TP8 : Exceptions

Objectifs

- Exceptions
- Lancement (throw), capture (try-catch) et propagation (throws) d'exceptions prédéfinies
- Lancement, capture et propagation d'exceptions créées par le programmeur
- Passage de paramètres sur la ligne de commande

Exercices

Exercice 68 – throw, throws, finally

Q 68.1 Donnez l'affichage produit par le programme ci-après. Expliquez les résultats.

```
1 public class MonException extends Exception {
2     public MonException(String s) {
3         super(s);
4         System.out.println("\n.MonException.:.constructeur");
5     }
6 }
7
8 public class TestFinally {
9     /** Exception deleguee a la methode appelante (ici main).*/
10    public static void test1() throws MonException {
11        if (true) throw new MonException("lancee.dans.test1");
12        System.out.println("test1.:.fin.de.la.methode");
13    }
14
15 /** Exception capturee (et pas deleguee) dans la methode test2 */
16 public static void test2() {
17    try {
18        if (true) throw new MonException("lancee.dans.test2");
19    } catch (MonException e) {
20        System.out.println("test2.:.capture.de.l'exception.:."+e);
21    }
22    System.out.println("test2.:.fin.de.la.methode");
23 }
24
25 /** Exception capturee (et pas deleguee) dans la methode test3 avec finally */
26 public static void test3() {
27    try {
28        if (true) throw new MonException("lancee.dans.test3");
29    } catch (MonException e) {
30        System.out.println("test3.:.capture.de.l'exception.:."+e);
31    } finally {
32        System.out.println("test3.:.finally.est.effectue");
33    }
34    System.out.println("test3.:.fin.de.la.methode");
35 }
36
37 /** Exception deleguee a la methode appelante (ici main) avec finally */
38 public static void test4() throws MonException {
39    try {
40        if (true)
41            throw new MonException("lancee.dans.test4");
42    } finally {
```

```
43          System.out.println("test4.::.finally.est.effectue");
44      }
45      System.out.println("test4.::.fin.de.la.methode");
46  }
47
48  /** Meme cas que le test4, mais ici l'exception n'est pas levee */
49  public static void test5() throws MonException {
50      try {
51          if (false) throw new MonException("lancee.dans.test5");
52      } finally {
53          System.out.println("test5.::.finally.est.effectue");
54      }
55      System.out.println("test5.::.fin.de.la.methode");
56  }
57
58  public static void main(String [] args){
59      try {
60          test1();
61      } catch (MonException e) {
62          System.out.println("main.::.test1.::.capture.de.l'exception."+e);
63      }
64      test2();
65      test3();
66      try {
67          test4();
68      } catch (MonException e) {
69          System.out.println("main.::.test4.::.capture.de.l'exception."+e);
70      }
71      System.out.println();
72      try {
73          test5();
74      } catch (MonException e) {
75          System.out.println("main.::.test5.::.capture.de.l'exception."+e);
76      }
77      System.out.println("Fin.du.programme");
78  }
79  }
```

Exercice 69 – MonTableau

Le but de l'exercice est de définir une classe MonTableau, gérant des tableaux ayant une longueur maximum fixée pour tous les éléments de la classe, et qui se prémunisse des dépassements de capacité de ses objets.

Q 69.1 Définir une classe MonTableau qui possède les variables tab (tableau d'entiers) et lgReelle (entier) donnant le nombre de cases de tab réellement utilisées dans le tableau. Au départ, lgReelle vaut 0. Ecrire un constructeur prenant en paramètre la taille du tableau, et une méthode ajouter(int n) qui ajoute la valeur *n* à la suite du tableau sans vérifier s'il reste de la place.

Q 69.2 Ecrire la méthode main qui crée un objet MonTableau de 3 cases et y ajoute 10 entiers. Exécutez le programme. Que se passe-t-il ?

Q 69.3 Capturer dans la méthode main l'exception précédemment levée, et afficher le texte "Depassement des bornes a la position 3" en utilisant la méthode getMessage() de l'exception levée.

Q 69.4 Définir un nouveau type d'exception appelée TabPleinException.

Q 69.5 Modifier la méthode ajouter pour lever cette exception quand le tableau est plein. Capturer cette exception dans la méthode main. Que retourne les méthodes getMessage() et toString() de cette exception ?

Exercice 70 – Classe Etudiant

On veut écrire une classe Etudiant dont les instances décrivent un étudiant ayant un nom et une liste de notes entières (au maximum 5 notes) implantée par un tableau.
Rappel de cours : toute instance de la classe Exception doit obligatoirement être attrapée ou signalée comme étant propagée par toute méthode susceptible de la lever.

Q 70.1 Écrire la classe Etudiant correspondant à la description ci-dessus avec un constructeur à un paramètre, le nom. La méthode toString() rend le nom de l'étudiant suivi de ses notes.

Q 70.2 Ajouter la méthode void entrerNote(int note) qui entre la note dans la liste des notes de cet étudiant. Elle lèvera une exception TabNotesPleinException (à définir) dans le cas où le tableau de notes de cet étudiant serait plein. Cette exception sera capturée dans le main.

Q 70.3 En supposant que la classe qui contient le main s'appelle TestEtudiants, on veut passer sur la ligne de commande une liste d'étudiants avec leurs notes, par exemple :

java TestEtudiants Anna 12 13 7 15 Tom Arthur 9 12 15 0 13 12 Karim 15 8 11 12
10 Melissa 12 6 18 10 12 6

On supposera que chaque donnée est correcte (pas de mélange entre lettres et chiffres), et que la première donnée est un nom.
Ces données sont de deux types : chaîne de caractères et entier. On va utiliser le fait qu'un entier ne fait pas lever d'exception à la méthode Integer.parseInt alors qu'une chaîne de caractères lui fait lever l'exception NumberFormatException.
Rappel : la méthode int Integer.parseInt(String s) rend l'entier représenté par la chaîne s, ou bien lève une exception NumberFormatException si la chaîne s ne repésente pas un entier.
Ecrire le code du main qui récupère les données et affiche pour chacune "c'est une note" ou bien "c'est un nom" suivant le cas. On utilisera obligatoirement le mécanisme d'exception pour ce faire.
Voici une exécution possible :

>java TestEtudiants Anna 12 13 7 15 Tom Arthur 9 12 15 0 13 12
Anna c'est un nom,
1 c'est une note, 13 c'est une note, 7 c'est une note, 15 c'est une note,
Tom c'est un nom,
Arthur c'est un nom,
9 c'est une note, 12 c'est une note, 15 c'est une note, 0 c'est une note, 13 c'est une note, 12 c'est une

Q 70.4 On souhaite gérer dans la classe Etudiant une liste au sens ArrayList d'étudiants. Une liste d'étudiants ne dépend pas d'un étudiant en particulier. Qu'en concluez-vous sur le type de variables que doit être la liste d'étudiants ? Ajouter les instructions nécessaires dans la classe Etudiant.

Q 70.5 Enrichir/modifier le code précédent pour qu'il traite les données de la façon suivante :
– si c'est une chaîne de caractères, il crée une nouvelle instance d'étudiant portant ce nom.
– si c'est une note, il ajoute cette note à la liste des notes de l'étudiant créé précédemment, puis affiche la liste des étudiants. On pensera à traiter les différentes exceptions levées (on rappelle qu'un étudiant a au maximum 5 notes).
Voici une exécution possible :

>java TestEtudiants Anna 12 13 7 15 Tom Arthur 9 12 15 0 13 12 Karim 15 8 11 12 10
Melissa 12 6 18 10 12 6
le tableau de notes de l'etudiant Arthur est plein
le tableau de notes de l'etudiant Melissa est plein
les 5 etudiants :
[Anna 12 13 7 15, Tom, Arthur 9 12 15 0 13, Karim 15 8 11 12 10, Melissa 12 6 18 10 12]