



Langage JAVA

Mama AMAR

Séquence 1 : Présentation du Langage

Chapitre 1 : Présentation du Langage

Introduction

Le langage Java est un langage généraliste de programmation synthétisant les principaux langages existants lors de sa création en 1995 par Sun Microsystems. Il permet une programmation orientée-objet (à l'instar de SmallTalk et, dans une moindre mesure, C++), modulaire (langage ADA) et reprend une syntaxe très proche de celle du langage C. Outre son orientation objet, le langage Java a l'avantage d'être modulaire (on peut écrire des portions de code génériques, c'est-à-dire utilisables par plusieurs applications), rigoureux (la plupart des erreurs se produisent à la compilation et non à l'exécution) et portable (un même programme compilé peut s'exécuter sur différents environnements).

La technologie Java est à la fois un langage de programmation et une plateforme.

L'objectif de ce chapitre est :

- De fournir un aperçu de la technologie Java dans son ensemble.
- De présenter le langage de programmation Java et de la plateforme java.
- De présenter les caractéristiques et de cette technologie.
- De présenter les étapes de création d'un programme avec un exemple en illustration

Le Langage de Programmation Java

Dans le langage de programmation Java, tout le code source est d'abord écrit dans des fichiers texte se terminant par l'extension **.java**. Ces fichiers sources sont ensuite compilés dans des fichiers **.class** par le compilateur **javac**. Un fichier **.class** ne contient pas de code qui est natif de votre processeur; il contient au lieu **bytecode** - la langue de la machine virtuelle Java (Java VM). L'outil lanceur de java exécute alors votre application avec une instance de la machine virtuelle Java.

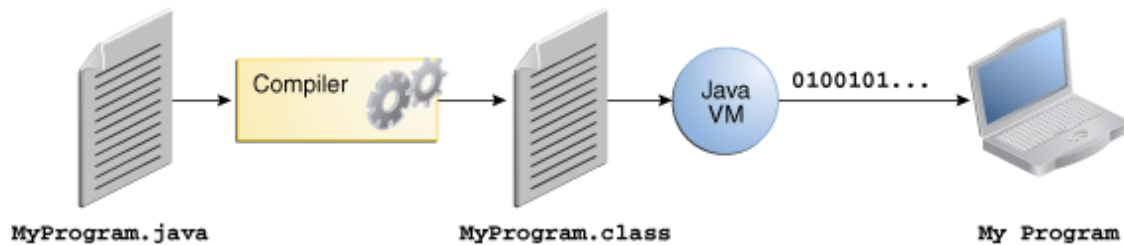


Figure 1 : Une vue d'ensemble du processus de développement logiciel en Java.

Puisque la machine virtuelle Java est disponible sur de nombreux systèmes d'exploitation différents, les mêmes fichiers **.class** sont capables de fonctionner sur Microsoft Windows, le système d'exploitation Solaris™ (SE Solaris), Linux ou Mac OS. Certaines machines virtuelles, telles que Java SE HotSpot at a glance, effectuent des étapes supplémentaires lors de l'exécution pour donner à votre programme une amélioration des performances. Cela inclut diverses tâches telles que trouver les goulots d'étranglement et recompiler (en code natif) les sections de code fréquemment utilisées.

I. La Plateforme Java

Une plateforme est l'environnement matériel ou logiciel dans lequel un programme est exécuté. Nous avons déjà mentionné certains des plateformes les plus populaires comme Microsoft Windows, Linux, Solaris et Mac OS. La plupart des plateformes peuvent être décrits comme étant une combinaison du système d'exploitation et du matériel sous-jacent. La plateforme Java diffère de la plupart des autres plateformes en ce qu'elle est une plateforme logicielle seule qui fonctionne sur d'autres plateformes basées sur le matériel.

Le schéma suivant est issu du site web www.oracle.com et présente les différentes composantes de la plateforme Java.

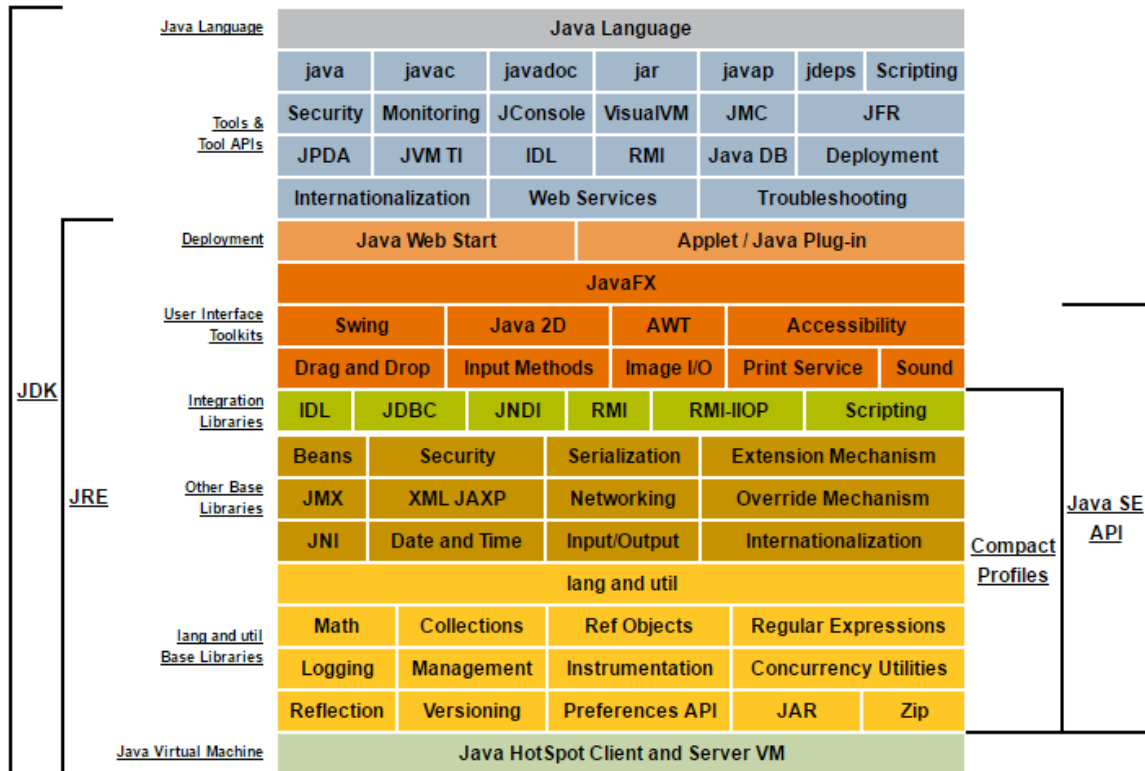


Figure 2 : Les composants de la plateforme Java

<https://docs.oracle.com/javase/8/docs/index.html>

La plateforme Java a deux composantes:

- La Machine Virtuelle Java
- L'interface de programmation API (Application Programming Interface) java

La machine virtuelle java (JVM)

La machine virtuelle est la base de la plateforme Java et est portée sur diverses plateformes basées sur le matériel. Elle est nécessaire pour l'exécution des programmes java. La machine virtuelle java s'occupe :

- du chargement des classes et du bytecode qu'elles contiennent : quand un programme invoque la création d'objets ou invoque des membres d'une classe, c'est la JVM qui s'occupe du chargement du bytecode qui doit être interprété.
- de la gestion de la mémoire : La JVM s'occupe entièrement de la gestion des pointeurs et donc de chaque référence fait à un objet. Ce procédé permet également à la JVM de s'occuper de la libération automatique de la mémoire (ramasse-miette) dès qu'un objet n'est plus référencé dans le programme, c'est-à-dire quand aucune variable n'y fait référence.
- de la sécurité : Au chargement, elle vérifie qu'il n'est pas fait appel à de la mémoire non initialisée, que des conversions de types illégales ne sont pas effectuées, que le programme ne manipule pas des pointeurs en mémoire. Dans le cas d'applets java, la JVM interdit au programme l'accès au réseau uniquement vers l'hôte qui diffuse l'applet.
- de l'interfaçage avec du code natif (par exemple, code écrit en langage C) : la plupart des API de base de java font appel à du code natif qui est fourni avec le JRE, afin d'interagir avec le système hôte.

L'API java

L'API est une grande collection de composants logiciels, prêts à l'emploi, qui offrent de nombreuses fonctionnalités utiles. Elles sont regroupées dans les bibliothèques de classes et interfaces connexes; ces bibliothèques sont connues sous forme de paquets.

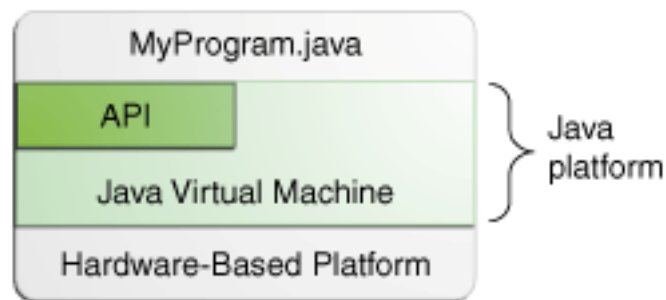


Figure 3 : L'API et la machine virtuelle java (Java)

L'API et la machine virtuelle Java isolent le programme à partir du matériel sous-jacent.

Comme un environnement indépendant de la plateforme, la plateforme Java peut être un peu plus lente que le code natif. Cependant, les progrès dans le compilateur et les technologies de la machine virtuelle apportent une performance proche de celle du code natif sans impacter la portabilité.

Les termes "Java Virtual Machine" et "JVM" signifie une machine virtuelle pour la plateforme Java.

II. Caractéristiques de Java

Le langage de programmation Java est un langage de haut niveau qui peut être caractérisée par :

- Simple
- Orienté objet
- Distribué
- Multithread
- Dynamique
- Architecture neutre
- Portable
- Haute performance
- Robuste
- Secure

Simple : La syntaxe de Java est similaire à celle du langage C et C++, mais elle omet des caractéristiques sémantiques de C et C++ qui rendent ces langages complexes et non sécurisés.

Orienté objet : En java tout est objet mis à part les types de données primitifs. En plus, il existe des classes préfabriquées qui permettent d'encapsuler les types primitifs dans des objets.

Distribué : Java implémente les protocoles réseaux standard, ce qui permet de développer des applications client/serveur en architecture distribuée, afin d'invoquer des traitements et/ou des manipulations de données sur des machines distants.

Multitâche : Java permet de développer des applications mettant en œuvre l'exécution simultanées de plusieurs threads (ou processus légers). Ceci permet d'effectuer plusieurs traitements à la fois, afin d'accroître la rapidité des applications, soit en partageant le temps CPU soit en partageant les traitements entre plusieurs processus.

Dynamique : En java, il est possible de modifier une ou plusieurs classes sans avoir à effectuer une mise à jour de ces modifications sur l'ensemble du programme. La vérification de l'existence des classes se fait au moment de la compilation et l'appel du code de ces classes ne se fait qu'au moment de l'exécution. Ce procédé permet d'avoir des applications allégées en taille mémoire.

Architecture neutre : Le compilateur produit du bytecode (langage binaire intermédiaire) qui est indépendant de toute architecture matérielle, de tout système d'exploitation et de tout dispositif de gestion de l'interface utilisateur graphique (GUI).

Portable : Étant un langage interprété, java est portable sur différentes plateformes. De plus, les bibliothèques de classes standard de java permet de déployer le code sur différentes plateformes sans adaptation.

Haute performance : Java met en œuvre un processus d'optimisation de l'interprétation du code appelé JIT (Just In Time) ou HotSpot. Ce processus permet de compiler à la volée le bytecode java en code natif, ce qui permet d'atteindre les mêmes performances qu'un programme en langage C ou C++.

Robuste : Java est un langage fortement typé et très strict. Le code est vérifié à la compilation et à l'exécution, ce qui permet de réduire les bugs et les problèmes d'incompatibilités de versions.

Sécurisé : Le moteur d'exécution de java (JRE Java Runtime Environment) s'occupe de la sécurité des applications et des systèmes

Historique des versions du langage

Initialement (1991) « Oak » était un projet de langage pour l'électronique grand public développé au sein de la société Sun (rachetée depuis par Oracle)

Principal concepteur : James Gosling

Transforme en langage pour le Web — sous le nom de « Java » — grâce à sa portabilité (1994/95)

Lancement officiel en mai 1995

1996 : Java 1.0

1997 : Java 1.1

Nombreuses optimisations, modification du modèle des événements pour AWT, ... ~400 classes prédéfinies

1998 : Java 1.2 (renomme Java2 ou J2SE 1.2)

Ajouts : collections, Swing, Java2D, ... ~1200 classes prédéfinies

2000 : Java 1.3 (J2SE 1.3)

2002 : Java 1.4 (J2SE 1.4)

2004 : Java 1.5 (J2SE 1.5)

Ajouts : généricité (proche des templates de C++), types énumérés, web services, ... ~2500 classes prédéfinies

2006 : Java 1.6 (ou « Mustang » ou Java SE 6)

Améliorations des performances (Swing notamment)

Ajouts : interactions avec scripts (PHP, Python, Ruby, JavaScript), ...

2011 : Java 1.7 (ou « Dolphin » ou Java SE 7)

Améliorations de la généricité et des instructions switch, catch, ...

NIO (new I/O), try-with, ... ~3000 classes prédéfinies

2014 : Java 1.8 (ou « Wolf » ou Java SE 8)

Améliorations de la gestion du temps et des durées (classes Instant, Duration, ...)

Ajouts : lambda expressions, streams, ... ~4000 classes prédéfinies

~2016 : Java 1.9

Les différentes « éditions » du langage Java

Plusieurs « éditions » (ou « distributions ») du langage coexistent :

- Java Standard Edition (J2SE ou Java SE)
- Java Enterprise Edition (J2EE ou Java EE) applications réparties (e.g. services web)
- Java Micro Edition (J2ME ou Java ME) applications embarquées (e.g. smartphones, TVs, ...)
- Java Card applications embarquées sur cartes à puce (smartcards)

Les différentes étapes de création d'un programme Java

I. Création des fichiers sources

La création d'un code java se fait à l'intérieur d'une classe qui elle-même est contenu dans un fichier portant l'extension java. Plusieurs classes peuvent exister dans un même fichier .java mais une seule peut être déclarée publique (À voir dans le prochain Chapitre) et c'est cette classe qui donne son nom au fichier.

Les fichiers sources Java sont des fichiers texte sans mise en forme. Un éditeur de fichier simple supportant le format texte ASCII, tel que bloc-notes sous Windows ou vi sous Unix permet ainsi d'écrire des fichiers sources java.

Toutefois, il existe des éditeurs open source comme Eclipse ou sous licence RAD d'IBM permettant d'écrire des applications java complexes.

Dans ce cours nous allons utiliser Eclipse qui est un outil de développement Java (IDE) open source, gratuit et qui offre beaucoup de fonctionnalités permettant de créer plus facilement des programmes java complexes.

II. Compilation du fichier source

Une fois le fichier source créé, il faut le compiler pour pouvoir l'exécuter. La compilation se fait avec l'outil de compilation **javac** fourni avec le kit de développement java (SDK Software Development Kit) disponible en téléchargement sur le site d'Oracle.

Les étapes de compilation d'un programme java sont :

- Ouvrir une fenêtre invite de commande
- Placer vous dans le dossier contenant le fichier java à l'aide de la commande **cd**
- Une fois dans le fichier contenant la source java, compiler le fichier avec la commande suivante : **javac <nomFichier.java>**

Javac : compilateur java fourni avec le SDK

<nomFichier.java> : nom du fichier source java avec l'extension .java

En cas d'erreur lors du lancement de la commande, un message d'erreur est retourné par la commande.

Il est possible de compiler plusieurs fichiers sources en même temps en utilisant la commande **javac** suivi des noms des fichiers à compiler avec l'extension .java en les séparant par un espace.

Syntaxe : **javac <nomFichier1.java> <nomFichier2.java> ...**

Le résultat de la compilation d'un fichier source java est la création d'un fichier binaire portant le même nom que le fichier source mais avec l'extension .class.

Un fichier binaire .class contient le pseudo-code java qui peut être interprété par la machine virtuelle java.

Par défaut les fichiers compilés sont créés dans le même répertoire que les fichiers sources. Il est possible d'indiquer le répertoire de création des fichiers binaires à l'outil **javac** avec l'option -d « directory ».

Syntaxe : **javac -d C:\apps\ <nomFichier.java>**

III. Exécution du programme Java

L'exécution d'un programme java nécessite d'un interpréteur java (la machine virtuelle java) qui charge la méthode **main()** de la classe principale de l'application.

Pour lancer l'exécution d'une application java, il faut utiliser l'outil **java** fourni avec le JDK.

- Ouvrir une fenêtre Invite de commandes

- Placez-vous dans le répertoire qui contient le ou les fichiers binaires (.class) de votre application.
- Exécuter le ou les fichiers binaires avec la commande suivante :

```
java <fichierMain> <argument1> <argument2>
```

java : outil en ligne de commande qui lance l'exécution de la machine virtuelle java.

<fichierMain> : est le nom du fichier binaire (.class) qui contient le point d'entrée de l'application, la méthode **main()**.

Note : ne pas mettre l'extension .class après le nom du fichier dans la commande car ceci est fait implicitement par la machine virtuelle java.

<argument1>, <argument2> : éventuels arguments de la commande java à passer à l'application à exécuter.

En cas d'erreur lors du lancement de la commande, un message d'erreur est retourné par la commande.

Les raisons qui peuvent causer une erreur sont entre autres :

- Le nom du fichier à exécuter ne porte pas le même nom que la classe (différence entre majuscules et minuscules).
- L'extension .class est ajoutée après le nom du fichier dans la commande
- Le fichier à exécuter ne contient pas de méthode main()
- Le fichier binaire (.class) n'est pas dans le répertoire où la commande est lancée.

Exemple de programme Java

I. Création du fichier source

Tout d'abord, créer le répertoire qui va contenir les fichiers java « C:\uvs\apps ».

Démarrez votre éditeur. Vous pouvez lancer l'éditeur de bloc-notes à partir du menu Démarrer en sélectionnant Programmes> Accessoires> Bloc-notes. Dans un nouveau document, tapez le code suivant:

```
/**
 * La classe BonjourMondeApp est une application java
 * qui affiche simplement "Bonjour le monde !" dans la sortie standard.
 */
class BonjourMondeApp {
    public static void main(String[] args) {
        System.out.println("Bonjour le monde !"); // Affiche le texte.
    }
}
```

Enregistrez le code dans un fichier avec le nom *BonjourMondeApp.java*. Pour ce faire dans le Bloc-notes, choisissez d'abord l'élément de menu Fichier> Enregistrer sous. Puis, dans la boîte de dialogue Enregistrer sous, sélectionner le répertoire C:\uvs\apps. Dans la zone de liste déroulante Encoding, laissez l'encodage comme ANSI.

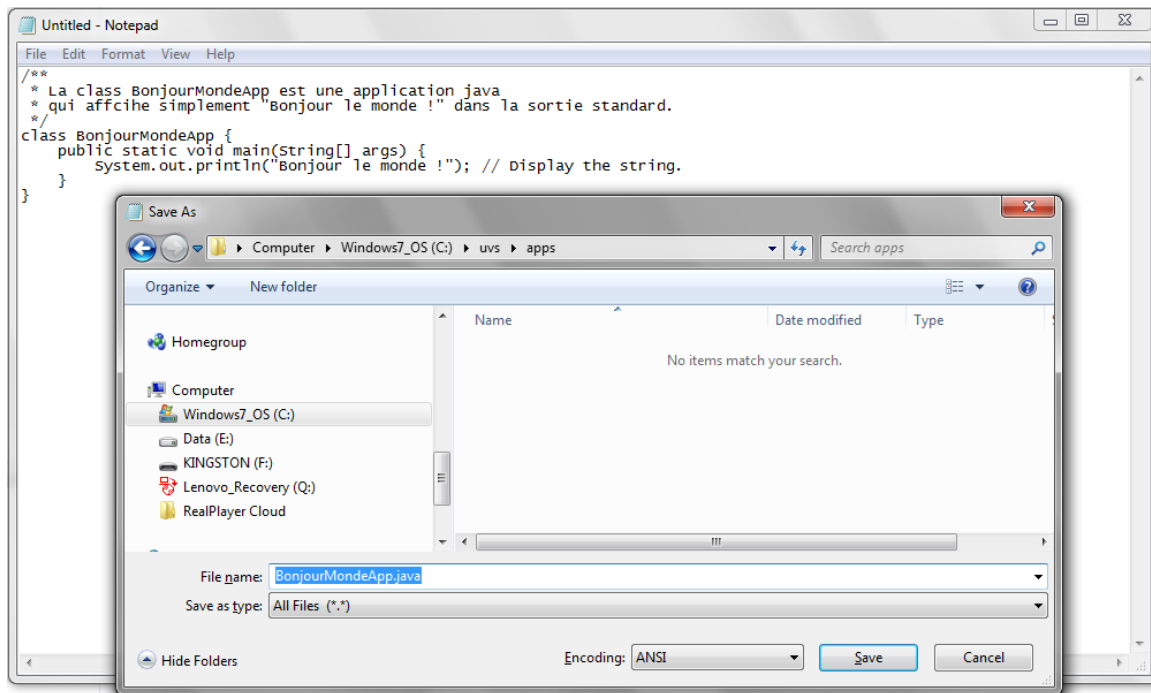


Figure 4 : Création du code source avec le Bloc-notes

II. Compilation du fichier

Compilez le fichier source dans un fichier .class

Lancer la fenêtre d'invite de commandes. Vous pouvez le faire à partir du menu Démarrer en choisissant Accessoires > Tous les programmes> Invite de commandes. La fenêtre doit ressembler à la figure suivante.

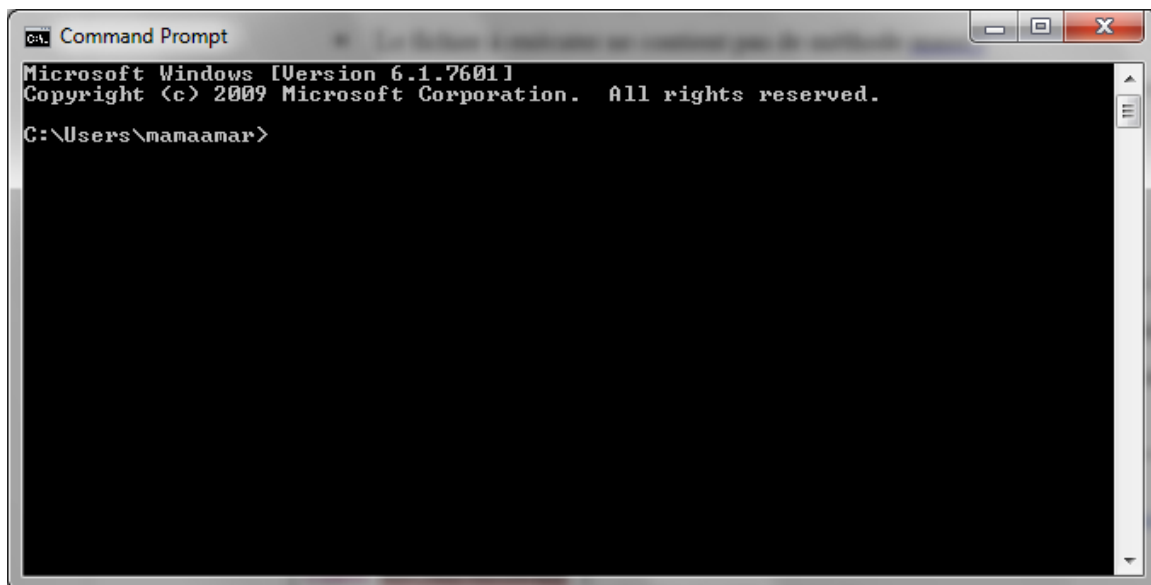
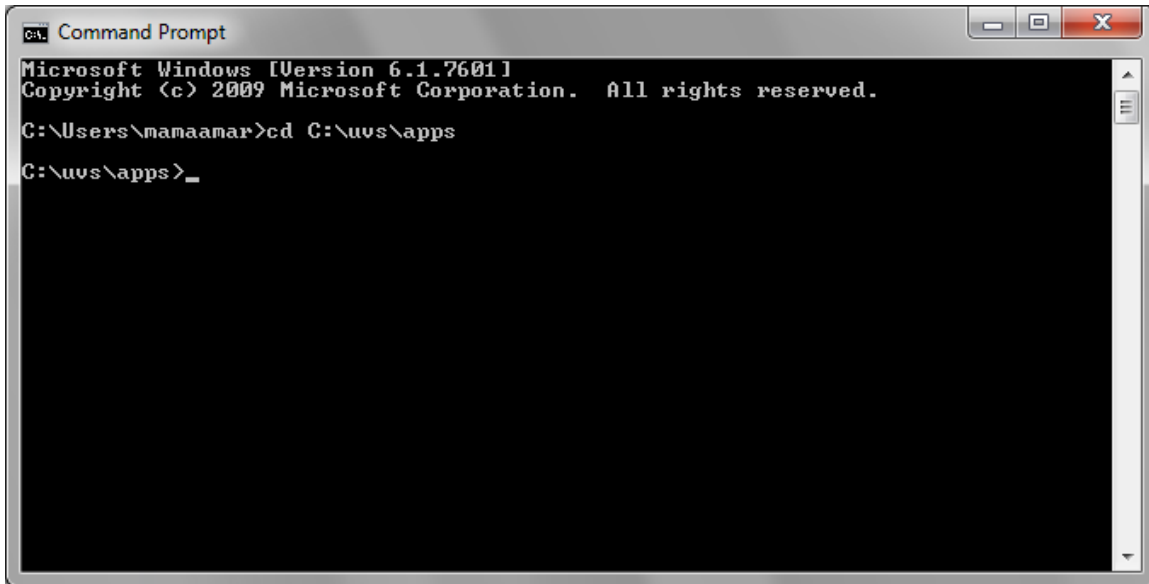


Figure 5 : L'invite de commandes windows

Placer vous dans le répertoire « C:\uvs\apps » où se trouve le fichier source « BonjourMondeApp.java » avec la commande : `cd C:\uvs\apps`.

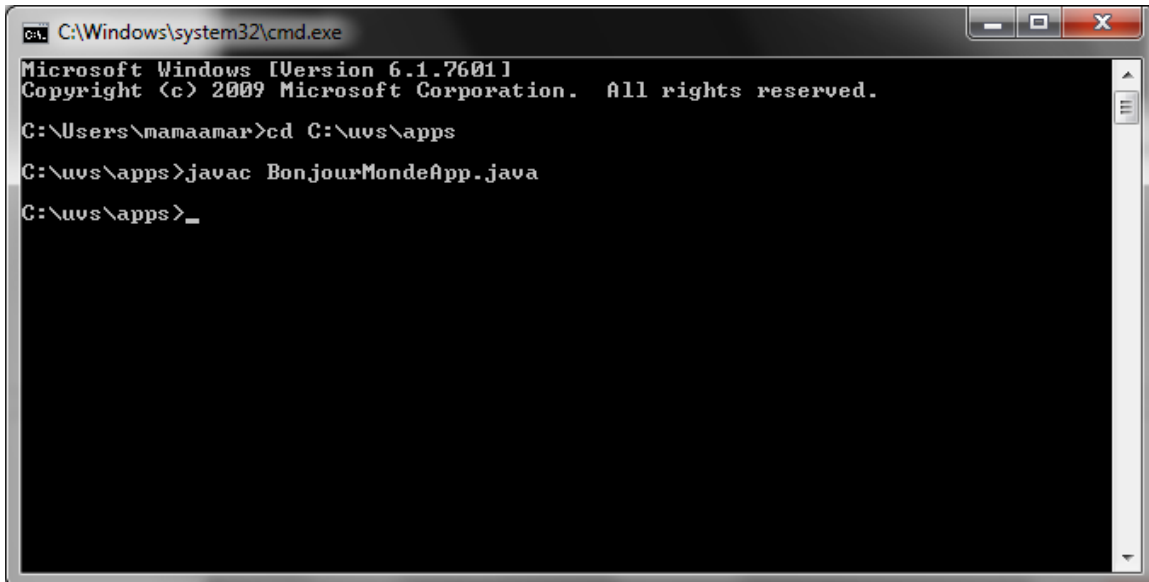


```
C:\ Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\mamaamar>cd C:\uvs\apps
C:\uvs\apps>_
```

Figure 6 : Positionnement dans le répertoire du code java

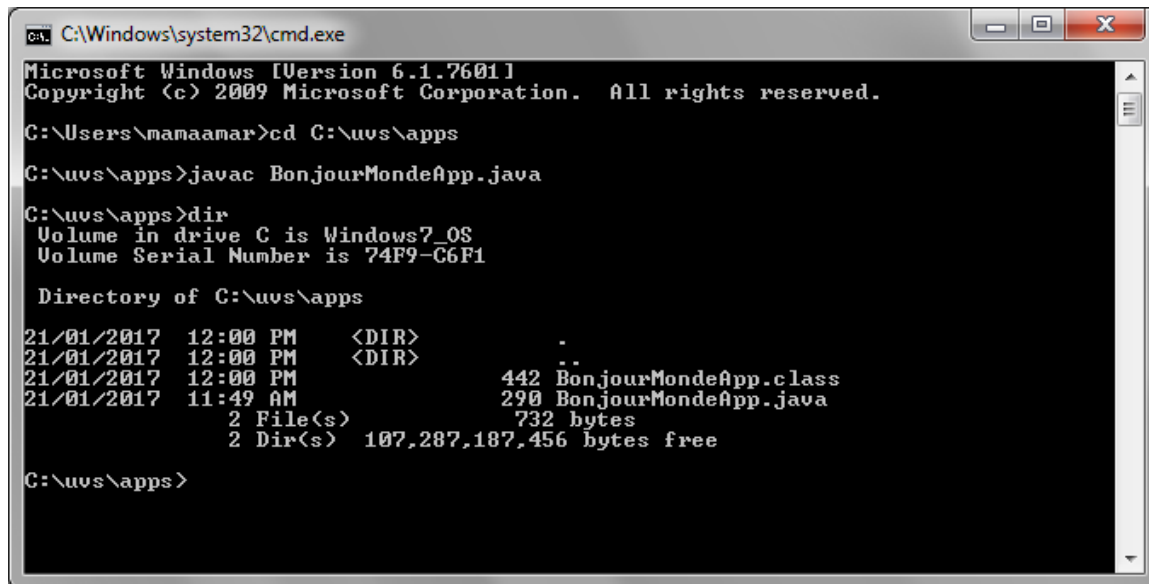
Maintenant, vous êtes prêt à compiler. À l'invite de commandes, tapez la commande suivante et appuyez sur Entrée :
javac BonjourMondeApp.java.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\mamaamar>cd C:\uvs\apps
C:\uvs\apps>javac BonjourMondeApp.java
C:\uvs\apps>_
```

Figure 7 : compilation du fichier en ligne de commande



```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\mamaamar>cd C:\uvs\apps
C:\uvs\apps>javac BonjourMondeApp.java
C:\uvs\apps>dir
Volume in drive C is Windows7_OS
Volume Serial Number is 74F9-C6F1

Directory of C:\uvs\apps

21/01/2017  12:00 PM    <DIR>          .
21/01/2017  12:00 PM    <DIR>          ..
21/01/2017  12:00 PM                442 BonjourMondeApp.class
21/01/2017  11:49 AM                290 BonjourMondeApp.java
                2 File(s)              732 bytes
                2 Dir(s)  107,287,187,456 bytes free

C:\uvs\apps>

```

Figure 8 : Affichage du contenu du répertoire du code source

III. Exécution du programme

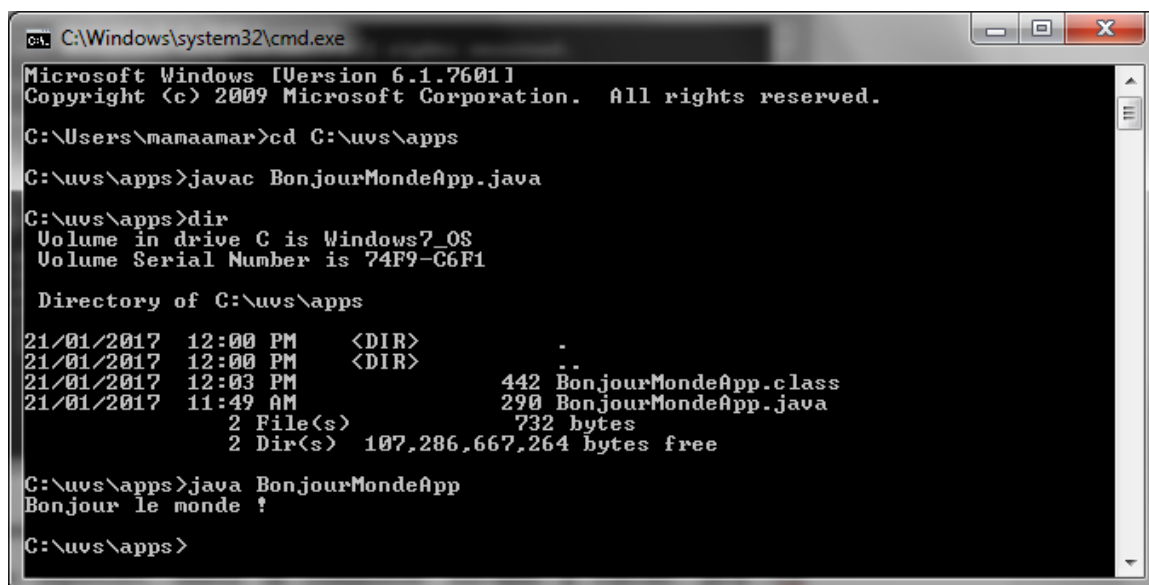
Dans le même répertoire, entrez la commande suivante à l'invite: `java BonjourMondeApp`

Vous devriez voir ce qui suit sur votre écran:

```
C:\uvs\dev>java BonjourMondeApp
```

Bonjour le monde!

```
C:\uvs\apps>
```



```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\mamaamar>cd C:\uvs\apps
C:\uvs\apps>javac BonjourMondeApp.java
C:\uvs\apps>dir
Volume in drive C is Windows7_OS
Volume Serial Number is 74F9-C6F1

Directory of C:\uvs\apps

21/01/2017  12:00 PM    <DIR>          .
21/01/2017  12:00 PM    <DIR>          ..
21/01/2017  12:03 PM                442 BonjourMondeApp.class
21/01/2017  11:49 AM                290 BonjourMondeApp.java
                2 File(s)              732 bytes
                2 Dir(s)  107,286,667,264 bytes free

C:\uvs\apps>java BonjourMondeApp
Bonjour le monde !

C:\uvs\apps>

```

Figure 9 : Exécution du programme java

Félicitations à vous! Votre programme fonctionne!