Correction TD n°7

Java Avancé

—M1 Apprentissage—

Exceptions

▸ Exercice 1. MyFIFO

Le but de l'exercice est d'écrire une structure de données FIFO (First In First Out). Pour cela, vous allez utiliser un tableau d'objets (de taille fixée) de manière circulaire, i.e., avec un indice de début et un de indice fin. Le premier indice donne le premier élément pouvant être enlevé, le second la première case libre. En cas d'ajout, l'indice fin est incrémenté, en cas de retrait, l'indice début est incrémenté. Si la fin de du tableau est atteinte, on retourne au début du tableau de manière circulaire. Faites un dessin si vous n'avez pas compris.

1. Créer une classe `MyFIFO`, avec un constructeur prenant en paramètre la taille max fixée de la file. Le constructeur peut lever une exception.
2. Écrire une méthode `offer` prenant un objet en paramètre et l'ajoutant dans la file. Attention aux préconditions. On ne veut pas pouvoir ajouter `null` dans la file. Comment vérifier que la file est pleine (attention, à ne pas confondre avec une file vide...). Que faire alors ?
3. Ecrire une méthode `poll` retournant et supprimant le premier élément de la file. Encore une fois, attention aux files vides. Attention également aux fuites de mémoires possible (le GC libère les objets sans références sur eux, est-ce le cas pour vous ?).
4. Tester dans un main (cas file vide, cas file pleine, etc).
5. Ajouter une méthode indiquant la taille de la file.
6. Ajouter une méthode d'affichage dans l'ordre qu'aurait `poll`. Attention aux différences dans le cas de files vides ou pleines. On veut un affichage à la java, avec des crochets avant et après, et les éléments séparés par des virgules.
7. Testez avec ces tests JUnit MyFIFOTest

Exercice 2 [Utilisation des Exceptions]
La méthode `parseInt` est spécifiée ainsi :

```
public static intparseInt(Strings)
                  throws NumberFormatException

    Parses the string argument as a signed decimal integer.
    The characters in the string must all be decimal digits,
    except that the first character may be an ASCII minus
    sign '-' ('\u002D') to indicate a negative value. The
    resulting integer value is returned, exactly as if the
    argument and the radix 10 were given as arguments to the
    parseInt(java.lang.String, int) method.

    Parameters:
        s - a String containing the intrepresentation to be parsed
    Returns:
        the integer value represented by the argument in decimal.
    Throws:
        NumberFormatException - if the string does not contain a parsable integer.
```

Utilisez cette méthode pour faire la somme de tous les entiers donnés en argument de la ligne de commande, les autres arguments étant ignorés.

Exercice 3 [Création des Exceptions]
Écrire une classe `Entreprise`. Une entreprise a un nombre d'employés, un capital, un nom, une mission, et une méthode `public String mission()` qui renvoie la mission de l'entreprise et qui déclare le lancement de l'exception `SecretMissionException`. On aura également une méthode `public int capital()` qui renvoie le capital et qui déclare le lancement de l'exception `NonProfitException`.
Écrire une classe `EntrepriseSecrete` qui hérite d'`Entreprise` et dont la méthode `mission` lance l'exception `SecretMissionException`. Écrire une classe `EntrepriseSansProfit` qui hérite d'`Entreprise` et dont la méthode `capital` lance l'exception `NonProfitException`.
Écrire une méthode qui prend en entrée un tableau d'entreprises et affiche la mission et le capital de toutes les entreprises (quand cela est possible). Tester la méthode sur les entreprises "Ford", "CIA", "Spectre", "CroixRouge", "Microsoft", "ParisDiderot".

Exercice 4 [Utilisation des exceptions dans les constructeurs]
`Toutou` est une classe avec deux propriétés privées `String nom` et `int nombrePuces`.
Écrire un constructeur `public Toutou (String n, int np)` qui propage des exceptions de

type `IllegalArgumentException` lorsque le nom `n` est `null` ou lorsque le nombre de puces `np` est négatif. Utiliser ce constructeur dans une méthode `main` pour contrôler les appels `new Toutou ("milou", 4)` et `new Toutou ("medor", -11)` et afficher les erreurs éventuelles lors de l'exécution des constructeurs.

Exercice 5 [Capture d'exceptions et rôle de `finally`.]
Executez la classe suivante, et expliquez la raison de son comportement.

```
import java.io.*;

public class Except1 {
  public void methodeA(String args[]) {
    System.out.println("  methodeA : debut");
    try {
      System.out.println("  methodeA : appel de methodeB");
      this.methodeB(args);
      System.out.println("  methodeA : retour de methodeB");
      if (args.length > 99)
        throw new IOException();
    } catch (IOException e) {
      System.out.println("  methodeA : capture : "+ e);
    } finally {
      System.out.println("  methodeA : execute finally");
    }
    System.out.println("  methodeA : fin");
  }

  public void methodeB(String args[]) {
    System.out.println("    methodeB : debut");
    try {
      System.out.println("    methodeB : tente d'acceder a args[99]");
      String s = args[99];
      System.out.println("    methodeB : a reussi a acceder a args[99]");
    } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("    methodeB : capture : "+ e);
    } finally {
      System.out.println("    methodeB : execute finally");
    }
    System.out.println("    methodeB : fin");
  }

  public static void main(String args[]) {
    System.out.println("main : debut");
    Except1 ex = new Except1();
    try {
      System.out.println("main : appel de methodeA");
      ex.methodeA(args);
      System.out.println("main : retour de methodeA");
    } catch (Exception e) {
```

```
      System.out.println("main : capture : "+e);
    }
    System.out.println("main : fin");
  }
}
```

Exercice 6 [Classes abstraites.]

1. Construisez une classe abstraite `TabTrie` qui correspond à un tableau trié d'objets. Cette classe doit notamment contenir :
   – un tableau d'`Object`, `tab`, initialisé avec une `capacite` définie par défaut (il faut penser aussi à stocker le nombre d'`Object` contenus dans le tableau),

   et différentes méthodes qui peuvent être implantées ou abstraites :
   – une méthode `plusGrand` qui compare deux objects et renvoie `true` si le premier est plus grand que le deuxième,
   – une méthode `ajouter` qui insère un objet dans le tableau en respectant l'ordre croissant,
   – une méthode `toString` qui renvoie une chaîne de caractères représentant le tableau.

   Lorsque la `capacite` du tableau est atteinte, l'insertion d'un nouveau element lancera une exception `TabPlein`.
2. Construisez la classe `TabTriCouple` qui hérite de `TabTri` et ordonne des objects de type `Couple` lexicographiquement.
3. Une solution était-elle envisageable uniquement avec des interfaces ? Quel est l'intérêt ici des classes abstraites ?