Analyse des arguments de la ligne de commande

Préambule : Lire attentivement les points suivants.

1. Ce travail est un **travail individuel**. Vous pouvez échanger entre vous sur la compréhension du sujet et sur la manière de le traiter mais vous devez chacun écrire vos réponses et votre code. Sinon le bonus ne sera pas pris en compte.
2. Vous devez répondre aux questions dans un document. Les diagrammes UML peuvent être dessinés à la main ou avec un outil tel que `https://app.diagrams.net/` ou `https://plantuml.com`. Idéalement, il faudrait me le partager.
3. Les fichiers fournis sont sur SVN et sur Moodle.
4. Le code et le document doivent être déposés sur votre SVN (dossier projet) ou rendu via un git (URL à me fournir).
5. Une première version doit être rendue à la fin de la séance à 12h30. Une deuxième version peut être rendue jusqu'au 16 avril.

Dans ces exercices, nous nous intéressons au traitement des arguments de la ligne de commande qui sont utilisés pour paramétrer l'exécution d'un programme. On parle de CLI : *Command Line Interface*. Nous prenons l'exemple d'un programme `PageRankClassique` [1] qui a comme paramètres un *indice* (option -K, 150 par défaut), la valeur d'*alpha*, un nombre réel compris entre 0 et 1 qui intervient dans les calculs (-A, 0.85 par défaut), une *précision* (-E, -1 par défaut) et un *mode* de représentation des matrices, pleines (-P) ou creuses (-C, par défaut). Ces paramètres (ou options) sont précisés sur la ligne de commande dans n'importe quel ordre et, dans le cas où un paramètre apparaît plusieurs fois, c'est sa dernière apparition qui est prise en compte.

Voici un exemple possible : « `java PageRankClassique -K 10 -A .90 -K 20 -P -K 30 -C` ». Il conduit à la configuration : « `alpha=0.9, epsilon=-1.0, indice=30, mode=CREUSE` ».

Ces différents paramètres du programme sont regroupés dans une classe `Configuration` (listing 1) qui s'appuie sur le type énuméré `Mode` (listing 2). La classe du listing 3 analyse les arguments de la ligne de commande et définit la configuration correspondante ; elle est utilisée par `PageRankClassique` (listing 4).

Exercice 1 : La classe Configuration

Cette classe est volontairement minimale.

1.1. Expliquer ce que signifie `@Override`. Pourquoi apparait-elle ici ? Est-elle obligatoire ?

1.2. Expliquer pourquoi les attributs ne devraient pas être déclarés publics et indiquer le droit d'accès qu'il faudrait leur donner.

1. *Page rank* est juste un exemple de programme. Inutile de connaître son but.

```java
public class Configuration {
    public double alpha = 0.85;
    public double epsilon = -1.0;
    public int indice = 150;
    public Mode mode = Mode.CREUSE;

    @Override public String toString() {
        return "alpha=" + alpha + ", epsilon=" + epsilon
            + ", indice=" + indice + ", mode=" + mode;
    }
}
```

Listing 1 – La classe Configuration

```java
public enum Mode { PLEINE, CREUSE };
```

Listing 2 – Le type énuméré Mode

1.3. On souhaite savoir si deux objets `c1` et `c2` de type `Configuration` sont égaux (même valeur des attributs). Est-ce qu'écrire `c1.equals(c2)` est possible ? Dans l'affirmative, est-ce que le résultat est le bon ? Dans la négative à l'une des deux questions précédentes, que faut-il faire pour que cette expression soit acceptée et donne le résultat attendu ?

Exercice 2 : Analyse des arguments de la ligne de commande : version naïve

Intéressons nous maintenant à la version naïve proposée au listing 3.

2.1. Écrire une classe de test JUnit qui ne fait qu'un seul test : l'exemple du sujet.

2.2. Quel devrait être le résultat de « `java PageRankClassique -C -K 21 -E .001 -K 100` » ?

2.3. Voici le résultat de l'exécution de : `java CLIClassique java PageRankClassique -K 15.5 -P`

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "15.5"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.base/java.lang.Integer.parseInt(Integer.java:652)
    at java.base/java.lang.Integer.parseInt(Integer.java:770)
    at CLIClassique.configuration(CLIClassique.java:11)
    at PageRankClassique.main(PageRankClassique.java:4)
```

Que peut-on en déduire sur le programme ?

2.4. L'exception `NumberFormatException` n'est pas vérifiée par le compilateur. Peut-on le déduire de la lecture du listing 3 ? La réponse doit être justifiée.

2.5. On veut afficher un message d'erreur à l'utilisateur si les valeurs de alpha, indice ou epsilon ne sont pas au bon format. Modifier ce programme en conséquence, sachant que le message pourrait apparaître plusieurs fois si plusieurs erreurs sont commises par l'utilisateur ?

2.6. Ce programme contient encore des erreurs. Lesquelles ?

Exercice 3 : Version réutilisable

Le traitement des arguments de la ligne de commande est spécifique de l'application considérée. Ainsi, la classe `CLIClassique` devrait en fait s'appeler `PageRankCLIClassique` car elle ne peut pas

```java
public class CLIClassique {

    public static Configuration configuration(String... args) {
        Configuration config = new Configuration();
        boolean finOptions = false;
        int i = 0;
        while (i < args.length && ! finOptions) {
            String arg = args[i];
            switch (arg) {
                case "-K": // Valeur de l'indice à calculer
                    config.indice = Integer.parseInt(args[++i]);
                    break;
                case "-E": // Valeur de la précision à atteindre
                    config.epsilon = Double.parseDouble(args[++i]);
                    break;
                case "-A": // Valeur de alpha
                    config.alpha = Double.parseDouble(args[++i]);
                    break;
                case "-C": // Mode matrice creuse
                    config.mode = Mode.CREUSE;
                    break;
                case "-P": // Mode matrice pleine
                    config.mode = Mode.PLEINE;
                    break;
                default:
                    finOptions = arg.length() == 0 || arg.charAt(0) != '-';
                    if (! finOptions) {
                        System.out.println("Option inconnue : " + arg);
                    }
            }
            i++;
        }
        return config;
    }
}
```

Listing 3 – La classe CLIClassique

```java
public class PageRankClassique {

    public static void main(String... args) {
        Configuration configuration = CLIClassique.configuration(args);
        System.out.println(configuration);
        // Le reste du programme... Omis.
    }
}
```

Listing 4 – La classe PageRankClassique

être utilisée par un autre programme qui aurait des options différentes et donc une classe de configuration différente.

On souhaite écrire un traitement des arguments de la ligne de commande qui pourra être réutilisé dans différentes applications. Les paragraphes suivants décrivent le framework que nous nous proposons de réaliser.

1. Une interface en ligne de commande (CLI) propose plusieurs options.
2. Une option a un accès (ici limité à une lettre, A, K, E, P et C dans l'exemple) et une description (le commentaire sur la ligne des `case` sur le listing 3).
3. Une option peut nécessiter une valeur (par exemple A, K et E mais ni P ni C) qui sera donc l'argument suivant sur la ligne de commande.
4. On peut ajouter une nouvelle option sur une CLI.
5. Une CLI permet d'analyser les arguments de la ligne de commande (un tableau de chaînes de caractères) en fonction de ses options.
6. Une action est associée à une option.
7. Une action peut-être exécutée.
8. Exécuter une action dépend de l'application et de l'option considérée. Par exemple pour l'option P de l'application du sujet, il s'agit de positionner le mode à PLEINE. Pour l'option K, on affecte l'indice avec la valeur entière donnée par l'argument qui suit `"-K"`.

3.1. Dessiner le diagramme de classe qui fait apparaître les éléments mentionnés dans les paragraphes précédents. Pour assurer la traçabilité entre le paragraphe et l'élément on mettra à côté de l'élément le numéro du paragraphe correspondant dans un cercle.
3.2. Écrire avec ce nouveau framework l'équivalent de la classe `PageRankClassique` du listing 4 (et de `CLIClassique` du listing 3). On l'appellera `PageRank`. Elle s'appuie sur le framework modélisé.

3.3. Indiquer la structure de données de l'API des collections à utiliser pour stocker les options d'une CLI. La réponse doit être justifiée.
3.4. Écrire les classes et interface qui remplacent la classe `CLIClassique`. On aura en particulier une classe `CLI` pour le traitement général des arguments de la ligne de commande, indépendamment du programme considéré. Ces classes et interfaces ne doivent pas faire référence au programme *PageRank* ni à sa classe de configuration.

Exercice 4 : Interface SWING

On souhaite développer une petite interface graphique avec Swing pour construire les arguments de la ligne de commande (figure 1). Les arguments apparaissent en bas. Ils sont complétés quand on clique sur Creuse (-C est ajouté), Pleine (-P est ajouté) ou un plus « + » (l'option correspondante et la valeur sont ajoutées).
4.1. Expliquer comment produire la vue.
4.2. Expliquer comment faire pour que « -C » soit ajouté en bas quand l'utilisateur clique sur « Creuse (C) ».
4.3. Écrire le code correspondant. Notons que cette classe est complètement indépendante des classes précédentes.

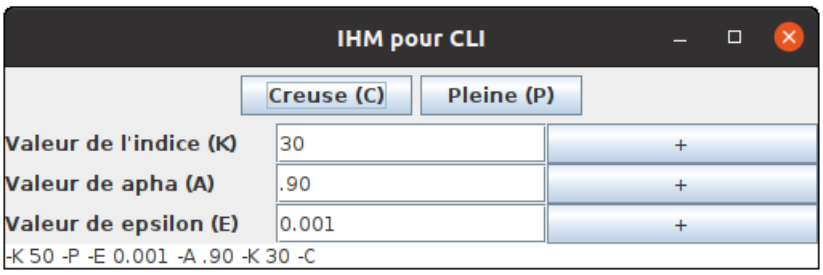

FIGURE 1 – Interface Swing pour construire les arguments de la ligne de commande

Exercice 5 : CLI depuis une classe de Configuration

Une classe telle que `Configuration` correspond aux options qui seront utilisées pour paramétrer un programme et qui pourront être initialisées via les arguments de la ligne de commande. Ici, nous souhaitons automatiser la création de l'objet CLI correspondant en adoptant les règles suivantes :

1. Tous les attributs de la classe, quelque soit leur droit d'accès, sont considérés comme des options possibles.
2. L'accès à l'option est l'initiale de l'attribut. La description de l'option est déduite du nom de l'attribut. Par exemple, l'attribut `alpha` donnera lieu à l'option d'accès « `a` » et de description « Valeur de alpha ».
3. Un traitement particulier est fait pour les attributs de type booléen. On leur associe deux options, la première avec l'initial en minuscule pour positionner l'attribut à vrai, la seconde avec l'initiale en majuscule pour le positionner à faux. La description de l'option est « Positionner xxx à vrai » (ou « faux »), xxx étant le nom de l'attribut.

5.1. Dans la classe CLIOutils, définir une méthode de classe `fromClass` qui retourne un objet de type CLI construit à partir du nom d'une classe de configuration en respectant les règles précédentes. On ne traitera pas les actions associées aux options.
5.2. Expliquer comment on pourrait traiter les actions associées aux options.

Exercice 6 : Sérialisation en XML

On veut produire un fichier XML (par exemple celui du listing 5) qui correspond à un objet CLI.
6.1. Écrire une DTD telle que le document du listing 5 lui soit conforme (fichier `cli.dtd`).
6.2. Écrire un programme Java qui engendre un fichier XML conforme à la DTD précédente à partir d'un objet de type CLI de l'exercice 3.
6.3. La définition des arguments de la ligne de commande pourrait se faire directement sous la forme d'un fichier XML comme le précédent. Expliquer comment il pourrait être exploité en Java pour produire un objet CLI.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cli SYSTEM "cli.dtd">

<cli>
    <argument acces="A" nb="1">Valeur de alpha</argument>
    <argument acces="I" nb="1">Valeur de indice</argument>
    <argument acces="P">Mode matrice pleine</argument>
    <argument acces="C">Mode matrice creuse</argument>
</cli>
```

Listing 5 – Exemple de fichier de configuration au format XML