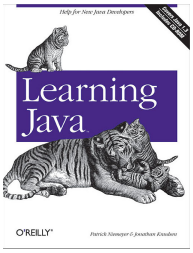
Sokoban : étape 1

1 - Ressources

- La documentation de Java 8 sur le site d'Oracle. Contient, en particulier, la spécification complète de l'API ainsi que des tutoriaux. La partie "Java SE API" sera votre documentation principale pour l'ensemble des TPs.
- 

Le livre : "Learning Java" (troisième édition) de Patrick Niemeyer et Jonathan Knudsen,
édité chez O'Reilly,
ISBN 10: 1-56592-718-4,
ISBN 13: 9781565927186

2 - Premier exercice : utilisation d'une classe de la bibliothèque standard

Dans cette partie, nous allons manipuler une classe de la bibliothèque standard nommée `Scanner`. Cette classe permet de lire des données de différent types sur un flux de caractères accessible en lecture.

1. Découverte :
   Commencez par saisir le programme suivant :

   ```
   import java.util.Scanner;

   class Essai_Scanner {
       public static void main(String [] args) {
           Scanner my_scanner;
           String ligne;

           my_scanner = new Scanner(System.in);
           System.out.println("Saisissez une ligne");
           ligne = my_scanner.nextLine();
           System.out.println("Vous avez saisi la ligne : " + ligne);
       }
   }
   ```

   compilez et exécutez ce code, vous pouvez constater qu'il vous demande simplement de saisir une ligne, puis affiche cette ligne.
2. Compréhension :
   Pour comprendre comment ce programme a été écrit, nous allons consulter la documentation de la classe `Scanner`, cherchez la dans la documentation sur l'API de java référencée ci-dessus. Nous pouvons y trouver:
   - la localisation de la classe dans la bibliothèque standard: `java.util`. Ceci indique que, pour utiliser cette classe, il faut ajouter en début de programme la ligne:

     ```
     import java.util.Scanner;
     ```

     ou encore

     ```
     import java.util.*;
     ```

     qui permet d'utiliser toutes les classes de `java.util`.
   - la description des constructeurs. Dans notre exemple, nous utilisons le constructeur permettant de lire des données à partir du flux d'entrée standard `System.in` (clavier).
   - la description des méthodes. Dans notre exemple, nous utilisons la méthode `nextLine` qui nous renvoie le contenu d'une ligne lue depuis le scanner.

   Vous pouvez aussi constater que les chaînes de caractères se concatènent à l'aide de l'opérateur `+`. A l'aide de ce même opérateur, il est aussi possible de concaténer un entier ou tout autre type d'objet à une chaîne de caractères (l'objet à concaténer sera alors convertit en sa représentation textuelle).
3. Entrée invalide :

exécutez le programme précédent en fermant l'entrée standard (Ctrl-D) avant d'avoir saisi le moindre caractère. Vous pouvez constater que le programme s'arrête brutalement suite à la levée d'une exception. Si vous consultez le détail de la documentation de la méthode `nextLine` vous pouvez voir qu'elle est susceptible de lever une exception de type `NoSuchElementException` (déclarée dans `java.util`). Modifiez votre programme pour qu'il affiche le message `"Aucune ligne saisie"` lorsque la méthode `nextLine` lève une exception.

4. Lire un entier :
   modifiez votre programme pour lire, puis afficher, un entier à la place d'une chaîne. Votre programme devra redemander la saisie de l'entier tant que la saisie de l'utilisateur ne correspond pas à un entier valide. Pour cela, après avoir étudié la documentation de la fonction de lecture d'un entier, vous pourrez utiliser une boucle `while` autour d'une variable de type `boolean` (qui peut prendre l'une des valeurs `true` ou `false`) et vous attraperez l'exception levée par la lecture de l'entier pour détecter une lecture invalide. Attention : `InputMismatchException` est une sous classe de `NoSuchElementException`, il faut donc l'attraper en premier.

2bis - Remarques sur l'ouverture de fichiers lors de l'exécution d'un programme java

Par la suite, vous allez devoir ouvrir des fichiers présents sur le système de fichiers de la machine exécutant votre programme. Comme java se sert du système de fichiers pour retrouver les classes impliquées dans un programme cela requiert une attention particulière :

- lors de l'ouverture d'un fichier, à l'aide de `FileInputStream` ou `FileOutputStream`, le nommage absolu est généralement inadapté (pas portable d'une machine à l'autre), il est donc préférable d'utiliser un nom relatif (exprimé par rapport au répertoire courant). Le répertoire dans lequel vous exécutez votre programme est donc à choisir en conséquence : cela correspond au répertoire courant dans le terminal ou au *working directory* dans la configuration d'exécution d'un IDE ;
- si les fichiers contenant vos classes ne se trouvent pas au même endroit que vos fichiers de données, cela ne pose pas de problème, il suffit d'indiquer à java à quel endroit ils se trouvent : via la variable d'environnement CLASSPATH ou l'option -cp de la command `java` dans le terminal, ou dans les réglages nommés *Build path* dans votre IDE.

3 - Lire et écrire un niveau

Dans cette partie, nous allons lire la description textuelle de plusieurs niveaux dans le jeu du Sokoban. Pour décrire un niveau, nous avons choisi de structurer les données selon un format largement utilisé pour le Sokoban et documenté ici (partie `Level` uniquement). Nous vous fournissons ce fichier de niveaux qu'il est possible de récupérer sur ce site qui contient beaucoup d'autres fichiers de niveaux pour le Sokoban.

Dans le jeu du Sokoban, un niveau est constitué d'une grille rectangulaire dont les dimensions sont variables d'un niveau à l'autre. Dans cette grille, chaque case peut contenir un ou plusieurs éléments comme un mur, un but, une caisse ou un pousseur. Les deux seuls cas dans lesquels une case peut contenir plusieurs éléments sont ceux :

- d'une caisse qui peut se trouver sur un but ;
- du pousseur qui peut se trouver sur un but.

Dans l'ensemble d'un niveau il est censé se trouver exactement un pousseur et autant de caisses que de buts.

Pour cette partie, nous vous proposons d'écrire une classe `Niveau` dotée des méthodes suivantes :

- `void fixeNom(String s)` : permet de fixer le nom du niveau ;
- `void videCase(int i, int j)` : supprime le contenu de la case à la ligne `i` et à la colonne `j` ;
- `void ajouteMur(int i, int j)`, `void ajoutePousseur(int i, int j)`, `void ajouteCaisse(int i, int j)` et `void ajouteBut(int i, int j)` : ajoutent respectivement un mur, un pousseur, une caisse ou un but à la case à la ligne `i` et à la colonne `j` ;

- `int lignes(), int colonnes(), String nom()` : renvoient respectivement le nombre lignes, le nombre de colonnes ou le nom du niveau ;
- `boolean estVide(int l, int c)` : renvoie vrai si la case à la ligne `i` et à la colonne `j` est vide ;
- `boolean aMur(int l, int c), boolean aBut(int l, int c), boolean aPousseur(int l, int c), boolean aCaisse(int l, int c)` : renvoient vrai si la case à la ligne `i` et à la colonne `j` contient respectivement un mur, un but, un pousseur ou une caisse.

Commencez par créer la classe `Niveau` comme une coquille vide, avec des méthodes ne faisant rien et vous complèterez ces méthodes au fil de l'avancée de l'apnée. Pour cette classe, vous aurez sans doute besoin d'utiliser des tableaux à plusieurs dimensions. Pour cela, rien de plus simple, il suffit d'ajouter autant de paires de crochets que de dimensions, par exemple :

- déclaration d'une référence à un tableau à deux dimensions d'entiers : `int[][] monTableau;`
- allocation du tableau :`monTableau = new int[42][42];`

Dans un tableau à deux dimensions, on peut accéder à une ligne entière en ne déréférençant qu'une seule dimension. La ligne est alors un tableau à une dimension avec ses attributs (dont `length`). Évidemment, ce principe est valable quelque soit le nombre de dimensions.

Pour créer nos niveaux, nous vous proposons d'écrire une classe `LecteurNiveaux` qui, étant donné un flux d'entrée (`InputStream`), renvoie un niveau lu à chaque appel de la méthode `lisProchainNiveau` ou `null` si la fin du flux a été atteinte. Sur le flux d'entrée, le niveau est décrit textuellement, ligne par ligne. Chaque ligne contient un caractère par case du niveau sur la ligne correspondante, selon le format de niveau cité précédemment (ATTENTION : pour ce projet, nous nous limitons à la partie nommée `Level` décrite dans ce format, pas de représentation de solutions ni de RLE). Nous ajoutons à ce format les conventions suivantes :

- deux niveaux successifs sont séparés par une ligne vide ;
- un caractère `;` débute un commentaire qui se termine à la fin de la ligne ;
- nous prendrons, par convention, le dernier commentaire rencontré comme nom de niveau.

Renseignez-vous sur la classe `String` (en particulier les méthodes `length, charAt` ou `substring` qui pourront vous être utiles ici). Ecrivez cette classe petit à petit en testant au fur et à mesure. Bien entendu, pour créer le niveau, cette classe pourra (devra) utiliser tout ce qui a été présenté sur la classe `Niveau`.

Pour que votre programme produise une sortie, et pour avoir la possibilité d'écrire sous forme textuelle nos niveaux, nous vous proposons d'écrire une classe `RedacteurNiveau` qui, étant donné un flux de sortie (`OutputStream`), à chaque appel de la méthode `ecrisNiveau`, écris le niveau passé en paramètre sur le flux de sortie donné. Pour écrire cette classe, renseignez-vous sur `PrintStream`.

Enfin, pour terminer, écrivez un programme principal qui ouvre le fichier de niveaux donné, charge chacun des niveaux qu'il contient en mémoire et, pour chaque niveau chargé, affiche le niveau sur la sortie standard. Si vous vous êtes bien débrouillés, la sortie de ce programme devrait être identique au contenu du fichier de niveaux donné.