. Elle devient par contre intéressante dans toutes les situations où la JVM doit transformer un objet en chaîne de caractère (transtypage vers String).

```
public static void main(String [] args){
    Calculatrice maCalc = new Calculatrice();
    maCalc.ajoute(10);
    System.out.println(maCalc);
}
```

Listing 5 – Exploitation de toString() de Calculatrice

Dans le code ci-dessus, la méthode `System.out.println` attend un String en paramètre. Puisqu'elle reçoit un objet, elle va automatiquement appeler sa méthode toString() pour le convertir en String.

3.1 Exercices

3.1.1 Tester toString()

Reprenez votre classe Calculatrice telle que vous l'avez laissée à l'exercice suivant. Dans la méthode `main`, créez un objet Calculatrice appelé `myCalc`, puis imprimez-le à la console avec l'instruction `System.out.println(myCalc)`. Que constatez-vous ?

Ajoutez ensuite la méthode `toString`, et réexécutez la méthode `main`. Que constatez-vous à présent ?

3.1.2 Mise à jour de vos classes

Pour chacune des classes implémentées jusqu'ici, ajoutez la méthode `toString()`. Notez qu'il est possible d'en générer automatiquement le squelette via Eclipse. Notez ici le chemin d'accès à cette commande :

4 Checkstyle

Lorsqu'on programme, il est de bonne pratique de respecter une convention de codage (voir `http://fr.wikipedia.org/wiki/RÃÍgles_de_codage`). Une convention de codage défini la manière dont on met le code en page et dont on choisit les noms de variable. Cela comprend notamment :

- La longueur des lignes de code (à l'origine, on recommandait 80 caractères à cause de la taille limitée des écrants. Ce n'est plus le cas actuellement.)
- La manière d'écrire les noms de variable : ma_variable, Ma_Variable, MaVariable (CamelCase), maVariable, MA_VARIABLE, etc.
- Les accolades pour délimiter les blocs de code : Accolade ouvrante en début de ligne ou en fin de ligne précédente
- D'éventuels espaces entre les opérandes et les opérateurs (int i = 0 ou int i=0)
- Des lignes vides entre les méthodes
- La manière d'organiser les commentaires (en ligne ou multi-lignes selon les circonstances)
- Les en-têtes des fichiers (noms d'auteurs, dates, version, licence, ...)
- L'utilisation des tabulations ou des espaces pour l'indentation
- ...

Pour ce qui est du Java, vous trouverez les conventions de codage recommandées par Sun/Oracle à l'adresse suivante : `http://www.oracle.com/technetwork/java/codeconvtoc-136057.html`.

Une version francophone de ce document est disponible ici : `http://anubis.polytech.unice.fr/cours/_media/2010_2011:cip2:algoprog:javaconventions.pdf`

Dans le cadre du cours de Java, nous vous demandons de respecter une convention de codage. Pour valider ce que vous avez écrit, vous allez utiliser l'outil Checkstyle, qui existe sous forme de plugin Eclipse (qu'il vous faudra sans doute installer via ce lien : `http://eclipse-cs.sourceforge.net/#!/install`). Allez dans les préférences d'Eclipse, et sélectionnez la rubrique Checkstyle. Vous allez importer la configuration convention_tp_java, disponible sur le Campus Virtuel. Cette convention est une version un peu simplifiée des règles proposées par Oracle.

Pour l'importation, dans les préférences de configuration Checkstyle, sélectionnez new, puis indiquez comme type de configuration `External configuration file`. Grâce au bouton `Browse`, sélectionnez le fichier depuis le système de fichiers. Donnez-lui un nom, puis appuyez sur OK. Il devrait apparaître dans la liste des configurations. Sélectionnez-le, et marquez-le comme configuration par défaut. Les curieux peuvent examiner les règles de ce fichier via le bouton `Configure`.

Pour appliquez Checkstyle sur votre code, il faut l'activer sur votre projet. Sélectionnez-donc ce dernier, faites un clic droit puis choisissez Checkstyle/Activate Checkstyle.

Pour vérifier votre code, faites un clic droit sur un fichier, un package ou un projet, et choisissez `Check Code with Checkstyle`. Checkstyle vous génère une série de Warning, à vous de modifier votre code pour qu'il n'en reste aucun. Cela prendra un peu de temps au début, le temps de vous approprier les conventions de codage, mais c'est un investissement pour apprendre à rédiger du code de qualité. N'hésitez pas à demander de l'aide au professeur si vous ne comprenez pas un Warning.

5 Bilan de fin de TP

A la fin de ce TP, vérifiez si chacune des classes que vous avez écrites possède :

- Des variables d'instance `private`,
- Un constructeur par défaut et un ou des constructeurs avec paramètres
- Les getters et setters
- Une méthode `toString()`
- Des éventuelles autres méthodes (cfr TP1)
- La Javadoc correctement formatée avec du **contenu pertinent**, et la documentation HTML générée
- Un diagramme UML complet et à jour

Vérifiez également que Checkstyle ne relève pas d'erreur de style dans votre code source.

6 Quelques classses en bonus...

Pour ceux qui avancent rapidement dans les TPs, voici quelques exercices supplémentaires. Pour chaque situation décrite ci-dessous, dessinez le diagramme UML puis implémentez la ou les classes Java correspondantes conformément aux éléments mentionnés dans la check-list du paragraphe précédent.

6.1 Fractions

Une fraction est représentée par un numérateur et un dénominateur. Le dénominateur ne peut pas être nul.

En plus de la représentation textuelle canonique avec la barre de fraction, la classe Fraction doit fournir une méthode getValeurReelle(), qui renvoie le double correspondant à la valeur numérique de la fraction.

Petit challenge pour les plus motivés : Essayez de faire en sorte que la représentation textuelle de la fraction renvoie la fraction simplifiée (ex : 1/2 au lieu de 2/4).

6.2 Ecole

Créez une classe Etudiant, qui, pour chaque étudiant, donne accès à son nom, son prénom et son matricule. Créez une classe Professeur, qui donne accès au nom, au prénom et à la spécialité d'un professeur. Créez une classe GroupeClasse, qui possède un professeur titulaire et un tableau de 10 étudiants (pour vos tests, utilisez des noms comme etu1, etu2 etc., ce qui vous permettra d'exploiter les boucles pour remplir le tableau).