Travail pratique #2

Ce travail doit être fait individuellement.

Notions mises en pratique : le respect d'un ensemble de spécifications, la conception et l'utilisation de méthodes (séparation fonctionnelle), et l'utilisation de la classe `String`.

1. Spécifications

1.1 Spécifications

Il s'agit d'écrire un programme qui permet de corriger l'indentation d'un algorithme écrit en pseudocode se trouvant dans un fichier texte. De plus, le programme permet de formater, soit en majuscules, soit en minuscules, tous les mots réservés du langage pseudocode. Vous devez écrire votre programme dans une classe nommée `IndenteurPseudocode`.

1) Démarrage de l'application.

Au démarrage de l'application, le programme affiche une brève description de l'application, et un menu de deux options, puis attend le choix de l'utilisateur :

```
Ce programme permet de corriger l'indentation d'un algorithme ecrit en pseudocode.


----
MENU
----
1. Indenter pseudocode
2. Quitter

Entrez votre choix :
```

Votre programme doit valider la saisie du choix au menu. ATTENTION, seules les réponses `1` ou `2` doivent être considérées comme valides. Par exemple, `11`, `ENTER`, `abc`, … ne sont pas des réponses valides, et votre programme doit le détecter. De plus, votre programme ne doit jamais « planter » à la saisie du choix au menu. Pour ce faire, vous devez considérer le choix au menu comme une chaine de caractères.

=> Comme spécifications complémentaires, voir le fichier **exempleExec_1.txt** fourni avec l'énoncé du TP.

2) Option 1 : Indenter pseudocode

Au choix de cette option, le programme demande tout d'abord d'entrer le chemin du fichier contenant le pseudocode à indenter correctement. Votre programme doit valider le chemin entré. Un chemin valide est un chemin vers un fichier texte qui existe sur votre disque, et qui peut être lu. Par exemple, le chemin vers le fichier `fic.txt` pourrait être, sur Windows : `C:\Users\toto\Desktop\fic.txt` ou sur Mac : `/Users/toto/Desktop/fic.txt`

Pour lire et valider le chemin de fichier entré par l'utilisateur, vous DEVEZ utiliser la méthode `lireFichierTexte` de la classe `TP2Utils`, fournie avec l'énoncé du TP. Lisez bien la Javadoc de cette méthode pour comprendre ce qu'elle fait, et comment l'utiliser. Cette méthode est `static`, vous devez donc l'appeler en préfixant son nom du nom de la classe où elle se trouve : `TP2Utils.lireFichierTexte(cheminFic)` où `cheminFic` est le chemin du fichier à lire (préalablement entré par l'utilisateur). Cette méthode retourne le contenu du fichier donné par `cheminFic`, sous forme d'une chaine de caractères ou bien retourne `null` si le fichier n'existe pas ou ne peut pas être lu. Si la méthode retourne `null`, c'est que le chemin du fichier entré n'est pas valide (votre programme doit alors afficher un message d'erreur et redemander à l'utilisateur d'entrer un chemin valide, etc.).

ATTENTION : Pour utiliser la méthode `lireFichierTexte`, vous DEVEZ inclure la classe `TP2Utils` dans votre projet. Vous NE DEVEZ PAS copier la méthode `lireFichierTexte` dans votre propre classe.

Notez que si le fichier de pseudocode à indenter se trouve dans le dossier de votre projet Java (à la racine de votre projet), le chemin du fichier entré par l'utilisateur peut n'être que son nom. Pour faire vos tests, donc, il serait plus convivial d'enregistrer tous vos fichiers de pseudocode à indenter, à la racine de votre projet. Par exemple, supposons que le fichier de pseudocode mal indenté suivant, disons `test1.psdo`, soit enregistré à la racine de votre projet :

```
Debut
afficher "Entrez un nombre : "
       lire nombre

     Si nombre mod 2 = 0 alors
afficher "C'est un nombre pair"
     Fin si

  Fin
```

L'appel de la méthode `TP2Utils.lireFichierTexte("test1.psdo")` retournera la chaine suivante :

```
"Debut      \n" +
"afficher \"Entrez un nombre : \"    \n" +
"       lire nombre   \n" +
"      \n" +
"     Si nombre mod 2 = 0 alors   \n" +
"afficher \"C'est un nombre pair\"     \n" +
"     Fin si    \n" +
" \n" +
"  Fin\n"
```

Notez que la fin d'une ligne est déterminée par un caractère saut de ligne `'\n'`. Notez aussi qu'il peut y avoir des **caractères blancs** (`'\n'`, `'\t'`, `' '`) avant ou après chaque ligne. On suppose aussi que le fichier donné par l'utilisateur, s'il existe, n'est pas vide, et contient bien du pseudocode correctement formé (valide), bien que possiblement mal indenté.

Lorsque le chemin du fichier entré par l'utilisateur est valide, le programme demande ensuite d'entrer la lettre `m` pour formater les mots réservés en minuscules ou `M` pour formater les mots réservés en majuscules. Seules ces deux réponses sont considérées comme valides, et le programme doit valider cette entrée. Par exemple, `MM`, `1`, `ENTER`, ne sont pas des entrées valides, et votre programme doit le détecter. Pour ce faire, vous devez considérer cette entrée comme une chaine de caractères. Notez que la liste des mots réservés est donnée à la section 1.2. Finalement, le programme affiche à l'écran, entre deux lignes de 20 tirets, le pseudocode correctement indenté, et dont les mots réservés ont été formatés en majuscules ou en minuscules, selon le cas. Le programme demande ensuite de taper <ENTRÉE> pour continuer (et revenir au menu). Exemple (avec le fichier `test1.psdo` montré ci-dessus, et avec le choix de transformer les mots réservés en majuscules) :

```
--------------------
DEBUT
   AFFICHER "Entrez un nombre : "
   LIRE nombre

   SI nombre mod 2 = 0 ALORS
      AFFICHER "C'est un nombre pair"
   FIN SI

FIN
--------------------

Tapez <ENTREE> pour continuer...
```

=> Comme spécifications complémentaires, voir le fichier **exempleExec_2.txt** fourni avec l'énoncé du TP.

3) Option 2 : Quitter

Au choix de cette option, le programme se termine en affichant un message de fin suivant :

```
F I N   N O R M A L E   D U   P R O G R A M M E
```

=> Comme spécifications complémentaires, voir le fichier **exempleExec_1.txt** fourni avec l'énoncé du TP.

1.2 Note sur l'implémentation

Définitions :

Les **caractères blancs** considérés dans ce TP sont : Espace (`' '`), tabulation (`'\t'`) ou saut de ligne (`'\n'`).

Les **lignes blanches** sont des lignes qui ne contiennent qu'un seul ou plusieurs caractères blancs.

Les **mots réservés** du pseudocode sont : `debut`, `fin`, `tant`, `que`, `si`, `alors`, `sinon`, `faire`, `afficher`, `ecrire`, `lire`, et `saisir`.

Mis à part l'indentation, le pseudocode dont on veut corriger l'indentation **est supposé correctement formé (valide),** c'est-à-dire qu'il respecte les règles suivantes :

1) Les instructions simples pouvant se trouver dans le pseudocode sont :

   Lecture : les mots réservés permis pour la lecture sont **`lire`** ou **`saisir`**.
   Ex. : **`lire`** `variable` ou **`saisir`** `variable`

   Écriture : les mots réservés permis pour l'écriture sont **`ecrire`** ou **`afficher`**
   Ex. : **`afficher`** `variable` ou **`ecrire`** `"Entrez un nombre : "`

   Affectation : sous la forme `variable <- valeur`

   Chaque instruction de lecture / écriture / affectation se trouve sur une seule ligne, et rien d'autre ne se trouve sur cette ligne (une seule instruction simple par ligne).

2) Les trois structures de contrôle permises dans le pseudocode sont :

Structure de contrôle	Exemple (mal indenté) qui respecte les règles.
Sélection : • Commence toujours avec un **`Si`** `condition` **`alors`** (début de bloc) • Peut contenir un ou plusieurs **`Sinon si`** `condition` **`alors`** (fin et début de bloc) • Peut optionnellement se terminer par un **`Sinon`** (fin et début de bloc) • Se termine toujours par **`Fin si`** (fin de bloc) • Les énoncés : o **`Si`** `condition alors` o **`Sinon si`** `condition` **`alors`** o **`Sinon`** o **`Fin si`** se trouvent toujours sur une seule ligne (et rien d'autre ne se trouve sur cette ligne).	**`Si`** `condition` **`alors`** `Instructions` **`Sinon si`** `condition` **`alors`** `Instructions` **`Sinon si`** `condition` **`alors`** `Instructions` **`Sinon`** `Instructions` **`Fin si`**

Boucle Tant que • Commence toujours par **`Tant que`** `condition` **`faire`** (début de bloc) • Se termine toujours par **`Fin tant que`** (fin de bloc) • Les énoncés : o **`Tant que`** `condition` **`faire`** o **`Fin tant que`** se trouvent toujours sur une seule ligne (et rien d'autre ne se trouve sur cette ligne).	**`Tant que`** `var > 6` **`faire`** `Instructions` **`Fin tant que`**
Boucle Faire… tant que • Commence toujours par **`Faire`** (début de bloc) • Se termine toujours par **`Tant que`** `condition` (fin de bloc) • Les énoncés : o **`Faire`** o **`Tant que`** `condition` se trouvent toujours sur une seule ligne (et rien d'autre ne se trouve sur cette ligne).	**`Faire`** `Instructions` **`Tant que`** `condition`

3) Le pseudocode ne contient aucune lettre accentuée.
4) Le pseudocode peut être écrit en majuscules, en minuscules ou un mélange des deux.
5) Le pseudocode peut contenir des lignes blanches.
6) Le pseudocode peut contenir des caractères blancs superflus au début ou à la fin de chacune de ses lignes.
7) Le pseudocode commence toujours par le mot réservé `debut`, et se termine toujours par le mot réservé `fin`. Il peut cependant y avoir des **caractères blancs** avant ou après ces deux mots réservés.
8) Les mots réservés sont séparés par un ou plusieurs caractères blancs.
9) Les chaines de caractères sont reprentées entre guillemets, et les caractères, entre apostrophes.
10) Les opérateurs de pseudocode vus dans le cours sont permis, et les expressions contenant ces opérateurs sont bien formées.
11) Aucune variable ne porte le nom d'un mot réservé comme `fin` ou `debut`…
12) Le pseudocode peut contenir des commentaires, mais seulement des commentaires de ligne (`//`). De plus :
    a. Il peut y avoir plusieurs lignes de commentaires, une à la suite de l'autre, par exemple :
    ```
    //Ceci est un commentaire qui se
    //poursuit sur cette deuxieme ligne
    ```
    b. Un commentaire doit se trouver seul sur sa ligne. Par exemple, on ne peut pas trouver une ligne du genre dans le pseudocode : `var <- 5 //initialisation`
    c. Les commentaires peuvent se trouver au-dessus de n'importe quelle instruction simple (lecture / écriture / affectation).
    d. Les commentaires peuvent se trouver au-dessus <u>du début</u> d'une structure de contrôle. Plus précisément, au-dessus d'un :
        - **`Si`** `condition` **`alors`**
        - **`Faire`**
        - **`Tant que`** `condition` **`faire`**

        Et non au-dessus d'un **`Fin tant que`** ou d'un **`Sinon`** ou d'un **`Sinon si`** `condition` **`alors`**, etc.

En vous basant sur le fait que le pseudocode à indenter respecte les règles précédentes, vous devez indenter correctement le pseudocode, et formater correctement les mots réservés (en majuscules ou en minuscules) en respectant les règles décrites ci-dessous.

Règles d'indentation et de formatage

1) Un niveau d'indentation doit correspondre à 3 espaces.
2) Lorsqu'on entre dans un bloc, on doit ajouter un niveau d'indentation. Deux blocs de même niveau (blocs "frères") doivent commencer, et se terminer au même niveau d'indentation.
3) S'il y a des caractères blancs avant `debut` ou après `fin`, on doit les supprimer.
4) S'il y a des lignes blanches entre `debut` et `fin`, celles-ci ne sont pas supprimées, mais après le traitement, elles ne doivent contenir qu'un seul caractère `'\n'`.
5) Lorsqu'il y a plus d'un caractère blanc qui sépare un mot réservé d'un autre mot, on les laisse tels quels. Par exemple (avec un formatage en majuscules), la ligne **`Si`** `condition` **`alors`** sera transformée comme ceci : **`SI`** `condition` **`ALORS`** (en conservant les caractères blancs entre `SI` et `condition` et entre `condition` et `ALORS`).
6) Lorsqu'une chaine de caractères (indiquée entre guillemets) contient un mot réservé, celui-ci ne doit pas être modifié. Par exemple, si l'utilisateur demande que les mots réservés soient en majuscules, et que le pseudocode à indenter / formater est : `afficher "Fin normale du programme"`, après le traitement, cette ligne deviendra `AFFICHER "Fin normale du programme"` où le mot `Fin`, dans la chaine de caractères, n'a pas été modifié.
7) Les mots réservés dans un commentaire ne doivent pas non plus être modifiés. Par exemple, lorsque l'utilisateur demande que les mots réservés soient en majuscules, le commentaire `//Afficher la fin du programme` ne doit pas être transformé en `//AFFICHER la FIN du programme`, mais plutôt demeurer tel quel.
8) Règle d'indentation des commentaires :

   Les commentaires se trouvant au-dessus d'une instruction simple (lecture / écriture / affectation) doivent être au même niveau d'indentation que cette instruction simple. Par exemple :

```
//Solliciter l'age
afficher "Entrez l'age : "
lire age
```

   Les commentaires se trouvant au-dessus du début d'une structure de contrôle doivent être au même niveau d'indentation que le début de cette structure de contrôle. Par exemple :

```
//Compter les nombres impairs
Tant que nombre > 0 faire

   //Tester si impair
   Si nombre mod 2 = 1 alors
      cpt <- cpt + 1
   Fin si

   nombre <- nombre - 1
Fin tant que
```

Indices sur la façon de procéder :

Avant de commencer à indenter correctement le pseudocode, je vous conseille de retirer tous les caractères blancs avant et après chacune des lignes du pseudocode, en ne laissant que le caractère `'\n'` à la fin de chaque ligne. Ceci aura pour effet de placer toutes les lignes du pseudocode au niveau d'indentation 0 (aucune indentation), et d'enlever tous les caractères blancs superflus en fin de ligne. Vous pouvez en même temps vous assurer que les lignes blanches (entre `debut` et `fin`) ne contiennent qu'un caractère `'\n'`.

La méthode ci-dessous fait ce travail. Vous pouvez l'utiliser telle quelle ou la modifier selon vos besoins, mais vous n'êtes pas tenus de l'utiliser. C'est en même temps un exemple de comment faire pour parcourir chacune des lignes du pseudocode en utilisant la méthode `indexOf` de la classe `String`.

```java
/**
 * Cette methode retourne une nouvelle chaine de caracteres qui represente
 * le pseudocode donne en parametre, auquel on a enleve tous les caracteres
 * blancs au debut de chaque ligne, et tous les caracteres blancs superflus
 * a la fin de chaque ligne (en conservant cependant le caractere '\n' qui
 * marque la fin d'une ligne).
 *
 * @param pseudocode le pseudocode a traiter.
 * @return une nouvelle chaine representant le pseudocode donne en parametre,
 *         auquel on a enleve tous les caracteres blancs au debut et a la fin
 *         de chaque ligne (en conservant le caractere '\n' a la fin de chaque
 *         ligne).
 */
public static String trimer (String pseudocode) {
   String sTrim = ""; //pseudocode a retourner
   String sTmp;
   int debut = 0; //indice du debut de la ligne courante
   int fin;       //indice de la fin de la ligne courante

   //parcourir les lignes, et enlever les caracteres blancs avant et après
   //chaque ligne sauf le \n a la fin de la ligne.
   pseudocode = pseudocode.trim() + "\n";
   fin = pseudocode.indexOf("\n", debut);

   while (fin != -1) {
      //extraire la ligne courante et enlever tous les caracteres blancs
      //en debut et fin de ligne
      sTmp = pseudocode.substring(debut, fin).trim();

      //concatener la ligne courante a la chaine a retourner, en ajoutant
      //le saut de ligne a la fin.
      sTrim = sTrim + sTmp + "\n";

      //ajuster le debut de la ligne courante suivante
      debut = fin + 1;

      //trouver la fin de la ligne courante suivante
      fin = pseudocode.indexOf("\n", debut);
   }

   return sTrim;
}
```

Ensuite, pour déterminer le niveau d'indentation, il s'agit de parcourir chacune des lignes du pseudocode (inspirez-vous de la méthode précédente), en commençant avec un niveau d'indentation égal à 0 (aucune indentation). Les mots réservés `debut` et `fin`, qui englobent l'algorithme, sont les seuls à ce niveau. Il s'agit ensuite de parcourir chacune des lignes du pseudocode, et de déterminer si l'on commence un nouveau bloc. Si c'est le cas, il faut augmenter le niveau d'indentation de 1 pour toutes les lignes suivantes tant que l'on ne rencontre pas la fin du bloc. Lorsqu'on rencontre la fin du bloc, on diminue le niveau d'indentation de 1.

Pour savoir si l'on entre dans un nouveau bloc, il faut déterminer si la ligne courante correspond au début d'une des structures de contrôle permises dans le pseudocode. Donc, si la ligne ne contient que le mot `debut`, c'est le début du bloc principal de l'algorithme, si la ligne commence par `tant que`, et se termine par `faire`, c'est le début d'un nouveau bloc pour une boucle tant que. Si la ligne ne contient que le mot `faire`, c'est le début du bloc d'une boucle faire… tant que. Si la ligne commence par `Si`, c'est le début d'une sélection. Lorsque la ligne ne contient que `fin`, `fin si`, `fin tant que` ou commence par `tant que` mais ne se termine pas par `faire`, c'est la fin d'un bloc, respectivement, bloc principal, `si`, `tant que`, et `faire… tant que`. Attention, cependant, aux instructions de sélections qui contiennent des blocs `Sinon si` ou un bloc `Sinon`. Par exemple, dans la sélection ci-dessous, la ligne **`Sinon si`** `nbr > 3` **`alors`** marque en même temps la fin du bloc **`Si`**, et le début du bloc **`Sinon si`**. De la même manière, la ligne **`Sinon`** marque en même temps la fin du bloc **`Sinon si`**, et le début du bloc **`Sinon`**. Cela veut dire

qu'on doit d'abord revenir d'un niveau d'indentation pour écrire la ligne **`Sinon si`**… (ou **`Sinon`**), et augmenter ensuite d'un niveau d'indentation pour les lignes suivantes qui sont des instructions se trouvant dans le bloc qui débute.

```
Si nbr > 5 alors
   instructions…
Sinon si nbr > 3 alors
   instructions…
Sinon
   instructions…
Fin si
```

Trois fichiers texte contenant du pseudocode valide, mais mal indenté, sont fournis avec l'énoncé du TP pour vous aider à tester votre programme : **`test1.psdo`**, **`test2.psdo`**, et **`test3.psdo`**. Vous devriez aussi vous créer d'autres fichiers de pseudocode valide, mais mal indenté, pour tester votre programme sur un plus grand échantillon de cas possibles. Notez que votre programme sera testé avec des fichiers de pseudocode différents que ceux fournis.

Méthodes qui pourraient être utiles (lire leur Javadoc pour comprendre leur utilité, et la manière de les utiliser) :

- Dans la classe `Character` : `isWhitespace`
- Dans la classe `String` (outre les méthodes vues en classe) : `startsWith`, `endsWith`

Fichiers fournis :

- **`TP2Utils.java`** => Classe à importer dans votre projet. NE PAS LA MODIFIER, car cette classe sera utilisée telle quelle dans les tests de votre programme, lors de la correction.
- Les fichiers texte contenant des exemples de pseudocode à indenter (pour tester votre programme) : **`test1.psdo`**, **`test2.psdo`**, et **`test3.psdo`**.
- Les fichiers texte **`test1-indente.psdo`**, **`test2-indente.psdo`**, et **`test3-indente.psdo`** contenant les résultats attendus après l'indentation / formatage des exemples de pseudocode contenus respectivement dans `test1.psdo`, `test2.psdo`, et `test3.psdo`. Ces exemples doivent être considérés comme des spécifications complémentaires à celles indiquées dans cet énoncé.

2. Précisions

- Votre classe `IndenteurPseudocode` doit se trouver dans le paquetage par défaut (il ne doit pas y avoir d'instruction de paquetage `package`… dans le haut de votre classe).
- Vous devez écrire votre programme dans une classe nommée `IndenteurPseudocode` qui contiendra votre méthode `main` et toutes les autres méthodes que vous aurez conçues.
- Prenez soin de bien découper votre code à l'aide de méthodes. Voici quelques lignes directrices pour vous guider dans la conception de vos méthodes :
  - Chaque fois que vous vous apercevez que différentes parties de votre code se ressemblent énormément, essayer d'en faire une méthode bien paramétrée (si possible).
  - Faites une méthode pour chaque petite tâche spécifique à accomplir dans la résolution du problème.
  - Si une méthode semble longue, posez-vous la question à savoir si elle pourrait être découpée en plusieurs méthodes plus spécialisées.
- **Toutes vos méthodes doivent être** `public static`**.**
- N'oubliez pas d'écrire les commentaires de documentation pour chacune de vos méthodes (sauf la méthode `main`).
- Lorsqu'on vous demande de valider une valeur saisie par l'utilisateur, cela sous-entend une BOUCLE de validation qui sollicite la valeur, lit la valeur, et lorsque la valeur saisie est invalide, affiche un message d'erreur, sollicite et lit de nouveau la valeur, et ce tant que la valeur saisie n'est pas valide.

- Vous DEVEZ utiliser la classe `Clavier.java` pour effectuer toutes les saisies.
- Les seules CLASSES PERMISES sont `TP2Utils`, `String`, `Character`, `Math`, `Clavier` (pour la saisie), et `System` (pour l'affichage).
- Vous **NE DEVEZ PAS utiliser les expressions régulières**, ni la classe `StringTokenizer` (qu'on ne verra pas dans ce cours).
- **Vous NE DEVEZ PAS utiliser les tableaux (qu'on n'a pas vus encore). ATTENTION, l'utilisation de tableaux peut engendrer une pénalité pouvant aller jusqu'à 30% puisque vous passerez à côté de ce qu'on veut noter dans ce TP sur la manipulation des chaines de caractères.**
- Vous DEVEZ respecter à la lettre les **informations, les messages**, et le **format** de l'affichage qui sont montrés dans les exemples d'exécution fournis avec l'énoncé du TP. Vous devez considérer les exemples d'exécution comme des spécifications complémentaires à celles décrites dans cet énoncé.
- Toute **VARIABLE GLOBALE** est **INTERDITE : toutes les variables doivent être déclarées à l'intérieur (et au début) d'une méthode** (sauf pour les boucles `for` où vous pouvez déclarer la variable de contrôle dans l'entête de la boucle)**.**
  - Pour passer une information du programme appelant (celui qui appelle la méthode) au programme appelé (la méthode appelée), vous devez passer cette information en paramètre.
  - Pour passer une information du programme appelé (une méthode quelconque) au programme appelant (celui qui appelle la méthode), utilisez la valeur de retour de la méthode appelée.
- Votre programme NE DOIT JAMAIS PLANTER lors d'une saisie faite par l'utilisateur.
- Utilisez des constantes (`final`) autant que possible : les messages affichés, les bornes de validation, etc.
- Les constantes doivent être déclarées et initialisées au niveau de la classe (constantes globales) : `public static final`…
- L'affichage des résultats doit se faire à la console.
- Utilisez des variables réelles uniquement lorsque requis. Ex. : un compteur ne doit pas être `float` ou `double`.
- Vous DEVEZ respecter le style Java.
- Votre fichier à remettre doit être encodé en **UTF8**.
- Votre code doit compiler et s'exécuter avec le **JDK 7**. Si vous n'utilisez que ce qu'on a vu en classe, ce sera le cas.

Le non-respect de toute spécification ou consigne se trouvant dans l'énoncé du TP (et les exemples d'exécution donnés avec l'énoncé du TP) est susceptible d'engendrer une perte de points.

Note : *Si quelque chose est ambigu, obscure, s'il manque de l'information, si vous ne comprenez pas les spécifications, si vous avez des doutes… vous avez la responsabilité de vous informer auprès de votre enseignante.*

3. Détails sur la correction

3.1 La qualité du code (40 points)

Concernant les critères de correction du code, lisez attentivement le document "**Critères généraux de correction du code Java"**. De plus, votre code sera noté sur le respect des conventions de style Java (dont un résumé se trouve dans le document **"Quelques conventions de style Java"**. Ces deux documents peuvent être téléchargés dans la section TRAVAUX PRATIQUES (ET BOITES DE REMISE) de la page Moodle du cours.

Note : Votre code sera corrigé sur la totalité ou une partie des critères de correction indiqués ci-dessus, sur le respect des spécifications/consignes mentionnées dans ce document, et sur le respect des bonnes pratiques de programmation.

3.2 L'exécution (60 points)

Un travail qui ne compile pas se verra attribuer la note 0 pour l'exécution.

Note : Votre code sera testé en tout ou en partie. Les points seront calculés sur les tests effectués. Ceci signifie que si votre code fonctionne dans un cas x et que ce cas x n'est pas testé, vous n'obtiendrez pas de points pour ce cas. Il est donc dans votre intérêt de vous assurer du bon fonctionnement de votre programme dans tous les cas possible. Notez que les exemples d'exécution fournis ne couvrent pas tous les cas à tester.

4. Date et modalité de remise

4.1 Remise

Date de remise : AU PLUS TARD le 19 novembre 2022 à 23h59.
Le fichier à remettre : `IndenteurPseudocode.java` (PAS dans une archive zip, rar, etc.)

Remise via Moodle uniquement.

Vous devez remettre (téléverser) votre fichier sur le site du cours (Moodle). Vous trouverez la boite de remise du TP2 dans la section TRAVAUX PRATIQUES (ET BOITES DE REMISE).

Vérifiez deux fois plutôt qu'une que vous avez remis le bon fichier, car c'est celui-là qui sera considéré pour la correction.

4.2 Politique concernant les retards

Une pénalité de 10% de la note finale, par jour de retard, sera appliquée aux travaux remis après la date limite. La formule suivante sera utilisée pour calculer la pénalité pour les retards : Nbr points de pénalité = m / 144, où m est le nombre de minutes de retard par rapport à l'heure de remise. Ceci donne 10 points de pénalité pour 24 heures de retard, 1.25 point de pénalité pour 3 heures, etc.

Aucun travail ne sera accepté après 1 jour (24 h) de retard, et la note attribuée sera 0.

4.3 Remarques générales

- **Aucun programme reçu par courriel ne sera accepté**. Plus précisément, un travail reçu par courriel sera considéré comme un travail non remis.

- Vous avez la responsabilité de conserver des copies de sauvegarde de votre travail (sur disque externe, Dropbox, Google Drive, etc.). La perte d'un travail due à un vol, un accident, un bris… **n'est pas une raison valable pour obtenir une extension pour la remise de votre travail**.

- **N'oubliez pas d'écrire (entre autres) votre nom complet et votre code permanent dans l'entête de la classe à remettre.**

N'attendez pas à la dernière minute pour commencer le travail, vous allez fort probablement rencontrer des problèmes inattendus !

Le règlement sur le plagiat sera appliqué sans exception. Vous devez ainsi vous assurer de ne pas échanger du code avec des collègues ni de laisser sans surveillance votre travail au laboratoire. Vous devez également récupérer rapidement toutes vos impressions de programme au laboratoire.

BON TRAVAIL !