TP 2

LES CLASSES EN JAVA

Nos souvenirs – TP2 «La classe String»

- **boolean equals()** La comparaison de deux chaînes
- **int length()** La détermination de la longueur d'une chaîne
- **char charAt(int i)** Retourne le caractère à l'indice spécifié
- **String substring(int d, int f)** Retourne une chaîne égale à la chaîne sans les espaces de début et de fin.
- **int indexOf(String str, int fromIndex)** Retourne l'indice de la première occurrence de la chaîne à partir de fromIndex
- **int lastIndexOf(String str, int fromIndex)** Retourne l'indice de la dernière occurrence de la chaîne à partir de fromIndex

Nos souvenirs – TP2 «La classe Scanner»

- Elle simplifie la lecture de données sur l'entrée standard (clavier) ou dans un fichier.
- Pour utiliser la classe Scanner, il faut d'abord l'importer :
  - import java.util.Scanner;
- Ensuite il faut créer un objet de la classe Scanner :
  - Scanner sc = new Scanner(System.in);

Nos souvenirs – TP2 «La classe Scanner»

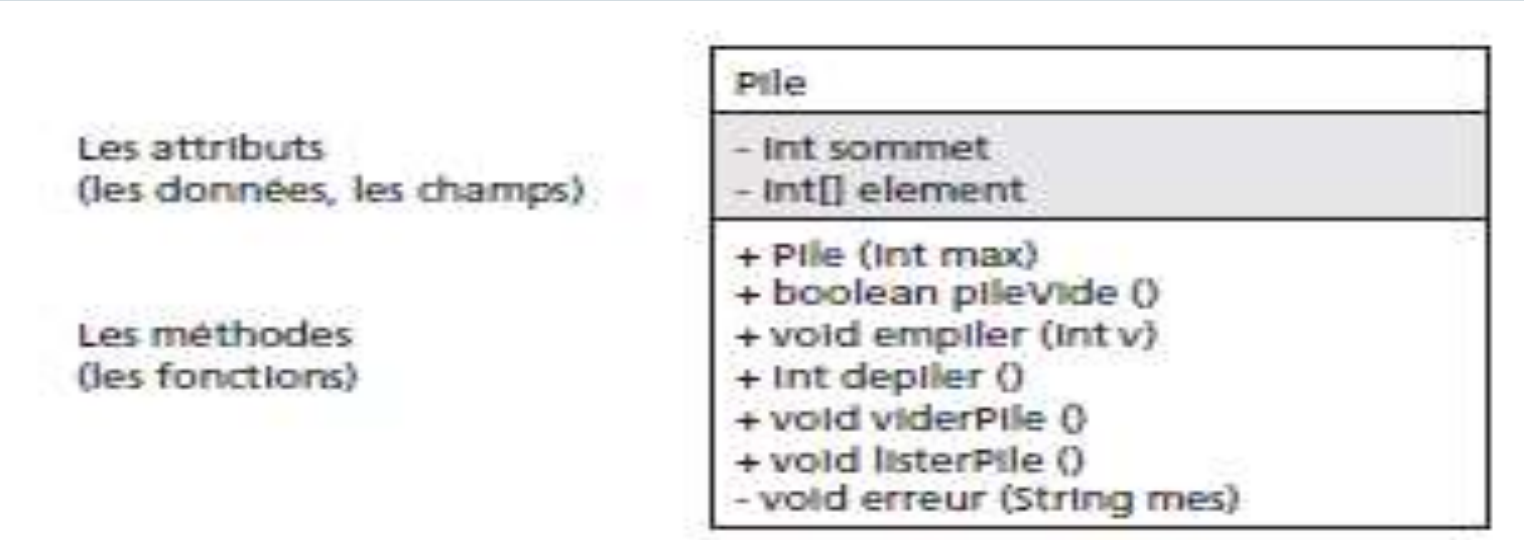
- Pour récupérer les données, il faut faire appel sur l'objet sc aux méthodes décrites ci-dessous. Ces méthodes parcourent la donnée suivante lue sur l'entrée et la retourne :
  - String next() donnée de la classe String qui forme un mot,
  - String nextLine() donnée de la classe String qui forme une ligne,
  - boolean nextBoolean() donnée booléenne,
  - int nextInt() donnée entière de type int,
  - double nextDouble() donnée réelle de type double.

Nos souvenirs – TP2 «La classe Scanner»

- Il peut être utile de vérifier le type d'une donnée avant de la lire :
  - boolean hasNext() renvoie true s'il y a une donnée à lire
  - boolean hasNextLine() renvoie true s'il y a une ligne à lire
  - boolean hasNextBoolean() renvoie true s'il y a un booléen à lire
  - boolean hasNextInt() renvoie true s'il y a un entier à lire
  - boolean hasNextDouble() renvoie true s'il y a un double à lire.

EXERCICE 1–L'Enoncé-

Il s'agit d'écrire le code en Java de la classe Pile. Le mot clé "private" indique que les attributs sommet et element, et la méthode erreur() sont privés.



EXERCICE 1–La Correction-

```
public class Pile {
    private int sommet; // repere le dernier occupe (le sommet)
    private int[] element ; // tableau d'entiers alloue dynamiquement
    public Pile (int max) {
    sommet = -1;
    element = new int [max]; // allocation de max entiers
    }
```

sommet et element sont des attributs prives

Le constructeur d'un objet : Elle porte seulement le nom de la classe en cours de définition. Cette méthode est un **constructeur chargé de construire l'objet (allouer de la mémoire et initialiser les** attributs de l'objet). Ce constructeur n'est jamais appelé directement ; il est pris en compte lors de la demande de création de l'objet avec new

EXERCICE 1–La Correction-

```
private void erreur (String mes) {
System.out.println ("***erreur : " + mes); }
public boolean pileVide () {
return sommet == -1; }
public void empiler (int v) {
if (sommet < element.length - 1) {
sommet++;
element [sommet] = v;} else {
erreur ("Pile saturee"); } }
public int depiler () {
int v = 0; // v est une variable locale a depiler()
if (!pileVide()) {
v = element [sommet];
sommet--; } else {
erreur ("Pile vide"); }return v; }
```

erreur est une methode privee utilisable seulement dans la classe Pile

EXERCICE 1–La Correction-

```
public void viderPile () {
sommet = -1; }
public void listerPile () {
if (pileVide()) {
System.out.println ("Pile vide");
} else {
System.out.println ("Taille de la pile : " + element.length);
for (int i=0; i <= sommet; i++) {
System.out.print (element[i] + " ");
}
System.out.println(); } } } /
```

EXERCICE 2–L'Enoncé-

Il s'agit d'écrire le programme PPPile qui crée l'objet pile1 de type Pile sur lequel on applique, en fonction des réponses au menu de l'utilisateur, des méthodes de la classe Pile.

GESTION D'UNE PILE

0 - Fin

1 - Initialisation de la pile

2 - La pile est-elle vide ?

3 - Insertion dans la pile

4 - Retrait de la pile

5 - Vidage de la pile

6 - Listage de la pile

Votre choix ?

EXERCICE 2–La Correction-

```
import java.util.Scanner;
class PPPile {
static int menu () {
System.out.print (
"\n\nGESTION D'UNE PILE\n\n" +
"0 - Fin\n" +
"1 - Initialisation de la pile\n" +
"2 - La pile est-elle vide ?\n" +
"3 - Insertion dans la pile\n" +
"4 - Retrait de la pile\n" +
"5 - Vidage de la pile\n" +
"6 - Listage de la pile\n\n" +
"Votre choix ? "
);
Scanner sc = new Scanner(System.in);
return sc.nextInt() ;
}
```

PPPile.java Programme Principal de la Pile

EXERCICE 2–La Correction-

```
public static void main (String[] args) {
Pile pile1 = new Pile (5) ; // creer un objet Pile
int valeur;
boolean fini = false;
Scanner sc = new Scanner(System.in);
while (!fini) {
switch (menu()) {
case 0 : // fin
fini = true; break;
case 1 : // initialisation - une pile par defaut est creee lors du new Pile (5)
System.out.print ("Taille de la pile ? ");
int taille = sc.nextInt() ; pile1 = new Pile (taille) ;
    System.out.println("Initialisation d'une pile vide"); break;
case 2 : // la pile est-elle vide ?
if (pile1.pileVide() ) { System.out.println ("La pile est vide"); } else {
System.out.println ("La pile n'est pas vide");}break;
```

pile1, valeur et fini sont des variables locales a main()

EXERCICE 2–La Correction-

```
case 3 : // empiler une valeur
System.out.print ("Valeur a empiler ? ");
valeur = sc.nextInt() ;
pile1.empiler (valeur) ; break;
case 4 : // depiler une valeur
valeur = pile1.depiler(); System.out.println ("Valeur depilee : " +
    valeur); break;
case 5 : // vider la pile
pile1.viderPile(); break;
case 6 : // lister la pile
pile1.listerPile(); break;
} // switch } // while (!fini) } // main } // class PPPile
```