



Institut Supérieur d'Informatique
et de Multimédia de Gabes

Programmation Orientée Objet JAVA

Sofiane HACHICHA

sofiane.hachicha@isimg.tn

CPI2

2025/2026



Objectifs du cours

- Comprendre le paradigme orienté objet et utiliser Java pour le mettre en œuvre
- Maîtriser la syntaxe du langage Java
- Modéliser et manipuler des objets du monde réel
- Comprendre et utiliser les classes de l'API Java
- Compiler et exécuter une application Java
- Comprendre et utiliser les exceptions





Contenu

- 1 Introduction générale à Java**
- 2 Concepts de base de la programmation orientée objet**
- 3 Syntaxe et éléments de base de Java**
- 4 Classe et Objet**
- 5 Héritage et Polymorphisme**
- 6 Classe abstraite et Interface**
- 7 Concepts avancés**
- 8 Classes de base en Java**
- 9 Gestion des exceptions**



Institut Supérieur d'Informatique
et de Multimédia de Gabes

POO : Java



Chapitre

1

Introduction générale à Java

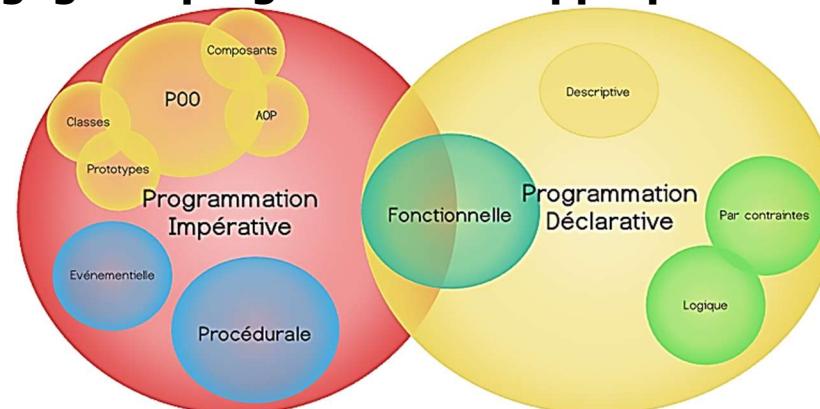
Sofiane HACHICHA
ISIMG 2025 - 2026

1- Introduction générale à Java

1.1 Evolution des langages de programmation (1/10)



- Dans les programmes informatiques, on distingue deux aspects : **les algorithmes et les données**.
 - Ceci n'a pas changé au cours de l'histoire de l'informatique.
 - Ce qui a évolué, c'est la relation existante entre ces deux aspects, une relation qui est définie par ce que l'on appelle **un paradigme de programmation**.
- Un **paradigme de programmation** est une manière d'aborder la programmation informatique et de traiter les solutions aux problèmes ainsi que leur formulation dans un langage de programmation approprié.





1- Introduction générale à Java

1.1 Evolution des langages de programmation (2/10)

Quelques paradigmes courants :

• Programmation impérative

- L'un des paradigmes de programmation les plus anciens et les plus couramment utilisés. Les langages de programmation comme C, Pascal et Fortran appartiennent à ce paradigme.
- La programmation impérative décrit les opérations sous forme de **séquences d'instructions** exécutées par l'ordinateur pour modifier l'état du programme.
- Elle se concentre sur la description de la manière (**comment**) dont un programme doit fonctionner pour résoudre un problème donné.

• Programmation structurée

- Un **sous-ensemble de la programmation impérative** qui vise à structurer les programmes impératifs en garantissant que le code est exécuté instruction par instruction, de manière séquentielle.
- Elle recommande d'éviter les instructions de sauts (**goto**) qui rendent l'exécution du programme non linéaire.





1- Introduction générale à Java

1.1 Evolution des langages de programmation (3/10)

● Programmation procédurale

- Une **extension de la programmation structurée** qui permet de subdiviser les algorithmes en parties plus petites et plus gérables, appelées **procédures**. Cela facilite la compréhension du code et évite les répétitions inutiles.
- Elle consiste à découper un programme en procédures, également appelées routines ou fonctions, qui :
 - Représentent un ensemble d'étapes.
 - Peuvent être récursives, c'est-à-dire s'appeler elles-mêmes.
- En programmation procédurale, les fonctions (ou procédures) peuvent prendre des paramètres en entrée, effectuer des opérations sur ces paramètres et renvoyer des résultats en sortie, ce qui améliore la structure et la lisibilité du code.





1- Introduction générale à Java

1.1 Evolution des langages de programmation (4/10)

● Programmation modulaire

- Elle est proche de l'approche procédurale et repose sur l'idée de subdiviser un programme de manière **ciblée** en blocs logiques, appelés **modules**, qui sont **indépendants** les uns des autres. Cette approche améliore la clarté du code et facilite le processus de débogage.
- Les différents modules peuvent être testés séparément avant d'être intégrés dans une application commune et peuvent être **réutilisés** dans d'autres applications.
- La programmation modulaire n'est pas liée à un paradigme de programmation spécifique ; elle peut structurer des éléments selon des approches impératives, orientées objet ou fonctionnelles.
- Elle permet de créer des **types de données abstraits** (TDA), qui sont des concepts essentiels dans ce paradigme.





1- Introduction générale à Java

1.1 Evolution des langages de programmation (5/10)

- Un programmeur peut développer une structure de données personnalisée en utilisant des modules, avec la possibilité de créer plusieurs variables (ou instances) de ce type.
- Les opérations nécessaires pour manipuler ces variables doivent être définies et **disponibles**.
- Les données associées à ce type doivent être **protégées** et ne peuvent être modifiées que par les opérations spécifiées.
- Les TDA peuvent être implémentés en utilisant la notion de **classe en programmation orientée objet**.
- Cette approche favorise la modularité, la réutilisation et une gestion efficace des données, en permettant de manipuler ces TDA comme s'ils étaient des types de données de base du langage.





1- Introduction générale à Java

1.1 Evolution des langages de programmation (6/10)

● Programmation déclarative

- Il peut être parfois essentiel d'atteindre un résultat précis sans se soucier de l'état actuel du programme ou de la machine. Autrement dit, on souhaite que la machine accomplisse une tâche sans avoir à décrire les différentes opérations nécessaires pour y parvenir.
- Les langages de programmation déclaratifs permettent de spécifier le résultat souhaité tout en laissant à la machine le soin de gérer les détails de l'exécution.
- Il existe une correspondance entre les langages de programmation déclaratifs et la logique mathématique, car le développeur se concentre sur le "**quoi**" (le résultat) plutôt que sur le "**comment**" (les opérations spécifiques).
- Un exemple de langage déclaratif est **HTML**, qui décrit ce que doit contenir une page (texte, titres, paragraphes, etc.) sans indiquer comment ces éléments doivent être affichés.



1- Introduction générale à Java

1.1 Evolution des langages de programmation (7/10)

● Programmation fonctionnelle

- Un **dérivé de la programmation déclarative** qui se caractérise par l'absence d'état mutable. Les données sont manipulées uniquement par des **évaluations de fonctions mathématiques**, sans modification directe des valeurs. Ce paradigme permet de décrire des calculs et des transformations de données à travers des fonctions.
- Le développement des applications **Big Data** nécessite la maîtrise de la programmation fonctionnelle.
- Lisp est considéré comme le premier langage fonctionnel.

● Programmation logique

- Elle repose sur l'utilisation d'un langage déclaratif où un programme est composé d'un ensemble de phrases exprimant des **faits** et des **règles** liés à un domaine spécifique.
- Elle est adaptée aux besoins de l'intelligence artificielle
- Prolog est un langage de programmation logique.





1- Introduction générale à Java

1.1 Evolution des langages de programmation (8/10)

● Programmation orientée objet

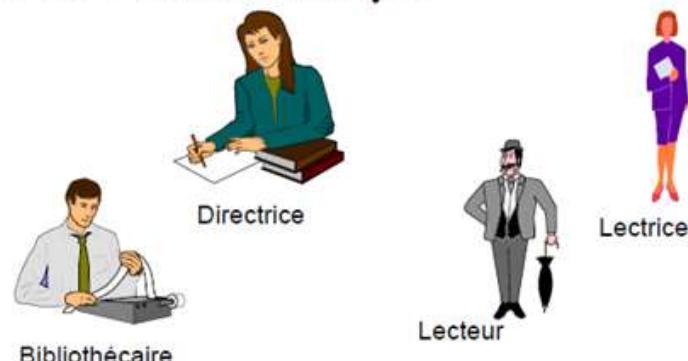
- Un paradigme qui complète et s'intègre avec la programmation impérative et la programmation fonctionnelle.
- Elle vise à regrouper les types de données manipulés par le programme avec les traitements qui peuvent s'y appliquer, organisant ainsi les parties corrélées du programme.
 - L'objectif est de favoriser la **modularité** à travers des concepts comme l'**encapsulation**.
- Les concepts de ce paradigme prennent souvent la forme de **classes** (qui définissent le modèle) et d'**objets** (instances dynamiquement créées à partir de ces classes).
- Ce paradigme est généralement complété par une technique appelée **héritage**, qui permet de :
 - Réutiliser des parties de code déjà écrites.
 - Redéfinir ou compléter ces parties pour les adapter à des contextes différents.



1- Introduction générale à Java

1.1 Evolution des langages de programmation (9/10)

Exemple: Gestion d'une bibliothèque



Approche procédurale :

- "Que doit faire mon programme ?"

Approche orientée objet :

- "De quoi doit être composé mon programme ?"

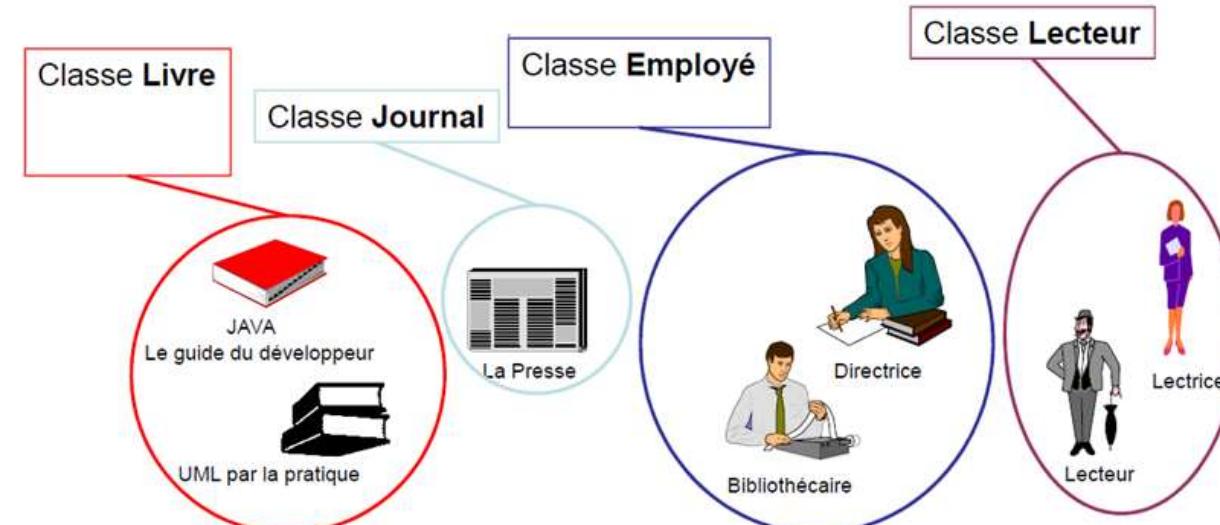


1- Introduction générale à Java

1.1 Evolution des langages de programmation (10/10)

Des **objets similaires** peuvent être informatiquement décrits par une même abstraction : **une classe**

- **même** structure de données et méthodes de traitement
- valeurs **différentes** pour chaque objet





1- Introduction générale à Java

1.2 Historique du langage Java (1/2)

- Développé en **1990** chez **Sun Microsystems** par **James Gosling, Mike Sheridan et Patrick Naughton**.
- Le nom **Oak** étant déjà utilisé, le langage est alors appelé **Java** (café en argot américain).
- En **1993**, avec l'intérêt grandissant d'Internet, ce langage, se métamorphose en langage dédié à Internet :
 - Sun diffuse le premier browser **HotJava** qui permet d'**exécuter** des programmes Java encapsulés dans des pages **WEB** (i.e. des **applets Java**).
- **1996** : **Java Development Kit (JDK)** est disponible gratuitement pour la plupart des machines du marché.
- La société **SUN** a été acquise en janvier **2010** par **Oracle**, qui est devenue le gardien officiel de Java.



1- Introduction générale à Java

1.2 Historique du langage Java (2/2)

- Java 1.0 publié en janvier 1996 : première version
- Java 1.1 publié en février 1997.
- Java 1.2 publié en décembre 1998.
...
- Java 8 (JDK 1.8), publié en mars 2014, est une version LTS (Long-Term Support) : supportée jusqu'en 2030.
- Java 9 (JDK 9) publié en septembre 2017.
...
- Java 17, publié en septembre 2021, est une LTS.
...
- Java 21, publié en septembre 2023, est une LTS.
...
- Java 25 sera disponible en septembre 2025.



1- Introduction générale à Java

1.3 Caractérisation du langage Java (1/7)

● Le langage Java est familier :

- Java est un langage familier et proche du **C++**.
 - Même types de base que C++ (int, float, double, etc.),
 - Même formes de déclarations que C++,
 - Même structure de contrôle que C++ (if, while, for, etc.).

● Le langage Java est simple :

- Java est un langage simple par rapport au langage **C** et **C++**.
 - Java évite des complexités du C++ comme la gestion manuelle des pointeurs et la manipulation directe de la mémoire.
 - Java intègre un ramasse-miettes (garbage collector) qui gère la libération automatique de la mémoire, ce qui rend la programmation plus simple.





1- Introduction générale à Java

1.3 Caractérisation du langage Java (2/7)

Le langage Java est orienté objet :

- Paquetages (ou packages) pour la **réutilisation** :
 - **java.lang** : classes de base
 - **java.io** : entrée/sortie
 - **java.awt** : gestion des interfaces graphiques
 - **java.sql** : interagir avec les bases données relationnelles
 - **java.util** : classes utilitaires

Le langage Java est distribué :

- Package **java.net** : fournit des classes pour les applications réseaux.
 - Classe **URL** : permet l'accès à des objets distants.
 - Classe **Socket** : facilite la programmation d'applications Client/Serveur.
- Il possède des API pour la programmation réseau (java.net) et la gestion d'objets distribués (java.rmi).



1- Introduction générale à Java

1.3 Caractérisation du langage Java (3/7)

- Java permet la manipulation **transparente** des ressources locales ou distantes, facilitant ainsi les applications réparties.

Le langage Java est interprété :

- Un programme Java n'est pas compilé directement en code machine ;
 - Il sera **compilé** en code intermédiaire appelé **bytecode**.
 - Lors de l'exécution, ce **bytecode** sera chargé et **interprété** par une machine virtuelle spécifique appelée **JVM**.

Le langage Java est portable et indépendant des plateformes :

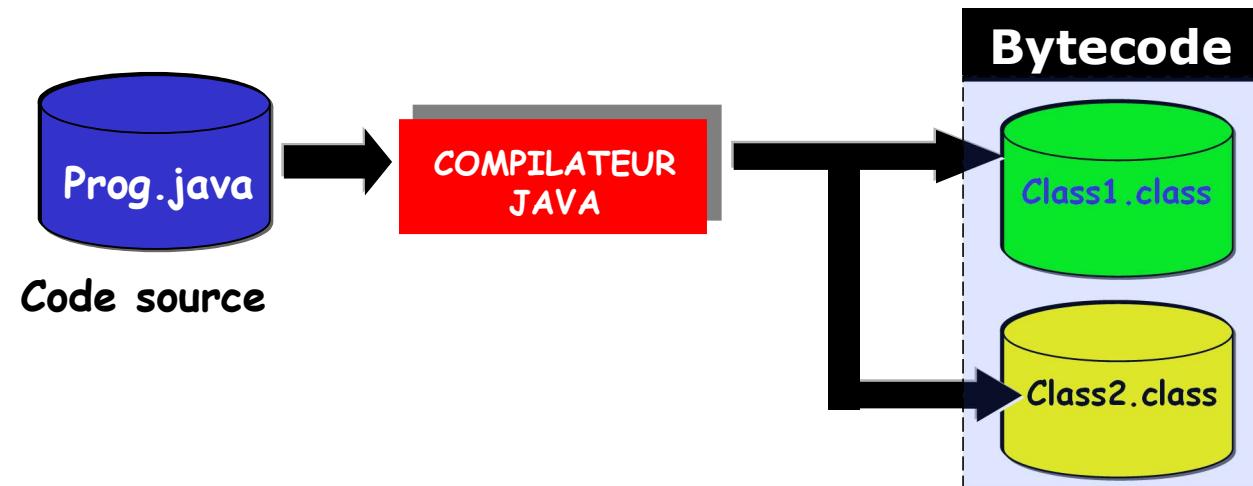
- Le **bytecode** est indépendant de la plateforme matérielle et du système d'exploitation.
 - Il peut être exécuté sur n'importe quelle machine disposant d'une machine virtuelle Java (JVM) adaptée à sa plateforme.



1- Introduction générale à Java

1.3 Caractérisation du langage Java (4/7)

Production d'une application Java standard



Bytecode = Pseudo code machine (ou code intermédiaire) qui, contrairement au code binaire natif, n'est pas exécuté directement par le processeur, ce qui peut ralentir l'exécution comparé à du code natif.



1- Introduction générale à Java

1.3 Caractérisation du langage Java (5/7)

Exécution d'une application Java standard



JVM = Programme capable d'interpréter les instructions contenues dans les fichiers **bytecode** Java afin de les exécuter sur n'importe quelle plateforme (Linux, Mac OS, Windows,...) sans aucune modification.





1- Introduction générale à Java

1.3 Caractérisation du langage Java (6/7)

Le langage Java est robuste et sûr :

- Java est un langage **fortement typé**, ce qui signifie que le type de chaque variable, paramètre ou attribut est **vérifié à la compilation**.
- Il permet la détection de nombreuses erreurs de type dès la compilation.
- Seules les conversions sûres sont effectuées automatiquement par le compilateur.

Fiabilité de la gestion de la mémoire :

- Java n'a pas de pointeurs explicites, ce qui élimine de nombreux risques liés à la manipulation directe des adresses mémoire.
- Il dispose d'un ramasse-miettes automatique (garbage collector) qui gère la mémoire et évite les fuites.
- Il effectue un contrôle automatique des accès aux tableaux et aux chaînes de caractères pour éviter des dépassements.





1- Introduction générale à Java

1.3 Caractérisation du langage Java (7/7)



Le langage Java est multi-thread :

- Java permet l'exécution en parallèle de plusieurs **threads**, facilitant la **programmation concurrente**.
- La classe **java.lang.Thread** fournit des méthodes permettant de démarrer, exécuter et stopper les threads.
- La classe **ThreadGroup** permet de regrouper des threads pour les gérer collectivement.
- Pour une **gestion avancée des threads**, les classes du **framework Executor** (dans `java.util.concurrent`) offrent un contrôle efficace des **pools de threads** et facilitent la programmation concurrente complexe, en gérant le cycle de vie des threads et la parallélisation des tâches.
- La **synchronisation** en Java permet de contrôler l'accès concurrent à des ressources partagées pour éviter les conflits entre threads.



1- Introduction générale à Java

1.4 Java par l'exemple (1/6)

Fichier «XXX.java»

// Commentaire

Définition d'une classe XXX

Méthode main ()

● **Classiquement, un programme Java contient une classe qui :**

- porte le même nom que le fichier ayant l'extension « .java » dans lequel elle est enregistrée.
- comporte (au moins) une méthode appelée **main**.



1- Introduction générale à Java

1.4 Java par l'exemple (2/6)

Fichier «PremierProgramme.java»

// Ce programme affiche le message < Bonjour ISIMG >

```
class PremierProgramme
{
    public static void main(String args[])
    {
        System.out.println("Bonjour ISIMG");
    }
}
```

- La première ligne du programme « PremierProgramme » est une ligne de commentaire :

- Elle commence par //.
- Tout ce qui est compris entre ces deux caractères et la fin de la ligne est ignoré par le compilateur.



1- Introduction générale à Java

1.4 Java par l'exemple (3/6)

- Le mot clé **class** permet de définir une nouvelle classe Java, suivi du nom de cette classe.
- En Java, les majuscules et les minuscules sont considérés comme des caractères différents.
 - PremierProgramme n'est pas identique à PREMIERProgramme.
- Les caractères « { » et « } » marquent le début et la fin du **bloc d'instructions** à réaliser par la classe.
- **main()** : représente la méthode principale de la classe.
- Un **interpréteur Java** a pour fonction d'exécuter les instructions contenues dans le **bloc d'instruction** de la méthode principale **main**, du programme qu'on lui soumet.





1- Introduction générale à Java

1.4 Java par l'exemple (4/6)

- Une méthode est une sorte de procédure (ensemble d'instructions) appartenant à une classe.
 - Une méthode peut prendre des paramètres de types précis et renvoie éventuellement une valeur de type aussi précis.
- Le mot clé **void** signifie que la méthode **main** ne renvoie aucune valeur. **args** est le nom du paramètre d'entrée de type **String[]** de la méthode **main**, qui permet de passer des arguments à la ligne de commande au programme.
- les mots clés **public** et **static** décrivent des caractéristiques de la méthode **main**.
- **System.out.println** est une commande permettant d'afficher la chaîne de caractère « **Bonjour ISIMG** » sur la sortie par défaut de votre machine qui est l'écran.



1- Introduction générale à Java

1.4 Java par l'exemple (5/6)



Depuis Java 21 (en mode preview) :

- La méthode **main()** peut être une méthode d'instance (non statique) avec ou sans tableau de chaînes de caractères en paramètre.

// Exemple.java

```
class Exemple {  
  
    void main(){  
        System.out.println("Bonjour");  
    }  
}
```

- Il n'est plus obligatoire de définir explicitement une classe. Dans ce cas, une **classe implicite** sera définie par le compilateur dans le package par défaut.

// Exemple.java

```
void main(){  
    System.out.println("Bonjour");  
}
```

En Java 23 et 24 en mode preview, il est possible d'appeler `println()` sans préfixe dans les classes implicites.



1- Introduction générale à Java

1.4 Java par l'exemple (6/6)



A partir de Java 25 (en mode standard finalisé) :

- La fonctionnalité des **classes implicites** permettra d'écrire du code simplifié où la classe **java.io.IO** est implicitement importée. Cette classe fournit des méthodes comme `println()`, `print()` et `readln()`. En Java 25 standard, il faut désormais utiliser la **syntaxe explicite** `IO.println()` car les méthodes ne sont plus visibles directement sans qualification.
- Dans ces classes implicites, il sera possible d'écrire :

// Exemple.java

```
void main(){  
    IO.println("Bonjour");  
}
```

sans avoir besoin d'écrire `System.out.println()` ni de faire un import explicite de `java.io.IO`.





1- Introduction générale à Java

1.5 Structure d'une application Java standard

Fichier «XXX.java »

// Commentaire

Définition d'une classe AAA

// Commentaire

Définition d'une classe BBB

:

Définition d'une classe XXX

Méthode main ()



Parmi les classes définies dans le fichier XXX.java, il ne peut y avoir qu'**une seule classe publique** et cette classe doit avoir le **même nom XXX** que le fichier.





1- Introduction générale à Java

1.6 Exemple d'une application Java standard

Fichier «**PremierProgramme.java** »

// Ce programme affiche le message « Bonjour ISIMG »

```
class PremierProgramme
{
    static void Affiche( )
    {
        System.out.println("Bonjour ISIMG");
    }

    public static void main(String args[] )
    {
        Affiche( );
    }
}
```





1- Introduction générale à Java

1.7 Le Java Development Kit : **JDK**

- **Le JDK est l'ensemble d'outils et bibliothèques nécessaires pour développer et exécuter des applications Java.**
 - Java est également soutenu par la communauté open source via le projet **OpenJDK**, lancé en 2006, qui fournit une implémentation libre et ouverte de la plate-forme Java.
- **Les environnements intégrés de développement (IDE) Java sont actuellement :**
 - commercialisés ([Codenvy](#), [IntelliJ IDEA Ultimate](#), [MyEclipse](#), ...)
 - distribués gratuitement à travers le Web ([Eclipse](#), [IntelliJ IDEA Community](#), [NetBeans](#), [BlueJ](#), [JDeveloper](#), [DrJava](#), ...).



1- Introduction générale à Java

1.8 Outils fournis par le JDK (1/3)

● Le compilateur **javac** :

- Compile les fichiers sources java de nom XXX.java.
- Traduit les fichiers sources en **bytecode** :
 - Produit autant de fichiers .class que de classes définies dans le fichier .java

● L'interpréteur **java/javaw** :

- Prend en paramètre le nom de la classe principale.
- Recherche le ou les fichiers .class correspondants.
- Appelle la méthode main de la classe principale.

● Le documenteur **javadoc** :

- Génère automatiquement une documentation sous la forme de fichiers html à partir des fichiers sources commentés.

● L'outil **jstat** :

- Un outil pour surveiller les statistiques de la JVM



1- Introduction générale à Java

1.8 Outils fournis par le JDK (2/3)

● L'outil **jconsole** :

- Un outil graphique permettant de contrôler et de gérer le comportement des applications.

● L'archive **jar** :

- **JAR** est l'abréviation de **Java ARchive**. C'est un format de fichier qui permet de regrouper des fichiers contenant du bytecode (fichier .class) et des données utilisées en tant que ressources (images, son, ...). Depuis Java 9, Il est possible d'utiliser l'archive **JMod**.

● L'outil **jlink** :

- Un outil permettant de produire une nouvelle « **JVM light** » dans laquelle on stocke uniquement les codes des modules utilisés par l'application.

● Le désassembleur de classe **javap** :

- Un programme permettant de désassembler les fichiers compilés en prenant en paramètre les fichiers **.class** et de produire un texte en clair de la classe afin d'examiner le bytecode.

1- Introduction générale à Java

1.8 Outils fournis par le JDK (3/3)



● **Le débogueur Java jdb :**

- Un programme permettant de faciliter la mise au point des programmes grâce à de nombreuses options permettant de surveiller leur exécution.
- Il est beaucoup plus facile à utiliser lorsqu'il est intégré à un IDE.

● **Un serveur web minimaliste jwebserver :**

- Depuis **Java 18**, un serveur web minimal est fourni pour servir uniquement des fichiers statiques.

● **Les librairies de classe standard :**

- Classes pour réaliser les entrées sorties.
- Classes pour réaliser des applications réseau.
- Classes utilitaires (listes, maps, etc.).

● **La documentation en ligne.**

● ...



1- Introduction générale à Java

1.9 Compiler et exécuter un programme Java en ligne de commande (1/3)



Etapes :

- Le fichier Java doit porter **exactement le même nom que la classe publique principale**, respectant la casse des caractères. Si la classe est nommée **PremierProgramme**, le fichier doit s'appeler **PremierProgramme.java**
 - Compiler le programme : **javac PremierProgramme.java**
 - Pour compiler en **mode preview** avec JDK 21 (par exemple) :

javac --enable-preview --release 21 PremierProgramme.java

- Un fichier **PremierProgramme.class** contenant du **bytecode** sera généré.





1- Introduction générale à Java

1.9 Compiler et exécuter un programme Java en ligne de commande (2/3)



- Exécuter le **bytecode** : **java PremierProgramme**
- Pour exécuter le ByteCode généré en **mode preview** :
java --enable-preview PremierProgramme
- Depuis **Java 11**, possibilité d'exécuter un programme depuis un fichier source unique (mono-fichier), sans étape de compilation préalable avec : **java PremierProgramm.java**
 - Pour exécuter un programme depuis **un fichier source unique** en utilisant une fonctionnalité **preview** (comme les **classes implicites**) avec JDK 21 (par exemple) :

java --enable-preview --source 21 Exemple.java



1- Introduction générale à Java

1.9 Compiler et exécuter un programme Java en ligne de commande (3/3)



- Depuis **Java 22**, il est possible d'exécuter un programme Java à partir d'un fichier source principal, et si ce fichier source utilise une classe définie dans un autre fichier source, ce second fichier sera aussi automatiquement compilé en mémoire. Seuls les fichiers sources réellement utilisés par le programme principal sont compilés en mémoire, ce qui simplifie l'exécution de programmes multi-fichiers sans avoir à compiler manuellement tous les fichiers au préalable.

// Prog.java

```
public class Prog {  
    public static void main(String[ ] args){  
        Utils.saluer();  
    }  
}
```

// Utils.java

```
public class Utils {  
    static void saluer(){  
        System.out.println("Hello");  
    }  
}
```

> java Prog.java
Hello



1- Introduction générale à Java

1.10 JVM (1/2)

- **La JVM est une machine abstraite qui permet à la machine physique (ordinateur) d'exécuter un programme Java.**
- **Le compilateur Java compile d'abord le code Java en bytecode. La JVM convertit ensuite ce bytecode en code machine natif (ensemble d'instructions que le processeur d'un ordinateur exécute directement).**
 - La JVM définit les spécifications hardware de la plate-forme.
- **Un compilateur JIT (Just In Time) est intégré à la JVM afin d'améliorer les performances de l'exécution du bytecode.**
 - L'idée d'un compilateur JIT est de compiler en code natif le bytecode d'une méthode invoquée plusieurs fois.
- **Par défaut, le bytecode est exécuté par la JVM HotSpot.**
- **La JVM possède trois tâches principales :**
 - Charger le code (**class loader**)
 - Vérifier le code (**bytecode verifier**)
 - Exécuter le code (**runtime interpreter**)



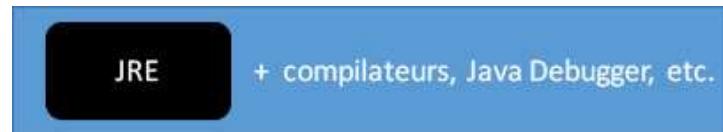
1- Introduction générale à Java

1.10 JVM (2/2)

- **JRE (Java Runtime Environment) est un progiciel qui fournit des bibliothèques de classes Java, ainsi que JVM, et d'autres composants permettant d'exécuter des applications écrites en Java.**



- **Lors du téléchargement du JDK, le JRE est également téléchargé**
 - pas besoin de télécharger le JRE séparément.
- **le JDK contient le JRE et plusieurs autres outils comme présenté ci-avant**





1- Introduction générale à Java

1.11 Java SE

- Java SE ou Java Platform, Standard Edition (anciennement Java 2 Platform, Standard Edition, ou J2SE), est une spécification de la plate-forme Java d'Oracle, destinée **aux applications pour poste de travail (desktop)**.
- Java SE est composée de plusieurs API (Application Programming Interface) Java :
 - Bibliothèques de classes Java standards sur lesquelles le programmeur peut s'appuyer pour écrire son code.
- À chaque version de Java SE correspond :
 - les Java Specification Requests (**JSR**), constituant les spécifications de la version considérée ;
 - un **JDK** ;
 - un **JRE** contenant le seul environnement d'exécution.
(Depuis Java 9, il est inclus de base dans le JDK).



1- Introduction générale à Java

1.12 Java vs C++

- Un programme Java doit contenir au moins une classe.
- Une méthode **main()** avec un type de retour **void** est requise comme point d'entrée pour l'exécution d'une application Java standard.
- L'instruction **include** (comme en C/C++) est remplacée en Java par l'instruction **import**.
- Java ne gère pas les pointeurs directement, mais utilise des **références** pour manipuler les objets.
- Java ne supporte pas l'héritage multiple de classes. Une classe ne peut hériter que d'une seule classe mère.