

7-Exercice de synthèses: (28/11/23)

Objectifs :

- Programmer une classe Pile (pile de caractères), représentée par un tableau et un indice de sommet de pile.
- Utilisation de chaînes de caractères sous forme de tableau de caractères, et trouver des solutions basées sur les caractéristiques de la variable char.
- Découvrir la méthode de stockage en pile (premier entré, dernier sorti).

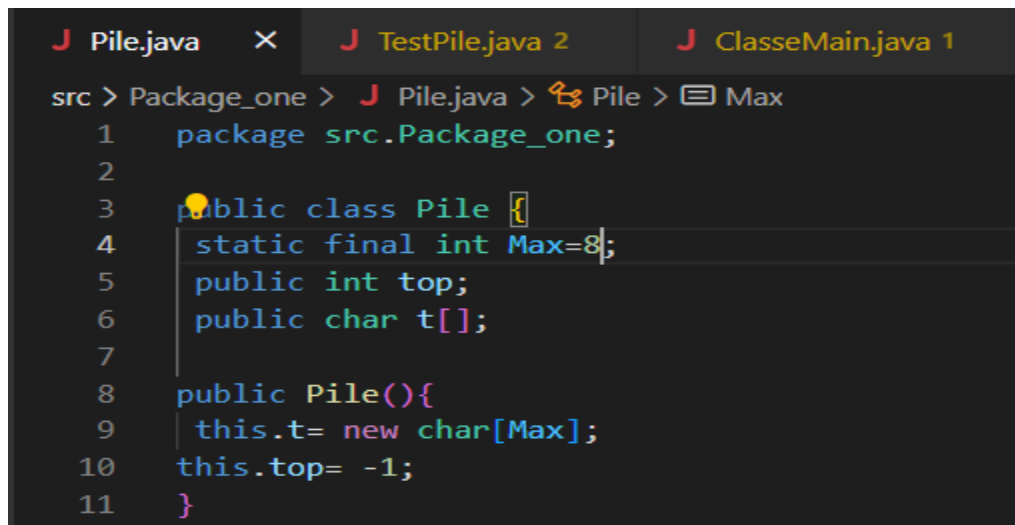
Introduction :

- Dans cet exercice, je crée trois classes :
 1. **ClasseMain** : c'est la classe principale du projet.
 2. **Pile** : une classe caractérisée par un tableau de caractères et un sommet.
 3. **TestPile** : une classe pour des opérations spécifiques en utilisant les fonctionnalités de la classe Pile.

1-la classe Pile :

1-1 : la réalisation

la première chose à faire est de déclarer trois paramètres (attributs), et un constructeur qui va initialiser les deux paramètres



```
J Pile.java  X  J TestPile.java 2  J ClasseMain.java 1
src > Package_one > J Pile.java > Pile > Max
1  package src.Package_one;
2
3  public class Pile {
4      static final int Max=8;
5      public int top;
6      public char t[];
7
8      public Pile(){
9          this.t= new char[Max];
10         this.top= -1;
11     }
```

-deuxièmes est la création de cinq fonctions (méthodes) :

1- Pour tester la mémoire du tableau :

- A. Si vide : puisque le variable 'top' est initialisée par -1 et à chaque fois on ajoute un élément on l'incrémente par un , donc il suffit de tester si le variable est incrémenté ou non :

```
public boolean estVide(){
    if(this.top== -1){
        return true;
    }else
        return false;
}
```

- B. Si pleine : du même façon si le variable 'top' est incrémenté jusqu'à la valeur de Max donc le pile est pleine , (pour le paramètre 'char c' et le condition de '#' sont relie avec la classe TestPile) :

```
public boolean estPleine(char c){
    if((c=='#')||(this.top==Max-1)){
        return true;
    }else{
        return false; }
}
```

2- Pour atteindre la sommet de pile : il suffit the prend variable 'top' comme indice de pile t[]

```
9 public char sommet(){
0 return this.t[this.top];
1 }
```

3- Pour dépiler la pile t[] : puisqu'on ne peut pas modifier un tableau char après l'avoir déclaré, j'ai donc décrémenté la valeur de la variable 'top', et j'ai ajouté une condition dans la classe TestPile en cas de rencontre d'un texte avec plus de parenthèses fermantes que de parenthèses ouvertes. :

```

public boolean depiler(){
    if (this.top== -1) {
        return false;
    }
    this.top--;
    return true;
}

```

1-2 : Conclusion

Je prends en considération que les méthodes de la classes Pile doivent être utilisable par la classe TestPile sans changer les méthodes du classe Pile.

2- la classe TestPile :

1-1 : traitement :

la classe TestPile est caractérisé par :

1- deux attributs : char c et Pile P , et un constructeur qui va instancier la classe Pile :



```

file.java  ●  J TestPile.java 2 X  J Class
> Package_one > J TestPile.java > TestPile >
package src.Package_one;
import java.util.*;

public class TestPile {
    Pile P ;
    public char c;

    public TestPile(){
        this.P = new Pile();
    }
}

```

2- Une méthode 'inverse' qui lit une chaîne de caractères et l'imprime inversée en respectant l'algorithme ; donc, je crée une boucle pour l'entrée et une autre d'inversée :

```
public void inverse(){
    Scanner scan = new Scanner(System.in);
    System.out.println(x:"\nsaisis un caractère : ");
    this.c = scan.nextLine().charAt(index:0);
    while (this.c!='#') {
        this.P.empiler(this.c);
        System.out.println(x:"\nsaisis un caractère : ");
        this.c = scan.nextLine().charAt(index:0);
    }
    this.P.empiler(this.c);
    while (P.estVide()!=true) {
        this.c = P.sommet();
        System.out.println(this.c);
        P.depiler();
    }
}
```

3- Une méthode pour vérifier les parenthèses : Pour faciliter la saisie du texte par l'utilisateur, j'ai créé une chaîne de caractères (string) pour enregistrer le texte et l'ai ensuite copiée dans un tableau char arr[]. Ensuite, une boucle for parcourt tous les caractères du tableau à la recherche de '(' et ')'. En cas de rencontre d'une ')', la méthode dépiler est appelée pour vérifier si 'top' est incrémenté ou non. Si ce n'est pas le cas, cela signifie qu'il y a plus de parenthèses fermantes que de parenthèses ouvertes, et la boucle est interrompue (break). Les deux autres conditions couvrent les cas restants :

```

public void parenthese(){
    Scanner scan = new Scanner(System.in);
    System.out.println(x:"entrez le texte contient des parenthèses \n");
    String text = scan.nextLine();
    text+='#';
    char arr[]=text.toCharArray();
    int i;
    for(i=0; i<arr.length;i++){
        if(arr[i]=='('){
            P.empiler(arr[i]);
        }
        if(arr[i]==')'){
            boolean ans= P.depiler();
            if(ans==false){
                System.out.println(x:"\n il y a plus de parenthèses fermantes que de parenthèses ouvrantes");
                break;
            }
        }
    }
    if(i==arr.length){
        if(P.estVide()){
            System.out.println(x:"\n l'expression est bien parenthésée");
        }else{
            System.out.println(x:"\n il y a plus de parenthèses ouvrantes que de parenthèses fermantes");
        }
    }
}

```

1-2 Conclusion

Le programme pour inverser le texte et celui pour vérifier les parenthèses sont déclarés dans la même classe TestPile afin d'éviter la surcharge des fonctions. Chaque fonction est capable de traiter les requêtes des deux programmes