JAVA - INTRODUCTION

Wijin

v1.0 2024

Objectifs

- Utiliser correctement le langage Java
- Mettre en œuvre les principes de la programmation orientée objet en Java
- Compiler, exécuter, déboguer un programme Java

Plan

- Présentation de Java
- Architecture
- Rappels
- Notion de classe et d'objet
- Portabilité
- Notion de type
- Gestion de la mémoire

Java - Présentation

Java - Présentation

- **Prérequis Java** : environnement de développement
  - Utilisation de l'IDE **Eclipse** pour la formation
  - D'autres IDE sont disponibles : IntelliJ, VS Code, …
  - Utiliser la dernière version d'Eclipse JEE : **Eclipse IDE for Enterprise Java Developers**

    https://www.eclipse.org/downloads/packages/

Java - Présentation

- **Prérequis Java** : JDK (**J**ava **D**evelopment **K**it)
  - Installer une version 17 de la JDK en version **OpenJDK**
  - Préconisation : privilégier une version **LTS** (« **L**ong **T**erm **S**upport »)

Java - Présentation

- **Prérequis Java** : JDK (**J**ava **D**evelopment **K**it)
  - Préconisation : privilégier une version **LTS** (« **L**ong **T**erm **S**upport »)
  - Une version **LTS** est maintenue jusqu'à la sortie de la prochaine **LTS** (environ tous les 3 ans)
    - → **Patchs correctifs et de sécurité !**
  - Une version non-LTS n'est **plus maintenue** dès lors que la version suivante est mise à disposition

Java - Architecture

Java - Architecture

- **3** composants
  - Le **langage**
  - La machine virtuelle (**JVM**)
  - La **plateforme**

Java - Architecture

- Le **langage** Java
  - Ecriture des programmes **Java**
  - Orienté **objet**
  - Compilation du code **Java** → **byte code** (langage machine portable)

Java - Architecture

- La **machine virtuelle** Java
  - **JVM** = **J**ava **V**irtual **M**achine
  - Interprète et exécute le **byte code**
  - **JVM** disponible pour chaque système d'exploitation
    - « Write once, run anywhere ! »

Java - Architecture

- Les **versions** de Java
  - Java a évolué au cours du temps
  - **Dernière version disponible** : Java **20** (mars 2023)
  - Dernière version avec **support à long terme** : Java **17** (support jusqu'en 2027)
  - Prochaine version **LTS** : Java **21** (septembre 2023)

Java - Architecture

- Gestion de la **mémoire**
  - L'interpréteur Java (JVM) sait quels emplacements mémoires il a **alloué**
  - Il sait déterminer quand un objet alloué n'est **plus référencé** par un autre objet ou une variable
  - Ramasse-miette (« **Garbage collector** ») : détecte et détruit ces objets non référencés (libération **automatique** de la mémoire)

Java - Quelques rappels

Java – Quelques rappels

- Notion de **variable**
  - En Java, une **variable** = une **donnée** avec un nom :
    - Peut être un objet issu d'une classe
    - Peut être un type primitif (int, char, float, etc…)
    - Peut être lue ou écrite
  - En Java, les variables sont typées, c'est-à-dire que chaque donnée possède un **type**

Java – Quelques rappels

- Notion de **variable**
  - En Java, les **noms** de variables :
    - peuvent comporter jusqu'à 247 caractères
    - ne peuvent comporter d'espace
    - ne peuvent commencer par un chiffre
  - Le nom d'une variable **ne peut pas être un mot clé** de Java, comme par exemple class, try, do, while, …
  - Le nom d'une variable est **sensible à la casse**

Java – Quelques rappels

- Notion de **variable**
  - Bonne pratique de **nommage** des variables :
    - Donner un nom **révélateur**
      - Eviter « a », « b », …
    - **Eviter un nom qui désinforme** sur l'usage
      - Eviter les abréviations qui peuvent avoir plusieurs sens
    - Utiliser des noms **distinctifs** pour distinguer les différentes variables
      - Eviter les noms trop courts (comme « a1 », « a2 », …)
    - **Ne pas préciser le type** dans le nom de la variable
      - Ne pas mettre « monObjetString » ou « sMonObjet », « iMonObjet »

Java – Quelques rappels

- Notion de **variable**
  - En Java, la déclaration d'une variable se fait de la sorte :

    TYPE nomvariable;

  - Exemple de déclaration d'un nombre de jours :

    int nombredejours;

Java – Quelques rappels

- Notion de **variable**
  - En Java, une variable doit être **déclarée avant d'être utilisée**
  - Il est possible **d'initialiser** les variables lors de leur déclaration : **affectation**

```
int nombredejours = 365;
```

Java – Quelques rappels

- **Portée** des **variables**
  - **3 niveaux** de portée en Java
  - Portée signifie **jusqu'où la variable reste accessible**, et **existante**
  - **Attention** : ne pas confondre portée et visibilité !
    - La visibilité permet de **limiter l'accès** à des propriétés ou des méthodes

Java – Quelques rappels

- **Portée** des **variables**
  - **Portée locale**
  - La portée la plus petite est celle d'une variable décrite dans une fonction

```
public String appel(String message) {
    String local = message;
    System.out.println(local);
    return "OK";
}
```

Java – Quelques rappels

- **Portée** des **variables**
  - **Portée locale**
    - Lors de l'appel à la fonction « appel » la variable locale n'aura vécu **que le temps d'exécution** de cette fonction, et sera perdue complètement à la sortie

```
public static void main(String[] args) {
    TestPortee portee = new TestPortee();
    portee.appel("machaine");
    System.out.println("OK");
}
```

Java – Quelques rappels

- **Portée** des **variables**
  - **Portée de niveau objet**
    - La portée de niveau supérieure est la portée de niveau **objet** :
      - Les variables déclarées dans une classe, et qui existeront réellement lorsque les objets seront créés, auront une **durée de vie égale à celle de l'objet**
      - C'est-à-dire qu'entre sa création et sa destruction, toutes les fonctions de cet objet peuvent **accéder** aux variables de cet objet, sans contrainte
      - Les variables, appelées propriétés de l'objet, sont **partagées** par les différentes fonctions de l'objet

Java – Quelques rappels

- **Portée** des **variables**
  - **Portée programme**
    - Niveau de portée supérieur dans Java
    - L'utilisation du mot clé **static** en Java permet de faire en sorte que la visibilité d'une variable soit remontée au niveau programme, dont accessible à n'importe quel objet

Java – Quelques rappels

- **Portée** des **variables**

  - **Synthèse**

Java – Quelques rappels

- Notion de **fonctions**
  - Une fonction est tout d'abord une **déclaration** :
    - Exposition de lignes de codes accessibles via un **nom**
    - Attente possible de **paramètres** en entrée, avec leur **type**
    - **Retour** possible de quelque chose en sortie, avec son **type**

Java – Quelques rappels

- Notion de **fonctions**
  - Exemple de déclaration :

```
11⊖    public Integer ajouter (Integer valeur1, Integer valeur2 ) {
12         Integer somme = valeur1 + valeur2;
13         return somme;
14     }
```

  - Exemple d'appel :

```
9          Integer resultat = ajouter (10,20);
```

Java – Quelques rappels

- **Algorithmique**
  - **Algorithme** = **recette** pour résoudre un **problème**
  - 3 structures principales
    - L'affectation
    - L'alternative
    - La répétition

Java – Quelques rappels

- **Algorithmique**
  - **Algorithme** = **recette** pour résoudre un **problème**
  - L'**affectation** :
    - Consiste à transférer le contenu d'une variable dans une autre variable
    - En général, cette affectation a pour raison la sauvegarde d'une variable dans une autre
    - Elle se traduit, dans plusieurs langages, par l'utilisation du caractère « **=** », qui signifie affecter, déplacer

Java – Quelques rappels

- **Algorithmique**
  - **Algorithme** = **recette** pour résoudre un **problème**
  - L'**alternative** :
    - Consiste à prendre des chemins de code différents selon la valeur d'une variable
    - Elle se traduit souvent dans les langages avec l'instruction « **if/else** » mais également souvent « **switch** »

Java – Quelques rappels

- **Algorithmique**
  - **Algorithme** = **recette** pour résoudre un **problème**
  - La **répétition** :
    - Consiste à exécuter une séquence de code donnée tant qu'une condition est réalisée (« boucle »)
    - Elle se traduit souvent dans les langages avec les instructions comme « for(;;) », « do.. », « while() », « forEach() », etc…

Java – Quelques rappels

- **Algorithmique**

  - Exemple

```
10  Variable X en Entier
20  Debut
30  X ← 0
40  Ecrire "Saisir un nombre entre 1 et 3"
50  TantQue X < 1 ou X > 3
60   Lire X
70    Si X < 1 ou X > 3 Alors
80      Ecrire "Mauvaise saisie, autre essai"
90    FinSi
100 FinTantQue
110 Fin
```

Affectation

Répétition

Alternative

Java – Quelques rappels

- **Algorithmique**

  - Exemple (en Java)

```
16          Scanner scanner = new Scanner(System.in);
17          Integer N;
18          N=0;
19          System.out.println("Entrez un nombre entre 1 et 3");
20          while (N < 1 || N > 3) {
21              N = scanner.nextInt();
22              if (N < 1 || N > 3) {
23                  System.out.println("Saisie erronée. Recommencez");
24              }
25          }
```

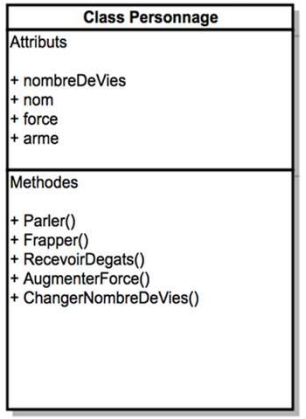
Java - Notion de classe et d'objet

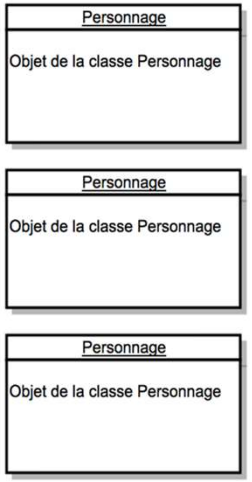
Java – Notions de classe et d'objet

- **Qu'est-ce qu'une classe ?**

  - Une classe est un « **moule** » qui nous permet de créer plusieurs objets à partir de ce moule

  - Exemple : Plusieurs personnages d'un jeux vidéo
    - On parle alors d'**instances** de la classe Personnages





Java – Notions de classe et d'objet

- **Comment créer une classe ?**
  - **Nom** : c'est quoi ?
    - Employé, compte bancaire, joueur, document, album...
  - **Attributs** : ce qui décrit la classe
    - Largeur, hauteur, couleur, type de fichier, score…
    - On les appelle aussi des propriétés
  - **Comportements** : que peut-elle faire ?
    - Jouer, ouvrir, chercher, enregistrer, imprimer…
    - On les appelle le plus souvent des méthodes

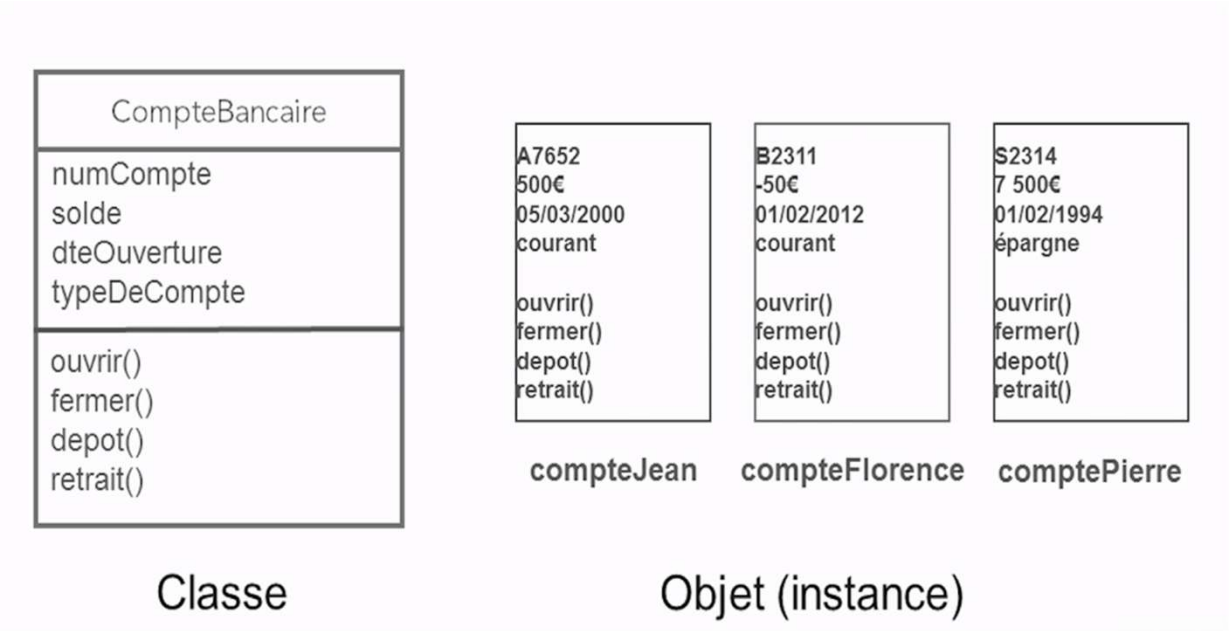
Java – Notions de classe et d'objet

- **Qu'est-ce qu'un objet ?**

  - **Un objet =** une instante **unique** d'une classe

Java – Notions de classe et d'objet

- **Classe / Objet**



Java – Notions de classe et d'objet

- **Programmation orientée objet en Java**
  - Un **programme** Java est défini sous forme de **classes**
  - Pas de code Java sans classe !
  - Les instructions Java sont positionnées dans des **méthodes** et toutes les méthodes sont implémentées dans des **classes**

Java – Notions de classe et d'objet

- **Les classes natives du langage Java**
  - La plupart des langages aujourd'hui proposent des classes préconçues prêtes à l'emploi
  - En Java, il s'agit de la **plateforme Java** !
  - On peut utiliser ces classes, en plus de créer les nôtres
  - String(), Date(), Array()....

Java – Notions de classe et d'objet

- **Exercice : créer une classe**

  - Dans Eclipse :
    - Créer un projet
    - Créer une classe « Test »

```
1  package formationjava;
2
3  public class Test {
4
5      public static void main(String[] args) {
6          int entier = 100;
7          System.out.println(entier);
8      }
9
10 }
```

Java - Portabilité

Java – Portabilité

- Dans plusieurs langages, comme le C, le résultat de la compilation produit des instructions qui sont **compréhensibles sur un type de processeur donné**
  - Le portage sur une autre architecture demande alors du travail de **conversion**, de **recompilation**
- Spécificité du langage **Java** = **interprété**
- Avec Java, le compilateur produit du « byte code », **non compris** par les différents types de processeurs

Java – Portabilité

- L'astuce est nommée « **WORA** » par Sun aux origines de Java (« Write Once, Run Anywhere ») :
  - Utiliser une « **JVM** » (Java Virtual Machine) qui va **lire** le byte code, et le **traduire** en code compréhensible par un type de processeur donné
  - Nécessite d'avoir une **JVM par type d'architecture**

Java – Portabilité

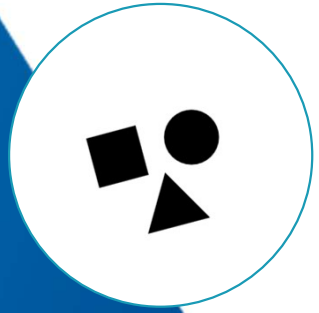
- **Conséquence importante** : Java peut s'exécuter sur des plateformes différentes, à condition qu'il existe une JVM

- Aujourd'hui, des JVM existent pour toutes les plateformes

Java – Portabilité

- **Exercice :**
  - Projet
    - src/ → créer une classe (Test)
    - Point d'entrée (main)
      - Déclaration int entier = 100;
      - System.out.println(entier);
    - Génération du fichier **.class** (résultat de la compilation : contient le **byte code**)
      - Navigator pour voir dans bin, le fichier .class

Java - Notion de type



Java – Notion de type

- **Typage fort** : Java est un langage **typé**

- Si **incompatibilité** entre information positionnée et type de la variable
  → **erreur à la compilation**

- Différence des langages non typés : erreur lors de **l'exécution**

Java – Notion de type

- **Synthèse** : une déclaration doit être typée dans Java

- Exception à partir de Java 10 : utilisation du mot clé **var**
  - mais nécessite quand même de fournir à un moment le type !

Java – Notion de type

- **Exercice :**
  - Reprendre l'exemple précédent
    - Voir les autres types (String, tableau, type correspondant à une classe, …)
    - float (float variable = 10.34f;)
    - Tenter de charger entier dans variable (erreur de typage à la compilation !)
    - Forcer le typage avec le « casting » (entier = (int) variable;)

Java - Gestion de la mémoire

Java – Gestion de la mémoire

- Le travail de libération de la mémoire allouée est réalisé **directement** par Java !

- Libération **automatique** lorsqu'un objet est « hors de portée »

Java – Gestion de la mémoire

- **Conséquence** : il n'existe pas dans Java d'instruction spécifique de suppression d'objet

- Solution pour **supprimer délibérément** un objet =

  charger sa référence à **null**

Java – Gestion de la mémoire

- La mémoire libérée n'est pas forcément rendue de suite !
- **Garbage Collector** : sous système de Java, qui se charge de récupérer la mémoire
- Il est possible de déclencher soi-même le Garbage Collector (System.gc), et donc de libérer effectivement la mémoire des objets devenus inaccessibles

Java – Gestion de la mémoire

- Exemple de **libération mémoire**

- Classe nommée **Memoire**

  - 1/ Un objet de type **Memoire** est créé à partir du programme principal, puis il est **supprimé explicitement** en lui affectant **null** dans la référence créée

  - 2/ Un objet créé disparaît de lui-même car sa portée est limitée à la fonction dans laquelle il a été déclaré et créé

Java – Gestion de la mémoire

- Exemple de **libération mémoire**

```
1 package formationjava;
2
3 public class Memoire {
4
5     public void creationObjet() {
6         Memoire memoire1 = new Memoire();
7     }
8
9 }
```

Java – Gestion de la mémoire

- Exemple de **libération mémoire**

```java
package formationjava;

public class TestMemoire {

    public static void main(String[] args) {

        Memoire memoireLocale = new Memoire();
        memoireLocale.creationObjet();
        memoireLocale = null;
    }
}
```

Java - Quiz

Java – Quiz

- Java est un langage **fortement typé**
  - Vrai
  - Faux

Java – Quiz

- Le résultat du compilateur Java est **directement exécutable** sur le processeur

  - Vrai

  - Faux

Java – Quiz

- La gestion et notamment la libération **mémoire** en Java est réalisée **dans le code**

  - Vrai

  - Faux

MERCI POUR VOTRE ATTENTION

Faites-moi part de vos remarques concernant le cours afin qu'il soit amélioré pour les prochaines sessions : nicolas.sanou@wijin.tech