1 Implémentation d'une pile par tableau

La pile est une structure de type LIFO (Last In, First Out), on accède uniquement au sommet et la première valeur accessible est la dernière ajoutée. Le type de données abstrait d'une pile définit les méthodes suivantes :

- `void push` : ajoute un élément au sommet de la pile. Cette méthode soulève une exception (`IllegalStateException`) si la pile est pleine.
- `Truc pop` : retourne l'objet situé au sommet de la pile et le retire. Si la pile est vide, cette méthode soulève une exception (`NoSuchElementException`).
- `boolean isEmpty` : teste si la pile est vide (retourne un booléen)
- `boolean isFull` : teste si la pile est pleine (retourne un booléen)
- `Truc peek` : retourne l'objet situé au sommet de la pile

. Cette méthode soulève aussi une exception si la pile est vide.

À partir de ces fonctions, nous allons implémenter une pile via la classe `Pile` et les méthodes associées. On se limite ici à des piles d'objets de type `Truc`, dont la taille est bornée par un entier `tailleMax`. On suppose que lorsque la pile est pleine, on ne peut plus empiler, et que lorsqu'elle est vide on ne peut pas dépiler. On peut par exemple utiliser la classe suivante pour implémenter une pile avec un tableau :

```
class Pile{
        private int tailleMax ; // taille maximale de la pile
        private Truc[] contenu ; // elements contenus dans la pile
        private int sommet ; // position du sommet de la pile dans le tableau
}
```

L'élément `contenu[sommet]` est l'élément qui se trouve en haut de la pile. Par convention, la pile est vide si sommet vaut -1.

Q 1. Écrivez un constructeur `Pile(int taille)` qui initialise une pile vide d'objets `Truc` avec une taille maximale donnée.

Q 2. Écrivez les méthodes `isEmpty`, `isFull`, `push`, `pop` et `peek`.

2 Utilisation d'une pile : vérification des délimiteurs

Nous allons étudier ici la vérification des délimiteurs dans une expression. On distingue ici trois délimiteurs, les parenthèses "(" et ")", les accolades "{" et "}" et les crochets "[" et "]". L'objectif est de déterminer si une chaîne de caractères (qui pourrait être un programme) est correctement délimitée, c'est-à-dire si chacun des délimiteurs ouvrant est suivi par un délimiteur fermant. Il s'agit d'une fonction implémentée dans vos chers compilateurs.

Q 1. Les exemples suivant sont-ils bien délimitées ?

a(b)c
a[b(c)}d
{a(b[c[d))]e]}
{a(b(d[d]))e}
(a[b]c

Q 2. Est-il possible de vérifier une chaîne uniquement en comptant les délimiteurs ? Si non, proposez un contre-exemple. Écrivez la méthode `boolean verifDelim(String s)` qui vérifie si la chaîne est correctement délimitée sans utiliser de pile. Cette méthode ne considérera que les délimiteurs "(" et ")".

Q 3. En vous inspirant des actions réalisées sur les compteurs, écrivez la méthode `boolean verifDelimStack(String s)` qui réalise vérification de délimiteurs à partir d'une pile. Vous utiliserez la classe `Pile` écrite en Section 1, en remplaçant le type Truc par le type adéquat.

Q 4. Modifiez votre algorithme de manière à afficher les indices des caractères qui posent problème, par exemple lorsqu'il manque un délimiteur fermant ou qu'il ne s'agit pas du délimiteur fermant correspondant au délimiteur ouvrant.

3 Implémentation d'une File avec un tableau

Pour la file (structure FIFO : First In, First Out), on commence par dépiler le premier élément mis dans la file. On peut par exemple utiliser la classe suivante pour implémenter une file avec un tableau :

```
class File{
  private Truc[] contenu ; // elements contenus dans la file
  private int tete ; // position de la tête de la file dans le tableau
  private int queue ; // position juste après le sommet de la file dans le tableau
```

```
    // i.e. première case de libre après la file

    public boolean isEmpty() { ...}
    public boolean isFull() { ...}
    public boolean offer(Truc t) {...}
    public Truc poll() {...}
}
```

Q 1. Écrivez les méthodes `isEmpty`, `isFull`, `offer` et `poll`.
Q 2. Si votre première version de la file fonctionnait avec des décalages, réfléchissez à une solution sans.