Présentation de Java

Table des matières

1 Historique

1.1 Pourquoi Java ?

Bill Joy, ingénieur chez SUN MICROSYSTEM, 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.

1.2 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.

1.3 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 HOT JAVA. 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.
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).

2 Caractéristiques 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.

2.1 Le langage de programmation Java

SUN/Oracle caractérise Java par le fait qu'il est simple, orienté objet, distribué, interprété, robuste, sécurisé, indépendant des architectures, portable, performant, multithread et dynamique.
Nous allons détailler chacune de ces caractéristiques :

2.1.1 Simple (C/C++ versus Java)

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 pas 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 (référencées 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++.

2.1.2 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...).

2.1.3 Distribué

Java implémente les protocoles réseau standards, 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 APIs 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.

2.1.4 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.

2.1.5 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.

2.1.6 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 byte code (ou langage binaire intermédiaire) contenu dans les classes Java. Le byte code 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.

2.1.7 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.

2.1.8 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 standards de Java facilitent l'écriture du code qui peut ensuite être déployé sur différentes plates-formes sans adaptation.

2.1.9 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 Hot Spot , 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++.

2.1.10 Multitâche

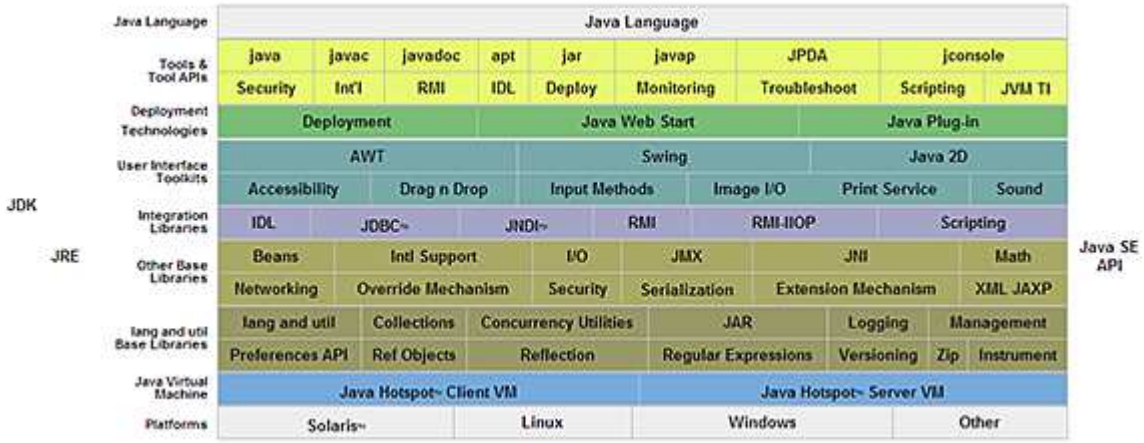
Java permet de développer des applications mettant en œuvre l'exécution simultanée de plusieurs Threads (ou processus légers). Ceci permet d'effectuer plusieurs traitements simultanément, afin d'accroître la rapidité des applications, soit en partageant le temps CPU, soit en partageant les traitements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

2.1.11 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.

2.2 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 de SUN/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 (APIs de bases, APIs d'accès aux données et d'intégration avec l'existant, APIs 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.

2.2.1 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 APIs 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 ou 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 SUN 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). La dernière version, la version 6, a encore optimisé le Java HotSpot.

2.2.2 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 6 est organisée en plus de 200 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 3800 classes, énumération et interfaces.
La plate-forme Java fournit des APIs de base. De nombreuses extensions peuvent être ajoutées et sont disponibles sur le site Java de SUN/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 :

2.2.2.1 Les APIs de base

Les APIs 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 standards tels que FTP, HTTP, UDP, TCP/IP plus les URLs 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 Autorization Service*),
    - d'échanger des messages en toute sécurité entre des applications communiquant via un service comme Kerberos (GSS-API - *Generic Security Service - Application Program 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 APIs DOM (*Document Object Model*) et SAX (*Simple API for XML*). Les APIs de base permettent aussi d'appliquer des transformations XSLT (*extensible Style Sheet Transformation*) à partir de feuilles de styles 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 applicative 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.

2.2.2.2 Les APIs d'accès aux données et d'intégration avec l'existant

Les APIs 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 2000, via l'API JNDI (*Java Naming and Directory Interface*).

2.2.2.3 Les APIs de gestion de l'interface des applications avec l'utilisateur

Les APIs 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.

2.2.3 Les outils de déploiement des applications

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

2.2.3.1 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.

2.2.3.2 Java Plug-in

Destiné à permettre le fonctionnement des Applets Java avec la machine virtuelle 6. 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 6 (le JRE étant composé d'une JVM et de l'ensemble des APIs) et à faire en sorte que les navigateurs Web utilisent cette JRE et non la leur.

2.2.4 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 JAVA SE.
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 APIs standards 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 déboguage 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".

2.3 Cycle de conception d'un programme Java

Pour développer une application Java, il faut d'abord se procurer la plate-forme JAVA SE de développement (JDK ou SDK - *Software Development Kit*) dédiée à sa machine et à son système d'exploitation, dont vous trouverez la liste sur le site Java de SUN/Oracle .
Par exemple : http://www.oracle.com/javase/downloads/index.jsp
Ensuite, vous pouvez utiliser les APIs standards 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 (HotJava, Netscape Navigator, Internet Explorer...). 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*).