Exercice 2

Ecrire une classe `Pile`, modélisant le fonctionnement d'une pile de caractères. Chaque pile encapsulera une liste chaînée d'éléments, dont la classe `Element` devra aussi être définie. La classe `Pile` sera pourvue des trois méthodes suivantes :

- `boolean estVide()`, renvoyant `true` si la pile est vide.
- `void empiler(char c)`, plaçant le caractère `c` au sommet de la pile,
- `char depiler()`, renvoyant le caractère au sommet de la pile, ou une valeur par défaut quelconque si la pile est vide.

Exercice 3

Un mot bien parenthésé est une suite de caractères `(` et `)` obtenu à l'aide des règles suivantes:

- le mot vide est bien parenthésé.
- si `m` est bien parenthésé, alors `(m)` l'est aussi.
- si `m` et `m`' sont bien parenthésés alors `m m`' l'est aussi.

Par exemple `(())()` est bien parenthésé, mais pas `(()`, ni `())`. On suppose que l'on dispose d'une méthode `static char getNextChar()`, renvoyant le dernier caractère entré au clavier, et donc un `"\n"` en cas de retour chariot.

1. En utilisant la classe `Pile`, écrire une méthode

   ```
   static boolean estBienParenthese()
   ```

   lisant une suite de parenthèses au clavier, et décidant si elles forment un mot bien parenthésée. Les caractères autres que les parenthèses seront ignorés.

2. Généraliser au cas où il y a des parenthèses, des accolades et des crochets, c'est-à-dire au cas où :

   - le mot vide est bien parenthésé.
   - si `m` est bien parenthésé, alors `(m)`, {`m`} et `[m]` le sont aussi.
   - si `m` et `m`' sont bien parenthésés alors `mm`' l'est aussi.

Remarque. L'implémentation de `getNextChar()` est par exemple :

```
static char getNextChar() {
   try { return (char) System.in.read(); }
   catch (Exception e) { return '\n'; }
}
```