Université Sultan Moulay Slimane
Ecole Supérieure de Technologie Béni Mellal
Département ITG
DUT- IDIA 1 - S2
M122 - Ingénierie Logicielle avec Java et UML
Année universitaire: 2024-2025

Support de cours

Ingénierie logicielle avec Java et UML

Partie I: Programmation Orientée Objet en Java

Chapitre I: Notions fondamentales du langage Java

Pr. Siham BAKKOURI
Email: s.bakkouri@usms.ma

Plan

- **INTRODUCTION AU LANGAGE JAVA**
- **OUTILS ET ENVIRONNEMENT DE DÉVELOPPEMENT JAVA**
- **SYNTAXE DE BASE DU LANGAGE JAVA**
- **STRUCTURES DE CONTRÔLE**
- **TABLEAUX ET GESTION DES DONNÉES**
- **EXERCICES**

Introduction au langage Java

Java est un langage de développement créé par **Sun** en **1995** puis racheté par **Oracle** en **2009** qui a réussi à obtenir une très grande notoriété en seulement quelques années grâce à ses qualités.

- Java représente la synthèse des bons côtés de plusieurs langages de programmation (notamment C++ et SmallTalk). J'apprécie sa portabilité, sa robustesse et la richesse de ses API fournies en standard ou par des tiers commerciaux ou libre.

- On peut faire de nombreux types de programmes avec Java :

  → Des applications, sous forme de fenêtre ou de console

  → Des applets, qui sont des programmes Java incorporés à des pages Web

  → Des applications pour appareils mobiles, comme les smartphones, avec J2ME (Java 2 Micro Edition)

  → Des sites web dynamiques, avec J2EE (Java 2 Enterprise Edition, maintenant JEE) et bien d'autres : JMF (Java Media Framework), J3D pour la 3D. . .

→ Les différentes éditions de Java

- ***Sun*** puis ***Oracle*** ont toujours fourni gratuitement un ensemble d'outils et d'API pour permettre le développement de programmes avec Java. Ce kit, nommé ***JDK***, est librement téléchargeable sur le site web d'Oracle : https://www.oracle.com/java/technologies/
- Le ***JRE*** (Java Runtime Environment) contient uniquement l'environnement d'exécution de programmes Java. Le JDK contient lui-même le JRE. Le JRE seul doit être installé sur les machines où des applications Java doivent être exécutées.
- Trois plate-formes d'exécution (ou éditions) Java sont définies pour des cibles distinctes selon les besoins des applications à développer :
  - → **Java Standard Edition** (J2SE / Java SE) : environnement d'exécution et ensemble complet d'API pour des applications de type desktop.
  - → **Java Enterprise Edition** (J2EE / Java EE) : environnement d'exécution reposant intégralement sur Java SE pour le développement d'applications d'entreprises
  - → **Java Micro Edition** (J2ME / Java ME) : environnement d'exécution et API pour le développement d'applications sur appareils mobiles.

→ Les différentes éditions de Java

- La séparation en trois plateformes permet au développeur de mieux cibler l'environnement d'exécution et de faire évoluer les plateformes de façon plus indépendante.
- Avec **différentes éditions,** les **types d'applications** qui peuvent être développées en Java sont nombreux et variés :
  - → Applications desktop
  - → Applications web : servlets/JSP, portlets, applets
  - → Applications pour appareil mobile (MIDP) : midlets
  - → Applications pour appareil embarqué (CDC) : Xlets
  - → Applications pour carte à puce (Javacard) : applets Javacard
  - → Applications temps réel

→ Les différentes versions de Java

Les versions majeures de Java : (1/2)

- **Java 1.0** (23 janvier 1996) :** Première version publique de Java.
- **Java 1.1** (19 février 1997) :** Ajout de classes internes, collections, et introduction de Swing.
- **Java 2** (J2SE 1.2) (8 décembre 1998) : Renommage en Java 2, introduction du compilateur JIT et de Swing.
- **Java 2** (J2SE 1.3) (8 mai 2000) : Améliorations de la performance et introduction de JNDI.
- **Java 2** (J2SE 1.4) (6 février 2002) : Intégration de HotSpot, ajout de l'API Logging, et améliorations de la sécurité.
- **Java SE 5.0** (30 septembre 2004) : Intégration de generics, énumérations, et boucle for-each.

→ Les différentes versions de Java

Les versions majeures de Java : (2/2)

- **Java SE 6** (11 décembre 2006): Intégration de HotSpot, gestion automatique des ressources.
- **Java SE 7** (28 juillet 2011) : Introduction des expressions lambda et améliorations de la gestion des exceptions.
- **Java SE 8** (18 mars 2014) : Introduction de JPMS, expressions lambda, API Stream, et java.time.
- **Java SE 9** (21 septembre 2017) : Intégration de JShell et projet Jigsaw pour les modules.
- **Java SE 11** (25 septembre 2018) : Version LTS avec améliorations de la performance et introduction de l'API HTTP Client.
- **Java SE 12-17** (2019 - 2021) : Versions successives avec des améliorations continues.
- **Java SE 18-21** (2022 - 2023) : Versions successives avec des améliorations continues.

→ Java : Un langage de programmation

Java se distingue en tant que langage de programmation grâce à ses caractéristiques avantageuses:

- **Simplicité et productivité**:
  - → Intégration complète de l'orientation objet
  - → Gestion automatique de la mémoire grâce au "Garbage collector".
- **Robustesse, fiabilité et sécurité**
- **Indépendance par rapport aux plateformes**
- **Ouverture**:
  - → Intégration native d'Internet.
  - → Connexion transparente aux bases de données via **JDBC** (Java DataBase Connectivity).
  - → Prise en charge des caractères internationaux.
- **Distribution et aspects dynamiques**
- **Performance**

→ Java : Une plateforme

Java en tant que Plateforme

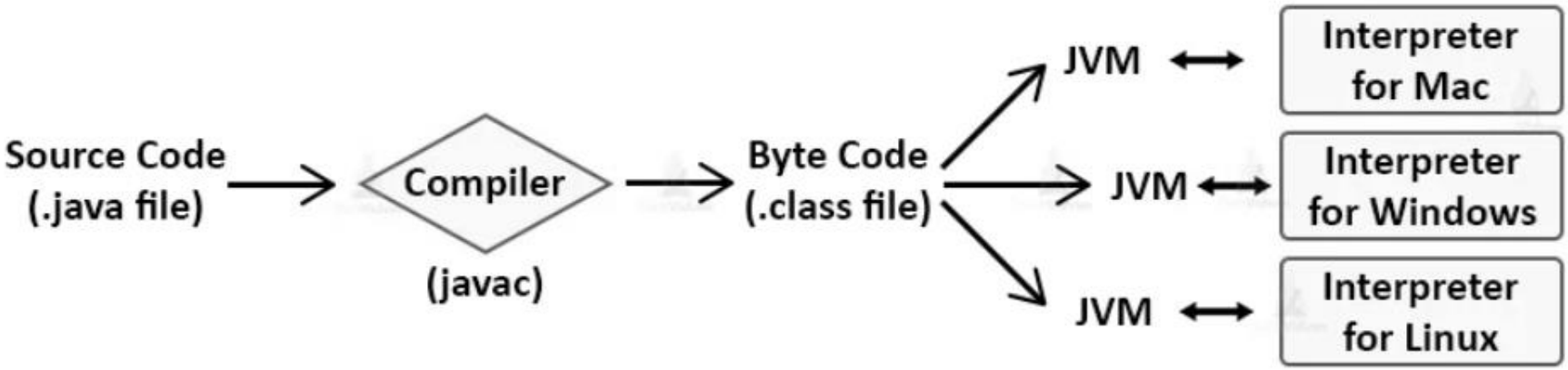
- L'Interface de Programmation Applicative (**API**) de **Java** est structurée en bibliothèques (packages).
- Ces packages regroupent des ensembles fonctionnels de composants (classes).
- Le cœur (core) de **l'API Java**, inclus dans toute implémentation complète de la plateforme Java, comprend notamment :
  - Essentials (types de données, objets, chaînes de caractères, tableaux, vecteurs, E/S, date, etc.)
  - Applet
  - Abstract Windowing Toolkit (AWT)
  - Basic Networking (URL, Socket - TCP ou UDP -, IP)
  - Evolved Networking (Remote Method Invocation)
  - Internationalization
  - Sécurité ... .

→ Essentiel du Développement Java : JVM

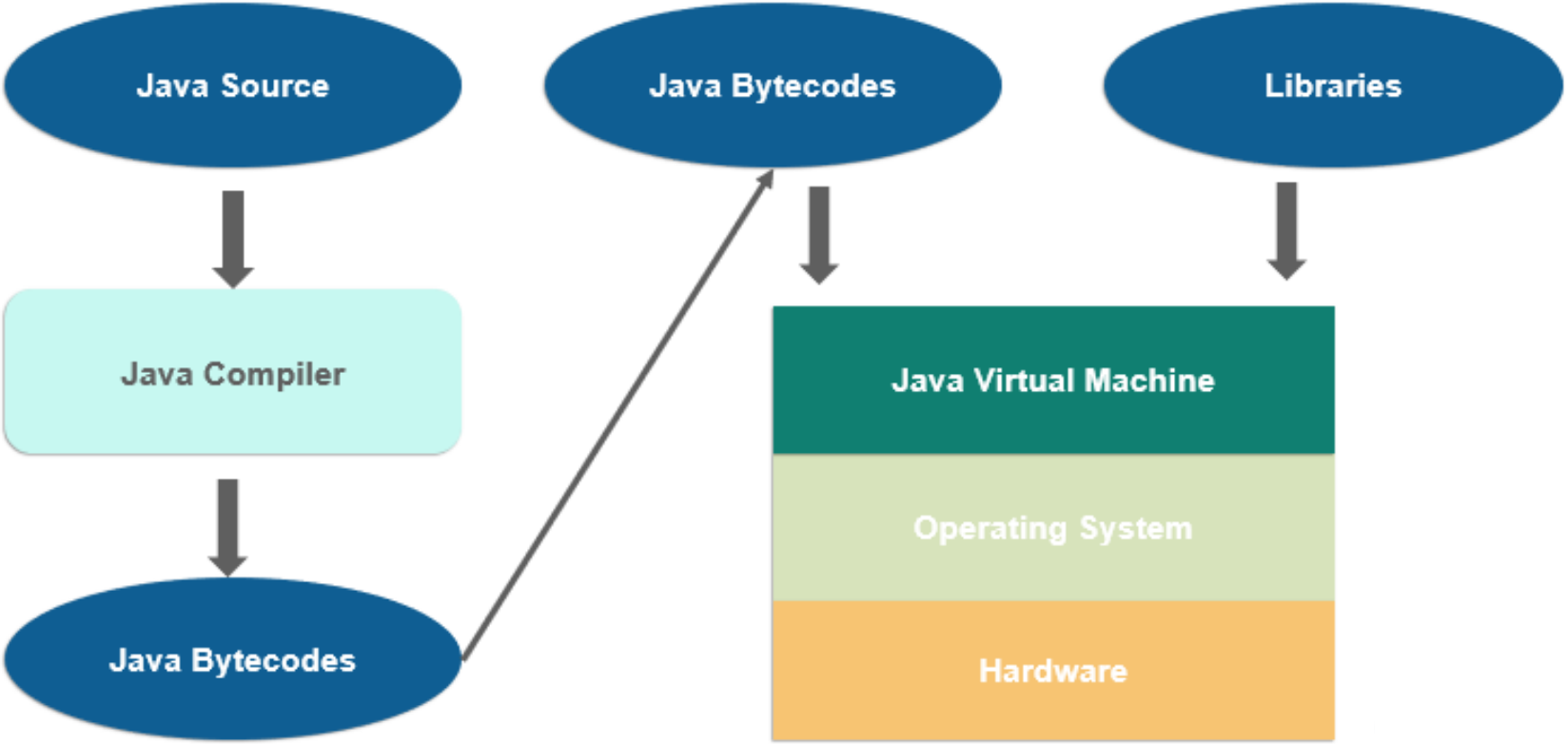
- **Java** est effectivement un langage de programmation compilé. On peut identifier trois grandes phases dans le cycle de vie d'un programme Java :
  - Phase d'écriture du code source
  - Phase de compilation : cette étape implique la conversion du code source Java en code intermédiaire appelé ***bytecode***. Cela se fait à l'aide du compilateur Java.
  - Phase d'exécution : lors de cette phase, le ***bytecode*** est interprété et exécuté par une machine virtuelle Java (**JVM**).
- Contrairement à certains autres langages compilés tels que le C ou le C++, le résultat de la compilation en **Java** n'est pas directement utilisable par l'ordinateur cible → La particularité de Java réside dans l'utilisation de la machine virtuelle.

→ Essentiel du Développement Java: JVM

- Après la compilation, le ***bytecode*** généré n'est pas directement converti en langage machine spécifique à l'ordinateur cible. Au lieu de cela, la **JVM** intervient pour interpréter et exécuter le ***bytecode***, assurant ainsi la portabilité du code Java sur différentes plates-formes.
- En résumé, Java utilise une approche **compilée avec une machine virtuelle**, ce qui offre à la fois la portabilité du code et la possibilité de l'exécuter sur **divers environnements** sans nécessiter une compilation spécifique pour chaque système cible.



→ Essentiel du Développement Java: JVM

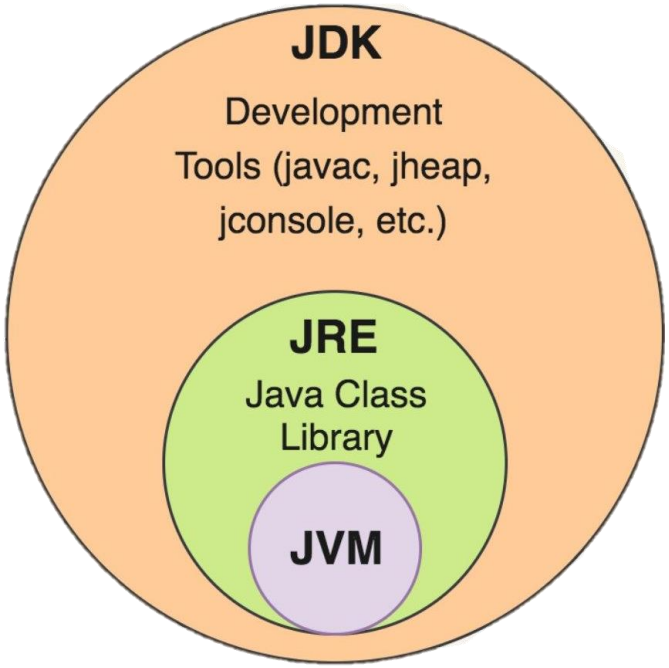


→ JVM, JRE et JDK

- Pour garantir la portabilité des programmes Java sur divers systèmes d'exploitation, la machine virtuelle Java (**JVM**) joue un rôle crucial. Pendant la compilation du code source, celui-ci est transformé en ***bytecode***, un langage intermédiaire ***interprétable*** uniquement par la **JVM**.

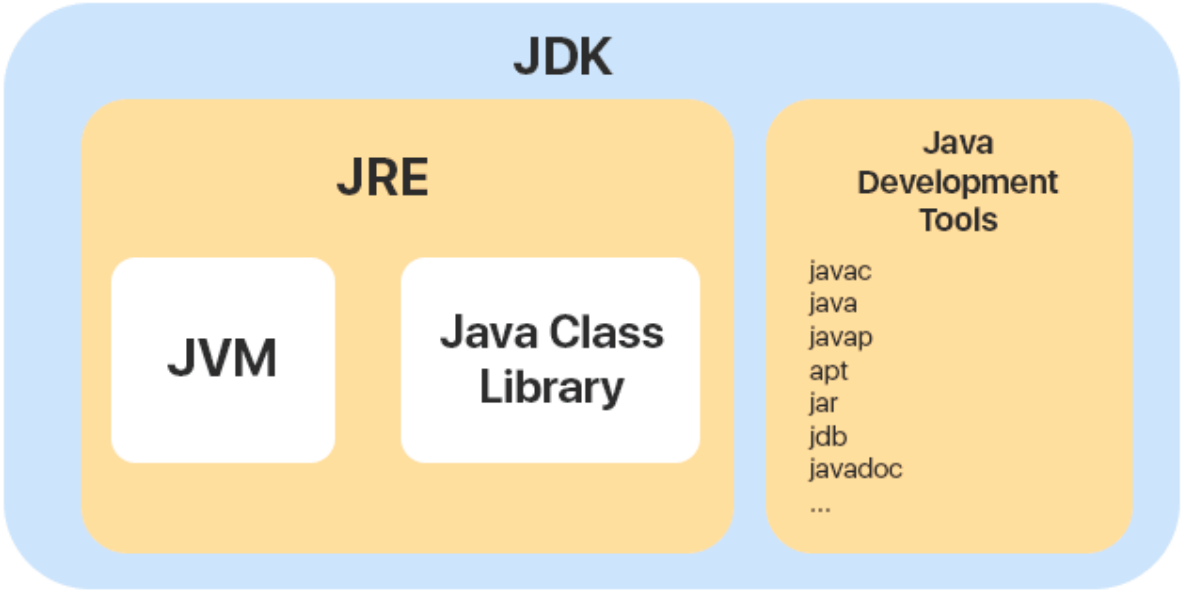
- Le **JRE** (***Java Runtime Environment***) englobe la **JVM**, permettant ainsi l'exécution des applications **Java** sans se soucier des détails liés à un OS particulier.
- Pour simplifier le développement, l'utilisation d'un ***IDE*** (Integrated Development Environment) est recommandée, nécessitant l'installation du **JRE**.

➔ En résumé, le **JDK (Java Development Kit)** est nécessaire pour développer des programmes Java, la **JVM** les exécute, et le **JRE** fournit l'environnement d'exécution, permettant ainsi une portabilité aisée sur différents systèmes d'exploitation.



Outils et environnement de développement java

→ JVM, JRE et JDK



→ Les utilitaires Java

- **Javac :** Compilateur, traduit fichier source .java en fichier bytecode .class
- **Java :** Interpréteur java, lance des programmes
- **Javadoc :** Générateur de documentation d'API
- **Jar :** Utilitaire d'archivage et de compression

→ Développer une application Java

❑ Deux façons d'écrire des programmes Java:

1. **En écrivant le code dans un simple éditeur de texte**

   → Compilation et exécution du code en ligne de commande (DOS)

2. **En utilisant un environnement de développement (IDE)**

   → Eclipse (Télécharger)

   → Netbeans (Télécharger)

   → Microsoft .Net Studio

   → BlueJ

   → JDeveloper…..



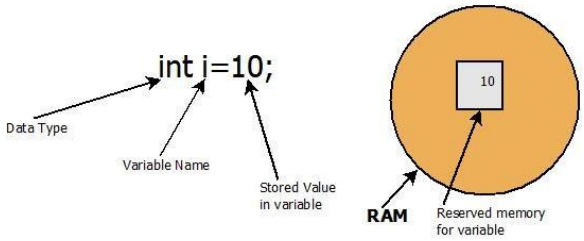

→ Les types de variables

- Une déclaration de variable se fait comme ceci :

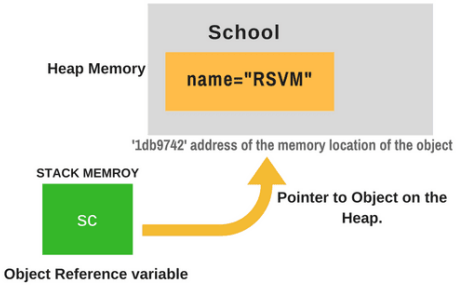
<Type de la variable> <Nom de la variable> ;

- En **Java**, nous avons deux **types de variables** :
  - Des variables de type **simple** ou **primitif :** elles stockent directement des valeurs comme des nombres entiers, des nombres à virgule flottante, des booléens ou des caractères.
  - Des variables de type **complexe** ou des **objets** (ou types de données de **référence**) : elles stockent des références vers des objets en mémoire plutôt que les objets eux-mêmes. Elles incluent les classes, les interfaces, les tableaux…

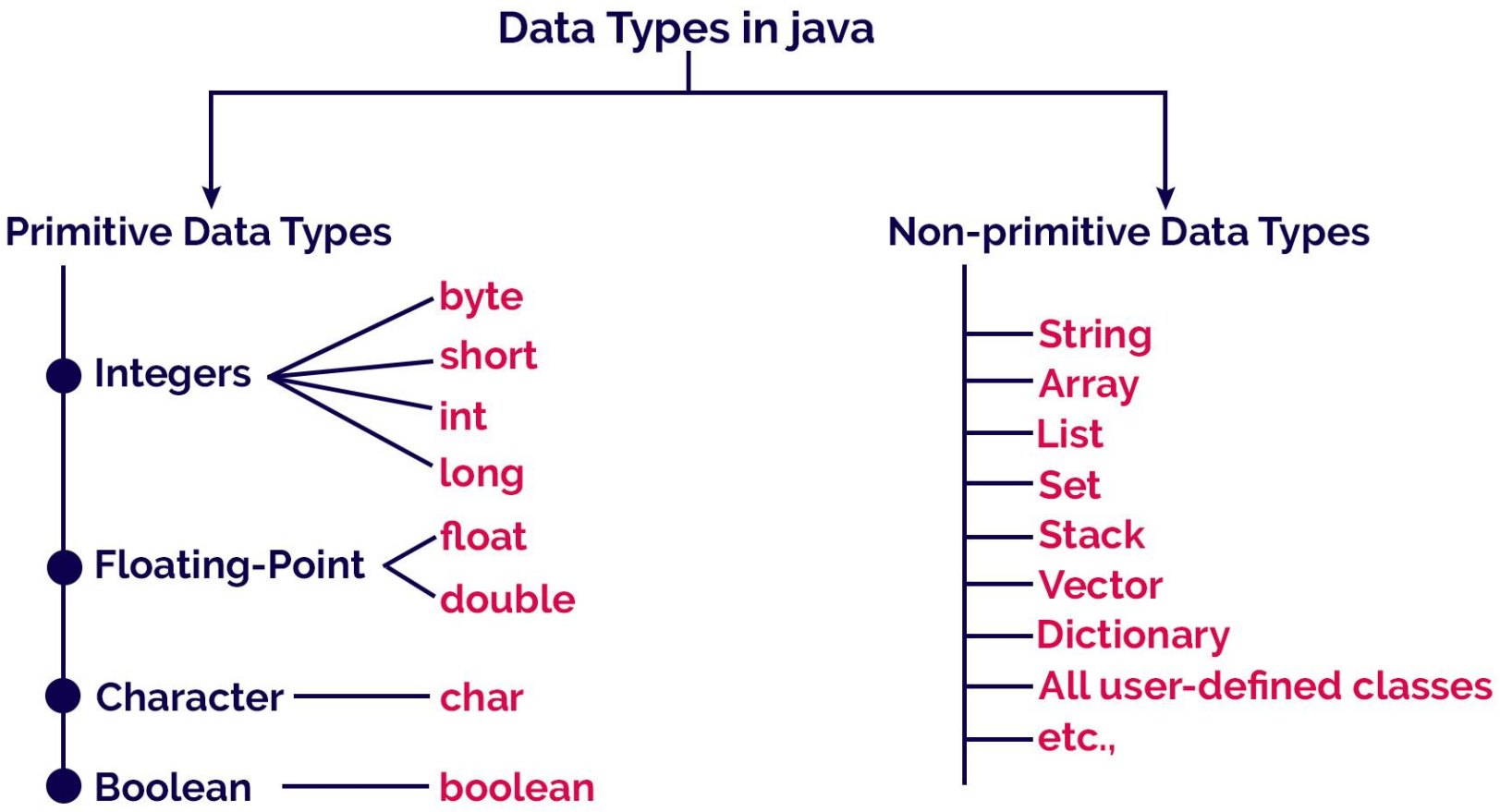
→ Variable de type primitif



→ Variable de type objet



→ Les types de variables



→ Les types de variables

- **Les variables de type simple/Primitifs**

Type	Size	Range	Default
boolean	1 bit	true or false	false
byte	8 bits	[-128, 127]	0
short	16 bits	[-32,768, 32,767]	0
char	16 bits	['\u0000', '\uffff'] or [0, 65535]	'\u0000'
int	32 bits	[-2,147,483,648 to 2,147,483,647]	0
long	64 bits	$[-2^{63}, 2^{63}-1]$	0
float	32 bits	32-bit IEEE 754 floating-point	0.0
double	64 bits	64-bit IEEE 754 floating-point	0.0

→ Les types de variables

❑ *Exemple*

```
int nombre;
String chaine;
float nbrfloat;
int entier = 32;
float pi = 3.1416f;
char carac = 'z';
String mot = new String("Coucou");
int nbre1 = 2, nbre2 = 3, nbre3 = 0;
```

String commence par une majuscule, et lors de l'initialisation, on utilise des guillemets doubles (" ").

→ Les opérateurs arithmétiques

→ **+** : permet d'additionner deux variables numériques (mais aussi de concaténer des chaînes de caractères ! Ne vous inquiétez pas, on aura l'occasion d'y revenir).

→ **-** : permet de soustraire deux variables numériques.

→ ***** : permet de multiplier deux variables numériques.

→ **/** : permet de diviser deux variables numériques (mais je crois que vous aviez deviné).

→ **%** : permet de renvoyer le reste de la division entière de deux variables de type numérique ; cet opérateur s'appelle le modulo.

Exemple

```
nbre2 = nbre2 * 2;
nbre3 = nbre2;
nbre3 = nbre3 / nbre3;
nbre1 = nbre1 - 1;
```

```
nbre1 = nbre1 + 1;
nbre1 += 1;
nbre1++;
++nbre1;
```

→ Les conversions, ou cast

Les conversions, ou "***casts***" en Java, consistent à changer le type d'une variable d'une forme à une autre. On distingue deux types principaux de conversions :

- **Conversion implicite (ou widening) :** Se produit automatiquement lorsque le type de données de petite taille est assigné à un type de données de plus grande taille. Il n'y a généralement pas de perte de données.
- **Conversion explicite (ou narrowing) :** Nécessite une intervention du programmeur pour convertir un type de données de grande taille en un type de données plus petit. Cela peut entraîner une perte de données, et une syntaxe spécifique est utilisée, appelée "***cast***".

→ Les conversions, ou cast

Exemple

```
// Conversion implicite (widening)
int entier = 42;
long longEntier = entier; // Pas besoin de cast, conversion implicite
// Conversion explicite (narrowing)
double reel = 3.14;
int entierArrondi = (int) reel; // Besoin d'un cast pour la conversion explicite
// Attention : perte de précision lors de la conversion
```

→ Ecrire/ afficher un message sur la console

En Java, l'écriture dans la console est généralement réalisée à l'aide de la classe **System.out**

- **`System.out.print` : u**tilisé pour afficher du texte sans saut de ligne.

Exemple

```
System.out.print("Bonjour, ");
System.out.print("comment ça va?");
// Affiche : Bonjour, comment ça va?
```

- **`System.out.println` : u**tilisé pour afficher du texte avec un saut de ligne.

Exemple

```
System.out.println("Bonjour");
System.out.println("monde!");
// Affiche :
// Bonjour
// monde!
```

→ Lire les entrées clavier

- **La classe Scanner**

❑ Pour permettre à Java de **lire l'entrée clavier**, on utilise un objet de type **Scanner**. Cet objet peut être configuré avec différents paramètres.

❑ Avant de lire l'entrée clavier, nous devons instancier un objet **Scanner**.

Exemple

Dans cet exemple, nous utilisons celui qui correspond à l'entrée standard de Java, ***System.in*** et importer la classe **Scanner** depuis le package **`java.util`**

```
import java.util.Scanner;

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

→ Lire les entrées clavier

Avant de lire l'entrée clavier, nous devons instancier un objet Scanner. Voici un exemple simple dans la méthode main d'une nouvelle classe :

```java
import java.util.Scanner; /* Importer la classe Scanner
                             depuis le package java.util */
public class LectureClavier {
    public static void main(String[] args) {
        // Instancier un objet Scanner pour lire l'entrée clavier
        Scanner scanner = new Scanner(System.in);

        // Lire une ligne entrée par l'utilisateur
        System.out.print("Entrez quelque chose : ");
        String userInput = scanner.nextLine();

        // Afficher la saisie de l'utilisateur
        System.out.println("Vous avez saisi : " + userInput);

        // Fermer le scanner pour éviter les fuites de ressources
        scanner.close();
    }
}
```

la méthode **nextLine()** pour lire une ligne de texte entrée par l'utilisateur.

→ Lire les entrées clavier

De façon générale, dites-vous que pour récupérer un type de variable, il vous suffit d'appeler :

next<Type de variable commençant par une majuscule>

Exemple

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
double d = sc.nextDouble();
long l = sc.nextLong();
byte b = sc.nextByte();
```

Attention: il y a un type de variables primitives qui n'est pas pris en compte par la classe Scanner : il s'agit du type **char**.

Voici comment on pourrait récupérer un caractère :

```
System.out.println("Saisissez une lettre :");
Scanner sc = new Scanner(System.in);
String str = sc.nextLine();
char carac = str.charAt(0);
System.out.println("Vous avez saisi le caractère : " + carac);
```

→ Les conditions

- **La structure `if... else`**

Pour créer et évaluer des conditions en Java, nous utilisons des opérateurs logiques tels que:

- `==` : permet de tester l'égalité.
- `!=` : permet de tester l'inégalité.
- `<` : strictement inférieur.
- `<=` : inférieur ou égal.
- `>` : strictement supérieur.
- `>=` : supérieur ou égal.
- `&&` : l'opérateur et. Il permet de préciser une condition.
- `||` : le ou. Même combat que le précédent.
- `? :` : l'opérateur ternaire.

→ Les conditions

Exemple 1

```
int i = 10;
if (i < 0)
System.out.println("le nombre est négatif");
else
System.out.println("le nombre est positif");
```

Exemple 2

```
int i = 0;
if (i < 0)
System.out.println("Ce nombre est négatif !");
else if(i > 0)
System.out.println("Ce nombre est positif !");
else
System.out.println("Ce nombre est nul !");
```

→ Les conditions

❑ La structure `switch - case`

Le switch-case en Java est une structure de contrôle qui permet d'évaluer une expression et d'exécuter des blocs de code en fonction de la valeur de cette expression.

Syntaxe

```
switch (expression) {
    case valeur1:
        // Code à exécuter si l'expression == valeur1
        break;
    case valeur2:
        // Code à exécuter si l'expression == valeur2
        break;
    // ... Autres cas ...
    default:
        // Code à exécuter si aucun des cas ne correspond
}
```

→ Les conditions

❑ La condition ternaire

La condition ternaire, également connue sous le nom d'opérateur ternaire, est une expression concise qui permet de prendre une décision basée sur une condition.

Syntaxe

variable = (condition) ? *valeurSiVrai* : *valeurSiFaux*;

→ Si la condition est **vraie**, la variable prend la valeur "**valeurSiVrai**".
→ Si la condition est **fausse**, la variable prend la valeur "**valeurSiFaux**".

Exemple

```
int a = 5;
int b = 10;
int plusGrand = (a > b) ? a : b;

System.out.println("Le plus grand nombre est : " + plusGrand);
```

→ Les boucles

- **La boucle `while` : u**tilisée lorsque la condition de répétition est vérifiée avant l'exécution du bloc.

```
while (condition) {
    // Bloc de code à répéter
}
```

- **La boucle `do-while` :** similaire à **`while`**, mais garantit au moins une exécution du bloc de code.

```
do {
    // Bloc de code à répéter
} while (condition);
```

- **La boucle `for` :** utilisée lorsque le nombre d'itérations est connu à l'avance.

```
for (initialisation; condition; mise à jour) {
    // Bloc de code à répéter
}
```

→ Les tableaux

En Java, un tableau est une structure de données qui permet de stocker des éléments de même type de manière contiguë. Voici un aperçu succinct des tableaux en Java :

Syntaxe

<*type du tableau*> <**nom du tableau**> **[] = {** <*contenu du tableau*>**};**

Ou <**nom du tableau**> **= new** <*type du tableau*>**[***taille du tableau***];**

❑ ***Exemple 1***

```
int tableauEntier[] = {0,1,2,3,4,5,6,7,8,9};
double tableauDouble[] = {0.0,1.0,2.0,3.0,4.0,5.0};
char tableauCaractere[] = {'a','b','c','d','e','f'};
String tableauChaine[] = {"chaine1", "chaine2"};
//Longueur du Tableau :
int taille = tableauEntier.length;
```

❑ ***Exemple 2***

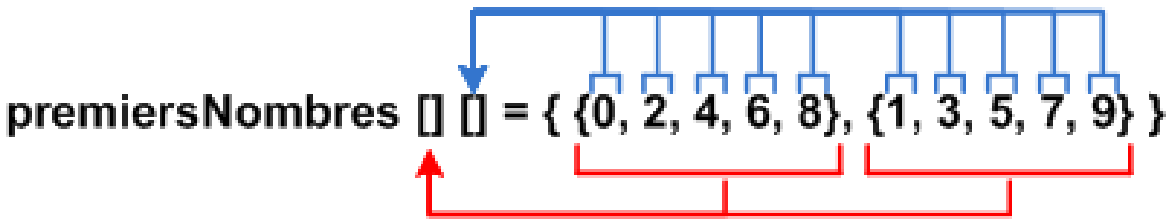
```
int[] nombres; // déclaration
nombres = new int[10]; // création
int[] nombres = new int[10]; // déclaration et création
nombres[0] = 28; // Le 1er élément est 28
```

→ Les tableaux multidimensionnels

❑ *Exemple 1*

```
// Déclaration et initialisation d'un tableau multidimensionnel
int premiersNombres[][] = { {0,2,4,6,8},{1,3,5,7,9} };

// Accès à un élément dans un tableau multimensionnel
int valeur = premiersNombres[1][2];
```



Nous changeons de colonne par le biais de la première paire de crochets

Nous choisissons le terme d'un tableau grâce à la deuxième paire de crochets

→ Les tableaux multidimensionnels

❑ *Exemple 2*

`int[][] matrice = new int[3][];`	« matrice » est une référence vers un tableau contenant lui-même **3** tableaux de taille non définie

`matrice[0] = new int[4];`
`matrice[1] = new int[5];`
`matrice[2] = new int[3];`
	Le premier élément de la matrice est une référence vers un tableau de **4** entiers,…

`matrice[0][0] = 25;`	Le premier élément du premier tableau de la matrice est un entier de valeur **25**

Exercices

Exercice 1

Écrivez un programme Java qui prend deux nombres entiers en entrée et affiche leur somme ainsi que leur différence.

→ Correction

Exercice 2

Écrivez un programme Java qui prend un nombre entier en entrée et vérifie s'il est premier ou non. Affichez un message indiquant si le nombre est premier.

→ Correction

Exercice 3

Recherche du minimum et du maximum dans un tableau

- Demandez à l'utilisateur d'entrer la taille du tableau.
- Demandez-lui ensuite d'entrer les éléments du tableau.
- Affichez la valeur minimale et la valeur maximale du tableau.

→ Correction

Exercice 4

Écrivez un programme Java qui affiche un menu permettant d'effectuer différentes opérations sur un tableau d'entiers :

1. **Remplir le tableau** : Demande à l'utilisateur d'entrer les valeurs du tableau.
2. **Afficher le tableau** : Affiche les éléments du tableau.
3. **Calculer la somme des éléments du tableau**.
4. **Rechercher un élément** : Demande un nombre et affiche son index s'il est trouvé.
5. **Trier le tableau** : Trie les éléments dans l'ordre croissant et affiche le tableau trié.
6. **Quitter le programme**.

ANNEXE

Exercice 1

```java
import java.util.Scanner;

public class SommeEtSoustraction {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Demande à l'utilisateur d'entrer deux nombres
        System.out.print("Entrez le premier nombre entier : ");
        int nombre1 = scanner.nextInt();

        System.out.print("Entrez le deuxième nombre entier : ");
        int nombre2 = scanner.nextInt();

        // Calcul de la somme et de la soustraction
        int somme = nombre1 + nombre2;
        int soustraction = nombre1 - nombre2;

        // Affichage des résultats
        System.out.println("Somme des nombres : " + somme);
        System.out.println("Différence des nombres : " + soustraction);

        scanner.close();
    }
}
```

Exercice 2

```java
import java.util.Scanner;

public class NombrePremierSimple {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Demande à l'utilisateur d'entrer un nombre entier
        System.out.print("Entrez un nombre entier : ");
        int nombre = scanner.nextInt();
        boolean estPremier = true;

        // Vérification si le nombre est premier
        if (nombre <= 1) {
            estPremier = false;
        } else {
            for (int i = 2; i < nombre; i++) {
                if (nombre % i == 0) {
                    estPremier = false;
                    break;
                }
            }
        }

        // Affichage du résultat
        if (estPremier) {
            System.out.println(nombre + " est un nombre premier.");
        } else {
            System.out.println(nombre + " n'est pas un nombre premier.");
        }
        scanner.close();
    }
}
```

Exercice 3

```java
import java.util.Scanner;

public class MinMaxTableau {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Entrez la taille du tableau : ");
        int taille = scanner.nextInt();
        int[] tableau = new int[taille];
        System.out.println("Entrez les éléments du tableau : ");
        for (int i = 0; i < taille; i++) {
            System.out.print("Élément " + (i + 1) + " : ");
            tableau[i] = scanner.nextInt();
        }
        // Initialisation du min et max avec le premier élément du tableau
        int min = tableau[0];
        int max = tableau[0];

        // Recherche du min et du max
        for (int i = 1; i < taille; i++) {
            if (tableau[i] < min) {
                min = tableau[i];
            }
            if (tableau[i] > max) {
                max = tableau[i];
            }
        }
        System.out.println("Valeur minimale : " + min);
        System.out.println("Valeur maximale : " + max);

        scanner.close();
    }
}
```