Java - les fondamentaux

EPSI 2023-2024

Planification des cours

- Vendredi 8 décembre de 16h à 20h
- Jeudi 21 décembre de 9h à 13h **à distance**
- Jeudi 1er février de 9h à 13h
- Vendredi 23 février de 9h à 13h et 14h à 18h **examen env. 2h**

Contact -> sebastien.guerlet@ecoles-epsi.net

Le cours dans le bloc de compétence

TPRE501	MSPR Bloc E6.1 CDA	Mise en situation professionnelle reconstituée : Développement et deploiement d'une application dans le respect du cahier des charges Client
		UF - Développement Front End et Back End (40 heures)
DEVE514	Dev Java	Developpement en Java
DEVE515	Java et Framework JS	Javascript et Framework JS au choix du campus
		UF - Développement mobile (32 heures)
MOBE516	Dev Env Mobile Frame	Développement environnement mobile avec utilisation d'un framework (langage et framework au choix du campus)
MOBE517	Atl Dev Mobile	Atelier Développement mobile (Swift ou Java pour androïd ou Kotlin pour androïd au choix du campus)
		UF - Développement & Testing (34 heures)
FRWE518	Atl Lang prog Ja Py	Atelier Le langage de programmation Java/Python et le framework Django/Spring (au choix du campus)
TESE519	Atl Méthodo test uni	Atelier de méthodologie autour des tests : tests unitaires et tests de charges
TESE510	Atl Industri test	Atelier : L'industrialisation des processus de test

Introduction

Un peu d’historique

Java est un langage orienté objet (OO) inventé par James Gosling au début des années 1990, chez Sun Microsystem

Première version publique en 1995

Une ouverture de l’environnement de développement et d’exécution en 2007 : OpenJDK

Rachat de Sun par Oracle en 2009

Java, les promesses

Java est un langage **multiplateformes**, dont le slogan - souvent décrié - est **write once, run anywhere (WORA)**

C'est un langage objet **fortement typé**

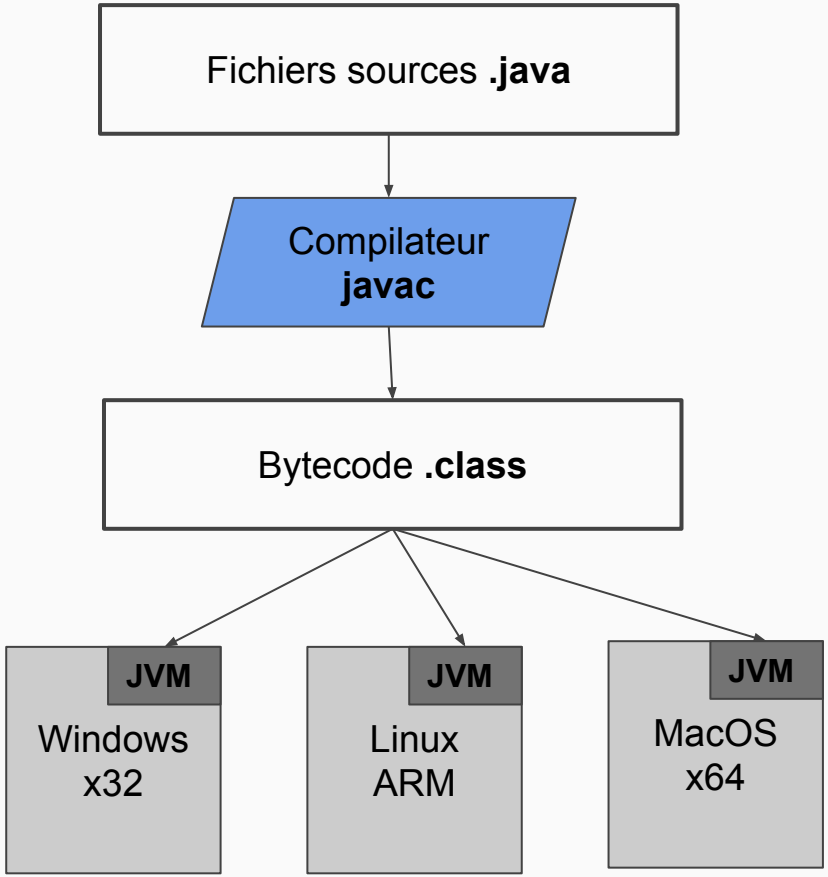
Le code source est d'abord compilé en un code intermédiaire appelé **bytecode**, indépendant de la machine. Puis exécuté au sein d'une machine virtuelle, la **JVM**

La JVM est outillée d'un **garbage collector**, qui se charge de la libération mémoire

La compatibilité **ascendante** est totale. Une force mais aussi une contrainte, l'ajout de nouvelles fonctionnalités au langage est complexe, il n'évolue que très lentement.

Compilation et exécution

A mi-chemin entre un langage compilé (directement exécutable) et interprété (exécution depuis le code source)



Java, JDK, J2EE, Java EE , Jakarta EE… ?!?

Pour une même terminologie, différents concepts

- Java le langage
- Java la plateforme, l'écosystème

Des termes et des versions qui évoluent avec le temps, les rachats ou donation, les licences

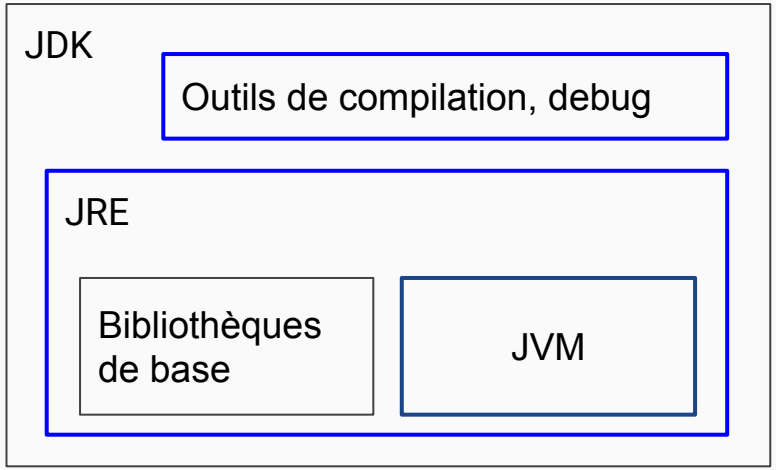
- Java 1.5 -> Java 5
- J2EE -> Java EE -> Jakarta EE

Composition de la plateforme

JVM - **Java Virtual Machine** - la machine virtuelle chargée d'exécuter le bytecode

JRE - **Java Runtime Environment** - l'environnement d'exécution complet, incluant la JVM et les librairies de base

JDK - **Java Development Kit** - l'environnement de développement complet, incluant le JRE et les outils de compilation, de debugging, etc.



Un langage Orienté Objet

Ou presque :

- Il reste des types primitifs : int, char, long…
- L'héritage multiple n'est pas possible

Rappel

- Un objet est un concept qui regroupe dans une même enveloppe
  - des propriétés (attributs) et
  - des comportements (méthodes/fonctions)
- Une application "objet" est composées de multiples acteurs qui interagissent par message (l'appel d'une fonction, avec ou sans paramètres, peut être considérée comme un envoi de message)

Les indispensables

OpenJDK - https://adoptium.net/ (anciennement AdoptOpenJDK)

Java Documentation - http://docs.oracle.com/javase

- La spécification du langage
- La documentation des API (javadoc)

Un éditeur de texte, IDE (IntelliJ, Eclipse, VS Code...),
Documentation browser -> doc offline et indexée (Zael : http://zealdoc.org ,
Dash sur macOS, http://devdocs.io , etc.)

Les fondamentaux du JDK

Les classes de base de la bibliothèque standard du JDK sont organisées en packages :

`java.lang` : les classes fondamentales (Object, String, System, …)

`java.io` : les classes relatives aux entrées/sorties (File, InputStream/OutputStream, Reader/Writer)

`java.util` : contient les collections et classes utilitaires (Collection, List, Hashmap, Date

TP - installer le JDK, accéder à la javadoc et aux spécifications du langage

Télécharger OpenJDK depuis adoptium.net

Découvrir son contenu

Installer Zeal/Dash ou créer un favori sur la javadoc/devdocs.io

Se familiariser avec la navigation dans la javadoc

Quelques spécifications du langage

Les types primitifs

boolean : true / false

char : caractère unicode

byte : octet

short : entier relatif court

une structure de base: le tableau (array) **[ ]**
Exemple : byte [ ]
On préfèrera cependant l'utilisation des objets Collection, List, etc.

int : entier relatif

long : entier relatif long

float : nombre décimal

double : nombre décimal à double précision

La syntaxe de base

Chaque instruction se termine par un point-virgule **;**

Un bloc d'instruction est entouré d'accolades **{ }**

L'affectation se fait avec le signe égal **'='** (à ne pas confondre avec la comparaison d'objets entre eux avec le double égal **'=='**)

Règles de nommage : nom de classe commençant par une majuscule, attribut et méthodes par des minuscules. Les constantes en majuscules
Le ***CamelCase*** est préconisé

La syntaxe de base

Les commentaires

```
// commentaire jusqu’a la fin de ligne

/*
    commentaire
    sur plusieurs
    lignes
*/
```

Définir une classe

Un package (classification, prenant la forme d’une arborescence de répertoires)

Un nom, le fichier contenant cette classe devant porter le même nom

Un ou plusieurs **modifiers** (public, final, abstract)

Des classes particulières : interface, classe abstraite

Un ensemble d’attributs et de méthodes

Exemple de classe

```
package fr.epsi;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

public class Programme {

   static LocalDate date = LocalDate.now();

   public static void main(String[] args) {
        System.out.printf("Date du jour : %s",
          DateTimeFormatter.ISO_LOCAL_DATE.format(date));
   }
}
```

Le point d’entrée de tout programme

La méthode main :

```
public static void main(java.lang.String[] args)
```

Méthode appelée par la JVM lors de l’exécution d’une classe, les arguments sont passés dans le tableau d’objets String

Compiler et exécuter

```
> javac ./fr/epsi/Programme.java
```

Le résultat de la compilation est un fichier **Programme.class** (bytecode indépendant du système) pouvant être exécuté par la JVM, quel que soit son architecture

```
> java fr.epsi.Programme
```

À noter que l'exécutable java nécessite un **nom de classe complet, avec son package**, pour déterminer le chemin relatif du fichier **.class** correspondant.

TP - réaliser un premier programme

Ecrire un programme affichant “Hello World” sur la sortie standard de la console (quelle originalité !)

Ensuite, le compiler (commande **`javac`**) et l’exécuter (commande **`java`**)

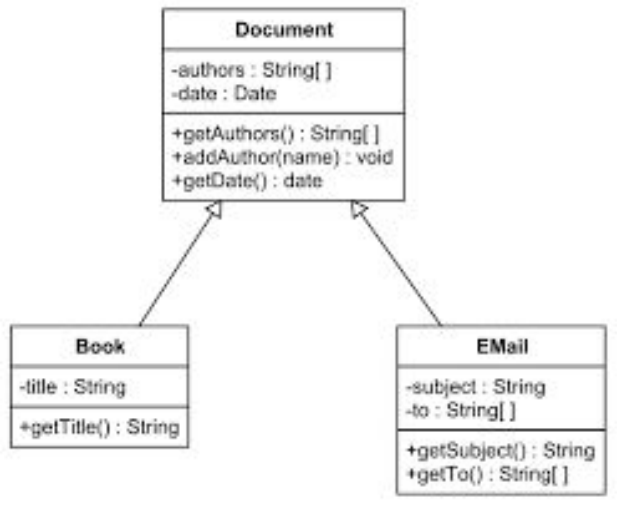
Modifier de programme pour qu’il accepte 2 arguments, supposément des nombres entiers, et affiche leur somme (indice : la signature de la méthode `main` accepte un tableau/liste d’objets de type `String`)

Rappel OOP : les classes / objets

Une classe est composées d'attributs/propriétés et de méthodes/fonctions

Un objet est l'instanciation d'une classe, référencée par un nom de variable et ayant son propre cycle de vie

Les concepts de l'OOP sont l'Abstraction, l'Encapsulation, L'Héritage et le Polymorphisme



Les objets

Un objet est une instance de classe, créé par l'appel du mot clef **`new`**

Un constructeur de la classe doit être appelé, il en existe un par défaut si absent

Un constructeur est défini comme une méthode qui doit porter le nom de la classe, mais ne doit retourner aucun type, pas même `void`
Une classe peut définir plusieurs constructeurs tant que leur signature est différentes (tout comme les fonctions)

Les attributs, les méthodes

Les attributs

- Forcément typés (type primitif ou classe)
- Supportent les `modifiers`
  - public, private, protected
  - static,
  - final

Les méthodes

- Retourne un type ou `void`
- Acceptent une liste d'argument, eux-même typés
- Cet ensemble, avec le nom de la méthode, forme sa signature
- Supportent les `modifiers`
  - public, private, protected
  - static,
  - abstract
  - synchronized

Les `modifiers`

`Access modifiers`

`public` : rend la classe, l'attribut, la méthode accessible depuis n'importe où

`protected` : uniquement depuis le même package, ou une sous-classe

`private` : uniquement depuis la classe où l'attribut, méthode ou innerclass est déclarée

`default` : accessible uniquement depuis le même package

Les `modifiers`

`Non-Access modifiers`

`final` : sur une classe, empêche l'héritage. Sur un attribut/méthode, ils ne peuvent pas être modifiés/surchargés

`static` : sur attribut/méthode, ils deviennent attachés à la classe et non l'objet, et donc unique pour toutes les instances

`abstract` : sur une classe, elle ne peut être utilisée pour créer des objets et doit donc être dérivée. Sur une méthode d'une telle classe, oblige la classe fille a implémenter le body

Accès aux éléments d’un objet

Création d’une instance (objet) d’une classe

```
TypeObjet instance = new TypeObjet();
```

Accès à un attribut `public`

```
System.out.print(instance.attribut)
instance.attribut = newValue;
```

Accès à une méthode `public`

```
instance.methode(parametre1, parametre2);
```

Conditions / boucles

```
if (condition) then {
    statement;
} else {
    statement;
}
```

Conditions / boucles

```
for (initialization; termination; increment) {
    statement;
}

for (Type item : Iterable<Type>) {
    statement;
}
```

Conditions / boucles

```
while (condition) {
    statement;
}

do {
    statement;
while (condition);
```

Conditions / boucles

Le mot-clef **`break`** permet de sortir immédiatement de la boucle

Le mot-clef **`continue`** permet de passer à la prochaine itération

TP - créer une classe avec une méthode utilitaire

Créer une classe dans un sous-package, avec une méthode de classe qui, à partir d'une chaîne de caractères, retourne cette même chaîne avec un "point" entre chaque lettre

Depuis la classe principale, appeler cette méthode avec le premier paramètre passé en ligne de commande et l'afficher sur la sortie standard

Héritage et interfaces

Un classe peut hériter des méthodes et attributs `public` et `protected` de son unique classe mère. Par défaut, toute classe étend `java.lang.Object`

L'héritage est réalisé par le mot-clef **`extends`**

Des méthodes peuvent être librement surchargées en respectant strictement leur signature. Les méthodes `abstract` doivent l'être systématiquement (sauf à créer une classe elle-même abstraite)
Il est possible d'invoquer les méthodes et constructeurs de la classe mère via **`super()`**

Héritage et interfaces

Un classe peut implémenter autant d'interfaces que souhaité.

La définition de la classe indique les interfaces implémentées avec le mot clef `implements`

Toutes les méthodes des interfaces **doivent** être implémentées par la classe

Ce comportement fait d'une Interface un **"contrat de service"**.

Héritage et interfaces

```
public class MyObject implements Comparable<MyObject> {

    Integer value;

    public int compareTo(MyObject other) {
        return this.value.equals(other.value);
    }

}
```

TP - Améliorons notre outillage

- Visual Studio Code
  - https://code.visualstudio.com/
  - Extension pack for Java (par Microsoft)

```
public class Application {
    Run | Debug
    public static void main(String[] args) {
        Run Java Program
```

TP - Polygones et surface

Modéliser un polygone, dont on pourra demander la surface (-> contrat de service)

Modéliser un quadrilatère rectangle, qui répond à la définition du polygone (= respecte son contrat d'interface) et définit 2 propriétés que sont sa longueur et sa largeur (-> concept "abstrait")

Modéliser un rectangle et un carré, 2 applications concrètes du quadrilatère rectangle

Écrire un programme de test qui instancie et manipule ces différentes classes

Classes - outils complémentaires

L'opérateur **instanceof** permet de tester si un objet est une instance d'une classe. Fonctionne également sur une interface implémentée ou une classe parente. Exemple :

```
Carre carre = new Carre(4); carre instanceof Carre; //true

carre instanceof Polygone; //true

carre instanceof QuadrilatereRectangle; //true

carre instanceof Rectangle; //false
```

Generics

Les *Generics* permettent de typer des classes ou des méthodes

La syntaxe pour leur utilisation est la suivante (entre chevrons) :

```
List<Type> list = new ArrayList<Type>();
```

Mise en pratique : Créer une liste ou tout autre *Collection* pouvant accepter des objets de types *Carre* ou *Rectangle*, puis itérer sur cette liste pour afficher leur description (représentation sous forme de chaîne de caractères) et leur surface

Principe d’encapsulation

En programmation orientée objet, l’encapsulation permet de masquer les propriétés et le comportement interne d’un objet, en exposant uniquement les méthodes nécessaires à l’utilisateur.

Assimilable au principe de la “boîte noire”, les attributs et méthodes cachés sont `private`, les attributs et méthodes exposés sont `public`

TP - sécuriser le comportement d’un objet grâce au principe d’encapsulation

Créer une classe modélisant un compte bancaire avec les propriétés suivante :

- Un pourcentage de retenue est appliqué à chaque dépôt, celui-ci est fixé à la création du compte et ne peut pas évoluer par la suite
- Une méthode permet d’ajouter de l’argent, une seconde d’en retirer, chacune mettant à jour le solde du compte
- Le solde du compte peut être consultable, mais pas modifiable par l’utilisateur

Ecrire un programme de test associé

Les Exceptions

Sur un appel de méthode, comment indiquer une erreur… ?

- Approche “historique” : retourner une valeur particulière (souvent `-1`).
  Exemple :
  ```
  int index = “chaîne où rechercher”.indexOf(“yolo”);
  ```
- Mais comment faire si aucune valeur particulière ne peut exister pour représenter un cas d’erreur ? Exemple:
  ```
  int value = Integer.parseInt(“-1”);
  ```

Les Exceptions

Une **`Exception`** représente une “exception” dans le comportement normal de l’exécution d’un programme, généralement un cas d’erreur non gérable par le développeur. Ex : `IndexOutOfBoundException` pour l’accès à un indice de tableau en dehors de sa taille prévue.

Une méthode définit les exceptions qu’elle est susceptible de “lever” dans la **signature** de sa méthode, après le mot clef **`throws`**. Exemple avec le constructeur FileInputStream :

```
public FileInputStream(File file) throws FileNotFoundException
```

Les Exceptions

Classe de base de toute exception : **`Exception`** (observer ses méthodes dans la javadoc) elle même étant classe fille de **`Throwable`**

Une exception peut être “levée” dans un programme par un appel à **`throw`**, suivi d’une instance d’exception. Ex:

```
throw new RuntimeException(“Une erreur s’est produite”);
```

Contrairement à un simple code d’erreur, une exception étant un objet, elle peut contenir davantage d’information (message d’erreur, exception d’origine, etc.)

Les Exceptions

Une exception peut être “attrapée” avec la syntaxe suivante

```
try {
    // traitement levant une potentielle exception
} catch (Exception ex) {
    // traiter ex
}
```

Les Exceptions

Lors de l'appel à une méthode levant une exception, la méthode appelante a deux options :

- Traiter l'exception à son niveau (try/catch)
- Lever à son tour cette exception (adapter sa signature pour ajouter le throws correspondant) -> signifie que ce n'est pas de la responsabilité de cette méthode de traiter l'erreur, mais de la méthode appelante

Une des 2 options doit être implémentée, sinon la classe **ne compile pas** (ici également, notion de contrat de service)

Les Exceptions

Quelle serait la meilleure option dans cette situation ?

```
public String chargerContenu(String filename) {
    FileReader fr = new FileReader(filename);
    fr.read
    ...
}
```

Potentielle exception

TP - Exceptions

Sur le TP précédent du compte bancaire :

- Créer une classe d'Exception représentant une opération bancaire interdite, permettant d'y renseigner une raison (message utilisateur, donnée de solde)
- Faire en sorte de lever cette exception en cas de retrait supérieur au solde

Ecrire un programme de test associé, sans gestion de l'exception (-> observer le comportement par défaut), et avec gestion "propre"

Les fichiers

Représentés par la classe **`java.io.File`**

Accessible en lecture/écriture avec les **`InputStream/OutputStream`** et **`Reader/Writer`**

Classe utilitaire depuis Java 7 : **`java.nio.file.Files`**

Les fichiers

Points d’entrée pour la lecture

FileInputStream

FileReader

BufferedReader

Scanner

Point d’entrée pour l’écriture

FileOutputstream

FileWriter

BufferedWriter

TP - Fichiers

A partir d’un fichier CSV que vous créerez (chaque ligne comprenant un nom d’élève puis une série de notes au format décimal, chaque élément étant séparé par “|”), écrire un programme qui lit ce fichier, calcule la moyenne de chaque élève et l’affiche au fur et à mesure

Introduisez une erreur dans une ligne (ex: format de note non numérique, oubli du “|”) : comment traitez-vous cette erreur pour qu’elle impacte le moins possible le fonctionnement du programme ?