Points et segments

Objectifs

— Comprendre les principaux concepts objets (encapsulation, constructeur) ;
— Écrire quelques méthodes Java ;
— Utiliser le JDK d'Oracle pour compiler, exécuter et documenter ;
— Définir, coder et tester une première classe.

L'objectif de ces exercices est de comprendre les concepts objets en exécutant en salle machine un exemple simple et en le complétant.

L'exemple que nous prenons consiste à implanter une version simplifiée d'un outil de dessin de schémas mathématiques. Un schéma peut contenir des points et des segments. La figure 1 montre un exemple de schéma : un triangle (composé de trois segments) et son centre de gravité.

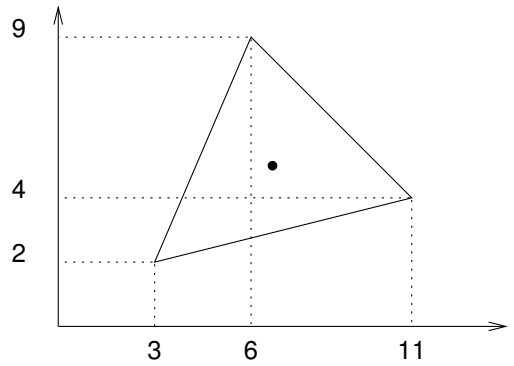

FIGURE 1 – Schéma mathématique composé de trois segments et de leur barycentre

Avertissement : Contrairement à ce que laisse supposer la figure 1, l'objectif n'est pas de faire du graphisme mais d'illustrer les concepts objets. Nous nous limiterons ainsi à une interface textuelle et à des rudiments de visualisation graphique.

Exercice 1 : Comprendre la classe Point
Le texte source de la classe Point correspond au diagramme d'analyse donné à la figure 2.

1.1 Expliquer comment on obtient le code Java d'une classe à partir de son diagramme d'analyse UML.

1.2 Indiquer, s'il y en a, les erreurs par rapport aux conventions de programmation en Java recommandées par Sun.

Exercice 2 : Compiler et exécuter
Dans cet exercice, nous allons utiliser les outils du JDK (Java Development Kit) pour compiler et exécuter une application.

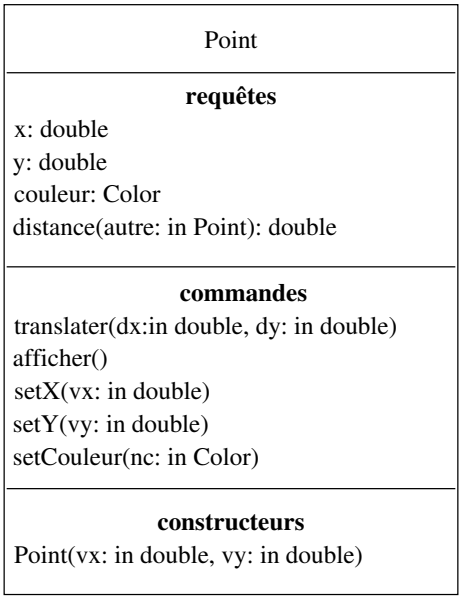

FIGURE 2 – Diagramme d'analyse de la classe Point

2.1 *Lire et comprendre le programme TestPoint.* Lire le programme TestPoint et dessiner l'évolution de l'état de la mémoire au cours de l'exécution du programme. Ceci doit être fait sur le listing, avant toute exécution du programme.

2.2 *Compiler TestPoint.* Le compilateur du JDK s'appelle `javac` (Java Compiler). Il transforme chaque classe (ou interface) contenue dans un fichier source Java (extension `.java`) en autant de fichiers contenant du byte code (extension `.class`). Remarquons que lorsqu'une classe est compilée, toutes les classes qu'elle utilise sont également automatiquement compilées. Par exemple, pour compiler la classe TestPoint contenue dans le fichier TestPoint.java, il suffit de taper la commande suivante :

```
1 javac TestPoint.java
```

Cette commande compile `TestPoint.java` et produit `TestPoint.class` mais aussi `Point.java` pour produire `Point.class` puisque TestPoint utilise la classe Point.
Attention : Pour que ceci fonctionne, il est nécessaire que le fichier porte exactement le même nom (y compris la casse) que l'unique classe publique qu'il contient.

Compiler le programme TestPoint fourni.
Remarque : On pourra utiliser l'option `-verbose` du compilateur `javac` pour lui faire afficher les différentes opérations qu'il réalise.

2.3 *Exécution de TestPoint.* Le byte code n'est pas directement exécutable par le système mais par la machine virtuelle de Java (Java Virtual Machine). La machine virtuelle fournie avec le JDK s'appelle `java`. Seules les classes qui contiennent la définition de la méthode principale peuvent être exécutées. La méthode principale est :

```
1 public static void main (String[] args)
```

Ainsi, la classe Point ne peut pas être exécutée et la classe TestPoint peut l'être. Pour exécuter TestPoint, il suffit de taper la commande :

```
1 java TestPoint
```

Attention : Il ne faut pas mettre d'extension (ni `.class`, ni `.java`), mais seulement le nom de la classe (en respectant la casse !).

Exécuter le programme TestPoint.

Remarque : Tous les appels suivants sont incorrects. Quels sont les messages d'erreurs indiqués ?

```
1 java TestPoint.java
2 java testpoint
3 java Point
4 java PasDeClasse
```

2.4 *Vérifier les résultats.* Vérifier que l'exécution donne des résultats compatibles avec l'exécution simulée à la question 2.1.

2.5 *Corriger le programme TestErreur.* Compiler le programme TestErreur. Le compilateur refuse de créer le point `p1`. Est-ce justifié ? Expliquer pourquoi. Quel est l'intérêt d'un tel comportement ? Corriger le programme.

2.6 *Durée de vie des objets.* Supprimer dans le fichier `Point.java` les commentaires devant l'affichage dans le constructeur (devant `System.out.println("CONSTRUCTEUR...");`) et les commentaires à la C (`/* */`) autour de la méthode `finalize` (vers la fin du fichier).

Compiler et exécuter le programme TestDestructeur. Que constatez-vous ? On pourra augmenter le nombre d'itérations jusqu'à 5000, 50000, voire plus.

Exercice 3 : Produire la documentation

Le JDK fournit un outil appelé javadoc qui permet d'extraire la documentation directement du texte des classes Java.

```
1 javadoc *.java
```

Seule est engendrée la documentation des classes contenues dans les fichiers donnés en argument de la commande `javadoc`. Par défaut, seuls les éléments publics sont documentés. On peut utiliser l'option `-private` pour engendrer la documentation complète (mais utile seulement pour le concepteur des classes).

La documentation doit bien entendu être donnée par le programmeur de la classe. Ceci se fait grâce aux commentaires qui commencent par `/**` et se terminent par `*/` et qui sont placés juste devant l'élément décrit. Des balises HTML peuvent être utilisées pour préciser une mise en forme du texte.

Il existe des étiquettes prédéfinies telles que *@author, @version, @see, @since, @deprecated* ou *@param, @return, @exception* pour la définition d'une méthode.

3.1 *Produire la documentation des classes.* Utiliser la commande javadoc pour produire la documentation des classes de notre application. Consulter la documentation ainsi engendrée.

3.2 *Consulter la documentation de la classe Segment.* En consultant la documentation engendrée pour la classe Segment, indiquer le sens des paramètres de la méthode `translater` et le but de la méthode `afficher`.

3.3 *Consulter la documentation en ligne.* La documentation des outils et des API Java est disponible en ligne à partir de l'URL suivante :

```
http://docs.oracle.com/javase/8/docs/api/
```

Exercice 4 : Comprendre et compléter la classe Segment

Une version incomplète d'une classe Segment est fournie ainsi que le programme TestSegment.

4.1 Compléter le diagramme de classes UML en faisant apparaître la classe Segment.

4.2 Compléter la classe Segment. On commencera par donner le code de la méthode `translater`, puis des méthodes `longueur` et `afficher`. Le code actuel n'est en effet pas le bon ! Le fichier `TestSegment` contient un programme de test.

Exercice 5 : Définir un schéma particulier

Écrire un programme Java qui construit le schéma de la figure 1. Ce schéma est composé de quatre éléments :

— trois segments (s_{12}, s_{23} et s_{31}) construits à partir de trois points (p_1, p_2 et p_3) ;
— et le barycentre de ces trois points.