Scanners et expressions régulières

1 Scanners

La classe Scanner se charge avec `import java.util.Scanner`. Un scanner permet de lire une entrée et de la découper en sous-parties selon des règles pré-définies. L'entrée peut prendre plusieurs formes, voilà les trois principales :

- Une chaîne de caractères :

```
String input = "1 fish 2 fish red fish blue fish";
Scanner sc = new Scanner(input);
```

- Un fichier :

```
Scanner sc;
try{  /* try catch pour gérer le cas où "mydoc.txt" ne s'ouvre pas */
  File f = new File("mydoc.txt");
  sc = new Scanner(f);
} catch(Exception e) {
  e.printStackTrace();
}
```

- L'entrée clavier : `Scanner sc = new Scanner(System.in);`

Il faut imaginer que le scanner a une tête de lecture initialement placée au tout début de l'entrée. Cette tête peut être déplacée vers la droite uniquement. L'entrée est représentée par une suite d'éléments (les **tokens**) séparés par des **délimiteurs**. Les délimiteurs sont déterminés par une *règle de délimitation* qui doit être spécifiée à l'avance. Par exemple, si on choisi comme règle de délimitation *"toute suite non vide d'espaces"* alors la chaîne

```
String s = "  1 fish 2    fish red  ";
```

contient 6 délimiteurs et 5 tokens. En représentant chaque délimiteur par le symbole |, l'entrée précédente correspond à

```
|1|fish|2|fish|red|
```

Le rôle d'un scanner est de renvoyer successivement les tokens constituant l'entrée. Les délimiteurs ne sont pas retournés (ils servent juste à séparer les tokens). L'obtention des tokens s'effectue via la méthode `next()` de la classe Scanner. Chaque appel à cette méthode retourne le token suivant et déplace la tête de lecture sur le token d'après. Par exemple

```
String s = "  1 fish 2    fish red  ";
Scanner sc = new Scanner(s);
System.out.println(s.next());
System.out.println(s.next());
```

va afficher `1` et `fish` (on a lu les deux premiers tokens). Comme on peut s'y attendre, `next()` provoque une erreur s'il n'y a plus de token disponible. Pour éviter cela, on peut tester l'existence d'un token avec la méthode `hasNext()` (cette méthode ne déplace pas la tête de lecture). Par exemple, pour afficher tous les tokens d'une chaîne de caractères :

```
String s = "  1 fish 2    fish red  ";
Scanner sc = new Scanner(s);

while (sc.hasNext())
  System.out.println(sc.next());
```

Pour fixer une règle de délimitation, il faut utiliser la méthode `useDelimiter()` de la classe Scanner qui prend en entrée une *expression régulière*. Une expression régulière est une manière concise de représenter une règle comme *"toutes les chaînes de caractères non vides constituées uniquement d'espaces"*. Voilà quelques exemples d'expressions regulières :

- `"a+"` : les chaînes de caractères non vides constituées uniquement de `'a'`

- `"ab+"` : les chaînes de caractères commençant par `'a'` suivi d'un ou plusieurs `'b'`
- `" +"` : les chaînes de caractères non vides constituées uniquement d'espaces
- `" *"` : les chaînes de caractères constituées uniquement d'espaces (la chaîne vide est autorisée)
- `"\n"` : les chaînes correspondant à un retour à la ligne

La syntaxe des expressions régulières est beaucoup plus riche que ça (s'entrainer ici : https://regexone.com). Certains caractères spéciaux (voir https://docs.oracle.com/javase/tutorial/essential/regex/pre_char_classes.html) simplifient également le filtrage, par exemple :

- `"A\\d"` : les chaînes de caractères commençant par `'A'` suivi d'un chiffre (`\\d` désigne n'importe quel chiffre)
- `"\\w0"` : les chaînes de caractères constituées de lettres et de chiffres (`\\w`), et finissant par un 0
- `"\\s+"` : les chaînes de caractères non vides constituées d'espaces, retours à la ligne (`\n`), tabulations (`\t`), retour chariot (`\r`), form feed (`\f`) et tabulation verticale (`\x0B`). Cette règle permet de filtrer la plupart des caractères invisibles.

Par défaut, la méthode `next()` utilise `next.useDelimiter("\\s+")` (suites de caractères invisibles). Si on souhaite utiliser comme règle de délimitation "les nombres commençant par 1 et finissant par 0", alors

```
String s = "  1760banana10 apple 18";
Scanner sc = new Scanner(s);
sc.useDelimiter("1\\d*0");

while (sc.hasNext())
  System.out.println(sc.next());
```

va retourner `"  "`, `"banana"`, `" apple 18"`.

2 Exercice 4

Pour appliquer les scanners à l'exercice 4, il faut un délimiteur qui sépare exactement les paragraphes (on veut 1 token = 1 paragraphe). L'expression régulière à utiliser est : `"\n\\s*\n"`. Cette expression désigne toutes les chaînes de caractères qui commencent par un retour à la ligne (`\n`), contiennent une suite de caracatères invisibles (`\\s*`) et finissent par un retour à ligne (`\n`). La méthode `read` s'écrira alors :

```
public void read(){
  sc.useDelimiter("\n\\s*\n");
  while(sc.hasNext()){  /* tant qu'il reste un paragraphe à lire */
    BoiteComposite p = readParagraphe();
    liste.add(p);
}
```

La méthode `readParagraphe()` doit récupérer le paragraphe suivant (grâce à `next`), le découper en mots et les stocker dans une boite composite. Pour cela on utilise un second scanner avec le délimiteur habituel `"\\s+"`.

```
 private BoiteComposite readParagraphe(){
   BoiteComposite paragraphe=new BoiteComposite();
   String para = sc.next();        /* on extrait le paragraphe suivant */
   Scanner s = new Scanner(para);  /* on initialise un nouveau scanner sur ce paragraphe */
   /* s.useDelimiter("\\s+");  pas nécessaire puisque \\s+ est le délimiteur par défaut */

   while(s.hasNext()){
     paragraphe.addBoite(new BoiteMot(s.next()));
     paragraphe.addBoite(new BoiteEspace());
   }

   return paragraphe;
 }
```