S01 - Type Abstrait, Piles

1. Compléter cette réalisation de la classe `CharStack` (pile de caractères). Utiliser en interne un tableau, et doubler sa taille lorsqu'il est rempli. Pour chaque méthode, préciser en commentaires les pré-conditions, c.-à-d. annoncer quelles hypothèses doivent être vérifiées pour que l'appel à la méthode soit valide (dans une spécification, une pré-condition décrit ce qu'il est interdit de faire, comme une mise en garde dans un mode d'emploi).

```
public class CharStack {
  private int topIndex;
  private char[] buffer;
  public CharStack() { this(…); }
  public CharStack(int estimatedSize) {…}
  public boolean isEmpty() {…}
  public char top()        {…}
  public char pop()        {…}
  public void push(char x) {…}
  public Enumeration<Character>
                   topDownTraversal() {…}
}
```

```
public interface Enumeration<E> {
  boolean hasMoreElements();
  E nextElement();
}
```

1b. On aimerait que l'objet construit par `topDownTraversal()` devienne "invalide" dès que la pile subira un `push()` ou un `pop()`. Adapter pour que les méthodes de l'interface lèvent au besoin une `ConcurrentModificationException`.

2. Ecrire une méthode qui teste si une chaîne de caractères avec parenthèses est "bien balancée". Utiliser votre classe `CharStack` ("utiliser" = seulement ce qui est public !). Un petit programme de test est mis à disposition.

3. Expliquer l'effet des méthodes `f()` et `g()`.

```
public class Parenth {
  static boolean isOpeningParenth(char c) {
    return (c == '(') || (c == '{');
  }
  static boolean isClosingParenth(char c) {
    return (c == ')') || (c == '}');
  }
  static boolean isMatchingParenth(char c1,
                                   char c2){
    return (  (c1=='(') && (c2==')'))
      ||   (  (c1=='{') && (c2=='}'));
  }
  //-------------------------------------
  public static boolean isBalanced(String l) {
    char c;
    CharStack s = new CharStack();
    for (int i=0; i<l.length(); i++) {
      c = l.charAt(i);
      ............
```

```
static int g(CharStack s, char e) {
  CharStack r = new CharStack();
  int c=0;
  while(!s.isEmpty())
    r.push(s.pop());
  while(!r.isEmpty()) {
    if (r.top()==e) c++;
    s.push(r.pop());
  }
  return c;
}
```

```
static String f(String w) {
  String r="";
  CharStack s = new CharStack();
  for(int i=0; i<w.length(); i++)
    s.push(w.charAt(i));
  while(!s.isEmpty())
    r += s.pop();
  return r;
}
```

Valable pour tous les TPs de l'année :

- nos TPs ne sont ***pas*** à faire en mode "*just ask Google!*", ni ChatGPT … , vous n'apprendrez vraiment qu'en faisant tous les exercices vous-mêmes;
- réfléchir sur papier *avant* de vous lancer sur l'IDE (c'est un gain de temps...);
- indiquer toujours dans quelle mesure vos codes ont été testés, avec les résultats des tests;
- travailler par groupe de 2;
- inclure tout ce qui permet d'évaluer le travail (réponses, commentaires, ...); mettre en évidence vos contributions (n'inclure que les parties de code qui étaient à faire, ou au moins les mettre en évidence d'une manière claire)
- déposer selon les indications fournies, indiquer noms et classe, respecter les délais.

Cours Moodle – `http://cyberlearn.hes-so.ch/course/view.php?id=19930`

Lecture – Une bonne partie de la matière du cours est inspirée du livre suivant, qui est un bon complément aux notes de cours :

[Weiss10] Mark Allen Weiss. "*Data Structures & Problem Solving Using Java*". Addison Wesley. 4th Edition, 2010. ISBN 0321546229.

La notion de structure de données et la spécification de la pile y sont discutées sections 6.1 et 6.6 (p. 266, 294). L'implémentation de la pile par tableau est discutée dans la section 16.1.1 (p. 626).