POO (Java)

BAC II Informatique Industrielle – Genie Logiciel – Management des SI

Par. Ir. Edouard Ngoy Mushame, Assistant, Doctorant en Genie Logiciel Industriel

Chercheur en Genie Logiciel, Intelligence Artificielle, Sciences de données

Objectifs pédagogiques

- Créer une application Java complète
- Maîtriser les bases de Java
- Configurer l'environnement de développement
- Maîtriser la Programmation Orientée Objet
- Exploiter les nouveautés de Java
- Manipuler les données efficacement
- Adopter les bonnes pratiques de développement
- Faire de vous un bon développeur Java.

Contenu du cours

- **Les bases du langage avec la syntaxe Java** : Les variables (types primitifs et type String), Les opérateurs, Les structures (if, else, switch, for, while), Les tableaux, Les méthodes.
- **La programmation orientée objet :** La création de classe (encapsulation et composition), L'héritage par extension, Le polymorphisme, L'abstraction.
- **Les bonnes pratiques du langage pour aller plus loin :** Les exceptions, Les patrons de conceptions, La généricité, Les intelligences artificielles, Les nouveautés du langage (Stream, lambda).

Pre-requis

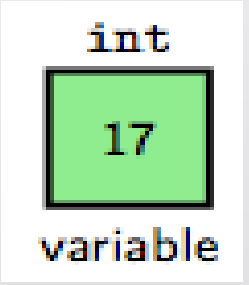
- Algorithmique et structures de donnees
- Conception Orientée Objet (UML)

Les bases du langage avec la syntaxe Java

- Les variables (types primitifs et type String)
- Les opérateurs
- Les structures (if, else, switch, for, while)
- Les tableaux
- Les méthodes.

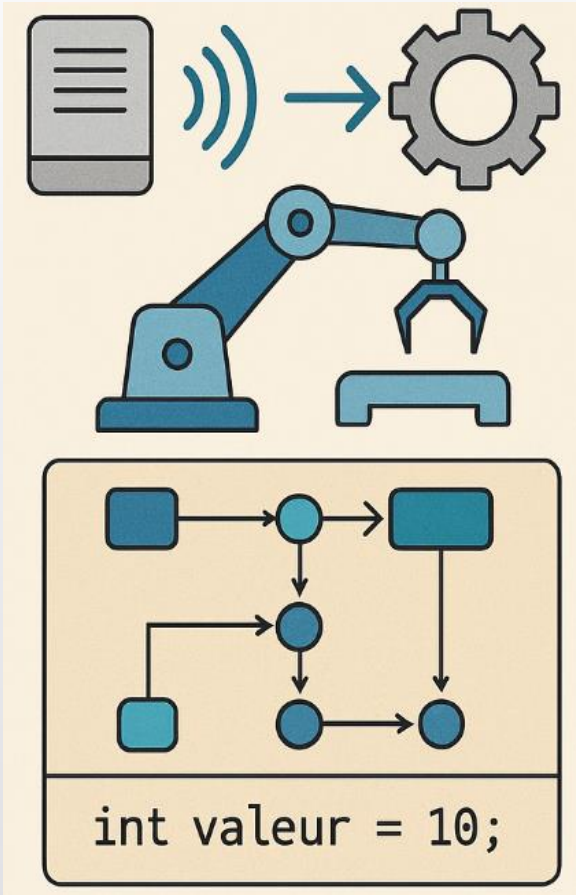
Les variables

- **Definition :** En informatique, une variable est un espace de stockage en mémoire qui associe un nom, appelé identifiant, à une valeur. Cette valeur peut être de différentes natures : nombre, texte, etc. Les variables sont implantées physiquement dans la mémoire du système programmé, qu'il s'agisse d'un ordinateur, d'une carte microprocesseur, etc.
- Il existe plusieurs types de variables, notamment :
  - ***Numériques*** : pour stocker des entiers ou des nombres réels.
  - ***Alphanumériques*** : pour les chaînes de caractères ou textes.
  - ***Booléennes*** : pour des valeurs logiques, comme VRAI ou FAUX.



Déclaration de variables en Java

- En informatique industrielle et instrumentation, les variables sont utilisées pour modéliser des données provenant de capteurs, pour commander des actionneurs, ou encore pour traiter des informations liées au fonctionnement des systèmes automatisés.
- En Java, chaque variable possède un type unique et immuable, ce qui garantit la fiabilité et la robustesse, deux aspects essentiels dans ces domaines.



Exemple de déclaration

- Pour déclarer une variable qui représente la vitesse d'un moteur industriel en tours par minute :
- ***int vitesseMoteur;***
- Dans un système industriel, les valeurs des variables changent constamment en fonction des mesures ou des consignes. Pour affecter une valeur à une variable, on utilise l'opérateur d'affectation :
- ***vitesseMoteur = 1500;***

Déclaration et initialisation combinées

- Dans les systèmes embarqués ou dans un programme informatique, il est courant de déclarer et d'initialiser les variables en même temps pour réduire les risques d'erreurs.
- ***int temperatureSeuil = 15;***
- ***Double pourcentageEtudiant = 63.2;***

L'utilisation de constantes

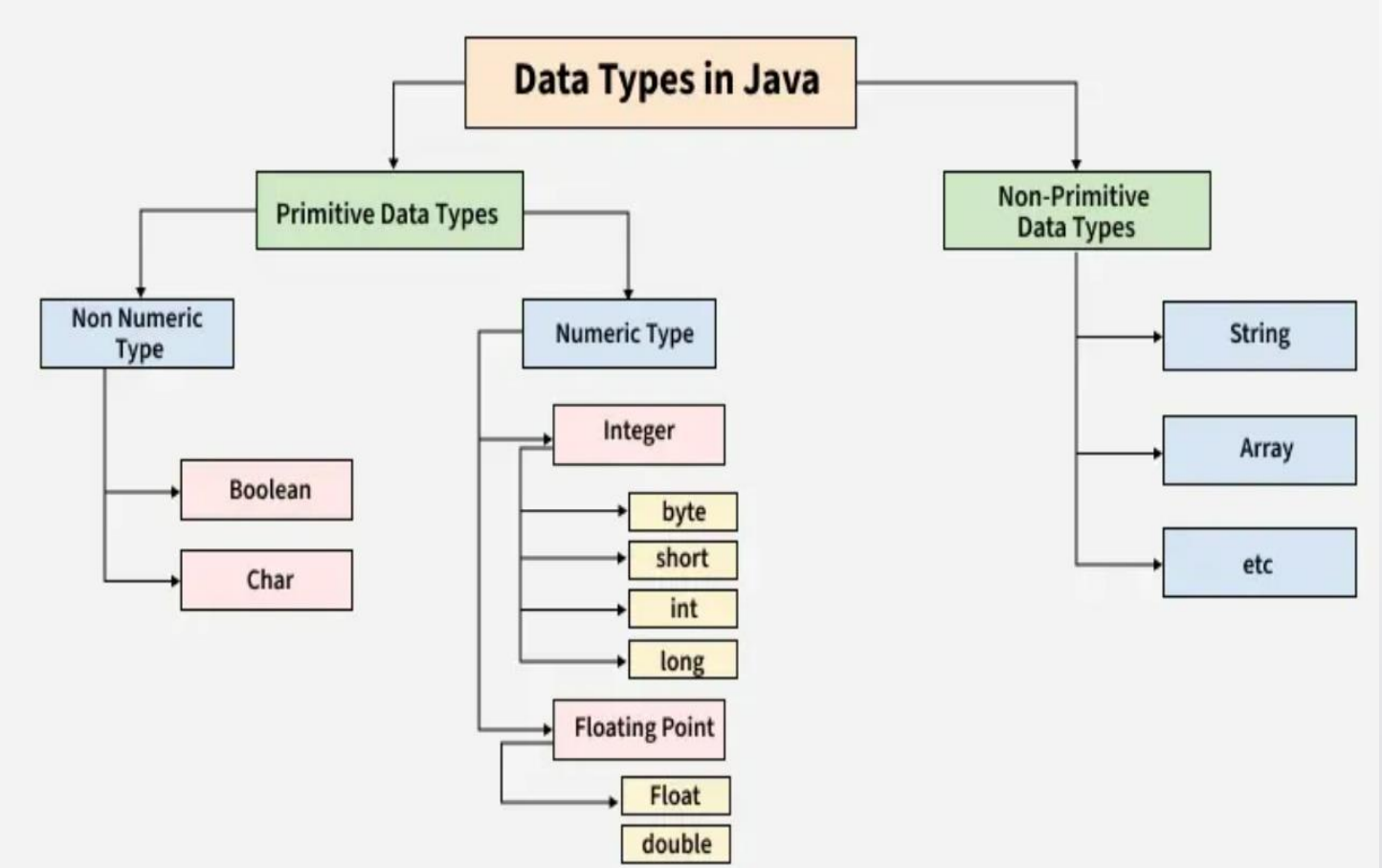
- Les constantes sont particulièrement utiles pour définir des seuils, des paramètres physiques, ou des identifiants d'équipements qui ne changent jamais.
- ***Exemple d'utilisation***
- Définissons une constante représentant le nombre maximal de cycles d'un moteur avant maintenance :
- ***Final int CYCLE_MAX = 15000000;***

Conventions de nommage adaptées

- Pour faciliter la maintenance et la compréhension des programmes industriels, il est important de nommer les variables et constantes en fonction de leur rôle dans le système :
- **Variables** : Utilisez la casse chameau pour les noms descriptifs.
  - Eg. ***double courantMoteur;***
- **Constantes** : Utilisez des majuscules pour les noms et des underscores pour séparer les mots.
  - Eg. ***final int MAX_TEMP_C = 1000;***

Types de données

- En Java, chaque donnée appartient à un type précis, qui détermine sa nature et les opérations qu'elle peut subir. Ces types se divisent en deux grandes catégories :
  - Les types primitifs
  - Les types objet



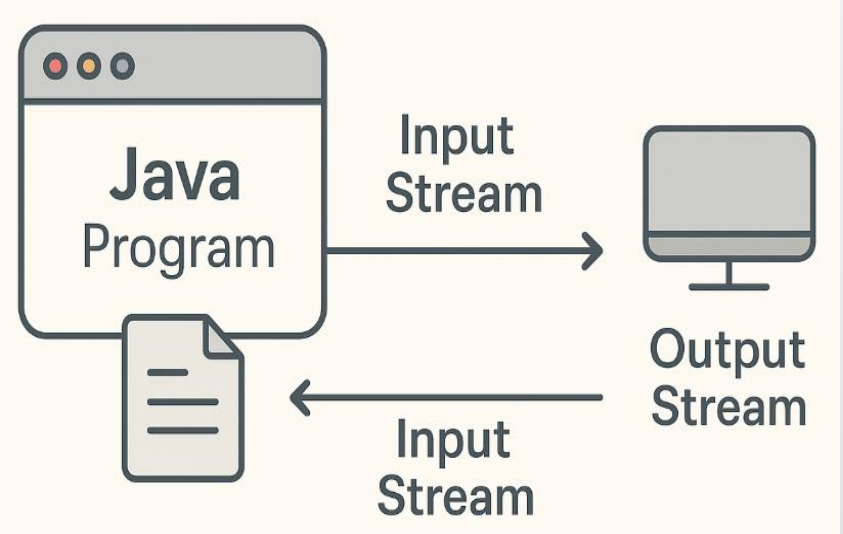
Les opérateurs

- Les **opérations** sur ces données. Ce point vous initiera aux **opérateurs**, ces outils qui permettent de manipuler les valeurs, et aux **expressions**, ces combinaisons d'opérateurs et de valeurs qui forment le langage de vos programmes.

Operators	Type
+,-,/,*,%	Arithmetic Operators
==, !=, >, <, <=, >=	Comparison Operators
&&, (II), !	Logical Operators
=, +=, -=, *=, /=, %=	Assignment Operators
++, --	Increment and Decrement Operators
&, |, ^,~, >>, <<	Bitwise Operators

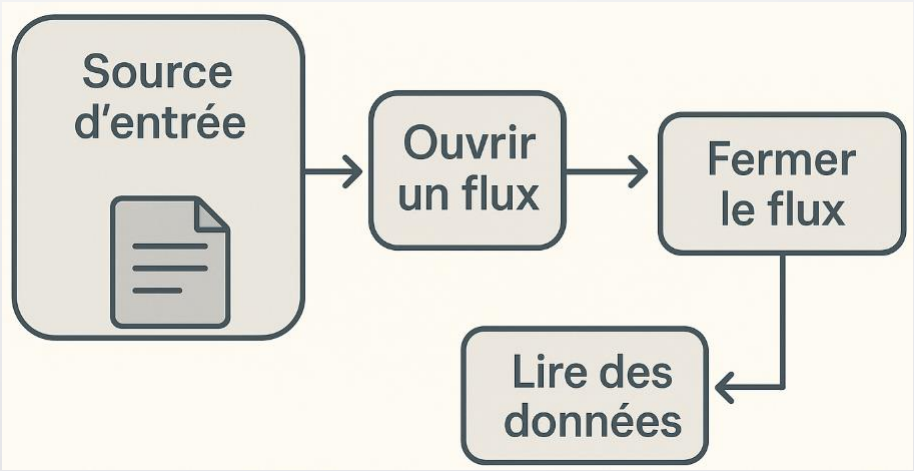
Les entrées / sorties

- Lorsqu'un programme doit écrire des données, il ouvre un flux vers une destination de sortie, telle qu'un fichier sur le disque dur, une zone mémoire dédiée, ou encore un périphérique d'affichage comme un écran ou une imprimante.
- L'écriture s'effectue elle aussi de façon séquentielle, ce qui signifie que les nouvelles données sont ajoutées les unes après les autres dans l'ordre spécifié par le programme.



La Lecture

- En Java, la lecture d'une information depuis une source (fichier, mémoire, entrée standard, base de données, etc.) suit plusieurs étapes essentielles. Cette opération peut être effectuée à l'aide de différentes classes de l'API Java, comme *`FileReader`*, *`BufferedReader`*, *`Scanner`* ou encore *`FileInputStream`* pour les fichiers binaires.



Lecture des flux Entrées à partir d'une saisie-écran (clavier)

- La création de flux Entrée à partir d'une saisie-écran est faite en combinant trois classes. D'abord une instance de la classe `InputStreamReader` dont l'argument d'instanciation est la variable `static System.in` dont l'appel invite l'utilisateur à faire une saisie via le clavier.
- Ensuite, cette instance de la classe `InputStreamReader` sert d'argument d'instanciation de la classe `BufferedReader` qui permet de bufféri ser les textes saisis par l'utilisateur

```
import  java.io.*;

public class Main {
    public static void main(String[] args) {
        BufferedReader rd = new BufferedReader(new InputStreamReader(System.in));
    }
}
```

- **NB: Ces classes viennent de la bibliothèque java.io**

Lecture d'un fichier de texte plat : usage de la classe FileReader

- Dans cette section nous montrons l'usage de la classe `FileReader` pour lire et afficher le contenu d'un fichier de texte plat. L'exemple ci-dessous illustre l'utilisation de la classe `FileReader`.

```
import  java.io.*;

public class Main {
    public static void main(String[] args) {
        try {
            BufferedReader br = new BufferedReader(new FileReader( fileName: "chemin/vers/le_fichier.ext"));
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Écriture de fichiers pour enregistrer des données

- A l'inverse des flux Entrées, les flux Sorties servent à conduire les données vers un système extérieur au programme Java. Tout comme les flux Entrées, les flux de Sorties peuvent être orientés vers un périphérique d'affichage (écran), un fichier, un réseau distant(socket), un espace mémoire, etc.
- Les classes de gestion des flux Sorties les plus couramment utilisées sont: `FileOutputStream`, `DataOutputStream`, `BufferedOutputStream`, `ObjectOutputStream`, `ByteArrayOutputStream`, `BufferedWriter`, `Filewriter`, `PrintWriter`, `BufferedWriter`, `CharArrayWriter`, `FilterOutputStream`, etc.

Ecriture sur la sortie standard et sur l'écran : les méthodes `System.out.print()` et `System.out.println()`

- Les méthodes `println()` et `print()` sont deux méthodes static de la classe `System.out` qui nécessitent peu de commentaires. La seule remarque qu'on peut faire est que, à chaque appel, la méthode `println()` affiche le résultat sur une nouvelle ligne dans la console, alors que la méthode `print()` affiche le résultat sur la même ligne que les appels précédents.

```
import  java.io.*;

public class Main {
    public static void main(String[] args) {
        BufferedReader rd = new BufferedReader(new InputStreamReader(System.in));
        double temperaturePiece;
        try {
            temperaturePiece = Double.parseDouble(rd.readLine());
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        System.out.println(temperaturePiece);
    }
}
```

Application 1 : Calcul du Rendement d'une Machine

- Dans une ligne de production industrielle, il est essentiel d'évaluer la performance d'une machine. On souhaite écrire un programme Java capable de calculer son **taux de rendement journalier**.
- Votre programme devra :
- **Demander à l'utilisateur :** le nombre total de pièces produites (**entier**), le nombre de pièces conformes (**entier**).
- **Déclarer une constante** représentant le pourcentage maximal : final int MAX_PERCENT = 100;
- **Calculer :** le nombre de pièces rejetées, le taux de rendement suivant la formule :
- $Rendement = \frac{Pieces\ OK}{Pieces\ Conformes} * 100$
- **Afficher** de manière claire : le total des pièces produites, le nombre de pièces conformes, le nombre de pièces rejetées, le taux de rendement (en %).

Application n°2 — Calcul de la Consommation Énergétique d'un Moteur

- Une machine industrielle fonctionne grâce à un moteur ayant une certaine puissance électrique exprimée en kilowatts (kW).
  On souhaite calculer :
- l'énergie consommée en **kilowattheures (kWh)**, puis le **coût total** de cette consommation.
- Votre programme Java doit :
- **Demander à l'utilisateur :** la puissance du moteur en kW (**double**) la durée de fonctionnement en heures (**double**) le prix du kWh en monnaie locale (**double**)
- **Déclarer une constante :** final double TAUX_CONVERSION = 1.0; // 1 kW * 1 heure = 1 kWh
- **Calculer :** l'énergie consommée : $Energie = Puissance * Duree$ et Cout total : $Cout = Energie * Prix\ du\ kWh$
- **Afficher clairement :** la puissance du moteur la durée de fonctionnement, l'énergie totale consommée (kWh), le coût total à payer