Thème 1 - TD

Objectifs

— Retour sur les exceptions
— Retour sur les entrées/sorties

Exercices

Exercice 1 – Lecture de données dans un fichier ASCII

Nous souhaitons écrire une méthode **`public void`** `initFromFile(String fichier)` permettant de lire des valeurs entières dans un fichier ASCII, dont le contenu se présente sous la forme suivante :

```
5
4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
```

Les deux premières lignes donnent respectivement le nombre de lignes de données dans le fichier et le nombre de valeurs qu'elles contiennent.

Voici les premières instructions que nous utilisons dans la méthode de lecture :

```
BufferedReader in;
in = new BufferedReader (
         new FileReader ( new File(fichier) ));
```

Question 1
Expliquez les différents éléments de l'initialisation de la variable `in`.

Question 2
Comment s'effectue la fermeture de ce flux ?

Question 3
Nous souhaitons stocker dans un tableau l'ensemble des valeurs lues sur une ligne. Nous disposons pour cela des méthodes :

- `String readLine()` dans la classe `BufferedReader`, qui lit une ligne de texte ;
- `String[] split(String regex)` dans la classe `String`, qui découpe la chaîne en utilisant *regex* comme séparateur.

Écrivez le code qui permet de stocker dans un tableau `valeurs` les données contenues dans la première ligne du fichier.

Question 4
Que manque-t-il pour pouvoir utiliser les données stockées dans le tableau afin de calculer, par exemple, une somme ?

Question 5
De quelle autre solution dispose-t-on pour lire les données du fichier comme une suite d'entiers ?

Exercice 2 – Saisie contrôlée de données

Nous souhaitons écrire un programme permettant d'effectuer une saisie contrôlée de données : l'utilisateur dispose de plusieurs tentatives pour saisir une donnée correcte. La donnée à saisir est un code PIN composé de 4 chiffres. Nous disposons pour cela de la classe `CodePin` dont le code est donné ci-dessous.

La méthode `saisirCode` lève une exception `NumberFormatException` dès qu'un caractère saisi n'est pas un chiffre. Si tous les caractères saisis sont des chiffres (ou si aucun caractère n'est saisi), elle peut aussi lever une exception `BadValueException` (c'est une classe que nous définissons) si le code ne contient pas exactement 4 caractères.

Avant de convertir la chaîne saisie en valeur entière, la méthode élimine tous les zéros en début de chaîne. En effet, si la méthode `decode` rencontre un `0` en début de chaîne, elle interprète la donnée comme une valeur octale (ou hexadécimale si le `0` est suivi d'un `x`).

```
import java.io.Console;
import java.lang.NumberFormatException;

public class CodePin {
   private static final int TAILLE_PIN = 4;
   private int val;

   public CodePin (int val) {
      this.val = val;
   }

   public int saisirCode(Console c) throws BadValueException {
      int essai, taille, i = 0;

      String code = new String(c.readPassword("Saisissez " + TAILLE_PIN + " chiffres :"));

      taille = code.length();
      if (taille == 0) {   // saisie vide
         throw new BadValueException();
      }
      while ( (i < TAILLE_PIN) && (code.charAt(i) == '0')) {
         i++;
      }

      if (i == TAILLE_PIN) {
         essai = 0;
      }
      else {
         essai = Integer.decode(code.substring(i));
         if (taille != TAILLE_PIN) {
            throw new BadValueException();
         }
      }
      return essai;
    }

    public boolean checkOK (int code) {
      return (code == val);
    }
}
```

Question 1
Pourquoi précise-t-on que la méthode `saisirCode` peut lancer une `BadValueException`, alors qu'on ne le fait pas pour `NumberFormatException` ?

Pourquoi choisit-on de ne pas rattraper ces exceptions directement dans la méthode ?

Question 2
Nous considérons pour l'instant que l'utilisateur ne fait que des saisies valides, i.e., des chaînes de 4 chiffres. Écrivez une classe `TestCodePin` dont la méthode `main` demande à l'utilisateur la saisie d'un code. Si le code correct n'a pas été saisi après trois tentatives, le programme affiche le message *"carte bloquée"*, sinon le message *"retrait autorisé"* est affiché.

Nous utiliserons la méthode `System.console()` qui permet d'accéder à l'unique instance de la classe `Console`, si elle existe (ce qui est le cas si la machine virtuelle est lancée depuis la ligne de commande, sans redirection des entrées/sorties).

Question 3
Que se passe-t-il s'il n'y a pas de console associée à la machine virtuelle ? Pourquoi ne précise-t-on pas que cette exception est levée par le programme ?

Question 4
Complétez la méthode `main` pour traiter le problème de la console, et pour qu'une saisie non conforme (un code qui n'est pas composé de 4 chiffres) ne soit pas comptabilisée dans les 3 tentatives de l'utilisateur.

Question 5
Modifiez la méthode `main` pour qu'un code comportant un caractère qui n'est pas un chiffre ne soit pas compté dans les tentatives. Tout autre code (donc une suite de chiffres de taille quelconque, éventuellement vide) doit être compté.

Question 6
Que faudrait-il faire pour qu'un code comportant des caractères qui ne sont pas des chiffres provoque la terminaison du programme, sans modifier le traitement des codes composés de chiffres ?