جامعة ابن زهر
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵉⴱⵏ ⵣⵀⵔ
UNIVERSITÉ IBN ZOHR

Filière d'excellence **ADIA**

TRAVAUX PRATIQUE DE JAVA

Réaliser par :

EL RHYATI Aymane

Tarzoukt Abderrahim

Encadré par :

M.OUKDACH Yassine

Exercice des piles :

Premièrement en doit compléter la classe Pile (du fichier Pile.java)

1. **Les Variables :**

```
public class Pile{
    static final int MAX = 8;
    char t[];
    int top;
```

- **MAX** : Représente la taille maximale de la pile.
- **t** : Tableau pour stocker les caractères, simulant la pile.
- **top** : Représente l'indice du sommet de la pile.

2. **Constructeur :**

```
public Pile(){
    t=new char[MAX];
    top=-1;
}
```

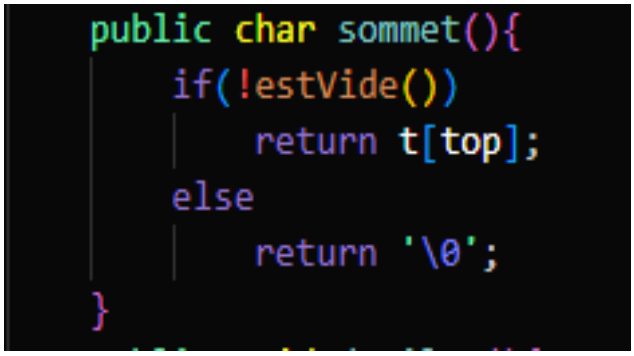
Initialise la pile avec la taille maximale spécifiée **MAX** et défini l'indice du sommet à -1, indiquant une pile vide.

3. **Méthode empiler :**

```
public void empiler (char c){
    if(!estPleine())
        t[++top]=c;
    else
        System.out.println("Pile pleine");
}
```

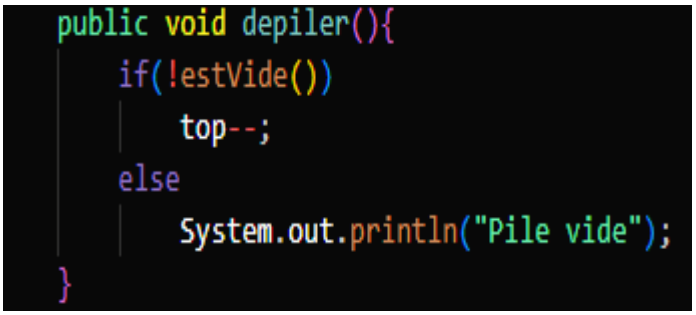
- Ajoute un caractère à la pile si la pile n'est pas pleine.
- Si la pile est pleine, affiche "Pile pleine".

4. **Méthode sommet :**



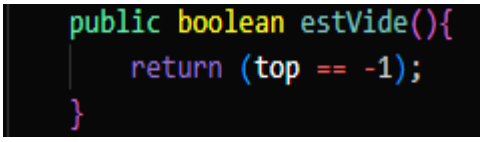
- Cette méthode retourne l'élément en haut de la pile sans le supprimer.
- Elle vérifie d'abord si la pile n'est pas vide (!estVide()).
- Si la pile n'est pas vide, elle retourne l'élément en haut (t[top]).
- Si la pile est vide, elle retourne le caractère nul ('\0').

5. **Méthode depiler :**



- Retire l'élément du sommet de la pile si la pile n'est pas vide.
- Si la pile est vide, affiche "Pile vide".

6. **Méthode estVide :**



- Retourne true si la pile est vide (le sommet est à -1), sinon retourne false.

7. **Méthode estPleine :**

```
    public boolean estPleine(){
        return (top == MAX - 1);
    }
}
```

- Retourne true si la pile est pleine (le sommet est à l'indice maximal), sinon retourne false.

On doit faire le fichier (TestPile.java) Pour le test :

1. **main Method :**

```
public class TestPile {

    Run | Debug
    public static void main(String[] args) {
        System.out.println("Entrez une chaîne de caractères (# pour terminer) : ");
        String chaine = lireChaine();

        System.out.println("Chaîne inversée : " + invChaine(chaine));

        System.out.println("\nEntrez un texte avec des parenthèses (# pour terminer) : ");
        String texte = lireChaine();

        if (verifierParentheses(texte)) {
            System.out.println("bien parenthésée.");
        } else {
            System.out.println("erreur");
        }
    }
```

- Demande à l'utilisateur d'entrer une chaîne de caractères, inverse la chaîne à l'aide de la méthode invChaine, puis affiche la chaîne inversée.
- Demande également à l'utilisateur d'entrer un texte avec des parenthèses, vérifie l'équilibre des parenthèses à l'aide de la méthode verifierParentheses, et affiche le résultat.

2. invChaine Method :

```java
private static String invChaine(String chaine) {
    Pile pile = new Pile();

    for (int i = 0; i < chaine.length(); i++) {
        pile.empiler(chaine.charAt(i));
    }

    StringBuilder chaineInverse = new StringBuilder();
    while (!pile.estVide()) {
        chaineInverse.append(pile.sommet());
        pile.depiler();
    }

    return chaineInverse.toString();
}
```

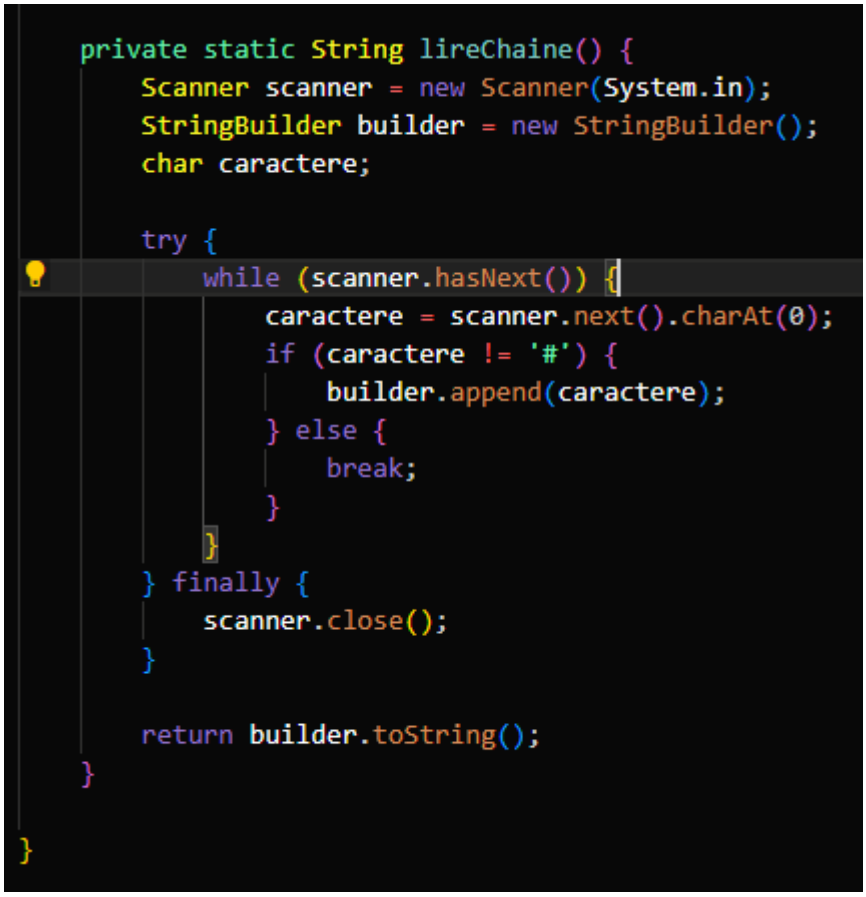
- Cette méthode prend une chaîne de caractères en entrée.
- Elle crée une instance de la classe Pile. Elle parcourt chaque caractère de la chaîne d'origine et l'empile sur la pile.
- Ensuite, elle dépile les caractères de la pile pour construire la chaîne inversée. La chaîne inversée est ensuite retournée.

3. Méthode verifierParentheses :

```java
private static boolean verifierParentheses(String texte) {
    Pile pile = new Pile();

    for (int i = 0; i < texte.length(); i++) {
        char caractere = texte.charAt(i);

        if (caractere == '(') {
            pile.empiler(caractere);
        } else if (caractere == ')') {
            if (!pile.estVide()) {
                pile.depiler();
            } else {
                return false;
            }
        }
    }

    return pile.estVide();
}
```

- Cette méthode prend une chaîne de texte en entrée.
- Elle crée une instance de la classe Pile.
- Elle parcourt chaque caractère du texte.
- Si elle rencontre une parenthèse ouvrante '(', elle l'empile sur la pile.
- Si elle rencontre une parenthèse fermante ')', elle vérifie si la pile n'est pas vide avant de la dépiler.
- Si la pile est vide, cela signifie qu'il y a une parenthèse fermante sans sa correspondante ouvrante, et la méthode retourne false.
- À la fin, elle vérifie si la pile est vide. Si la pile est vide, cela signifie que toutes les parenthèses ouvrantes ont une correspondante fermante, et la méthode retourne true. Sinon, elle retourne false.

4. **Méthode lireChaine:**



```
    private static String lireChaine() {
        Scanner scanner = new Scanner(System.in);
        StringBuilder builder = new StringBuilder();
        char caractere;

        try {
            while (scanner.hasNext()) {
                caractere = scanner.next().charAt(0);
                if (caractere != '#') {
                    builder.append(caractere);
                } else {
                    break;
                }
            }
        } finally {
            scanner.close();
        }

        return builder.toString();
    }

}
```

- Cette méthode utilise la classe Scanner pour lire l'entrée de l'utilisateur depuis la console.
- Elle lit chaque caractère jusqu'à ce que l'utilisateur entre '#' pour indiquer la fin de la saisie.

- Les caractères lus sont concaténés pour former une chaîne qui est ensuite retournée.