



# JAVA 8 – PRESENTATION

JSE – SEQUENCE 1

LYCEE PASTEUR MONT-ROLAND  
Julian.courbez@gmail.com

## Table des matières

1.	Avant-propos.....	3
2.	Historique.....	3
2.1.	Pourquoi Java ?.....	3
2.2.	Objectifs de la conception de Java.....	3
2.3.	Essor de Java.....	4
3.	Caractéristique de Java.....	5
3.1.	Le langage de programmation Java.....	5
	Simple.....	5
	Orienté objet.....	6
	Distribué.....	6
	Interprété.....	6
	Robuste.....	7
	Sécurisé.....	7
	Indépendant des architectures.....	7
	Portable.....	8
	Performant.....	8
	Multitâche.....	8
	Dynamique.....	8
3.2.	La plate-forme Java.....	8
	La machine virtuelle Java (JVM).....	9
	L'API Java.....	10
	Les outils de déploiement des applications.....	12
	Les outils d'aide au développement.....	12
3.3.	Cycle de conception d'un programme Java.....	13
4.	Installation du SDK version Win32 pour l'environnement Windows.....	13
4.1.	Téléchargement.....	13
4.2.	Installation.....	13
4.3.	Configuration.....	15
4.4.	Test de la configuration du SDK.....	15
4.5.	Installation de la documentation du SDK et des API standard.....	16
5.	Les différentes étapes de création d'un programme Java.....	17
5.1.	Création des fichiers source.....	17
5.2.	Compiler un fichier source.....	18
5.3.	Exécuter une application.....	20
6.	Votre première application Java.....	20

6.1. Squelette d'une application .....	20
public .....	21
static .....	21
void .....	21
main.....	21
6.2. Arguments en ligne de commande .....	22
Principes et utilisation .....	22
Passage d'arguments à une application Java au moment de l'exécution.....	22

## 1. Avant-propos

Lorsque les ingénieurs de Sun Microsystems ont développé en 1991 le langage Java, ils ne pensaient certainement pas que vingt ans plus tard il serait l'un des langages les plus utilisés au monde. À l'origine prévu pour le développement d'applications destinées à des systèmes embarqués, il est maintenant présent dans tous les domaines de l'informatique. Il apparaît comme un des langages les plus demandés dans la majorité des offres d'emploi dans le domaine du développement.

C'est un langage dont la syntaxe est simple mais rigoureuse. Il permet donc de prendre rapidement de bonnes habitudes lorsque l'on débute.

Nous allons faire découvrir les bases de ce langage pour vous permettre ensuite d'évoluer vers le développement d'applications importantes utilisant les nombreuses technologies disponibles avec ce langage (JEE, JME...).

## 2. Historique

### 2.1. Pourquoi Java ?

Bill Joy, ingénieur chez Sun Microsystems, et son équipe de chercheurs travaillaient sur le projet "Green" qui consistait à développer des applications destinées à une large variété de périphériques et systèmes embarqués (notamment téléphones cellulaires et téléviseurs interactifs).

Convaincus par les avantages de la programmation orientée objet (POO), ils choisissaient de développer avec le langage C++ éprouvé pour ses performances.

Mais, par rapport à ce genre de projet, C++ a rapidement montré ses lacunes et ses limites. En effet, de nombreux problèmes d'incompatibilité se sont posés par rapport aux différentes architectures matérielles (processeurs, taille mémoire) et aux systèmes d'exploitation rencontrés, ainsi qu'au niveau de l'adaptation de l'interface graphique des applications et de l'interconnexion entre les différents appareils.

En raison des difficultés rencontrées avec C++, il était préférable de créer un nouveau langage autour d'une nouvelle plate-forme de développement. Deux développeurs de chez Sun, James Gosling et Patrick Naughton se sont attelés à cette tâche.

La création de ce langage et de cette plate-forme s'est inspirée des fonctionnalités intéressantes offertes par d'autres langages tels que C++, Eiffel, SmallTalk, Objective C, Cedar/ Mesa, Ada, Perl. Le résultat est une plate-forme et un langage idéaux pour le développement d'applications sécurisées, distribuées et portables sur de nombreux périphériques et systèmes embarqués interconnectés en réseau mais également sur Internet (clients légers), et sur des stations de travail (clients lourds).

D'abord surnommé C++- (C++ sans ses défauts) puis OAK, mais il s'agissait d'un nom déjà utilisé dans le domaine informatique, il fut finalement baptisé Java, mot d'argot voulant dire café, en raison des quantités de café ingurgité par les programmeurs et notamment par ses concepteurs. Et ainsi, en 1991, est né le langage Java.

### 2.2. Objectifs de la conception de Java

Par rapport aux besoins exprimés, il fallait un langage et une plate-forme simples et performants, destinés au développement et au déploiement d'applications sécurisées, sur des systèmes hétérogènes dans un environnement distribué, devant consommer un minimum de ressources et fonctionner sur n'importe quelle plate-forme matérielle et logicielle.

La conception de Java a apporté une réponse efficace à ces besoins :

- Langage d'une syntaxe simple, orienté objet et interprété, permettant d'optimiser le temps et le cycle de développement (compilation et exécution).
- Les applications sont portables sans modification sur de nombreuses plates-formes matérielles et systèmes d'exploitation.
- Les applications sont robustes car la gestion de la mémoire est prise en charge par le moteur d'exécution de Java (*Java Runtime Environment*), et il est plus facile d'écrire des programmes sans erreur par rapport au C++, en raison d'un mécanisme de gestion des erreurs plus évolué et plus strict.
- Les applications et notamment les applications graphiques sont performantes en raison de la mise en œuvre et de la prise en charge du fonctionnement de multiples processus légers (thread et multithreading).
- Le fonctionnement des applications est sécurisé, notamment dans le cas d'applet Java où le moteur d'exécution de Java veille à ce qu'aucune manipulation ou opération dangereuse ne soit effectuée par l'applet.

### 2.3. Essor de Java

Malgré la création de Java, les développements du projet "Green" n'ont pas eu les retombées commerciales escomptées et le projet fut mis de côté.

À cette époque, l'émergence d'Internet et des architectures client/serveur hétérogènes et distribuées a apporté une certaine complexité au développement des applications.

Les caractéristiques de Java se trouvent alors également fort intéressantes pour ce type d'applications.

En effet :

- Un programme Java étant peu encombrant, son téléchargement à partir d'un site Internet prend peu de temps.
- Un programme Java est portable et peut donc être utilisé sans modification sous n'importe quelle plate-forme (Windows, Macintosh, Unix, Linux...).

Java se trouve alors un nouveau domaine d'application sur le réseau mondial Internet, ainsi que sur les réseaux locaux dans une architecture intranet et client/serveur distribuée.

Pour présenter au monde les possibilités de Java, deux programmeurs de SUN, Patrick Naughton et Jonathan Peayne ont créé et présenté au salon SunWorld en mai 1995 un navigateur Web entièrement programmé en Java du nom de HotJava. Celui-ci permet l'exécution de programmes Java, nommés applets, dans les pages au format HTML.

En août 1995, la société Netscape, très intéressée par les possibilités de Java, signe un accord avec Sun lui permettant d'intégrer Java et l'implémentation des applets dans son navigateur Web (Netscape Navigator). En janvier 1996, Netscape version 2 arrive sur le marché en intégrant la plate-forme Java.

C'est donc Internet qui a assuré la promotion de Java.

Fort de cette réussite, Sun décide de promouvoir Java auprès des programmeurs en mettant à disposition gratuitement sur son site Web dès novembre 1995, une plate-forme de développement dans une version bêta du nom de JDK 1.0 (*Java Development Kit*).

Peu après, Sun crée une filiale du nom de JavaSoft (<http://java.sun.com>), dont l'objectif est de continuer à développer le langage.

Depuis, Java n'a fait qu'évoluer très régulièrement pour donner un langage et une plate-forme très polyvalents et sophistiqués, et de grandes compagnies telles que Borland/Inprise, IBM, Oracle, pour ne citer qu'eux, ont misé très fortement sur Java.

Début 2009, IBM effectue une tentative de rachat de Sun. Aucun accord n'étant trouvé sur le montant de la transaction, le projet de rachat échoue. Peu de temps après, Oracle fait à son tour une proposition de rachat qui cette fois se concrétise.

Java est aujourd'hui le premier langage objet enseigné dans les écoles et universités en raison de sa rigueur et de sa richesse fonctionnelle.

La communauté des développeurs Java représente plusieurs millions de personnes et est plus importante en nombre que la communauté des développeurs C++ (pourtant une référence).

### 3. Caractéristique de Java

Java est à la fois un langage et une plate-forme de développement.

Cette partie vous présente ces deux aspects, elle vous donnera un aperçu des caractéristiques de Java et vous aidera à évaluer l'importance de l'intérêt porté à Java.

#### 3.1. Le langage de programmation Java

Sun caractérise Java par le fait qu'il est simple, orienté objet, distribué, interprété, robuste, sécurisé, indépendant des architectures, portable, performant, multithread et dynamique.

Ces caractéristiques sont issues du livre blanc écrit en mai 1996 par James Gosling et Henry Mc Gilton et disponible à l'adresse suivante : <http://www.oracle.com/technetwork/java/langenv-140151.html>

Nous allons détailler chacune de ces caractéristiques.

##### Simple

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

- En Java, il y a seulement trois types primitifs : les numériques (entiers et réels), le type caractère et le type booléen. Les numériques sont tous signés.
- En Java, les tableaux et les chaînes de caractères sont des objets, ce qui en facilite la création et la manipulation.
- En Java, le programmeur n'a pas à s'occuper de la gestion de la mémoire. Un système nommé le "ramasse-miettes" (*garbage collector*) s'occupe d'allouer la mémoire nécessaire lors de la création des objets et de la libérer lorsque les objets ne sont plus référencés dans le contexte courant du programme (quand aucune variable n'y fait référence).
- En Java, pas de préprocesseur et pas de fichier d'en-tête. Les instructions `define` du C sont remplacées par des constantes en Java et les instructions `typedef` du C sont remplacées par des classes en Java.
- En C et C++, on définit des structures et des unions pour représenter des types de données complexes. En Java, on crée des classes avec des variables d'instance pour représenter des types de données complexes.
- En C++ , une classe peut hériter de plusieurs autres classes, ce qui peut poser des problèmes d'ambiguïté. Afin d'éviter ces problèmes, Java n'autorise que l'héritage simple mais apporte un mécanisme de simulation d'héritage multiple par l'implémentation d'une ou de plusieurs interfaces.

- En Java, il n'existe pas la célèbre instruction goto, tout simplement parce qu'elle apporte une complexité à la lecture des programmes et que bien souvent, on peut se passer de cette instruction en écrivant du code plus propre. De plus, en C et C++, le goto est généralement utilisé pour sortir de boucles imbriquées. En Java, nous utiliserons les instructions `break` et `continue` qui permettent de sortir d'un ou plusieurs niveaux d'imbrication.
- En Java, il n'est pas possible de surcharger les opérateurs, tout simplement pour éviter des problèmes d'incompréhension du programme. On préférera créer des classes avec des méthodes et des variables d'instance.
- Et pour finir, en Java, il n'y a pas de pointeurs mais plutôt des références sur des objets ou des cases d'un tableau (référéncées par leur indice), tout simplement parce qu'il s'est avéré que la manipulation des pointeurs est une grosse source de bugs dans les programmes C et C++.

### Orienté objet

Mis à part les types de données primitifs, tout est objet en Java. Et de plus, si besoin est, il est possible d'encapsuler les types primitifs dans des objets, des classes préfabriquées sont déjà prévues à cet effet.

Java est donc un langage de programmation orienté objet conçu sur le modèle d'autres langages (C++, Eiffel, SmallTalk, Objective C, Cedar/Mesa, Ada, Perl), mais sans leurs défauts.

Les avantages de la programmation objet sont : une meilleure maîtrise de la complexité (diviser un problème complexe en une suite de petits problèmes), un réemploi plus facile, une meilleure facilité de correction et d'évolution.

Java est fourni de base avec un ensemble de classes qui permettent de créer et manipuler toutes sortes d'objets (interface graphique, accès au réseau, gestion des entrées/sorties...).

### Distribué

Java implémente les protocoles réseau standard, ce qui permet de développer des applications client/serveur en architecture distribuée, afin d'invoquer des traitements et/ou de récupérer des données sur des machines distantes.

Pour cela, Java fournit de base deux API permettant de créer des applications client/serveur distribuées :

- RMI (*Remote Method Invocation*), qui permet de faire communiquer des objets Java s'exécutant sur différentes machines virtuelles Java et même sur différentes machines physiques.
- CORBA (*Common Object Request Broker Architecture*), basé sur le travail de l'OMG (<http://www.omg.org>), qui permet de faire communiquer des objets Java, C++ , Lisp, Python, Smalltalk, COBOL, Ada, s'exécutant sur différentes machines physiques.

### Interprété

Un programme Java n'est pas exécuté, il est interprété par la machine virtuelle ou JVM (*Java Virtual Machine*), ce qui le rend un peu plus lent. Mais cela apporte des avantages, notamment celui de ne pas être obligé de recompiler un programme Java d'un système à un autre car il suffit, pour chacun des systèmes, de posséder sa propre machine virtuelle Java.

Du fait que Java est un langage interprété, vous n'avez pas à faire l'édition des liens (obligatoire en C++) avant d'exécuter un programme. En Java, il n'y a donc que deux étapes : la compilation puis

l'exécution. L'opération d'édition des liens est réalisée par la machine virtuelle au moment de l'exécution du programme.

### Robuste

Java est un langage fortement typé et très strict. Par exemple, la déclaration des variables doit obligatoirement être explicite en Java.

Le code est vérifié (syntaxe, types) à la compilation et également au moment de l'exécution, ce qui permet de réduire les bugs et les problèmes d'incompatibilité de versions.

De plus, la gestion des pointeurs est entièrement prise en charge par Java et le programmeur n'a aucun moyen d'y accéder, ce qui évite des écrasements inopportuns de données en mémoire et la manipulation de données corrompues.

### Sécurisé

Vu les domaines d'application de Java, il est très important qu'il y ait un mécanisme qui veille à la sécurité des applications et des systèmes. C'est le moteur d'exécution de Java (JRE) qui s'occupe entre autres de cette tâche.

Le JRE s'appuie notamment sur le fichier texte `java.policy` qui contient des informations sur le paramétrage de la sécurité.

En Java, c'est le JRE qui gère la planification mémoire des objets et non le compilateur comme c'est le cas en C++.

Comme en Java il n'y a pas de pointeurs mais des références sur des objets, le code compilé contient des identifiants sur les objets qui sont ensuite traduits en adresses mémoire par le JRE, cette partie étant complètement opaque pour les développeurs.

Au moment de l'exécution d'un programme Java, le JRE utilise un processus nommé le `ClassLoader` qui s'occupe du chargement du bytecode (ou langage binaire intermédiaire) contenu dans les classes Java. Le bytecode est ensuite analysé afin de contrôler qu'il n'a pas fait de création ou de manipulation de pointeurs en mémoire et également qu'il n'y a pas de violation d'accès.

Comme Java est un langage distribué, les principaux protocoles d'accès au réseau sont implémentés (FTP, HTTP, Telnet...). Le JRE peut donc être paramétré afin de contrôler l'accès au réseau de vos applications :

- Interdire tous les accès.
- Autoriser l'accès seulement à la machine hôte d'où provient le code de l'application. C'est le cas par défaut pour les applets Java.
- Autoriser l'accès à des machines sur le réseau externe (au-delà du firewall), dans le cas où le code de l'application provient également d'un hôte sur le réseau externe.
- Autoriser tous les accès. C'est le cas par défaut pour les applications de type client lourd.

### Indépendant des architectures

Le compilateur Java ne produit pas du code spécifique pour un type d'architecture.

En fait, 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).



L'avantage de ce bytecode est qu'il peut facilement être interprété ou transformé dynamiquement en code natif pour des besoins de performance.

Il suffit de disposer de la machine virtuelle dédiée à sa plate-forme pour faire fonctionner un programme Java. C'est elle qui s'occupe de traduire le bytecode en code natif.

#### Portable

Ce qui fait tout d'abord que Java est portable, c'est qu'il s'agit d'un langage interprété.

De plus, contrairement au langage C et C++, les types de données primaires (numériques, caractère et booléen) de Java ont la même taille, quelle que soit la plate-forme sur laquelle le code s'exécute.

Les bibliothèques de classes standard de Java facilitent l'écriture du code qui peut ensuite être déployé sur différentes plates-formes sans adaptation.

#### Performant

Même si un programme Java est interprété, ce qui est plus lent qu'un programme natif, Java met en œuvre un processus d'optimisation de l'interprétation du code, appelé JIT (*Just In Time*) ou HotSpot, qui permet de compiler à la volée le bytecode Java en code natif, ce qui permet d'atteindre les mêmes performances qu'un programme écrit en langage C ou C++.

#### Multitâche

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

#### Dynamique

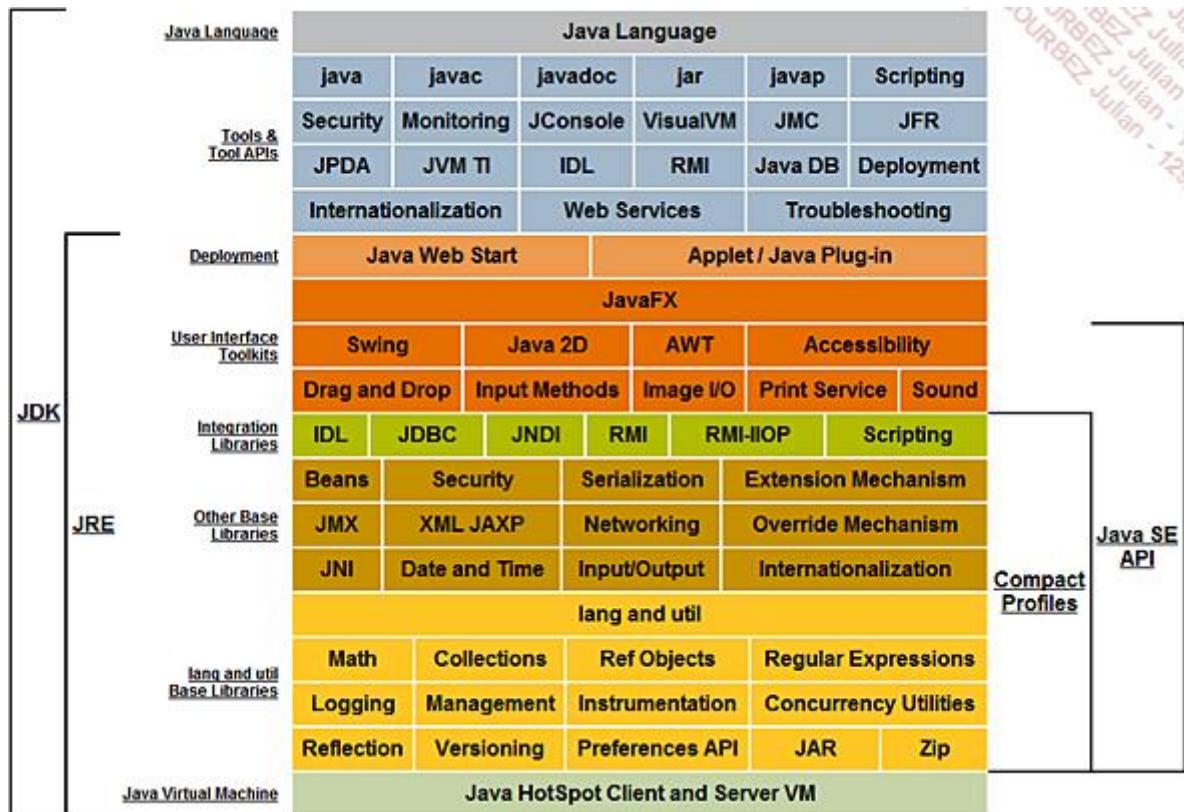
En Java, nous l'avons dit, le programmeur n'a pas à faire l'édition des liens (obligatoire en C et C++). Il est donc possible de modifier une ou plusieurs classes sans avoir à effectuer une mise à jour de ces modifications pour l'ensemble du programme. La vérification de l'existence des classes se fait au moment de la compilation et l'appel du code de ces classes ne se fait qu'au moment de l'exécution du programme. Ce procédé permet de disposer d'applications allégées en taille mémoire.

### 3.2. La plate-forme Java

Par définition, une plate-forme est un environnement matériel ou logiciel sur lequel peut s'exécuter un programme. La plupart des plates-formes actuelles sont la combinaison d'une machine et d'un système d'exploitation (ex : PC + Windows).

La plate-forme Java diffère par le fait qu'elle ne se compose que d'une partie logicielle qui s'exécute sur de nombreuses plates-formes matérielles et différents systèmes d'exploitation.

Le schéma suivant est issu du site web d'Oracle sur le langage Java et présente les différents composants de la plate-forme Java :



Comme le montre le schéma, elle est composée des éléments suivants :

- La machine virtuelle Java (JVM),
- L'interface de programmation d'application Java (API Java), qui est décomposée en trois catégories (API de bases, API d'accès aux données et d'intégration avec l'existant, API de gestion de l'interface avec l'utilisateur),
- Les outils de déploiement des applications,
- Les outils d'aide au développement.

Voyons en détail ces différents éléments.

### La machine virtuelle Java (JVM)

La machine virtuelle est la base de la plate-forme Java, elle est nécessaire pour l'exécution des programmes Java. La JVM est disponible pour de nombreux types d'ordinateurs et systèmes d'exploitation.

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 faite à un objet. Ce procédé permet également à la JVM de s'occuper de la libération automatique de la mémoire (ramasse-miettes) 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 : c'est l'une des opérations les plus complexes effectuées par la JVM. Au chargement du programme, 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 aux périphériques de la machine sur laquelle l'applet s'exécute et autorise 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. Vous pouvez également utiliser ce procédé pour des accès à des périphériques ou à des fonctionnalités qui ne sont pas implémentés directement où voir pas du tout en Java.

Le fait que Java soit interprété apporte des avantages et des inconvénients. Depuis toujours, on reproche à Java d'être moins performant que des langages natifs, ce qui était surtout le cas pour les applications avec interface utilisateur graphique. Afin de combler cette lacune et de perdre cette mauvaise image injustifiée, les développeurs de chez Oracle ont énormément travaillé sur l'optimisation de la JVM.

Avec la version 1.2, on avait un compilateur JIT (*Just In Time*) qui permettait d'optimiser l'interprétation du bytecode en modifiant sa structure pour le rapprocher du code natif. Depuis la version 1.3, la JVM intègre un processus nommé HotSpot (client et serveur) qui optimise davantage l'interprétation du code et d'une manière générale les performances de la JVM. HotSpot apporte un gain de performance allant de 30 % à 40 % selon le type d'application (on le remarque énormément au niveau des interfaces graphiques).

### L'API Java

L'API Java contient une collection de composants logiciels préfabriqués qui fournissent un grand nombre de fonctionnalités.

L'API Java dans sa version 8 est organisée en plus de 220 packages, l'équivalent des librairies en langage C. Chaque package contient les classes et interfaces préfabriquées et directement réutilisables. Vous avez donc à votre disposition environ 4300 classes et interfaces.

La plate-forme Java fournit des API de base. De nombreuses extensions peuvent être ajoutées et sont disponibles sur le site Java d'Oracle : gestion des images en 3D, des ports de communication de l'ordinateur, de la téléphonie, des courriers électroniques...

L'API Java peut être décomposée en trois catégories :

### Les API de base

Les API de base permettent de gérer :

- Les éléments essentiels comme les objets, les chaînes de caractères, les nombres, les entrées/sorties, les structures et collections de données, les propriétés système, la date et l'heure, et plus encore...
- Les applets Java dans l'environnement du navigateur Web.
- Le réseau, avec les protocoles standard tels que FTP, HTTP, UDP, TCP/IP plus les URL et la manipulation des sockets.
- L'internationalisation et l'adaptation des programmes Java, en externalisant les chaînes de caractères contenues dans le code dans des fichiers de propriétés (.properties). Ce procédé permet d'adapter le fonctionnement des applications par rapport à des paramètres changeants (nom serveur, nom d'utilisateur, mot de passe...) et d'adapter la langue utilisée dans les interfaces graphiques par rapport aux paramètres régionaux de la machine.

- L'interfaçage avec du code natif, en permettant de déclarer que l'implémentation d'une méthode est faite au sein d'une fonction d'une DLL par exemple.
- La sécurité, en permettant :
  - De crypter/décrypter les données (JCE - *Java Cryptography Extension*),
  - De mettre en œuvre une communication sécurisée via SSL et TLS (JSSE - *Java Secure Socket Extension*),
  - D'authentifier et de gérer les autorisations des utilisateurs dans les applications (JAAS - *Java Authentication and Authorization Service*),
  - D'échanger des messages en toute sécurité entre des applications communiquant via un service comme Kerberos (GSS-API - *Generic Security Service - Application Programming Interface*),
  - De créer et valider des listes de certificats nommées Certification Paths (*Java Certification Path API*).
- La création de composants logiciels du nom de JavaBeans réutilisables et capables de communiquer avec d'autres architectures de composants tels que ActiveX, OpenDoc, LiveConnect.
- La manipulation de données XML (*eXtensible Markup Language*) à l'aide des API DOM (*Document Object Model*) et SAX (*Simple API for XML*). Les API de base permettent aussi d'appliquer des transformations XSLT (*eXtensible Style Sheet Transformation*) à partir de feuilles de style XSL sur des données XML.
- La génération de fichiers de journalisation (logs) permettant d'avoir un compte rendu du fonctionnement des applications (activité, erreurs, bugs...).
- La manipulation de chaînes de caractères avec des expressions régulières.
- Les erreurs système et applicatives avec le mécanisme des exceptions chaînées.
- Les préférences utilisateur ou système, en permettant aux applications de stocker et récupérer des données de configuration dans différents formats.

#### *Les API d'accès aux données et d'intégration avec l'existant*

Les API d'intégration permettent de gérer :

- Des applications client/serveur dans une architecture distribuée, en permettant la communication en local ou par le réseau entre des objets Java fonctionnant dans des contextes de JVM différents, grâce à l'API RMI (*Remote Method Invocation*).
- Des applications client/serveur dans une architecture distribuée, en permettant la communication en local ou par le réseau entre des objets Java et des objets compatibles CORBA tels que C++ , Lisp, Python, Smalltalk, COBOL, Ada, grâce au support de l'API CORBA (*Common Object Request Broker Architecture*), basé sur le travail de l'OMG (<http://www.omg.org>).
- L'accès à pratiquement 100 % des bases de données, via l'API JDBC (*Java DataBase Connectivity*).
- L'accès aux données stockées dans des services d'annuaire au protocole LDAP (*Lightweight Directory Access Protocol*) comme par exemple l'Active Directory de Windows, via l'API JNDI (*Java Naming and Directory Interface*).

#### *Les API de gestion de l'interface des applications avec l'utilisateur*

Les API de gestion de l'interface utilisateur permettent de gérer :

- La conception des interfaces graphiques avec l'API AWT (*Abstract Window Toolkit*) d'ancienne génération, ou l'API SWING de nouvelle génération.

- Le son, avec la manipulation, la lecture et la création de fichiers son de différents formats (.wav ou .midi).
- La saisie de données textuelles par d'autres moyens que le clavier, comme par exemple des mécanismes de reconnaissance vocale ou de reconnaissance d'écriture, avec l'API Input Method Framework.
- Les opérations graphiques de dessin avec l'API Java 2D et de manipulation d'images avec l'API Java Image I/O.
- L'accessibilité des applications aux personnes handicapées avec l'API Java Accessibility qui permet de s'interfacer par exemple avec des systèmes de reconnaissance vocale ou des terminaux en braille.
- Le déplacement ou transfert de données lors d'une opération glisser/déposer (*Drag and Drop*).
- Des travaux d'impression de données sur tout périphérique d'impression.

### Les outils de déploiement des applications

La plate-forme Java fournit deux outils permettant d'aider au déploiement des applications :

- Java Web Start : destiné à simplifier le déploiement et l'installation des applications Java autonomes. Les applications sont disponibles sur un serveur, les utilisateurs peuvent en lancer l'installation sur leur machine via la console Java Web Start et tout se fait alors automatiquement. Ce qui est intéressant, c'est qu'ensuite à chaque lancement d'une application, Java Web Start vérifie si une mise à jour est disponible sur le serveur et procède automatiquement à son installation.
- Java Plug-in : destiné à permettre le fonctionnement des applets Java avec la machine virtuelle 8. En effet, lorsque vous accédez, via votre navigateur web, à une page html qui contient une applet, c'est la machine virtuelle du navigateur qui est chargée de la faire fonctionner. Le problème, c'est que les machines virtuelles des navigateurs supportent d'anciennes versions de Java. Afin de ne pas être limité au niveau des fonctionnalités et donc de ne pas rencontrer des problèmes d'incompatibilité entre les navigateurs, vous pouvez installer le Java Plug-in sur les postes clients. Le Java Plug-in consiste à installer un moteur d'exécution Java 8 (le JRE étant composé d'une JVM et de l'ensemble des API) et à faire en sorte que les navigateurs Web utilisent cette JRE et non la leur.

### Les outils d'aide au développement

La plupart des outils d'aide au développement sont contenus dans le répertoire bin sous le répertoire racine de l'installation du J2SE.

Les principaux outils d'aide au développement permettent :

- De compiler (javac.exe) vos codes source .java en fichier .class.
- De générer la documentation (javadoc.exe) automatique de vos codes source (nom de classe, package, hiérarchie d'héritage, liste des variables et méthodes) avec le même style de présentation que la documentation officielle des API standard fournies par Sun.
- De lancer l'exécution (java.exe) des applications autonomes Java.
- De visualiser, à l'aide d'une visionneuse (appletviewer.exe), l'exécution d'une applet Java dans une page HTML.

Deux autres technologies sont également intéressantes. Elles sont destinées à des outils de développement tiers afin qu'ils puissent les intégrer :

- JPDA (*Java Platform Debugger Architecture*), qui permet d'intégrer un outil de débogage au sein de son IDE de développement, apportant des fonctionnalités telles que les points d'arrêts, le pas à pas, l'inspection des variables et expressions...
- JVMPI (*Java Virtual Machine Profiler Interface*), qui permet d'effectuer des analyses et de générer des états sur le fonctionnement des applications (mémoire utilisée, objets créés, nombre et fréquence d'invocation des méthodes, temps de traitement...) afin d'observer le bon fonctionnement des applications et de repérer où sont les "goulets d'étranglement".

### 3.3. Cycle de conception d'un programme Java

Pour développer une application Java, il faut d'abord se procurer la plate-forme J2SE de développement (SDK - *Software Development Kit*) dédiée à sa machine et à son système d'exploitation, dont vous trouverez la liste sur le site Java

d'Oracle : <http://www.oracle.com/technetwork/java/index.html>

Ensuite, vous pouvez utiliser les API standard de Java pour écrire vos codes sources. En Java, la structure de base d'un programme est la classe et chaque classe doit être contenue dans un fichier portant l'extension java. Plusieurs classes peuvent être contenues dans un même fichier .java, mais une seule de ces classes peut être déclarée publique. Et c'est le nom de cette classe déclarée publique qui donne son nom au fichier .java.

Au cours du développement, vous pouvez procéder à la phase de compilation en utilisant l'outil javac.exe. Vous obtenez comme résultat au moins un fichier portant le même nom mais avec l'extension .class. Le fichier .class compilé reste tout de même indépendant de toute plate-forme ou système d'exploitation.

Ensuite, c'est l'interpréteur (java.exe) qui exécute les programmes Java. Pour l'exécution des applets, l'interpréteur est incorporé au navigateur Internet compatible Java. Pour l'exécution d'applications Java autonomes, il est nécessaire de lancer l'exécution de la machine virtuelle fournie soit avec la plate-forme de développement Java (SDK), soit avec le kit de déploiement d'applications Java (JRE - *Java Runtime Environment*).

## 4. Installation du SDK version Win32 pour l'environnement Windows

### 4.1. Téléchargement

Dans un premier temps, il vous faut télécharger la dernière version du SDK pour l'environnement Windows (Win32) à partir du site web

d'Oracle : <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Actuellement, le fichier à télécharger se nomme jdk-8u111-windows-i586.exe et fait 190 Mo. Dans tous les cas, téléchargez toujours la dernière version disponible.

Puisque vous êtes sur le site web d'Oracle, profitez-en pour télécharger un autre élément qui s'avère indispensable pour programmer en Java : la documentation des API standard.

Actuellement, le fichier à télécharger se nomme jdk-8u111-apidocs.zip et fait 85 Mo. Pour pouvoir le décompresser sur votre machine, il vous faut 300 Mo d'espace disque disponible.

### 4.2. Installation

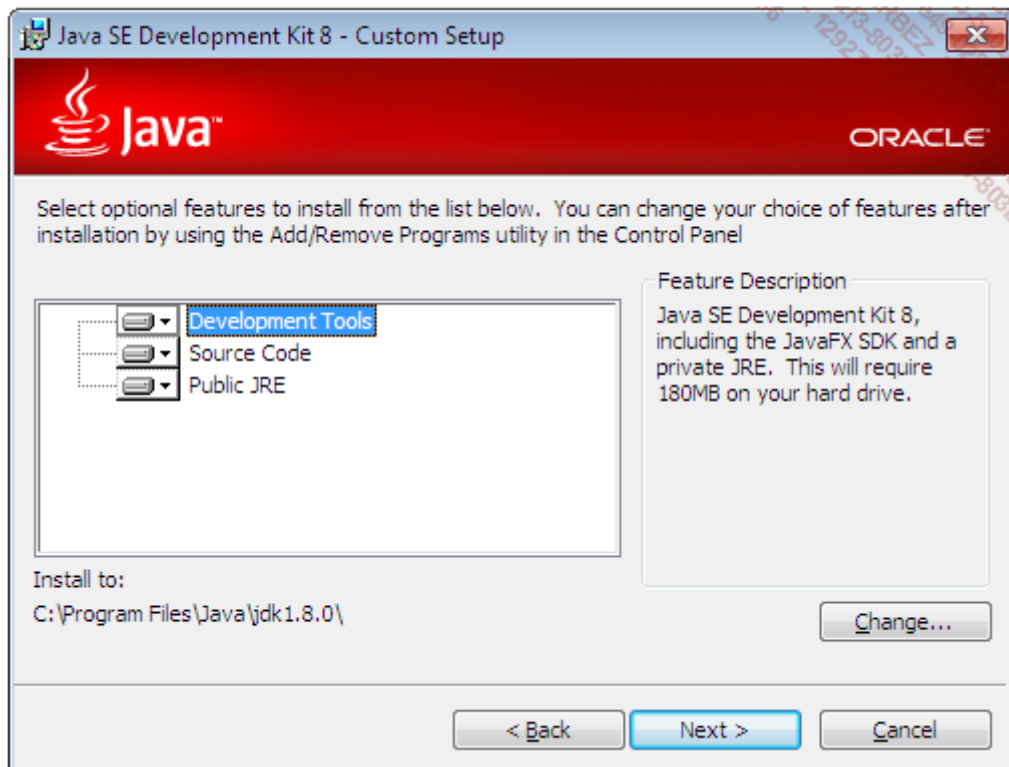
Avant d'installer le SDK sur votre ordinateur, assurez-vous qu'il n'y a aucun autre outil de développement Java d'installé, ceci afin d'éviter les problèmes de conflits de configuration.

Pour commencer l'installation, double cliquez sur le fichier d'installation précédemment téléchargé : jdk-8u111-windows-i586.exe.

Tout d'abord une boîte de message **Welcome** apparaît vous indiquant que vous êtes sur le point d'installer le SDK et vous demande si vous voulez poursuivre l'installation.

Cliquez sur **Next**.

Une nouvelle fenêtre apparaît, **Custom Setup** qui vous permet de sélectionner les éléments du SDK à installer et le répertoire de destination de l'installation.



Après avoir fait vos choix ou avoir laissé la sélection par défaut, cliquez sur **Next**. Le programme installe alors les fichiers sur votre ordinateur. Après quelques instants, la boîte de dialogue suivante vous informe sur le succès de l'installation.





#### 4.3. Configuration

Il reste à configurer le système, en indiquant dans quel répertoire sont stockés les outils tels que `java.exe` (machine virtuelle) `appletviewer.exe` (visionneuse d'applets) ou encore `javac.exe` (compilateur). Pour ce faire, il faut modifier la variable d'environnement `PATH` pour ajouter le chemin d'accès vers le répertoire `bin` du `jdk`. Si vous avez conservé les options par défaut lors de l'installation, ce chemin doit être `C:\Program Files\Java\jdk1.8.0\bin`.

#### 4.4. Test de la configuration du SDK

Vous allez tester si l'ordinateur a bien pris en compte les modifications que vous venez d'apporter à la variable `PATH` et donc vérifier s'il trouve le chemin où sont situés les outils du SDK.

Pour tester la configuration du SDK, utilisez une fenêtre **Invite de commandes**.

À l'invite de commandes, saisissez la commande suivante qui va permettre de déterminer si l'installation du SDK est correcte ou non : `java -version`

Vous devez voir le message suivant apparaître en réponse à la ligne que vous avez saisie :

```
Microsoft Windows [version 10.0.14393]
(c) 2016 Microsoft Corporation. Tous droits réservés.

C:\Users\Moff>java -version
java version "1.8.0_112"
Java(TM) SE Runtime Environment (build 1.8.0_112-b15)
Java HotSpot(TM) 64-Bit Server VM (build 25.112-b15, mixed mode)

C:\Users\Moff>
```



Cette commande affiche des informations concernant la version de la machine virtuelle Java.

Si vous obtenez un message du style : 'java' n'est pas reconnu en tant que commande interne ou externe, un programme exécutable ou un fichier de commandes, cela signifie que le répertoire où sont stockés les outils du SDK n'a pas été trouvé par votre système.

Dans ce cas, vérifiez si la variable PATH contient bien les modifications que vous avez apportées et que vous n'avez pas fait d'erreur de syntaxe en spécifiant le chemin du répertoire bin.

#### 4.5. Installation de la documentation du SDK et des API standard

À l'aide d'un utilitaire de décompression tel que WinZip, ouvrez le fichier que vous avez précédemment téléchargé. Extrayez tous les fichiers qu'il contient vers la racine d'installation du SDK, c'est-à-dire par défaut sous : `C:\Program Files\Java\jdk1.8.0`

Il faut prévoir 270 Mo d'espace disque disponible pour installer la documentation.

Une fois tous les fichiers extraits, fermez l'utilitaire. Sous l'explorateur Windows, dans le répertoire `C:\Program Files\Java\jdk1.8.0`, vous devez avoir un nouveau répertoire `docs`. C'est le répertoire qui contient l'ensemble de la documentation du SDK au format HTML.

Dans ce répertoire `docs`, double cliquez sur le fichier `index.html`. Ce fichier contient des liens hypertextes vers l'ensemble de la documentation Java, qui est soit installée sur votre ordinateur, soit accessible sur un site Web.

Le plus important de la documentation se trouve dans le sous-répertoire `api`, en double cliquant sur le fichier `index.html`. Ce fichier contient les spécifications de l'API Java, ou plus précisément la description de l'ensemble des classes de la librairie Java. Sans cette documentation, vous ne pourrez pas développer efficacement en Java.

Il est recommandé de placer sur votre bureau un raccourci vers ce document.



Plusieurs classes peuvent exister dans un même fichier .java mais une seule peut être déclarée publique, et c'est cette classe qui donne son nom au fichier.

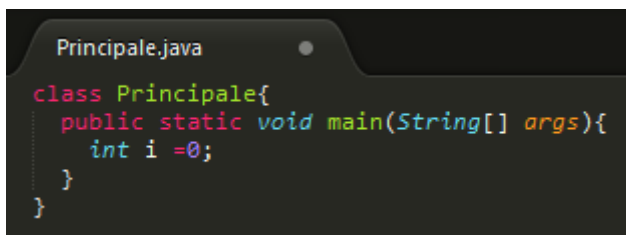
Comme beaucoup d'autres langages de programmation, les fichiers source Java sont des fichiers de texte sans mise en forme.

Un simple éditeur de texte capable d'enregistrer au format texte ASCII, tel que le Bloc-notes de Windows ou VI sous Unix, est suffisant pour écrire des sources Java.

Une fois le code de votre fichier source écrit, il faut l'enregistrer avec l'extension `java` qui est l'extension des fichiers source.

Si vous utilisez le Bloc-notes de Windows, faites attention lors de l'enregistrement de votre fichier que le Bloc-notes n'ajoute pas une extension `.txt` au nom de votre fichier. Pour éviter ce problème, nommez votre fichier avec l'extension `java`, le tout placé entre guillemets.

Voici un exemple de code java :



```
Principale.java
class Principale{
    public static void main(String[] args){
        int i =0;
    }
}
```

Il y a toutefois mieux qu'un simple éditeur de texte pour développer. Vous pouvez, moyennant le coût d'une licence, utiliser des outils commerciaux ou encore mieux utiliser des produits open source comme l'excellent **Eclipse**. Il s'agit au départ d'un projet IBM mais de nombreuses sociétés se sont jointes à ce projet (Borland, Oracle, Merant...). C'est un outil de développement Java excellent et gratuit auquel on peut lier d'autres applications tierces via un système de plug-in. Oracle propose également **NetBeans** un outil très efficace et simple d'utilisation.

## 5.2. Compiler un fichier source

Une fois votre fichier source réalisé et enregistré avec l'extension .java, il faut compiler votre fichier.

Pour compiler un fichier source Java, il faut utiliser l'outil en ligne de commande `javac` fourni avec le SDK.

Ouvrez une fenêtre **Invite de commandes**.

À l'invite de commandes, placez-vous dans le dossier contenant votre fichier source (.java), à l'aide de la commande `cd` suivie d'un espace puis du nom du dossier qui contient votre fichier source.

Une fois que vous êtes dans le bon dossier, vous pouvez lancer la compilation de votre fichier à l'aide de la commande suivante à l'invite de commandes : `javac <nomFichier>.java`.

`javac` : compilateur Java en ligne de commande, fourni avec le JDK.

`<nomFichier>` : nom du fichier source Java.

`.java` : extension qui indique que le fichier est une source Java.

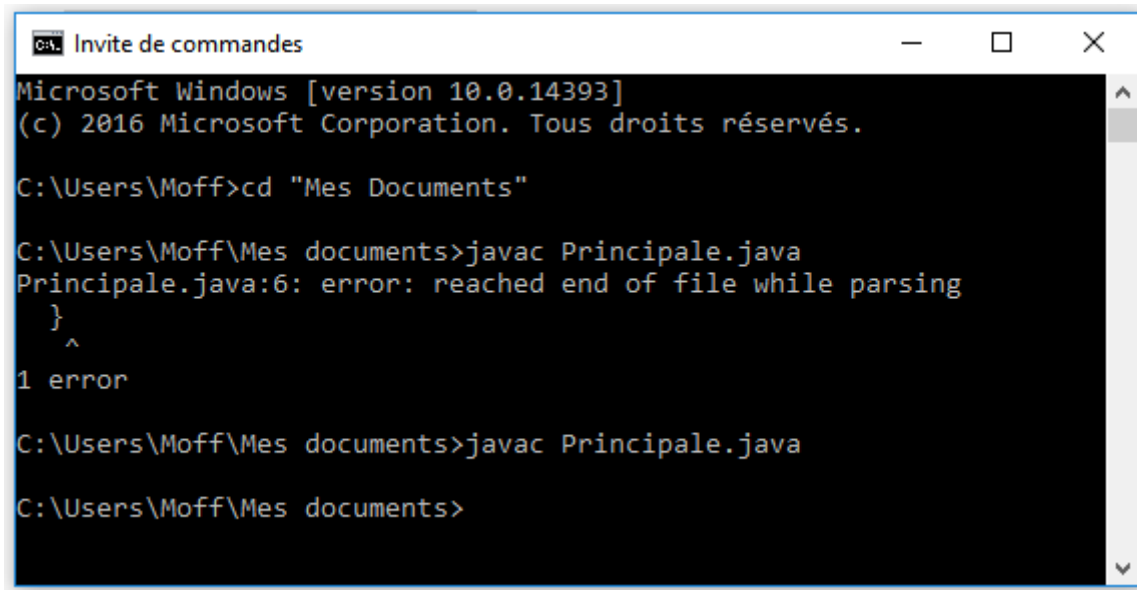
Si vous voulez compiler plusieurs fichiers source en même temps, il suffit de saisir la commande précédente, puis d'ajouter les autres fichiers à compiler en les séparant par un espace.

`javac <nomFichier1>.java <nomFichier2>.java`

Si au bout de quelques secondes vous voyez apparaître de nouveau l'invite de commandes, c'est que votre fichier ne contient pas d'erreur et qu'il a été compilé. En effet le compilateur n'affiche pas de message quand la compilation se déroule correctement.

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. Si par contre, vous voyez apparaître une suite de messages dont le dernier vous indique un nombre d'erreurs, c'est que votre fichier source contient des erreurs et que `javac` n'a donc pas réussi à le compiler.



```
Microsoft Windows [version 10.0.14393]
(c) 2016 Microsoft Corporation. Tous droits réservés.

C:\Users\Moff>cd "Mes Documents"

C:\Users\Moff\Mes documents>javac Principale.java
Principale.java:6: error: reached end of file while parsing
    }
    ^
1 error

C:\Users\Moff\Mes documents>javac Principale.java

C:\Users\Moff\Mes documents>
```

Dans ce cas, il vous faut corriger votre fichier source.

Pour vous aider à trouver les erreurs dans votre ou vos fichiers source, `javac` vous fournit des informations très utiles :

`<nomFichier.java> : <numLigne> : <message> <ligne de code>`

`<nomFichier>`

Nom du fichier source Java qui contient une erreur.

`<numLigne>`

Numéro de la ligne de votre fichier source où `javac` a décelé une erreur.

`<message>`

Message indiquant le type de l'erreur.

`<ligne>`

Ligne de code contenant une erreur, javac indique par une flèche où est située l'erreur dans la ligne.

Après avoir corrigé votre code, recompilez votre fichier. Si javac vous indique toujours des erreurs, renouvelez l'opération de correction puis de recompilation du fichier jusqu'à obtenir la création du fichier binaire .class.

Par défaut les fichiers compilés sont créés dans le même répertoire que vos fichiers source. Vous pouvez indiquer à l'outil javac de les créer dans un autre répertoire à l'aide de l'option -d "directory".

### 5.3. Exécuter une application

Une application Java est un programme autonome, semblable aux programmes que vous connaissez, mais qui, pour être exécuté, nécessite l'emploi 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 en ligne de commande `java` fourni avec le JDK.

Ouvrez une fenêtre **Invite de commandes**. Placez-vous dans le répertoire qui contient le ou les fichiers binaires (.class) de votre application, puis saisissez la commande avec la syntaxe suivante :

```
java <fichierMain> <argumentN> <argumentN+1>
```

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

`<fichierMain>` : est obligatoirement le nom du fichier binaire (.class) qui contient le point d'entrée de l'application, la méthode `main()`. Important : ne mettez pas l'extension .class après le nom du fichier car ceci est fait implicitement par la machine virtuelle Java.

`<argumentN> <argumentN+1>` : éventuels arguments de ligne de commande à passer à l'application à l'exécution de celle-ci.

Si vous avez lancé l'exécution correctement (syntaxe correcte, avec le fichier contenant la méthode `main()`), vous devez voir apparaître les éventuels messages que vous avez insérés dans votre code. Si par contre vous voyez apparaître un message d'erreur semblable à `Exception in thread "main" java.lang.NoClassDefFoundError : ...` c'est que votre programme ne peut pas être exécuté.

Plusieurs raisons peuvent en être la cause :

- Le nom du fichier à exécuter ne porte pas le même nom que la classe (différence entre majuscules et minuscules).
- Vous avez saisi l'extension .class après le nom du fichier à exécuter sur la ligne de commande.
- Le fichier que vous exécutez ne contient pas de méthode `main()`.
- Vous essayez d'exécuter un fichier binaire (.class) qui est situé dans un autre répertoire que celui d'où vous lancez l'exécution.

## 6. Votre première application Java

### 6.1. Squelette d'une application

Une application Java est un programme autonome qui peut être exécuté sur n'importe quelle plateforme disposant d'une machine virtuelle Java.

Tout type d'application peut être développé en Java : interface graphique, accès aux bases de données, applications client/serveur, multithreading...

Une application est composée au minimum d'un fichier `.class` qui doit lui-même contenir au minimum le point d'entrée de l'application, la méthode `main()`.

#### Exemple de la structure minimum d'une application

```
public class MonApplication {  
    public static void main(String arguments[]) {  
        /* corps de la méthode principale */  
    }  
}
```

Si l'application est importante, il est possible de créer autant de classes que nécessaire. Les classes qui ne contiennent pas la méthode `main()` sont nommées classes auxiliaires.

La méthode `main()` est le premier élément appelé par la machine virtuelle Java au lancement de l'application.

Le corps de cette méthode doit contenir les instructions nécessaires pour le lancement de l'application, c'est-à-dire la création d'instances de classe, l'initialisation de variables et l'appel de méthodes.

Idéalement, la méthode `main()` peut ne contenir qu'une seule instruction.

La déclaration de la méthode `main()` se fait toujours suivant la syntaxe suivante :

```
public static void main(String <identificateur>[ ] ) {...}
```

`public`

Modificateur d'accès utilisé pour rendre la méthode accessible à l'ensemble des autres classes et objets de l'application, et également pour que l'interpréteur Java puisse y accéder de l'extérieur au lancement de l'application.

`static`

Modificateur d'accès utilisé pour définir la méthode `main()` comme étant une méthode de classe. La machine virtuelle Java peut donc appeler cette méthode sans avoir à créer une instance de la classe dans laquelle elle est définie.

`void`

Mot clé utilisé pour indiquer que la méthode est une procédure qui ne retourne pas de valeur.

`main`

Identificateur de la méthode.

`String <identificateur>[ ] :`

Paramètre de la méthode, c'est un tableau de chaînes de caractères. Ce paramètre est utilisé pour passer des arguments en ligne de commande au lancement de l'application. Dans la plupart des programmes, le nom utilisé pour <identificateur> est `argument` ou `args`, pour indiquer que la variable contient des arguments pour l'application.

## 6.2. Arguments en ligne de commande

### Principes et utilisation

Une application Java étant un programme autonome, il peut être intéressant de lui fournir des paramètres ou des options qui vont déterminer le comportement ou la configuration du programme au lancement de celui-ci.

Les arguments en ligne de commande sont stockés dans un tableau de chaînes de caractères. Si vous voulez utiliser ces arguments sous un autre format, vous devez faire une conversion de type, du type `String` vers le type désiré lors de l'utilisation de l'argument.

Dans quels cas faut-il utiliser les arguments en ligne de commande ?

Les arguments en ligne de commande sont à utiliser au lancement d'une application dès qu'une ou des données utilisées à l'initialisation de votre programme peuvent prendre des valeurs variables en fonction de l'environnement. Par exemple :

- Nom du port de communication utilisé dans le cas d'une communication avec un périphérique matériel.
- Adresse IP d'une machine sur le réseau dans le cas d'une application client/serveur.
- Nom d'utilisateur et mot de passe dans le cas d'une connexion à une base de données avec gestion des permissions d'accès.

Par exemple, dans le cas d'une application qui accède à une base de données, il faut en général fournir un nom d'utilisateur et un mot de passe pour ouvrir une session d'accès à la base. Des utilisateurs différents peuvent accéder à la base de données, mais avec des permissions différentes. Il peut donc exister plusieurs sessions différentes. Il n'est pas envisageable de créer une version de l'application pour chaque utilisateur.

De plus, ces informations sont susceptibles d'être modifiées. Il n'est donc pas judicieux de les intégrer dans votre code, car tout changement vous obligerait à modifier votre code source et à le recompiler et à détenir une version pour chaque utilisateur.

La solution à ce problème réside dans les arguments en ligne de commande.

Il suffit dans votre code d'utiliser le tableau d'arguments de la méthode `main` qui contient les variables (nom et mot de passe) de votre application.

Ensuite, selon l'utilisateur du programme, il faut au lancement de l'application par l'instruction `java`, faire suivre le nom de la classe principale par la valeur des arguments de ligne de commande de l'application.

### Passage d'arguments à une application Java au moment de l'exécution

Le passage d'arguments à une application Java se fait au lancement de l'application par l'intermédiaire de la ligne de commande. L'exemple de programme suivant montre comment utiliser le passage d'arguments en ligne de commande dans une application Java.

```

/* Déclaration de la classe principale de l'application */
public class MaClasse
{
/* Déclaration de la méthode point d'entrée de l'application*/
public static void main(String args[])
{

/* Affichage des arguments de la ligne de commande */

for (int i = 0 ; i < args.length; i++)

    System.out.println("Argument " +i + " = " + args[i]);

}

/* Conversion de deux arguments de la ligne de commande de
String vers int, puis addition des valeurs entières, et
affichage du résultat */

int somme;
somme=(Integer.parseInt(args[3]))+(Integer.parseInt(args[4]));
System.out.println("Argument 3 + Argument 4 = " + somme);
}
}
|

```

Après compilation, le programme s'exécute avec la ligne de commande suivante :

```
java MaClasse cours JSE "cours JAVA" 2 5
```

L'exécution du programme affiche les informations suivantes :



Argument	0	=	cours
Argument	1	=	JSE
Argument	2	=	cours JAVA
Argument	3	=	2
Argument	4	=	5
Argument	3	+	Argument 4 = 7