Université de Paris **Programmation Orientée Objet**
L2 **Année 2020-2021**

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 Boîte qui représente les objets composant le texte.

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

— une *Boîte 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.

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

— une *Boîte composite* 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 boîtes

Exercice 1
Créez une classe `Boite`, qui contient deux méthodes publiques : `length()`, de type entier, `toString()`, de type String. Par défaut, on considérera que la longueur est 0 et `toString()` renvoie la chaîne vide.

Exercice 2
Écrivez les définitions de deux classes qui héritent la classe `Boite` : `BoiteEspace` et `BoiteMot`. Une `BoiteEspace`

a une longueur de 1, et se convertit en la chaîne réduite à un espace `" "` (à ne pas confondre avec la chaîne vide !). Une `BoiteMot` représente une chaîne arbitraire, sa méthode `toString` retourne cette chaîne, et la méthode `length()` retourne sa longueur.

Exercice 3
Écrivez maintenant la classe `BoiteComposite`. Une boîte composite contient une suite de boîtes (une `LinkedList`) ; sa longueur est la somme des longueurs des boîtes qu'elle contient, et sa représentation sous forme de chaîne est la concaténation des représentations des boîtes qu'elle contient. En théorie, une boîte composite peut contenir une boîte composite même si dans ce TP ce ne sera jamais le cas. Ne pas exclure ce cas ne devrait pas vous poser de problème.
En plus des méthodes héritées de la classe `Boite`, la classe `BoiteComposite` aura une méthode publique `isEmpty` qui détermine si une boîte composite est vide, ainsi qu'une méthode publique `addBoite` qui ajoute une boîte à la fin d'une boîte composite.

3 Formateur de texte très simple

Exercice 4
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). Chaque paragraphe sera une boîte composite contenant des mots séparés par des espaces.
L'autre méthode `print()` affichera ce texte en insérant une ligne vide entre chaque paragraphe. Les attributs de `Formateur` seront un `Scanner sc` et un liste chaînée `LinkedList<BoiteComposite> liste` 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.*` ; et `import java.io.*` ; 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 fic){
sc = null;
try {
    sc = new Scanner(new File(fic));
}
catch(Exception e){
    System.out.println("Erreur lors d'ouverture fichier:");
    e.printStackTrace();
    System.exit(1);
}
liste = new LinkedList<BoiteComposite>();
}
```

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

et de la méthode `next()`.

Il faudra ajouter une boite 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 `BoiteComposite`).
Il vous faudra définir ce que rend `readParagraph()` s'il n'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. Là encore, on peut factoriser en programmant une méthode privée `printParagraphe(BoiteComposite b)` qui imprime le paragraphe mais pas de ligne vide.

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 avec taille de ligne limitée

Exercice 5
On crée une nouvelle classe `FormateurLimite` qui aura un attribut supplémentaire correspondant à la longueur maximum d'une ligne (i.e. la largeur de la page). Ce formateur va stocker les paragraphes sous forme de liste de lignes et on stockera donc une liste de listes de boîtes. On aura donc un type attribut de type `LinkedList<LinkedList<BoiteComposite>>`

Adaptez la classe précédente : Quel type de résultat doit retourner `readParagraphe` ? La méthode `readParagraphe` passe maintenant à une nouvelle boîte composite dès lors qu'ajouter le nouveau mot à la boîte courante lui ferait dépasser la largeur de la page. Cependant, on ne passe jamais à une nouvelle boîte si la boîte 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).

5 Justification (facultatif)

Ce que nous allons faire Le texte produit par notre `FormateurLimite` n'est pas justifié puisque la marge droite n'est pas alignée. Pour justifier le texte, nous introduirons une nouvelle classe `BoiteEtirable` qui hérite de `Boite`. Les objets de `BoiteEtirable` pourront être convertis en des chaînes de longueur plus grande que la longueur des boites, ceci sera fait en insérant des espaces supplémentaires. Cela nous permettra de justifier les lignes.
Il faudra ensuite ajouter une méthode `printJustifie()` à la classe `FormateurLimite` pour qu'elle imprime le texte de manière justifié.

Adaptation des Boîtes

Exercice 6
Commencez par ajouter à la classe `Boite` une nouvelle méthode booléenne `isEtirable` qui retournera false.

Exercice 7
Définissez maintenant une nouvelle classe `BoiteEtirable` qui étend `Boite` en lui ajoutant une méthode `toString(int n)` de type `String`, elle retournera une chaîne vide. Dans les autres classes qui hériteront de `BoiteEtirable` cette nouvelle méthode doit convertir une boîte en une chaîne en ajoutant `n` espaces supplémentaires aux endroits où cela peut se faire.
Quelles sont les classes qui doivent hériter de `BoiteEtirable` ?

Exercice 8
Modifiez maintenant la définition de la classe `BoiteEspace` pour qu'elle hérite de `BoiteEtirable`. Toutes les `BoiteEspaces` sont étirables (la méthode `isEtirable` retourne toujours true), et `toString(n)` retourne simplement une chaîne de `n+1` espaces (l'espace d'origine, et `n` espaces ajoutés).
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
Le cas d'une boîte composite est un peu plus compliqué. Une boîte composite peut être étirée dès qu'une des boîtes qu'elle contient peut l'être : la méthode `isEtirable` devra donc vérifier si c'est le cas.

La méthode `toString(n)` devra ajouter un certain nombre d'espaces à chaque boîte étirable contenue. Malheureusement, ce nombre n'est pas toujours constant : si une boîte composite contient deux boîtes étirables et qu'il faut distribuer trois espaces, il faudra ajouter deux espaces à la première mais un seul à la seconde. (On utilisera le fait que si `n` et `p` sont des entiers, `n/p` fait une division entière et que `n%p` retourne le reste de cette division.)

Impression

Exercice 10
Ecrivez la méthode `printJustifie` de la classe `FormateurLimite` pour qu'elle étire les lignes étirables afin d'arriver à une largeur uniforme correspondant à l'attribut correspondant.

Exercice 11
Le programme précédent a un défaut flagrant : il justifie toutes les lignes, même celles qui sont à la fin d'un paragraphe. Il va donc falloir le modifier pour résoudre ce problème.