Université Paris Cité **Programmation Orientée Objet**
L2 **Année 2022-2023**

TP n° 5

Héritage

Remarques générales

— Nous vous rappelons qu'il est utile et nécessaire de tester votre code au fur et à mesure.

— Il pourra être utile de consulter la documentation à la page : `https://docs.oracle.com/javase/10/docs/api/` en particulier pour la classe `Scanner` (Cliquez sur une méthode pour avoir une description détaillée de son fonctionnement)

— Un mémo sur la classe Scanner et les expressions régulières est également disponible sur Moodle. N'hésitez pas à le consulter.

— Vous pouvez toujours ajouter des méthodes intermédiaires pour mieux factoriser le code même si le sujet ne le mentionne pas.

1 But général et structure général du code

Le but du TP sera de réaliser un formateur de texte fonctionnant sur le même principe que le formateur de texte `fmt` sous Unix.
Le formateur fait une lecture du fichier et le découpe en paragraphes. Ces paragraphes étant eux mêmes composés de lignes. Lors de l'écriture le formateur imprime la liste des paragraphes en insérant une ligne vide entre chaque paragraphe. Il justifiera éventuellement ces lignes. On aura ainsi éliminé les lignes vides, les espaces et tabulations inutiles.
On modélise le problème de la façon suivante : on introduit le concept de ChaineChar qui représente les objets composant le texte.

— une *ChaineCar* est un élément du texte formaté qui a une taille et peut être affiché.

— un *Espace* est un élément du texte formaté qui séparera les mots. Ce sera un seul espace dans le cas où le texte n'est pas justifié, et potentiellement plusieurs espaces sinon.

— un *Mot* est un élément du texte formaté qui représentera un mot.

— une *Ligne* représentera une ligne de texte.

— le *Formateur* utilise la classe `Scanner` et les classes précédemment décrites pour construire le texte formaté.

Ce qu'on appelle *mots* est en fait toute suite de caractères sans espaces, ni tabulation ni retour à la ligne. Par exemple, `"1.2m"` et `"Hello!"` sont des mots.

2 Les Chaînes de caractère

Exercice 1
Créez une classe `ChaineCar`, qui contient deux méthodes publiques : `len()`, de type entier, `toString()`, de type String. Par défaut, on considérera que la longueur est 0 et `toString()` renvoie la chaîne vide.
(Si vous connaissez les classes abstraites, vous pouvez déclarer que `ChaineCar` et la méthode `len()` sont abstraites)

Exercice 2
Écrivez les définitions de deux classes qui héritent la classe `ChaineCar` : `Espace` et `Mot`. Un `Espace` a une longueur de 1, et se convertit en la chaîne réduite à un espace `" "` (à ne pas confondre avec la chaîne vide !). Un `Mot` représente une chaîne arbitraire, sa méthode `toString` retourne cette chaîne, et la méthode `len()` retourne sa longueur.

Exercice 3
Écrivez maintenant la classe `Ligne` qui hérite la classe `ChaineCar`. Une Ligne contient une suite de ChaineCar (une `LinkedList`) ; sa longueur est la somme des longueurs des ChaineCar qu'elle contient, et sa représentation est la concaténation des représentations des ChaineCar qu'elle contient.
En plus des méthodes héritées de la classe `ChaineCar`, la classe `Ligne` aura une méthode publique `isEmpty` qui détermine si une Ligne est vide, ainsi qu'une méthode publique `addChaine` qui ajoute une ChaineCar à la fin d'une Ligne.

Exercice 4 Définissez une classe `Paragraphe`. Un paragraphe hérite aussi de ChaineCar et contient une suite de lignes. Cette classe contient aussi les méthodes `isEmpty` et `addChaine` (pensez à utiliser les méthodes définies dans la classe `Ligne`. Remarquez que cette version de paragraphe contiendra au plus une ligne, ceci sera changé par la suite).

3 Formateur de texte très simple

Exercice 5
La classe `Formateur` contiendra deux méthodes principales. L'une, `read()`, lit le texte sur un fichier et en stockera la partie logique (liste des paragraphes).
L'autre méthode, `print()`, affichera ce texte en insérant une ligne vide entre chaque paragraphe. Les attributs de `Formateur` seront un `Scanner sc` et une liste chaînée `LinkedList<Paragraphe> texte` pour stocker la liste des paragraphes.

Constructeur Dans le constructeur, on va ouvrir le fichier et attacher un `Scanner` à celui-ci. Cela nous permettra de lire le fichier de la même manière qu'on lit des entrées clavier. A noter que la lecture se fait comme si on avait le doigt sur le caractère qu'on lit. Quand on a lu le caractère, le doigt est déjà sur le caractère d'après et on ne peut pas revenir en arrière : il est donc impossible de relire plusieurs fois le même caractère.
Il faudra déclarer `import java.util.Scanner` ; et `import java.io.File` ; Nous vous donnons le code du constructeur :

```
//fic est le nom du fichier,
//chemin compris s'il n'est pas dans le même répertoire
public Formateur(String filename) {
    sc = null;
    try {
        sc = new Scanner(new File(filename));
    }
    catch(Exception e) {
        System.out.println("Erreur lors d'ouverture fichier:");
        e.printStackTrace();
        System.exit(1);
    }
    texte = new LinkedList<Paragraphe>();
}
```

Méthode read La méthode `read` va remplir l'attribut `texte` avec la liste des paragraphes du texte contenu dans le fichier. Pour cela, il peut être utile de faire une méthode privée `Paragraphe readParagraphe()`. Le principe est le suivant : on utilise deux Scanner, l'un, `sc`, sert à lire paragraphe par paragraphe, l'autre sera attaché à chaque paragraphe lue pour pouvoir le découper en mots. Vous aurez besoin du constructeur `Scanner (String s)` et de la méthode `next()`.

Il faudra ajouter une boîte espace après chaque boîte mot, sauf à la fin du paragraphe. Il peut être plus simple de faire cet ajout systématiquement et de supprimer le dernier espace à la fin du paragraphe. (Ajoutez une méthode appropriée dans `Paragraphe`).
Il vous faudra définir ce que rend `readParagraph()` s'il n'y a plus que des lignes vides à lire. Pensez aux lignes vides qui peuvent être au tout début du texte ou à la fin, à celles qui peuvent être redondantes.

Pour plus d'aide avec la méthode `read`, lisez attentivement le mémo sur la classe Scanner et les expressions régulières, disponible sur Moodle.

Méthode Print la méthode `print` imprime les paragraphes et imprime une ligne vide après chacun d'eux sauf le dernier.

Test Testez votre code avec les quatre textes fournis sur Moodle. Les fichiers `texte` et `texteBis` contiennent le même texte aux espaces, tabulations et lignes vides près. Le fichier `vide` est vide ! et le fichier `videBis` ne contient que des espaces, des tabulations et des lignes vides. Vérifier que les résultats sont corrects.

4 Formateur de texte (un peu) plus avancé

Nous avons programmé un formateur simple qui contient les fonctionnalités de base. Maintenant, on veut un formateur qui affiche les paragraphes de manière plus jolie, avec des indentations et justifié.

4.1 Longueur limite

Pour l'instant, nos paragraphes ne sont constitués que d'une seule ligne. On veut qu'ils contiennent une liste de lignes dont la longueur ne dépasse pas une certaine valeur.

Exercice 6
Créez une nouvelle classe `ParagrapheJoli` qui hérite de `Paragraphe`, et `FormateurJoli` qui hérite de `Formateur` où on implémentera les nouvelles fonctionnalités. Ajoutez à `ParagrapheJoli` et `FormateurJoli` un attribut supplémentaire correspondant à la longueur maximum d'une ligne (i.e. la largeur de la page). Mettez à jour les constructeurs.

Exercice 7
Modifiez la méthode `addString` de `ParagrapheJoli` pour qu'elle passe à une nouvelle ligne si en ajoutant le nouveau mot à la ligne courante lui ferait dépasser la largeur de la page. Cependant, on ne passe jamais à une nouvelle ligne si la ligne courante est vide (pourquoi ?).

Testez en utilisant plusieurs valeurs pour la largeur de la page. Testez en particulier avec un petit nombre (7, par exemple) et aussi avec un nombre "raisonnable" (50, par exemple).

4.2 Indentation

Nous voulons indenter les paragraphes.

Exercice 8
Pour cela on doit introduire les tabulations. Modifiez la classe `Espace` pour qu'elle puisse avoir une longueur modifiable. On aura besoin d'un attribut `size` et d'une méthode `setSize(int)`. Il faudra aussi modifier les méthodes déjà définies. Pour la suite, on considère les tabulations comme des espaces de taille 4.
Pour obtenir une chaîne avec `k` fois un espace, ajoutez `import java.util.Arrays;` comme première ligne, et faites

```
char[] array = new char[k];
Arrays.fill(array, ' ');
return new String(array);
```

Exercice 9
Dans le constructeur de `ParagrapheJoli`, faites lui ajouter une tabulation au début. Modifiez `readParagraphe()` pour que `FromateurJoli` utilise des `ParagrapheJoli`. Vérifiez que les paragraphes sont bien indentés.

4.3 Justification (facultatif)

Le texte produit par notre `FormateurJoli` n'est pas justifié puisque la marge droite n'est pas alignée.

Exercice 10
Ajoutez une méthode `justifier(int longueur)` à la classe `Ligne` pour qu'elle étire les espaces dans la ligne pour que la ligne atteigne la longueur souhaitée. Malheureusement, ce nombre n'est pas toujours constant : si une ligne contient deux espaces et qu'il faut augmenter la longueur de 3, il faudra ajouter deux espaces à la première mais un seul à la seconde.

Exercice 11
A quel endroit dans la classe `ParagrapheJoli` doit-on justifier les lignes ?
Modifiez la classe pour que toutes ses lignes (sauf potentiellement la dernière) aient une largeur uniforme correspondant à l'attribut correspondant, et testez.