Présentation de Java

Historique et caractéristiques de Java

- **Historique :** Développé par Sun Microsystems en 1995.
  - Conçu pour être portable, robuste et sécurisé.
  - Utilise la JVM (Java Virtual Machine) pour exécuter le code sur différentes plateformes.
- **Caractéristiques :**
  - Langage orienté objet.
  - Forte gestion de la mémoire (garbage collector).
  - Large écosystème de bibliothèques et frameworks.

Comprendre JDK et JRE

- **JDK (Java Development Kit) :**
  - Contient tout ce qui est nécessaire pour développer des applications Java.
  - Inclut le compilateur (`javac`), les outils de débogage et le JRE.
- **JRE (Java Runtime Environment) :**
  - Nécessaire pour exécuter des applications Java.
  - Contient la JVM (Java Virtual Machine) et les bibliothèques nécessaires.

Résumé :

- **Développer :** Utilisez le JDK.
- **Exécuter :** Le JRE suffit.

Évolution des versions (1)

- **Java 2 (1998) :**
  - Introduction de Swing pour les interfaces graphiques.
  - Amélioration des performances de la JVM.
- **Java 5 (2004) :**
  - Généricité
  - Enumérations
  - Boucles simplifiées, arguments multiples (varargs)
- **Java 8 (2014) :**
  - Introduction des expressions lambda et de l'API Stream.
  - Date et heure avec `java.time`.

Évolution des versions (2)

- **Java 11 (2018) :**
  - Première version LTS (Long-Term Support) après Java 8.
  - Nouveautés : API HTTP Client, retrait de JavaFX du JDK.
- **Java 17 (2023) :**
  - Ajout de fonctionnalités comme les classes scellées (sealed).
- **Java 21 (2021) :**
  - *main* très simplifié

Cycle de vie :

- Versions LTS tous les 3 ans pour les entreprises.
- Versions intermédiaires tous les 6 mois.

Installation de l’environnement

1. **Télécharger et installer le JDK** (Java Development Kit).
    - Site officiel : oracle.com/java
2. **Choisir un IDE** :
    - IntelliJ IDEA (recommandé pour sa robustesse).
    - Eclipse.
3. **Vérifier l’installation** :

```
java -version
javac -version
```

Structure d’un programme Java

Exemple minimal :

```
// La base
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

- **Fichier `.java` :** Contient une classe publique portant le même nom que le fichier.
- **Méthode `main` :** Point d’entrée du programme.
- La compilation (en byte-code) génère un fichier `.class` interprété par la JVM

Syntaxe de base en Java

Types de données primitifs, variables, opérateurs

- **Types primitifs :**
  - `int`, `double`, `boolean`, `char`, etc.
- **Déclaration de variables :**

```
int age = 25;
double price = 19.99;
boolean isActive = true;
char initial = 'A';
final int LONGUEUR_MAX = 25;
```

- **Opérateurs :**
  - Arithmétiques : `+`, `-`, `*`, `/`, `%`
  - Logiques : `&&`, `||`, `!`
  - Comparaison : `==`, `!=`, `<`, `>`, `<=`, `>=`

Les chaînes de caractères en Java (1)

- **Déclaration et concaténation :**

```
String greeting = "Bonjour"; String name = "Alice";
String message = greeting + ", " + name + "!";
System.out.println(message);
```

- **Méthodes utiles :**
  - Longueur :

```
System.out.println(greeting.length());
```

  - Comparaison :

```
if (name.equals("Alice")) {
    System.out.println("Nom correspond");
}
```

Les chaînes de caractères en Java (2)

- Sous-chaîne :

```
System.out.println(greeting.substring(0, 3)); // Affiche "Bon"
```

- Conversion en majuscules/minuscules :

```
System.out.println(greeting.toUpperCase()); // "BONJOUR"
```

- Remarque :

  `String` est une classe de `java.lang` (paquetage importé par défaut)

```
String s; // s vaut null
```

Les tableaux en Java

- **Déclaration et initialisation :**

```
int[] numbers = {1, 2, 3, 4, 5};
String[] names = new String[3];
names[0] = "Alice";
names[1] = "Bob";
names[2] = "Charlie";
```

- **Accès aux éléments :**

```
System.out.println(numbers[0]); // Affiche 1
```

- **Propriété `length` :**

```
System.out.println("Taille du tableau : " + numbers.length);
```

Limitations des tableaux (1)

- **Taille fixe :**
  - La taille d'un tableau doit être définie au moment de sa création et ne peut pas être modifiée.

```
int[] numbers10 = new int[10]; // Taille fixe de 10, 10 cases à 0 (valeur par défaut)
```

- **Difficulté à ajouter ou supprimer des éléments :**
  - Nécessité de copier les éléments dans un nouveau tableau pour modifier sa taille.

```
int[] numbers20 = Arrays.copyOf(numbers10, 20);
```

Limitations des tableaux (2)

- **Absence de méthodes utilitaires :**
  - Les tableaux ne disposent pas de méthodes comme `add`, `remove` ou `contains`.
- **Manipulation manuelle des index :**
  - Les opérations nécessitent de gérer explicitement les indices, augmentant le risque d'erreurs.

Présentation d'ArrayList

- **Classe fournie par le package `java.util`.**
- **Taille dynamique :**
  - La taille peut augmenter ou diminuer dynamiquement en fonction des besoins.
- **Avantages :**
  - Corrige tous les défauts d'un tableau statique
- **Création d'une ArrayList :**

```
import java.util.ArrayList;

ArrayList<String> names = new ArrayList<>();// crée 10 cases

ArrayList<boolean> generated = new ArrayList<>(50);// crée 50 cases
```

Méthodes principales d'ArrayList (1)

- **Ajouter des éléments :**

```
names.add("Alice");
names.add("Bob");
```

- **Accéder à un élément :**

```
String firstName = names.get(0); // Index 0
```

- **Modifier un élément :**

```
names.set(1, "Charlie"); // Remplace Bob par Charlie
```

Méthodes principales d'ArrayList (2)

- **Obtenir la taille :**

```
int size = names.size();
```

- **Supprimer un élément :**

```
names.remove(0); // Supprime Alice
```

- **Parcourir la liste :**

```
for (String name : names) {
    System.out.println(name);
}
```

Tableaux et chaînes de caractères : manipulation par adresse (1)

- **Références et adresses :**
  - En Java, les tableaux et les chaînes de caractères (`String`) sont manipulés par adresse (référence).
  - Lorsqu'on compare deux objets avec `==`, cela compare leur adresse en mémoire et non leur contenu.
- **Exemple avec des chaînes de caractères :**

```
String s1 = new String("test");
String s2 = new String("test");
System.out.println(s1 == s2); // false (adresses différentes)
System.out.println(s1.equals(s2)); // true (contenu identique)
```

Tableaux et chaînes de caractères : manipulation par adresse (2)

- **Bonne pratique :** Toujours utiliser `.equals()` pour comparer le contenu des chaînes de caractères.
- **Pour les tableaux :**
  - La comparaison avec `==` se fait également sur les adresses.

```
int[] arr1 = {1, 2, 3};
int[] arr2 = {1, 2, 3};
System.out.println(arr1 == arr2); // false
```

Structures de contrôle (1)

Conditions :

```
if (age >= 18) {
    System.out.println("Adulte");
} else {
    System.out.println("Mineur");
}
```

```
System.out.println(age>=18 ? "Adulte" : "Mineur");
```

Structures de contrôle (2)

```
int day = 3;
switch (day) {
    case 1:
        System.out.println("Lundi");
        break;
    case 2:
        System.out.println("Mardi");
        break;
    default:
        System.out.println("Jour inconnu");
}
```

Structures de contrôle (3)

Boucles :

For :

```
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

While :

```
int count = 0;
while (count < 5) {
    System.out.println(count);
    count++;
}
```

Structures de contrôle (4)

Do-While :

```
int count = 0;
do {
    System.out.println(count);
    count++;
} while (count < 5);
```

For-Each :

```
int[] tab = {1, 2, 3, 4, 5};
for (int i : tab) {
    System.out.println(i);
}
```

Fonctions et procédures en Java (1)

- Une fonction est une méthode qui retourne une valeur ou `void` si ce n'est pas le cas.
- **Exemples :**

```java
// Procédure
public static void afficherMessage(String message) {
    System.out.println(message);
}

// Fonction
public static int addition(int a, int b) {
    return a + b;
}
```

Fonctions et procédures en Java (2)

- **Passage de paramètres :**
  - En Java, les arguments sont toujours passés **par valeur**.
  - Pour les types primitifs : la valeur est copiée.
  - Pour les objets : la référence est copiée (référence = adresse).
- **Exemple de passage par valeur :**

```
public static void incrementer(int valeur) {
    valeur++;
}
public static void main(String[] args) {
    int x = 5;
    incrementer(x);
    System.out.println(x); // Affiche 5, pas 6
}
```

Fonctions et procédures en Java (3)

- **Modifications d'objets :**

```java
public static void ajouterElement(int[] tab) {
    tab[0]=1;
}

public static void modifier(int[] tab) {
    tab = new int[12];
    tab[0] = 5;
}

public static void main(String[] args) {
    int[] tab = new int[5];
    ajouterElement(tab);
    System.out.println(Arrays.toString(tab)); // [ 1 ...]
    modifier(tab);
    System.out.println(Arrays.toString(tab)); // [ 1 ...]
}
```

Saisie de données utilisateur avec `Scanner` (1)

- **Importer la classe Scanner :**

```
import java.util.Scanner;
```

- **Initialisation :**

```
Scanner scanner = new Scanner(System.in);
```

- **Méthodes courantes :**
  - Lire une chaîne de caractères :

```
System.out.print("Entrez votre nom : ");
String nom = scanner.nextLine();
System.out.println("Bonjour, " + nom + "!");
```

Saisie de données utilisateur avec `Scanner` (2)

- Lire un entier :

```
System.out.print("Entrez votre âge : ");
int age = scanner.nextInt();
System.out.println("Vous avez " + age + " ans.");
```

- Lire un nombre décimal :

```
System.out.print("Entrez un nombre décimal : ");
double valeur = scanner.nextDouble();
System.out.println("Vous avez entré : " + valeur);
```

- Fermer le Scanner après utilisation peut avoir des effets étranges :

```
scanner.close(); // autant éviter
```

Conversion de chaîne en entier ou réel

- **Conversion avec `Integer` et `Double` :**

```
String intStr = "42";
int intValue = Integer.parseInt(intStr);

String doubleStr = "3.14";
double doubleValue = Double.parseDouble(doubleStr);
```

- **Gestion des exceptions :**
  - Si la chaîne n'est pas convertible, une exception est levée.

```
try {
    int value = Integer.parseInt("abc");
} catch (NumberFormatException e) {
    System.out.println("Conversion échouée : " + e.getMessage());
}
```

Conversion de nombres en chaîne

- **Utiliser `String.valueOf` ou la concaténation :**

```
int number = 42;
String numberStr = String.valueOf(number);

double pi = 3.14;
String piStr = "" + pi;
```

- **Exemple pratique :**

```
int age = 30;
String message = "Votre âge est : " + age;
System.out.println(message);
```

Évolutions récentes du langage Java

- **Déclaration avec `var` (Java 10) :**
  - Permet de déclarer des variables locales sans préciser explicitement leur type.

```
var message = "Hello!"; // Type déduit comme String
```

- **`main` sans `static` et `String[] args` (Java 21) :**
  - Exemple simplifié :

```
void main() {
    System.out.println("Exécution simplifiée");
}
```