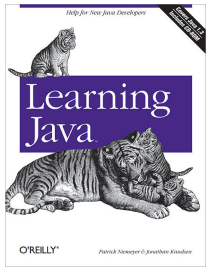
Apnée de Programmation numéro 1

Introduction au langage JAVA

1 - Ressources

- La documentation de Java 8 sur le site d'Oracle contient en particulier la spécification complète de l'API et des tutoriaux. La partie "Java SE API" sera votre référence documentaire 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 - Exercice 1 : 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 donnée à partir du flux d'entrée `System.in` (clavier).
   - la description des méthodes. Dans notre exemple nous utilisons la méthode `nextLine` qui nous renvoie le contenue 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 (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 constater qu'elle est susceptible de lever une exception de type `NoSuchElementException` (déclaré 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 chaine. 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.
5. Lire une structure de données
   récupérez l'archive Apnee_Intro.zip et décompressez la. Dans le répertoire `src` créé, vous trouverez l'implementation de la classe `BriqueInitiale` qui modélise une brique dans un jeu de casse briques (et qui s'accompagne de l'interface `ComposantInitial`). Vous pouvez constater qu'une brique contient comme information sa resistance (un entiers) ainsi que sa position (un `Point` constitué de deux nombres à virgule flottante, ses coordonnées cartésiennes). Utilisez un Scanner pour lire ces informations au clavier, créer une brique correspondante et l'afficher. Vous pouvez constater que la méthode

`System.out.println` sait afficher une brique ! Cela vient du fait que cette méthode sait afficher tout objet qui définit la méthode `toString` qui est alors utilisée pour fabriquer la représentation textuelle de l'objet à afficher.

3 - Lire un niveau

Dans cette partie, nous allons lire la description textuelle des objets constituant un niveau dans un jeu de casse briques, à savoir briques, bonus et leur position dans le niveau. Pour décrire un niveau, nous avons choisi de structurer les données selon le format Yaml qui a l'avantage d'être très lisible. Dans ce format, les données sont organisées hiérarchiquement (de manière analogue à l'organisation dans d'autres formats comme Json ou xml) et c'est l'indentation qui délimite les différents niveaux de la hiérarchie. Nous n'utiliserons que les tables d'association `clé:valeur`, qui sont une des structures de données supportées par Yaml ainsi que le caractère `|` qui permet de marquer le début d'un bloc de texte brut utilisé comme valeur (en conservant les retours charriots).

Une section est constituée d'un nom et d'une table d'association (qui joue ici le rôle de valeur). Un fichier décrivant un niveau est composé des deux sections suivantes, présentes dans cet ordre :

- la section `Composants`, qui décrit l'ensemble des modèles de composants graphiques présents dans le niveau. La valeur de cette section est une table d'association qui associe la description d'un composant à un caractère. Un composant est lui-même décrit par une table d'association contenant une partie des entrées suivantes :
  - `type` : donne le type de composant. Pour nous, cela sera `brique`, `bonus` ou `raquette`, doit être la première entrée
  - `resistance` : uniquement dans le cas d'un composant de type `brique`, donne la resistance de la brique (0 pour l'infini)
  - `nature` : uniquement dans le cas d'un composant de type `bonus`, correspond au type de bonus
- la section `Couches`, qui décrit l'organisation spatiale du niveau. Un niveau est constitué de couches de composants, qui n'ont pas d'existence dans le jeu mais qui sont destinées à simplifier l'écriture du fichier de description d'un niveau. Ainsi nous pouvons avoir autant de composants que nous le souhaitons au même endroit dans le niveau, il suffit de les placer dans des couches distinctes. Dans la suite, par convention, la couche 0 sera réservée aux bonus, la couche 1 aux briques, aux bords, à la (aux) balle(s) ainsi qu'à la raquette. Notre section `Couches` contient des entrées associant à un numéro de couche une description du contenu de la couche sous forme d'un bloc de texte avec retour chariots dans lequel :
  - une ligne de `.` est à ignorer et correspond à une bordure supérieure ou inférieure
  - une colonne de `.` est à ignorer et correspond à une bordure gauche ou droite (et sert aussi à distinguer le contenu du niveau de l'indentation)
  - une ligne de # correspond à une bordure supérieure ou inférieure concrétisée par une brique spéciale cachée en dehors du niveau
  - une colonne de # correspond à une bordure gauche ou droite concrétisée là encore par une brique spéciale cachée en dehors du niveau (un seul par colonne)
  - un caractère espace correspond à une position dans la couche ne contenant aucun composant
  - un autre caractère correspond à un composant décrit dans la section `Composants` dont une copie se trouve à la position courante

  Dans cette description, les coordonnées sont exprimées dans un espace logique démarrant en (0,0) et tel que :
  - tout caractère d'une ligne autre que `.` ou # correspond à un déplacement de (1,0)
  - tout changement de ligne correspond à un déplacement de (0,1)

Dans le répertoire `rsc` de l'archive, le fichier `Niveaux/Niveau-1.txt` contient une description dans ce format, tout comme le fichier `Niveaux/Niveau-2.txt`.

Votre travail est d'écrire une classe `ChargeurNiveaux` contenant :

- un constructeur qui accepte comme arguments formels un `InputStream` (depuis lequel un niveau sera lu) et une `FabriqueComposantsInitiale` (permettant de fabriquer les composants du niveau).
- une méthode :
  `void lisNiveau(Niveau n)`
  qui prend en argument un niveau et le peuple des composants décrits par le flux d'entrée passé au constructeur. Les composants décrits dans la section `Composants` doivent être créés par la méthode apprprié de la `FabriqueComposantsInitiale`. Les composants placés dans les couches de la section `Couches` doivent être créés par la méthode `copie` de la fabrique en lui fournissant le modèle créé dans la section composants, puis ajouté au niveau via la méthode `ajouteComposant`.

Attention, il ne faudra créer qu'un instance de chaque bord (horizontal ou vertical) malgré le fait qu'une ligne ou une colonne de caractères `#` est lue. A la fin de la lecture de chaque couche, la méthode `fixerDimensionsMax` du niveau en cours de lecture devra être appelée en lui passant les coordonnées logiques maximales atteintes lors de la lecture Après tous les points précédents, la méthode `nouvelleBalle` devra être appelée. Pour analyser la description textuelle du niveau, vous pourrez vous servir des méthodes de la classe `String` comme `charAt`, `split`, `trim`, `substring`, `equals` ou encore de la méthode `Integer.parseInt`.

ATTENTION : dans l'archive fournie vous disposez du programme `Etape1` qui se trouve, comme les autres, dans le répertoire `src`. Celui-ci lit les niveaux décrits dans le répertoire `rsc` à l'aide d'un chargeur qui regarde dans les répertoires de chargement des classes. Ceci est fait de cette manière afin de pouvoir fonctionner aussi bien avec le système de fichiers de la machine hôte qu'avec une archive exécutable java (fichier `.jar`) que nous étudierons ultérieurement. Pour l'heure, afin de faire fonctionner `Etape1`, il faut que la machine virtuelle java cherche ses classe à la fois dans `src` et dans `rsc`. En vous plaçant dans `Apnee_Intro` executez :

```
export CLASSPATH=src:rsc
```

Vous devriez alors pouvoir exécuter `Etape1` qui devrait trouver les niveaux sans problèmes :

```
java Etape1
```

Le programme `Etape1` doit, pour les deux fichiers de niveau donnés, afficher :

```
Niveau : Niveau 1
Bonus en (16.0, 2.0), nom Aleatoire
Bonus en (16.0, 3.0), nom Colle
Bonus en (16.0, 4.0), nom Aleatoire
Bonus en (16.0, 5.0), nom Colle
Bonus en (16.0, 6.0), nom Aleatoire
Bonus en (16.0, 7.0), nom Colle
```

```
Bonus en (4.0, 8.0), nom Multiballes
Bonus en (8.0, 8.0), nom Laser
Bonus en (12.0, 8.0), nom Multiballes
Bonus en (16.0, 8.0), nom Aleatoire
Bonus en (20.0, 8.0), nom Multiballes
Bonus en (24.0, 8.0), nom Laser
Bonus en (28.0, 8.0), nom Multiballes
Brique en (4.0, 2.0), resistance 1
Brique en (6.0, 2.0), resistance 1
Brique en (8.0, 2.0), resistance 0
Brique en (10.0, 2.0), resistance 0
Brique en (12.0, 2.0), resistance 1
Brique en (14.0, 2.0), resistance 1
Brique en (16.0, 2.0), resistance 1
Brique en (18.0, 2.0), resistance 1
Brique en (20.0, 2.0), resistance 1
Brique en (22.0, 2.0), resistance 0
Brique en (24.0, 2.0), resistance 0
Brique en (26.0, 2.0), resistance 1
Brique en (28.0, 2.0), resistance 1
Brique en (2.0, 3.0), resistance 1
Brique en (4.0, 3.0), resistance 1
Brique en (6.0, 3.0), resistance 1
Brique en (8.0, 3.0), resistance 1
Brique en (10.0, 3.0), resistance 1
Brique en (12.0, 3.0), resistance 1
Brique en (14.0, 3.0), resistance 1
Brique en (16.0, 3.0), resistance 1
Brique en (18.0, 3.0), resistance 1
Brique en (20.0, 3.0), resistance 1
Brique en (22.0, 3.0), resistance 1
Brique en (24.0, 3.0), resistance 1
Brique en (26.0, 3.0), resistance 1
Brique en (28.0, 3.0), resistance 1
Brique en (30.0, 3.0), resistance 1
Brique en (2.0, 4.0), resistance 1
Brique en (4.0, 4.0), resistance 1
Brique en (6.0, 4.0), resistance 1
Brique en (8.0, 4.0), resistance 2
Brique en (10.0, 4.0), resistance 2
Brique en (12.0, 4.0), resistance 2
Brique en (14.0, 4.0), resistance 2
Brique en (16.0, 4.0), resistance 1
Brique en (18.0, 4.0), resistance 2
Brique en (20.0, 4.0), resistance 2
```

```
Brique en (22.0, 4.0), resistance 2
Brique en (24.0, 4.0), resistance 2
Brique en (26.0, 4.0), resistance 1
Brique en (28.0, 4.0), resistance 1
Brique en (30.0, 4.0), resistance 1
Brique en (2.0, 5.0), resistance 1
Brique en (4.0, 5.0), resistance 1
Brique en (6.0, 5.0), resistance 1
Brique en (8.0, 5.0), resistance 1
Brique en (14.0, 5.0), resistance 1
Brique en (16.0, 5.0), resistance 1
Brique en (18.0, 5.0), resistance 1
Brique en (24.0, 5.0), resistance 1
Brique en (26.0, 5.0), resistance 1
Brique en (28.0, 5.0), resistance 1
Brique en (30.0, 5.0), resistance 1
Brique en (2.0, 6.0), resistance 1
Brique en (4.0, 6.0), resistance 1
Brique en (6.0, 6.0), resistance 1
Brique en (14.0, 6.0), resistance 1
Brique en (16.0, 6.0), resistance 1
Brique en (18.0, 6.0), resistance 1
Brique en (26.0, 6.0), resistance 1
Brique en (28.0, 6.0), resistance 1
Brique en (30.0, 6.0), resistance 1
Brique en (4.0, 7.0), resistance 1
Brique en (6.0, 7.0), resistance 1
Brique en (8.0, 7.0), resistance 1
Brique en (10.0, 7.0), resistance 1
Brique en (12.0, 7.0), resistance 1
Brique en (14.0, 7.0), resistance 1
Brique en (16.0, 7.0), resistance 1
Brique en (18.0, 7.0), resistance 1
Brique en (20.0, 7.0), resistance 1
Brique en (22.0, 7.0), resistance 1
Brique en (24.0, 7.0), resistance 1
Brique en (26.0, 7.0), resistance 1
Brique en (28.0, 7.0), resistance 1
Brique en (4.0, 8.0), resistance 1
Brique en (8.0, 8.0), resistance 1
Brique en (12.0, 8.0), resistance 1
Brique en (16.0, 8.0), resistance 1
Brique en (20.0, 8.0), resistance 1
Brique en (24.0, 8.0), resistance 1
Brique en (28.0, 8.0), resistance 1
```

```
Raquette en (14.0, 23.0)
Brique en (-1.0, 0.0), resistance 0
Brique en (34.0, 0.0), resistance 0
Brique en (0.0, -1.0), resistance 0
Le niveau a une largeur de 34.0 et une hauteur de 25.0
Prêt à jouer, en attente de balle !
Niveau : Niveau 2
Bonus en (10.0, 4.0), nom Elargit
Bonus en (12.0, 4.0), nom Elargit
Bonus en (26.0, 4.0), nom Elargit
Bonus en (28.0, 4.0), nom Elargit
Bonus en (14.0, 5.0), nom Laser
Bonus en (16.0, 5.0), nom Laser
Bonus en (18.0, 5.0), nom Laser
Bonus en (20.0, 5.0), nom Laser
Bonus en (22.0, 5.0), nom Laser
Bonus en (24.0, 5.0), nom Laser
Bonus en (10.0, 6.0), nom Elargit
Bonus en (12.0, 6.0), nom Elargit
Bonus en (26.0, 6.0), nom Elargit
Bonus en (28.0, 6.0), nom Elargit
Brique en (2.0, 2.0), resistance 1
Brique en (4.0, 2.0), resistance 1
Brique en (34.0, 2.0), resistance 1
Brique en (36.0, 2.0), resistance 1
Brique en (6.0, 3.0), resistance 1
Brique en (8.0, 3.0), resistance 1
Brique en (30.0, 3.0), resistance 1
Brique en (32.0, 3.0), resistance 1
Brique en (10.0, 4.0), resistance 1
Brique en (12.0, 4.0), resistance 1
Brique en (26.0, 4.0), resistance 1
Brique en (28.0, 4.0), resistance 1
Brique en (14.0, 5.0), resistance 2
Brique en (16.0, 5.0), resistance 2
Brique en (18.0, 5.0), resistance 2
Brique en (20.0, 5.0), resistance 2
Brique en (22.0, 5.0), resistance 2
Brique en (24.0, 5.0), resistance 2
Brique en (10.0, 6.0), resistance 1
Brique en (12.0, 6.0), resistance 1
Brique en (26.0, 6.0), resistance 1
Brique en (28.0, 6.0), resistance 1
Brique en (6.0, 7.0), resistance 1
Brique en (8.0, 7.0), resistance 1
```

```
Brique en (30.0, 7.0), resistance 1
Brique en (32.0, 7.0), resistance 1
Brique en (2.0, 8.0), resistance 1
Brique en (4.0, 8.0), resistance 1
Brique en (34.0, 8.0), resistance 1
Brique en (36.0, 8.0), resistance 1
Raquette en (7.0, 28.0)
Brique en (-1.0, 0.0), resistance 0
Brique en (40.0, 0.0), resistance 0
Brique en (0.0, -1.0), resistance 0
Le niveau a une largeur de 40.0 et une hauteur de 30.0
Prêt à jouer, en attente de balle !
Exception in thread "main" java.lang.RuntimeException: Niveaux/Niveau-3.txt introuvable
        at JeuInitial.prochainNiveau(JeuInitial.java:29)
        at JeuInitial.rafraichit(JeuInitial.java:34)
        at Etape1.main(Etape1.java:31)
```