Examen (1h30, feuille A4 autorisée)

— En cas de doute sur le sujet, notez vos choix sur la copie. Aucune réponse ne sera donnée par les surveillants.
— Il est conseillé de lire complètement le sujet avant de commencer à y répondre !
— Le code est à écrire en Java. Ne pas mettre les clauses d'importation.
— Inutile de mettre les commentaires de documentation.
— Barème indicatif (sur 20 points) :

Exercice	1	2	3	4	5
Points	5	3	4	5	3

Exercice 1 Répondre de manière concise et précise aux questions suivantes.

1.1. On considère une poignée `p` de type `T` initialisée avec un objet de la classe C. Indiquer comment est traité en Java l'appel de la méthode `m` suivant, `a` et `b` étant des variables. On précisera ce que l'on doit connaître de `a` et `b` pour traiter cette question.

```
T p = new C();
p.m(a, b);
```

1.2. Considérons la déclaration du listing suivant. Nous supposons qu'elle apparaît dans la classe `Point` vue en cours, TD et TP.

```
1 public class Point {
2     public static final Point origine = new Point(0, 0);
3     ...
4 }
```

1.2.1. Expliquer ce que signifient les mots-clés **`public`**, **`static`** et **`final`**.
1.2.2. Indiquer, en justifiant la réponse, si on peut être sûr que l'attribut `origine` de la classe `Point` aura toujours pour coordonnées $(0, 0)$.

1.3. On veut écrire une méthode `classer` qui ajoute chaque élément d'une première liste soit dans une liste `petits` soit dans une liste `grands` suivant que l'élément est plus petit ou plus grand qu'une valeur pivot. Un appel à cette méthode pourrait être le suivant :

```
classer(pivot, elements, petits, grands);
```

1.3.1. Donner la signature de cette méthode.
1.3.2. Écrire le code de cette méthode.

Analyse des arguments de la ligne de commande

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 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 : `-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). Le programme du listing 3 analyse les arguments de la ligne de commande et définit la configuration correspondante.

```
1  public class Configuration {
2      public double alpha = 0.85;
3      public double epsilon = -1.0;
4      public int indice = 150;
5      public Mode mode = Mode.CREUSE;
6
7      @Override public String toString() {
8          return "alpha=" + alpha + ", epsilon=" + epsilon
9              + ", indice=" + indice + ", mode=" + mode;
10     }
11 }
```

Listing 1 – La classe Configuration

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

Listing 2 – Le type énuméré Mode

Exercice 2 : La classe Configuration
Cette classe est volontairement minimale.
2.1. Expliquer ce que signifie `@Override`. Pourquoi apparait-elle ici ? Est-elle obligatoire ?
2.2. Expliquer pourquoi les attributs ne devraient pas être déclarés publics et indiquer le droit d'accès qu'il faudrait leur donner.
2.3. 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 3 : Analyse des arguments de la ligne de commande : version naïve
Intéressons nous maintenant à la version naïve proposée au listing 3.
3.1. Écrire une classe de test JUnit qui ne fait qu'un seul test : l'exemple du sujet.
3.2. Voici le résultat que l'on obtient quand on exécute : `java CLIClassique -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 CLIClassique.main(CLIClassique.java:37)
```

Que peut-on en déduire sur le programme ?

3.3. 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.
3.4. On veut afficher un message d'erreur à l'utilisateur si les valeurs de alpha, indice ou epsilon ne sont pas au bon format. Comment modifier ce programme pour obtenir ce résultat, sachant que le message pourrait apparaître plusieurs fois si plusieurs erreurs sont commises par l'utilisateur ?
3.5. Ce programme contient encore des erreurs. Lesquelles ?

Exercice 4 : Version réutilisable

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 CLI permet d'analyser les arguments de la ligne de commande (un tableau de chaînes de caractères) en fonction de ses options.
3. On peut ajouter une nouvelle option sur une CLI.
4. 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).
5. Une option peut nécessiter une valeur (par exemple A, K et E mais pas P et C) qui sera donc l'argument suivant sur la ligne de commande.
6. Une action est associée à une option.
7. Une action peut-être faite.
8. Faire 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"`.

4.1. 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.
4.2. Écrire avec ce nouveau framework l'équivalent du programme du listing 3 en se limitant aux options K et P.
4.3. Indiquer la structure de données de l'API des collections à utiliser pour stocker les options d'une CLI. La réponse doit être justifiée.

Exercice 5 : 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).
5.1. Expliquer comment produire la vue.
5.2. Expliquer comment faire pour que « -C » soit ajouté en bas quand l'utilisateur clique sur « Creuse (C) ».


```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;
    }

    public static void main(String[] args) {
        System.out.println(configuration(args));
    }
}
```

Listing 3 – La classe CLIClassique

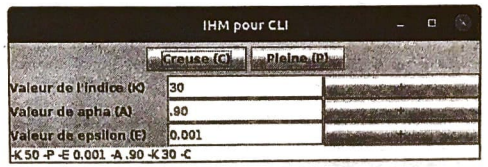

FIGURE 1 – Interface Swing pour construire les arguments de la ligne de commande