POO – Programmation Orientée Objet en Java

Fiche de TP numéro 1

La documentation

Il y a deux sources indispensables de documentation sur Java : la documentation sur le JDK incluant un manuel de référence complet sur les classes disponibles en standard avec le JDK, et le *tutorial*, qui permet d'apprendre Java par l'exemple. Cette documentation est disponible au format HTML. Une dernière source importante est la documentation des outils (`java`, `javac`, `javadoc`, ...). L'ensemble de cette documentation est accessible à partir de `https://docs.oracle.com/en/java/javase/11/`.

- Ouvrez un navigateur HTML, et accédez à la documentation en ligne de Java : `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/module-summary.html`
- Le tutorial est disponible à l'adresse : `https://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html` Vous pourrez commencer par la première leçon de *Learning the Java Language* !
- La documentation sur les outils disponibles en standard avec le kit de développement Java (Java Development Kit, JDK) : `https://docs.oracle.com/en/java/javase/11/tools/tools-and-command-reference.html`.

Ces adresses sont données sur l'espace du cours dans Moodle. Vous pouvez d'ores et déjà ouvrir ces trois pages dans votre navigateur.

Exercice 1 : Écrire, compiler et exécuter un programme Java

Créez chez vous un répertoire `java`, puis, dans ce répertoire, un nouveau répertoire `tp1`, et dans `tp1` trois nouveaux répertoires `doc`, `src`et `bin`. Pour ce faire, vous pouvez soit utiliser un gestionnaire de fichiers, soit ouvrir une console, et tapez les commandes suivantes (utilisez la touche `Tab` pour bénéficier de la complétion automatique) :

```
$ mkdir java
$ ls
$ mkdir java/tp1
$ cd java/tp1
$ mkdir doc
$ mkdir bin
$ mkdir src
$ ls
```

Vous allez utiliser l'éditeur `visual studio code`.

Saisissez le code Java suivant dans un fichier nommé `Premiere.java`, dans le répertoire `java/tp1/src`. Il définit une classe `Premiere` qui affiche la chaîne de caractères *"Bonjour le monde"* sur la console. N'oubliez pas les commentaires.

```
/**
 * Ma premiere classe en Java
 *
```

```
 * @author Nom Prenom
 * @version 1.0
 */
public class Premiere {

    public static void main(String [] args) {
        System.out.println("Bonjour le monde!");
    }
}
```

Pour compiler cette classe, on utilise le compilateur Java fournit en standard dans le JSDK de Sun : `javac` (il en existe d'autres : `guavac, jikes`). Dans une console, tapez
`javac -d bin -cp bin:src src/Premiere.java`

Un fichier `Premiere.class` est créé dans répertoire `bin`. Il contient la classe compilée, que vous allez maintenant pouvoir interpréter. Il suffit d'appeler la machine virtuelle Java suivie du nom de la classe contenant la méthode `main` **sans l'extension .class** :
`java -cp bin Premiere`

Exercice 2 : Générer la documentation Parce que la documentation est importante, au moins aussi importante que le code, le JSDK fournit en standard un générateur de documentation HTML à partir du code source de la classe. Cet outil utilise pour cela les commentaires commençant par `/**`. Il recherche ensuite des balises particulières : `@author`, `@version`, `@param`, `@return`, `@exception`, etc.

Pour plus d'informations, consultez la documentation de Javadoc avec un navigateur HTML (vous avez l'adresse dans vos signets !).

On change un peu la classe `Premiere` : on lui ajoute un attribut privé `message` et une méthode `affiche`. La méthode `main` créé maintenant un objet `Premiere` avant d'appeler sa méthode `affiche`.

```
/**
 * Ma première classe en Java
 *
 * @author Nom Prenom
 * @version 1.0
 */
public class Premiere {

    private String message = "Bonjour le monde!";

    /**
     * On utilise le constructeur par defaut
     */

    /**
     * Cette methode affiche le contenu de l'attribut privé
     * message à l'ecran.
     */
    public void affiche() {
        System.out.println(message);
    }
```

```
    /** methode principale de mon premier programme. */
    public static void main(String [] args) {
        // Cree un objet de type Premiere
        Premiere prem = new Premiere();
        prem.affiche();
    }
}
```

Recompilez `Premiere`, réinterprétez-la. Rien n'a changé.

Lancez le générateur de documentation sur votre premier programme java :

```
javadoc -charset utf8 -d doc Premiere.java
```

Pour visualiser la documentation, il suffit alors d'ouvrir le fichier `doc/index.html` dans votre navigateur HTML.

Exercice 3 : Enrichir cette première classe

On désire maintenant enrichir la classe `Premiere`.

— déclarez un constructeur qui prend en paramètre une chaîne de caractères qui sera utilisée pour initialiser l'attribut message.

— déclarez un accesseur `getMessage()` et un modificateur `setMessage()` pour l'attribut `message`.

— modifiez la méthode `main` pour prendre en compte ces changements.

Vous ferez attention à bien documenter chaque nouvelle méthode.

Exercice 4 : Organisation du code

On utilisera la convention suivante pour les tous TPs :

```
TPx // repertoire pour le TP en cours (1 <= x <=12)
 |
 ---- src // fichiers source Java (.java)
 |
 ---- bin // fichiers compiles (.class)
 |
 ---- doc   // documentation (.html)
```

Pour cela vous avez utilisé l'option `-d` (destination) de `javac` et de `javadoc`, et l'option `cp` (classpath) de `java` qui indique

— pour le compilateur `javac`, où trouver soit les sources des classes à compiler, soit les `.class` (classes déjà compilées)

— pour l'interprète `java`, où trouver les `.class`.

```
$ cd TPx
$ javac -d bin -cp bin:src src/Premiere.java
$ java -cp bin Premiere
$ javadoc -charset utf8 -d doc src/Premiere.java
```

L'ajout de `-charset utf8` demande à javadoc de préciser dans le `.html` que les fichiers générés sont encodés en `utf8`.

Cette organisation est importante car les applications Java sont toujours constituées d'un grand nombre de fichiers, que de ce soit des sources (`.java`), des fichiers bytecode (`.class`), ou de la documentation.

Exercice 5 : La classe Etudiant Programmez la classe `Eutdiant` vue en cours. Écrivez dans un fichier `src/Etudiant.java` :

```
/** classe qui modélise la gestion d'un étudiant dans une université.
*/
public class Etudiant {
    public final static int MAXNOTES = 10;
    public final static int MINVALUE = 0;
    public final static int MAXVALUE = 20;

    private String nom;
    private String prenom;

    //private Formation formation;

    private float[] lesNotes = new float[MAXNOTES];
    private int nbNotes; // nombre actuel de notes de l'étudiant

    /** Constructeur d'un étudiant.
    @param nom : le nom de l'étudiant
    @param prenom : le prénom de l'étudiant
    */
    public Etudiant(String nom, String prenom) {
        this.nom = nom;
        this.prenom = prenom;
    }

    /** accesseur pour le nom de l'étudiant
    @return le nom de l'étudiant */
    public String getNom(){
        return nom;
    }

    /** accesseur pour le prénom de l'étudiant
    @return le prénom de l'étudiant */
    public String getPrenom(){
        return prenom;
    }

    /** retourne une présentation textuelle d'un étudiant :
    son nom, son prénom, son nombre de notes */
    public String toString(){
        return prenom+" "+nom+", "+nbNotes+" note(s).";
    }
```

```
    /** ajoute une note à un étudiant dans la limite de la capacité de l'étudiant
    @param note : la note qu'on veut ajouter à l'étudiant
    @return vrai si l'ajout a pu être fait */
    public boolean ajouteNote(float note){
        if (nbNotes < lesNotes.length){
            lesNotes[nbNotes] = note;
            nbNotes++;
            return true;
        }
        return false;
    }
}
```

Écrivez ensuite la classe `TestEtudiant` pour tester ce que vous venez d'écrire (dans un autre fichier `src/TestEtudiant.java`) :

```
/** Classe qui permet de lancer une application qui teste la classe Etudiant.
 */
public class TestEtudiant {

    public static void main(String[] args){
        Etudiant e1 = new Etudiant("Dupont","Jean");
        Etudiant e2 = new Etudiant("Durand","Paul");
        System.out.println(e1);
        System.out.println(e2);
        e1.ajouteNote(15);
        e2.ajouteNote(17);
        e1.ajouteNote(13);
        e2.ajouteNote(8);
        System.out.println(e1);
        System.out.println(e2);
    }
}
```

N'oubliez pas les commentaires. Compilez, interprétez, générez la documentation.

Exercice 6 : Les étudiants

On modifie un peu le cahier des charges pour la gestion des étudiants. On s'occupe maintenant de la gestion des notes d'une promotion d'étudiants. Dans cette promotion, on conserve pour chaque étudiant son nom, son prénom, les notes qu'il a obtenues, et s'il a été déclaré admis à son diplome. Pour obtenir le diplome, il faut avoir 5 notes, et une moyenne de 10 sur ces 5 notes. Lorsqu'un étudiant a moins de 10 de moyenne, il continue alors de passer des épreuves et d'obtenir des notes. Chaque nouvelle note obtenue remplace la plus mauvaise de ses notes précédentes, si elle était inférieure à la nouvelle note (si sa plus mauvaise note était un 4 et qu'il a obtenu 8, alors il y a remplacement ; s'il a obtenu 2, il garde son 4). Il n'y a pas d'autre possibilité pour lui modifier une note.

Un étudiant a toujours un nom et un prénom, qui ne peuvent pas être modifiés. Ses notes lui sont données au fur et à mesure, une par une. C'est lui qui calcule sa moyenne. Il ne communique d'ailleurs

que sa moyenne (sur 5 notes ou moins) aux autres. Il est capable de se présenter, de dire s'il est diplomé ou non, d'accepter une nouvelle note (s'il est déjà diplomé, il ne fait rien).

Les promotions sont d'au plus 30 étudiants. Pour ajouter un étudiant dans la promotion, on doit fournir le nom et le prénom de l'étudiant. Pour donner une note à un étudiant,

— soit on passe par la promotion, et alors on donne la note et l'indice de l'étudiant dans la promotion ;
— soit on connaît directement l'étudiant, (et il suffit de lui donner sa note).

Étape 1 : *Modifier autant que nécessaire la classe* `Etudiant`*, ses attributs, les entêtes de ses méthodes et de son constructeur. Les seuils de 10, pour la moyenne, et 5 pour le nombre de notes sont non modifiables et conservés dans des attributs de cette classe.*

Étape 2 : *Écrire maintenant les corps des méthodes et du constructeur de la classe* `Etudiant`*.*

Étape 3 : *Modifier la méthode* `main` *de* `TestEtudiant` *pour prendre en compte les modifications apportées.*

Étape 4 : *Ecrire la classe* `Promotion`*, ses attributs,les entêtes de ses méthodes et de son constructeur.*

Étape 5 : *Écrire maintenant les corps des méthodes et du constructeur* `Promotion`*.*

Étape 6 : *Écrire une méthode qui affiche tous les étudiants diplômés de la promotion.*

Étape 7 : *Écrire une méthode qui retourne le nombre d'étudiants diplômés, et une autre qui retourne le nombre total d'étudiants.*

Étape 8 : *Écrire une classe* `TestPromotion` *qui contient seulement une méthode* `main`*, et qui créé une promotion, quelques étudiants, afin de tester les classes écrites.*

Étape 9 : *Écrire une méthode* `nouvellePromotion(String nomPromo)` *qui créé et retourne une nouvelle promotion avec les étudiants diplômés de l'actuelle promotion. Les étudiants de la nouvelle promotion sont reconstruits comme de nouveaux étudiants (même nom, même prénom, pas de notes). Ils doivent être supprimés de la promotion courante, et le tableau des étudiants de la promotion courante doit être resserré (pas de cellule vide dans le tableau, et vous devez connaître à tout moment le nombre d'étudiants de la promotions=).*

Étape 10 : *Adaptez* `TestPromotion` *pour tester cette nouvelle méthode.*