Chapitre 2

Introduction à la programmation orientée objet et installation des outils de travail pour le langage Java

1. Critique de la programmation procédurale
2. Définitions, principes et langages de la Programmation Orientée Objet (POO)
3. Présentation du langage Java
4. Installation des outils et première exécution d’un programme Java

1 Critique de la programmation procédurale

2 Définitions, principes et langages de la Programmation Orientée Objet (POO)

3 Présentation du langage Java

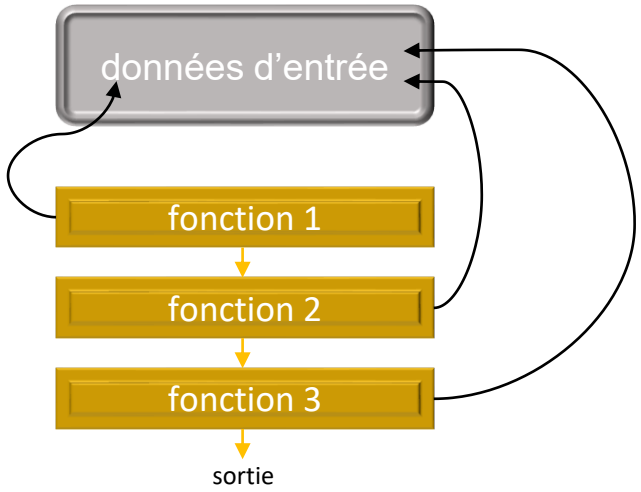
4 Installation des outils et première exécution d’un programme Java

Critique de la programmation procédurale (1/2)

❑ **Rappel des principes :**

- ✓ C'est une sous-catégorie du paradigme de programmation dite **impérative**
- ✓ Un programme correspond donc à une série d'instructions qui donne des ordres à un ordinateur et sont exécutées séquentiellement suivant leur ordre
- ✓ Des groupes d'instructions peuvent être découpés en des blocs appelés **procédures** ou **fonctions**
- ✓ Un programme est donc vu comme étant un ensemble de données sur lesquelles agissent des procédures et des fonctions

Critique de la programmation procédurale (2/2)



❑ **Problèmes :**

- ✓ Programmer revient à définir des variables contenant les données d'entrées d'une part et d'écrire des fonctions pour les manipuler d'autre part. Elles sont donc complètement décorrélées des fonctions qui les traitent
- ✓ Exécuter un programme se réduit alors à appeler ces fonctions dans un ordre décrit par le séquençage des instructions et en leur fournissant les données nécessaires à l'accomplissement de leurs tâches

données et **fonctions** sont traitées indépendamment les unes des autres sans tenir compte des relations étroites qui les unissent

1 Critique de la programmation procédurale

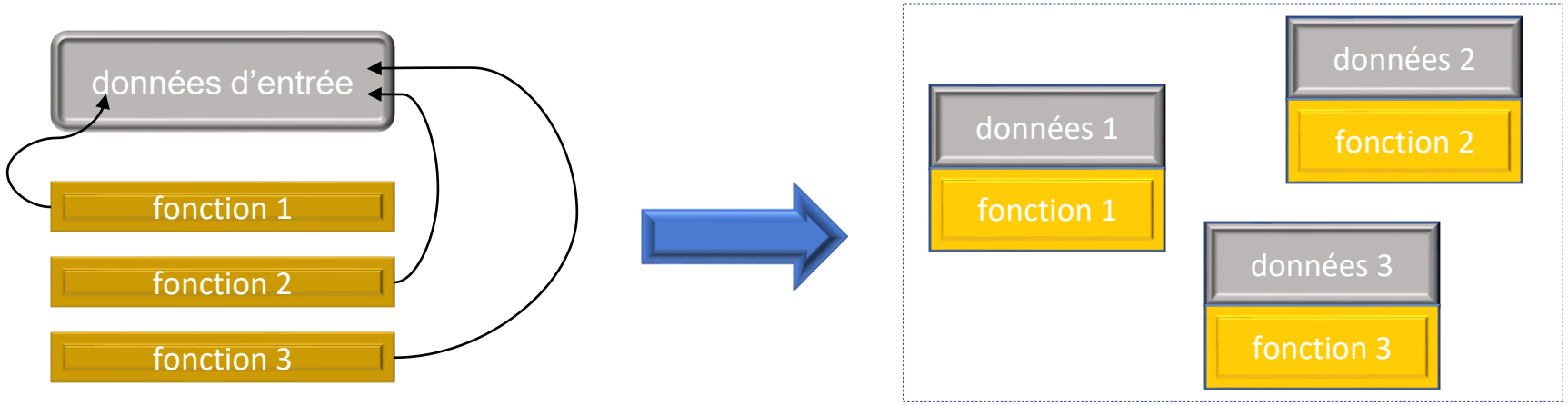
2 Définitions, principes et langages de la Programmation Orientée Objet (POO)

3 Présentation du langage Java

4 Installation des outils et première exécution d’un programme Java

Définitions et principes (1/4)

L'idée de la POO est de rompre avec l'indépendance entre les données et les fonctions qui les traitent, mais de les discriminer et d'assembler celles ayant une corrélation dans des groupes logiques appelés **Objets**



Définitions et principes (2/4)

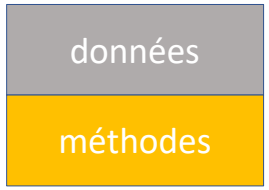
❑ **Qu'est-ce donc la POO ?**

- ✓ Paradigme de programmation issu de la programmation **impérative** et né de la critique de l'approche **procédurale**
- ✓ Consiste en la définition et l'assemblage de briques logicielles appelées **objets**
- ✓ Un programme consiste en l'**interaction** entre les objets dans le but de résoudre le problème traité
- ✓ Un **objet** représente un *concept*, une *idée* ou toute entité du monde physique telle qu'une *voiture*, une *personne* un *stylo*, une *maison*, un *livre*, un arbre, un téléviseur, etc
- ✓ La difficulté dans la résolution d'un problème par l'approche objet réside dans un travail préalable consistant à analyser, discriminer et modéliser les données du problème en des objets cohérents entre eux. Pour cela, il existe des méthodologies comme **UML** qui permettent de le faire, mais ce n'est pas un impératif

Définitions et principes (3/4)

❑ Objet : définition

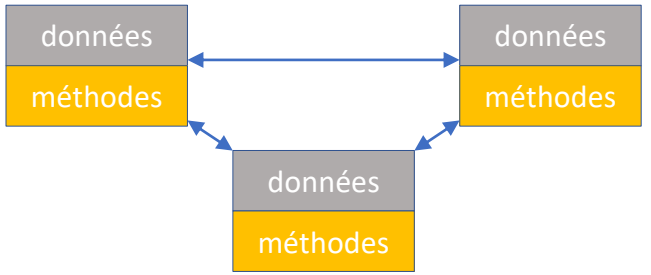
Un objet dans la POO est le composant de base d'un programme. Théoriquement, il dispose d'une part d'un ensemble de **données** qui définissent son état à un instant *t*, et d'autre part d'un ensemble de **méthodes** (des *fonctions* en fait) qui permettent d'agir sur ces données pour modifier son état. On dit en d'autres termes que ces méthodes correspondent aux comportements de l'objet



Objet

❑ Objet : interaction

Dans la POO, un programme se caractérise par l'interaction qu'il y a entre les différents objets qui le constituent. Cette interaction se traduit par des échanges de messages entre objets au moyen des relations qu'il y a entre eux. Un message correspond à une donnée transmise d'un objet à un autre via leurs méthodes



Définitions et principes (4/4)

❑ POO : définition finale en un mot

Méthode de programmation dans laquelle les programmes sont organisés sous formes de collections coopératives d'objets. L'interaction entre les objets via leurs relations permet de concevoir et réaliser les fonctionnalités attendues et de mieux résoudre le ou les problèmes. Un objet dans un programme en cours d'exécution correspond à une boîte noire qui contient toute la logique impérative d'un bout d'une fonctionnalité attendue

Les langages existants en POO

- *Simula* (première version en 1967)
- *Smalltalk* (première version en 1980)
- *Eiffel* (première version en 1986)
- *AS3* (première version en 1998)
- *Ada* (première version en 1980)
- *Objective-C* (première version en 1983)
- *C++* (première version en 1983)
- *PHP* (première version en 1994)
- *Ruby* (première version en 1995)
- *C#* (première version en 2001)
- *Python* (première version en 1991)
- **Java**
- *Groovy* (première version en 2003)
- *JavaScript* (première version en 1996)
- *Kotlin* (première version en 2011)
- Ocaml (première version en 1996)
- *VB.Net* (première version en 2001)
- *Perl* (première version en 1987)
- *Self* (première version en 1987)
- *TypeScript* (première version en 2012)
- *Delphi* (première version en 1995)
- *Swift* (première version en 2014)
- *Scala* (première version en 2004)
- *etc…*

1 Critique de la programmation procédurale

2 Définitions, principes et langages de la Programmation Orientée Objet (POO)

3 Présentation du langage Java

4 Installation des outils et première exécution d’un programme Java

Présentation du langage Java

Présentation du langage Java

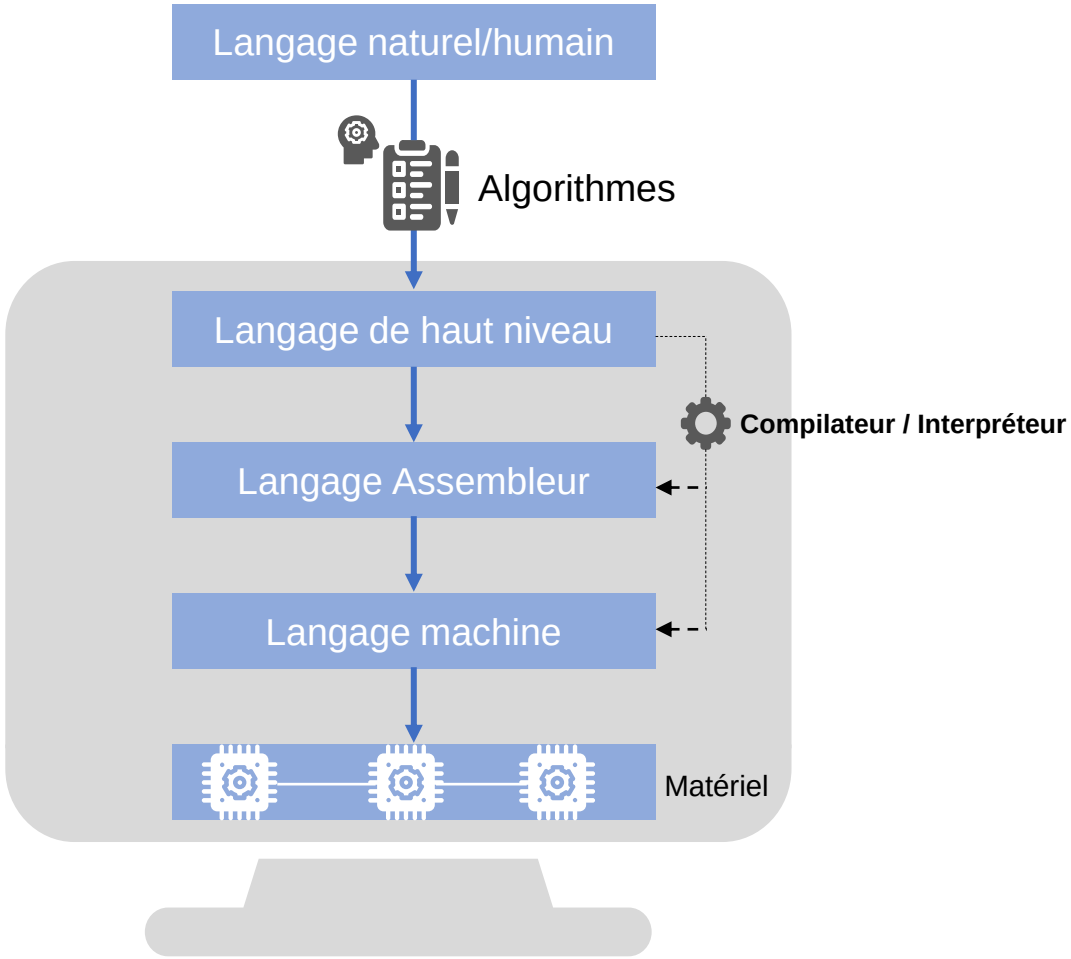
Historique & Caractéristiques (1/2)

- créé en **1995** par **James Gosling** et **Patrick Naughton** ingénieurs chez ***Sun Microsystems***
- **2009 : *Sun Microsystems*** *est racheté par* ***Oracle.*** *Java appartient et est désormais maintenu par* ***Oracle***

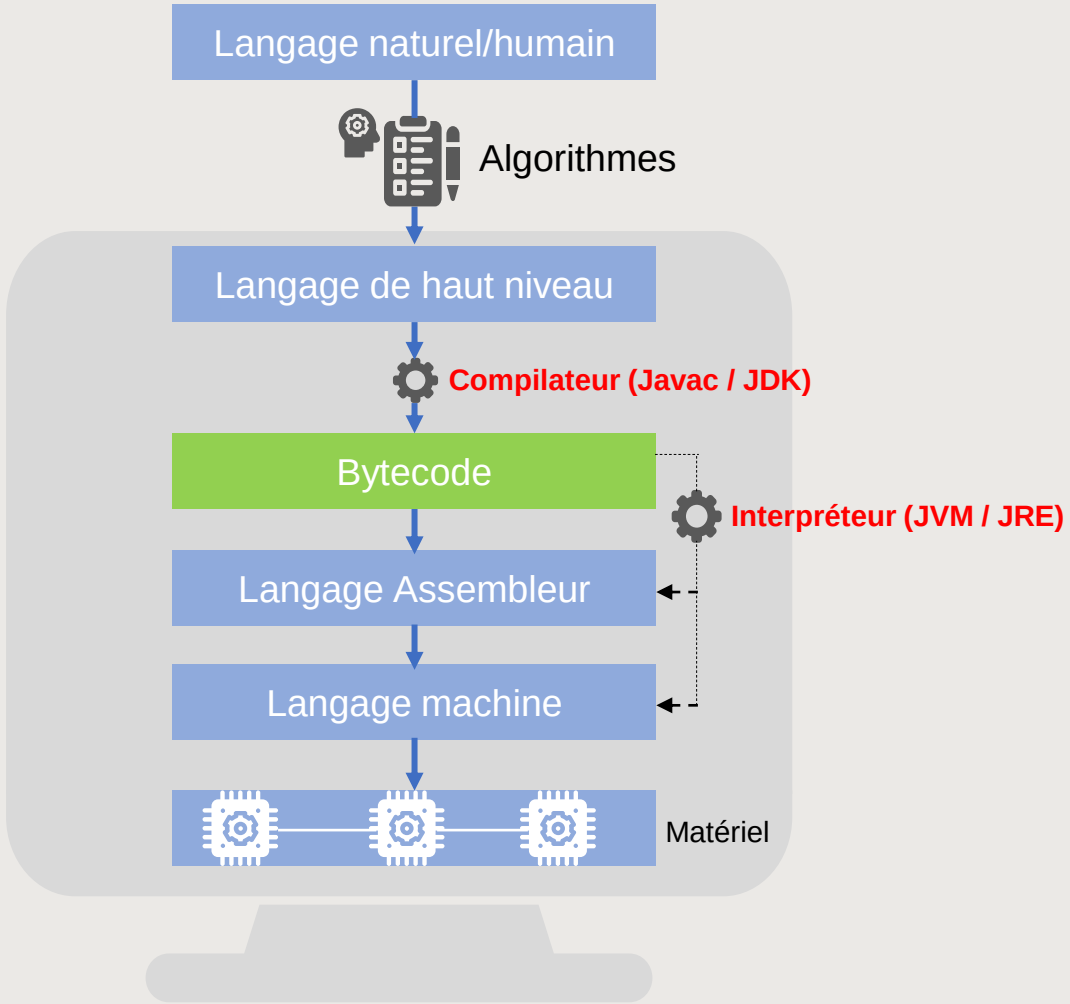
- Java est un langage **interprété**, mais dispose d'une étape de *compilation* en amont. C'est-à-dire que le code source est compilé et traduit dans un langage intermédiaire qualifié de **pseudo-code** ou **Bytecode** avant d'être exécuter ensuite par un **interpréteur** en langage machine
- Le choix de traduire le code source en *Bytecode* rend les **programmes Java portables**, c'est-à-dire, exécutables sur tout type d'ordinateurs ayant des caractéristiques et des architectures différentes.
- Le ***compilateur*** du code source Java en *Bytecode* s'appelle ***Javac*** inclut dans l'outil appelé ***JDK (Java Development Kit)***
- L'***interpréteur*** du Bytecode en langage machine s'appelle ***JVM (Java Virtual Machine)***, lui-même inclut dans un outil appelé ***JRE (Java Runtime Environment)*** à installer absolument sur un ordinateur pour exécuter une application Java
- Le **Bytecode** est un fichier binaire correspondant à la conversion du code source texte en tableaux d'octets universellement lisible par tout type de système d'exploitation, d'où sa portabilité sur tout type d'ordinateur.

Historique & Caractéristiques (2/2)

Rappel : Modèle de compilation classique



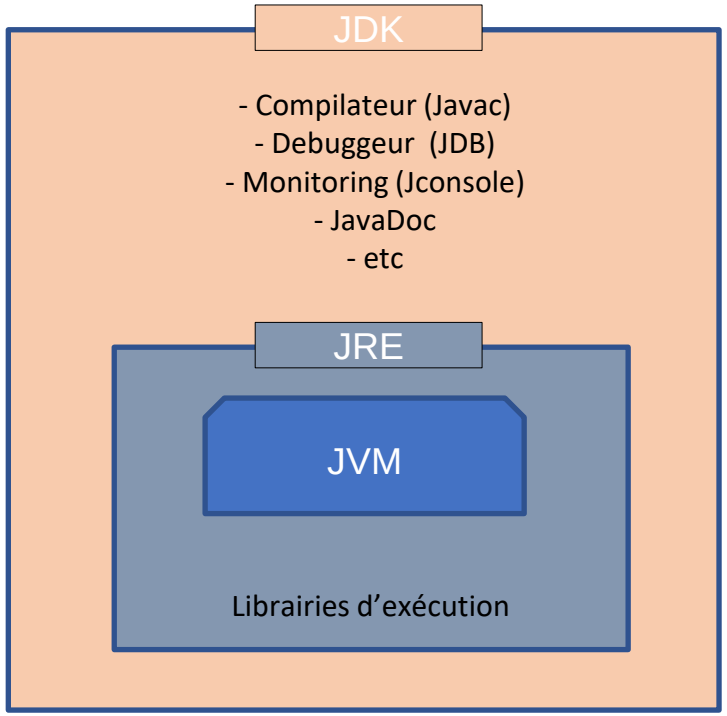
Modèle de compilation Java



Différence entre JDK, JRE, JVM (1/2)

JDK	JRE	JVM
Java Development Kit	**Java Runtime Environment**	**Java Virtual Machine**
Logiciel utilisé pour développer les applications Java	Logiciel d'exécution des applications Java	Logiciel d'exécution du Bytecode Java
Permet de développer des programmes exécutables sur n'importe quel ordinateur ayant un JRE/JVM d'installé. Convertit le code source en Bytecode	Permet d'exécuter une application dont le Bytecode et ses dépendantes externes sont déjà empaquetés ensemble	Permet de convertir le Bytecode en langage machine directement interprétable par un processeur
Il contient une JRE (et donc une JVM) qui permet au développeur de pouvoir exécuter son programme Java pendant la phase de développement	Il contient une JVM pour traduire le Bytecode en langage machine. Il contient aussi des librairies et fichiers nécessaires à la JVM pour l'exécution	Est contenu dans un JRE. En est fondamentalement indépendant, mais exécute un programme en utilisant les librairies fournies par le JRE
Il contient de riches outils pour développer, debugger et monitorer un programme	Il ne contient aucun outils permettant de faire du développement (debugger, monitoring, etc)	Il ne contient pas de librairie ni outils de développement, mais ceux de gestion bas niveau
Il dispose d'un installateur qui permet d'installer le nécessaire pour **développer** et **exécuter** du code Java	Il dispose d'un installateur qui permet d'installer tout le nécessaire pour **exécuter** un programme	Ne dispose pas d'un installateur spécifique. Il est directement inclut dans un JRE
Est **plateforme dépendant** : une distribution différente pour les systèmes Linux, Windows, Mac…	Est **plateforme dépendant** : une distribution différente pour les systèmes Linux, Windows, Mac…	Est **plateforme indépendant**. Une JVM s'adapte donc à tout type d'architecture matérielle et OS

Différence entre JDK, JRE, JVM (2/2)



Les différentes éditions Java

- ***Java Standard Edition (JavaSE)*** **:** plateforme de programmation principale Java. Il contient toutes les bibliothèques et les outils que tout programmeur Java devrait apprendre pour développer des applications normales fonctionnant sur ordinateur. Il fournit avec l'outil **JDK** nécessaire pour faire les développements.

- ***Java Enterprise Edition (JavaEE)*** **:** extension de JavaSE pour développer des applications beaucoup plus complexes et qui ont besoin d'autres types de bibliothèques permettant de réaliser par exemples des accès aux bases de données, des services web, etc; et d'être déployés à plus grande échelle et sur des serveurs. Utilise l'outil **Java EE SDK** pour faire les développements

- ***Java Micro Edition (JavaME)*** **:** extension de JavaSE pour développer des applications de types mobiles. Utilise l'outil **Java ME SDK** pour faire les développements

Les versions du langage Java

Communément, lorsqu'on parle de Java (tout court), on fait référence (sauf mention explicite) à son **édition standard** incarnée par le **JDK** . Parler donc des versions de Java, revient à parler des versions du JavaSE :

Version	Année sortie
Java 1.0	1996
Java 1.1	1997
JavaSE 1.2	1998
JavaSE 1.3	2000
JavaSE 1.4	2002
JavaSE 5	2004
JavaSE 6	2006
JavaSE 7	2011
JavaSE 8	2014

Version	Année sortie
JavaSE 9	09/2017
JavaSE 10	03/2018
JavaSE 11	09/2018
JavaSE 12	03/2019
JavaSE 13	09/2019
JavaSE 14	03/2020
JavaSE 15	09/2020
JavaSE 16	03/2021
JavaSE 17	09/2021
JavaSE 18	03/2022

Depuis le 09/2017, Oracle a décidé de sortir une version de Java tous les 6 mois et Java est devenu payant en utilisation professionnelle

1 Critique de la programmation procédurale

2 Définitions, principes et langages de la Programmation Orientée Objet (POO)

3 Présentation du langage Java

4 Installation des outils et première exécution d’un programme Java

Les outils

Pour développer en *Java*, il faut :

❖ disposer d'un ordinateur ayant un Système d'exploitation de type ***Windows, Linux*** ou ***MacOS***

❖ y installer un des JDK suivant :

- ***OpenJDK*** : c'est une version totalement libre du JDK développée et maintenue conjointement entre l'entreprise du même nom OpenJDK, Oracle et la communauté Java
- ***Oracle JDK*** : c'est une version du JDK uniquement développée et maintenue par l'entreprise Oracle. Elle l'y ajoute des fonctionnalités qui le rend payant en utilisation de production

❖ y installer un ***IDE (Integrated Development Environment).*** Il en existe une flopée sur le marché, dont les plus connus sont :

- ***IntelliJ IDEA***
- ***Eclipse***
- ***NetBeans***
- ***Visual Studio Code (VsCode)***

Installer les outils

Pour le reste du cours, nous allons installer et utiliser *OpenJDK* et *Eclipse* sur *Windows*

Le processus d'installation de ces deux outils est très simple et j'ai concocté pour vous un tutoriel qui vous explique comment faire.

➔ *Voir le document sur Moodle*

Note : Le document traite de l'installation de Java 16, mais vous êtes libre d'installer la dernière version à ce jour

A la découverte d'Eclipse

Je vous ai préparé une petite vidéo qui vous explique succinctement les premières fonctionnalités d'Eclipse que vous devez connaître pour écrire, compiler et exécuter des programmes Java. Mais vous devez vous documenter un peu plus pour avoir une bonne maîtrise de cet outil qui vous sera utile le long du cours pour vos exercices d'applications.

➔ *Voir la vidéo sur Moodle*

Compilation et exécution d'un programme en ligne de commande

Découverte des commandes *Javac* et *java*

Comme cela a déjà été expliqué précédemment, *Javac* est le compilateur du JDK et *java* est une commande de même nom que le langage utilisée pour actionner la JVM afin qu'elle interprète et donc exécute le programme. Eclipse ne fait que d'ailleurs s'appuyer sur ces deux commandes pour proposer une couche abstraction plus conviviale.

Exercice :

Dans un dossier vide que vous avez créer (et dont le nom ne comporte pas d'espace), y créer un fichier à l'intérieur avec par exemple Bloc-notes. Copiez le code source ci en face dans le fichier et faites ***enregistrer sous*** pour le sauvegarder sous le nom **HelloWorld.java**

```
class HelloWorld {

   public static void main( String[] args ) {
      System.out.println( "Hello World !" );
   }

}
```

Application des commandes et constats :

1. Ouvrir une invite de commande qui pointe dans ce dossier et exécutez la commande *javac HelloWorld.java*
   ➔ Que constatez-vous ?
   Réponse : un fichier ***HelloWorld.class*** sera créée dans le répertoire : c'est le **Bytecode**

2. A la suite de la première commande, exécutez celle-ci
   java HelloWorld
   ➔ Que constatez-vous ?
   Réponse : vous verrez afficher sur la console le mot « Hello World !»

C'est à ce type de type de travail archéologique qu'Eclipse nous affranchira en quelques clics


SUCCESS