« The alternative to modeling the machine is to model the problem »
Bruce Eckel « Thinking in Java ».

1.) Le langage Java.

1.1.) Origines du langage.

Contrairement à la rumeur, JAVA ne signifie pas « *Just Another Vague Acronym* ». Le nom de « JAVA » a simplement été choisi « parce qu'il sonnait bien ». Les origines de Java remontent en 1990 alors que le World Wide Web n'existait pas encore et que l'Internet ne bénéficiait pas encore de la publicité et de la couverture actuelle. Le PC (Personal Computer) était en pleine phase ascendante. Chez Sun, beaucoup pensaient avoir raté un tournant important dans l'informatique des années 90. Les stations Sun, malgré leur succès, étaient réputées compliquées, laides et trop spécialisées pour le grand public. Sun avait donc besoin d'un nouveau fer de lance pour relancer son activité.

1.1.1.) PRINTEMPS 90

Naughton, Gosling et Sheridan définissent quelques principes de base pour un nouveau projet : le consommateur est le centre du projet, il faut construire un environnement de petite taille avec une petite équipe et intégrer cet environnement dans une nouvelle génération de machine, des ordinateurs simples pour les " gens normaux ".

1.1.2.) PRINTEMPS 91

On trouve des microprocesseurs dans l'électronique grand public. L'équipe, baptisé " green team " décide de construire le prototype d'une machine qui contrôlerait l'électroménager, la téléphonie, …

1.1.3.) AOUT 91

La machine a maintenant pour ambition d'interfacer l'homme au " cyberespace ". Gosling, déçu par le C++ crée un langage robuste, industriel, orienté objet d'abord appelé « *C++ minus minus*[1] » puis « OAK[2] ». Naughton développe une interface graphique et des animations.

Figure 1 : James Gosling

[1] C++ sans ses inconvénients
[2] « Chêne » en anglais.

1.1.4.) AOUT 92

Une démonstration est fait à Scott Mc Nealy. On y voit un personnage nommé DUKE guider l'utilisateur à travers une maison cartoonesque pour mettre en marche une cafetière ou enregistrer un film au magnétoscope.

1.1.5.) MARS 1993

L'équipe devenu société " FirstPerson " réponds à des appels d'offres officiels, s 'essaie à ses premières alliances stratégiques. En Juin 1993, le logiciel Mosaic est lancé par NSCA, le Web décolle.

1.1.6.) ETE 1993

FirstPerson n'est pas économiquement viable. Le programmeur légendaire Bill Joy sauve le projet et Eric Schmidt le directeur technique de Sun leur demande d'adapter Oak à l'Internet naissant. Gosling travaillera sur le code interne alors que Naughton s'attachera à développer une application stratégique.

1.1.7.) JANVIER 95

Oak est un nom déposé, le projet s'appellera maintenant Java[3]. L'application de Naughton, un interpreteur pour un navigateur Web se nomme HotJava.

1.1.8.) AOUT 95

La première licence de Java est vendue à Netscape.

Mais cela c'est l'histoire ou la préhistoire de Java. Depuis le 23 Mai 1995, jour où Sun a officiellement présenté Java, le langage a beaucoup fait parler de lui d'abord par sa relation avec le très médiatisé Internet puis par ses qualités intrinsèques qui en font un langage à part entière .

1.2.) Particularités.

Sun s'est engagé à maintenir le langage et les outils java de base dans le domaine public. La société a mis en place une politique de licence / partenariat.

De nombreuses licences ont déjà été accordés pour des outils de développement (Symantec, Microsoft, Borland (Inprise), RogueWare, Macromédia, Ilog, SunSoft, Silicon Graphique Inc., Adobe, Metrowerks), des implémentations de machines virtuelles au niveau du système d'exploitation (Apple, Microsoft, IBM, OSF, Oracle, Mitsubishi) ; de navigateurs Web (Microsoft, IBM, Netscape, Spyglass, Oracle…), serveurs GroupWare, Web ou SGBD (Documentum, Lotus, Netscape, Microsoft, Oracle, Sybase, O2 Technology, Informix…).

[3] Java se traduit exactement par " Kawa " en français soit la dénomination argotique pour « café ». Ce nom atypique donne naissance à de nombreuses déclinaisons pour les produits connexes, tels que « Caféine », « Café », « Kawa » et de nombreux jeux de mots tels " *Wake up and smell the Java* ", ou "*Java, hot technology ?*"

Sun développe également des outils de développement et toute une activité centrée autour d'une nouvelle société JavaSoft.

Java hérite des concepts de nombreux langages plus anciens comme Cedar-Mesa, SmallTalk, Objective-C, C++ et Eiffel, USCD P.

1.2.1.) UNE SYNTAXE SIMPLE ET AGREABLE.

Java est un langage constitué de règles simples. Contrairement au C++, il ne permet pas de surcharge de type ou d'opérateurs. Il ne possède pas de pré-processeur (donc pas de macros, d'expression #define ou de directives de compilation conditionnelles), ni de fichier d'en-tête.

Un seul héritage de classe mais il permet l'héritage multiple pour les interfaces. (Le mécanisme des interfaces a été emprunté à Objective C)

1.2.2.) UN LANGAGE ORIENTE OBJET

Java adopte les quatre principes fondamentaux des langages objets. Java respecte l'abstraction (Tout est objet, on ne peut écrire que des classes. Hors objet, pas de salut !), l'encapsulation, la communication par message et l'héritage.
Java n'utilise pas l'héritage multiple mais offre le mécanisme d'**interfaces**, hérité des **protocoles** de l'Objective-C, qui peut être utilisé pour implémenter de nombreuses fonctionnalités normalement mise en place grâce à l'héritage multiple.

1.2.3.) UN LANGAGE DISTRIBUE

Les appels aux fonctions d'accès réseau (sockets) et les protocoles Internet les plus utilisé tels HTTP, FTP, Telnet sont intégrés dans Java.
On peut écrire très simplement une architecture client-serveur sous Java et utiliser des fichiers sur des systèmes distants comme s'il se trouvaient sur un disque dur local.

1.2.4.) UN LANGAGE **ROBUSTE**

Java insiste beaucoup sur le contrôle de type, tant à la compilation qu'à l'exécution. Chaque structure de données doit être explicitement définie et typée à la différence de JavaScript. Par conséquent il demande une grande rigueur lors de la conception initiale. Java offre un modèle de pointeur, non accessible par le programmeur, totalement différent de celui du C ou C++ qui empêche à une application d'écraser une zone mémoire ou d'altérer des données. Il est impossible par exemple de **caster** un entier arbitraire en pointeur, en référence d'un objet.

1.2.5.) UN LANGAGE **SUR, SECURISE**

Qui dit réseau, implique risque, paranoïa, virus et infiltrations. De part son architecture, il est courant de télécharger un application Java distante pour son exécution locale. Il est donc important pour l'interpréteur de s'assurer qu'une application n'a pas été altérée depuis sa compilation. Un mécanisme de contrôle complet a été incorporé afin de vérifier la légitimité du code.

1.2.6.) UN LANGAGE **INDEPENDANT DU MATERIEL**

L'Internet est multiple. De nombreux systèmes d'exploitation cohabitent, de nombreuses versions de logiciels, d'architecture matérielles. Le maître mot semble être hétérogénéité.
Java se doit d'être indépendant de ces couches qu'il ne maîtrise pas.
C'est pourquoi le compilateur Java ne génère pas des instructions machines spécifiques mais un programme en byte-code, qui peut être décrit comme un langage machine pour un processeur virtuel qui n'a pas d'existence physique[4].
Ce code objet compilé peut alors être exécuté par un interpréteur Java qui n'est ni plus ni moins qu'un émulateur de processeur virtuel : la Machine Virtuelle Java.

Cette idée n'est pas nouvelle, elle a été inventée par les chercheurs de l'USCD (University of California at San Diego) dans les années 70.

1.2.7.) UN LANGAGE **PORTABLE**

Java respecte de nombreux standards tant au niveau réseau, qu'interne ou qu'interface.
En effet dans Java, les types sont identiques quel que soit l'implémentation et le matériel (Alpha 64 bits à 250 MHz ou 68030, 25 MHz 16/32 bits) . Java définit explicitement dans " The Language Spécification " de Sun, qu'un "Int " est toujours un entier de 32 bits en complément à deux signé. Les formats numériques respectent la norme IEEE 754 et les chars la norme Unicode.

1.2.8.) UN LANGAGE **HAUTE PERFORMANCE**

Tout cela est bien beau, mais on nous apprend que les interpréteurs sont lents, lourds et gourmands. L'interpréteur de Java ne respecte pas cette règle puisque plus qu'un interpréteur, il fait tourner un programme généré par un compilateur optimisé sur une machine virtuelle. On n' atteint pas les performances du C ou C++ mais l'écart est de plus en plus réduit avec les nouvelles techniques d'optimisation et les JIT.

1.2.9.) UN LANGAGE **MULTITHREADE**

Dans les programmes multithréadés, plusieurs processus peuvent s'exécuter simultanément. Java inclut le support des thread (ou processus légers) multiples allégeant considérablement l'écritures de programmes qui utilisent ces fonctionnalités. Le langage contient un jeu de primitives de synchronisation basé sur les travaux de Hoare.

1.2.10.) UN LANGAGE **DYNAMIQUE**

Les bibliothèques externes de Java qui offrent ces possibilités ainsi que les bibliothèques écrites par le dévelopeur ne sont pas gérées de la même manière qu'en C++. Il n'est pas nécessaire de modifier le programme si une de ses bibliothèque change, les classes sont

[4] N'existe pas **encore**. Sun a annoncé pour le milieu de l'année 1997 les processeurs Java picoJAva (<$25), microJava ($25-$50) et ultraJava (~$100) de puissances croissantes qui implémentent en dur la machine " virtuelle " Java. Ces processeurs permettrons des ordinateurs dédiés Internet (les fameux network computer de Oracle, Sun et autres) ainsi que des cartes d'accélération Java. D'autres procédés permettent d'accélérer l'exécution de programmes Java comme les JIT (Just In Time compiler) et l'optimisation intelligente mettant en œuvre une bonne connaissance du travail du compilateur *javac.*

chargées dynamiquement. Ainsi Java résout le problème classique de C++ nommé le problème de **Superclasse fragile**.

2.) Java sur le net.

L'une des raisons du succès de Java, est sa capacité d'utilisation sur le net.

2.1.) Sun Microsystems..

Ce site contient les dernières versions publiées du JDK (à télécharger gratuitement) ainsi que d'autres ressources destinées aux développeurs, des communiqués de presse sur les produits relatifs à Java, toute la documentation sur Java et des exemples de programmes tournant sur le Web.

Le site de Sun se trouve à http://java.sun.com.

2.2.) Javaworld.

Il existe un magazine en ligne pour les développeurs Java et Internet. Ce magazine propose des articles pratiques, des actualités sur le développement Java etc. Le site Web JavaWorld se trouve à http://www.javaworld.com.

2.3.) Developer.com

developer.com est le plus grand registre de programmes Java, de ressources de développement et autres informations concernant le langage. Il se trouve à http://www.developer.com/.

2.4.) Cours disponibles sur internet.

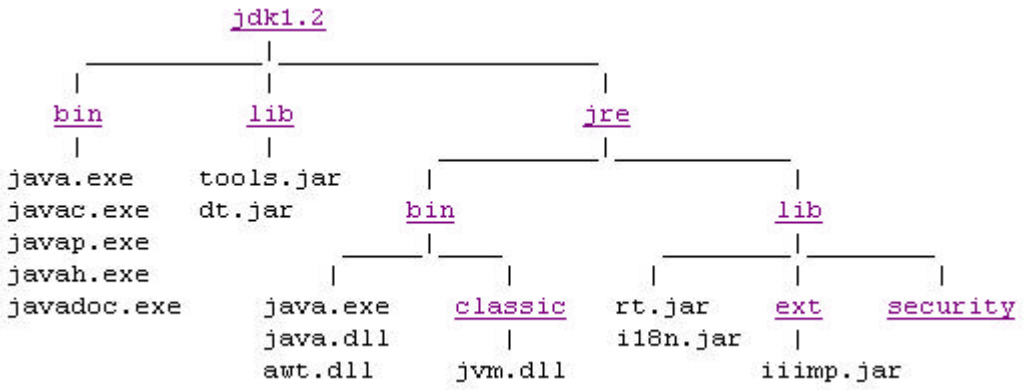
`http://www.imaginet.fr/ime/java.htm`	Gille Maire : "Un nouveau Guide Internet"
`http://java.sun.com/docs/books/jls/index.html`	James Gosling, Bill Joy and Guy Steele : "The Java Language Specification"
`http://cuiwww.unige.ch/java/cours/notes/`	Michel Bonjour, Gilles Falquet, Jacques Guyot et André Le Grand "Notes de cours JAVA"
`http://java.sun.com/docs/books/tutorial/index.html`	Elliotte Rusty Harold : "Brewing Java : A tutorial"
`http://gbm.esil.univ-mrs.fr/~tourai/Java/`	Touraïvane. "Le langage Java"
`http://www.mindview.net/TIJ2/index.html`	Bruce Eckel "Thinking in Java"[5]

[5] Le meilleur cours sur Java avec un rapport qualité prix incomparable !
« I really appreciate your work and your book is good. I recommend it here to our users and Ph.D. students. » Hugues Leroy // Irisa-Inria Rennes France, Head of Scientific Computing and Industrial Tranfert
« It's the best Java book I have ever read - and I read some. » Jean-Yves MENGANT, Chief Software Architect NAT-SYSTEM, Paris, France

3.) L'environnement de développement.

3.1.) Le JDK

Depuis l'été 1995, Sun distribue le JDK (Java Development Kit).



Répertoire	Contenu
`\jdk1.2\bin`	Programmes qui constituent le kit de développement Java, le compilateur, l'interpréteur, etc...
`\jdk1.2\lib`	Classes java standard; fournies avec le JDK
`\jdk1.2\jre`	Environnement d'exécution des programmes Java. (Java Runtime Environment).
`\jdk1.2\jre\bin`	Fichiers exécutables et DLLs pour les outils et les librairies utilisés par la plate-forme java.
`\jdk1.2\jre\bin\classics`	Contient des DLLs utilisés par la machine virtuelle *classique.* Quand une nouvelle version de la machine virtuelle apparaît, celle-ci est stockée dans un nouveau sous-répertoire de `jre\bin`.
`\jdk1.2\jre\lib`	Librairies et ressources utilisés par l'environnent d'éxécution.
`\jdk1.2\jre\lib\ext`	Répertoire par défaut pour les extensions de Java.
`\jdk1.2\jre\lib\security`	Contient des fichiers utilisés pour la gestion de la sécurité.

3.2.) Les outils de l'environnement Java.

Le JDK contient une collection d'outils qui peuvent être utilisés avec les programmes Java pour réaliser un certain nombre de fonctions.

3.2.1.) LE COMPILATEUR JAVA.

Le compilateur javac est un programme qui qui transforme un fichier codant une classe (ou une ensemble de classes) nom_de_fichier.java en un fichier bytecodé (compilé) nom_de_fichier.class.

La syntaxe du compilateur est la suivante :

```
javac [options] nom_de_fichier.java
```

Le compilateur Java se trouve dans le répertoire \Java\bin.

Option	Description
-nowrite	Indique au compilateur de compiler le code sans créer de fichier .class. Cela permet de tester la syntaxe sans créer de fichier.
-nowarn	Désactive les messages d'avertissement.
-verbose	Demande au compilateur d'afficher des informations sur ce qu'il fait pendant la compilation.
-d *répertoire*	Indique au compilateur qu'il doit utiliser le répertoire répertoire comme premier répertoire d'un package.
-classpath *dirs*	Indique au compilateur qu'il doit regarder dans les répertoires listés dans dirs pour les classes incluses dans le fichier source.
-sourcepath *dirs*	Indique au compilateur où se trouvent les fichiers source.
-debug	Exécute le compilateur en mode débogage. Ajoute des informations nécessaires au débogage.
-O	Demande au compilateur d'optimiser la vitesse d'éxécution. En contrepartie, le programme résultat sera plus gros.
-g	Utilisée pour préparer le pseudo-code au débogage.

Pour compiler plusieurs classes en même temps, il est possible de lister dans un fichier toutes les classes à compiler et d'appeler javac avec @nom_de_fichier en paramètre.

Par exemple, si on veut compiler les classes c1,c2 et c3 en même temps, on peut créer un fichier mes_classes qui ressemblera à :

c1.class
c2.class
c3.class

Et on lancera javac de la manière suivante :

```
javac @mes_classes
```

3.2.2.) LA MACHINE VIRTUELLE JAVA.

La machine virtuelle Java est un programme qu'on utilise pour tester une application déjà compilée. La syntaxe pour la lancer est la suivante :

```
java [options] classname
```

Option	Description
-cp *repertoires*	Chemin de recherche
-D	
-verbose	
-version	Affiche la version de la machine virtuelle.
-? ou -Help	Affiche une aide sur les options possibles
-X	Affiche une aide sur les options non standard

Le jeu d'instruction de la machine virtuelle Java est optimisé être petit et compact. Destiné à être véhiculé à travers le réseau Internet, il a du sacrifier rapidité d'interprétation pour être de petite taille.

Le code source Java est compilé en bytecode et stocké dans un fichier class. Une instruction bytecode est constituée d'un code opérateur (opcode) d'un octet qui identifie l'instruction, puis de zéro ou plus opérandes, chacune pouvant peser plus d'un octet et qui constituent les paramètres nécessaires à l'instruction.

Les bytecodes interprètent les données dans la mémoire d'exécution comme faisant parti d'un ensemble limité de type de données. Ces types primitifs sont ceux que manipulent Java :

- 4 types entiers (byte de 8 bits, short de 16 bits, int de 32 bits, long de 64 bits) ;
- 1 type non signé (char de 16 bits) ;
- 2 types flottants (un float de 32 bits, un double de 64 bits) ;
- 1 type référence à un objet (un type de pointeur sur 32 bits).

Certaines instructions comme dup traite la mémoire comme des données brutes sans se préoccuper du type.

Comme dans tous les assembleurs ont retrouve les concepts classiques de jeu d'instructions, ensemble de registres, pile, heap avec ramasse-miettes ainsi qu'un espace de stockage pour les méthodes et un pool de constantes.

Les quatre registres (de 32 bits) sont pc (program counter classique), optop (un pointeur vers le haut de la pile des opérandes), frame (un pointeur vers l'environnement d'exécution de la méthode courante), vars (pointeur vers la première variable locale de la méthode courante).

La machine virtuelle reposant principalement sur la pile, elle n'utilise pas de registres pour passer et recevoir des arguments. La pile contient l'état d'une seule méthode, et est utilisée pour passer des arguments aux bytecodes et méthodes et pour recevoir leurs résultats.
Le heap est cette partie de la mémoire qui contient les nouvelles instances qui sont allouées. Les objets Java sont référence indirectement dans l'environnement runtime via des handles qui sont des sortes de pointeurs vers la heap.

3.2.3.) LE DESASSEMBLEUR JAVA

Le désassembleur Java est utilisé pour désassembler des pseudo-codes Java déjà compilés. Une fois le code désassemblé, les informations sur les variables et sur les méthodes sont affichées. La syntaxe pour le désassembleur est :

```
javap [options] classname.
```

Option	Description
-b	Compatibilité avec java 1.1
-c	Désassemble le code
-classpath *repertoires*	chemin de recherche
-extdirs *dirs*	
-help	Affiche un écran d'aide
-l	Affiche les numéros de lignes et les variables locales
-public	N'affiche que les classes publiques
-protected	Affiche les classes « protected »
-package	Affiche les classes « package/protected/public »
-private	Affiche toutes les classes
-verbose	Affiche la taille de la pile, les arguments des méthodes

3.2.4.) LE GENERATEUR DE DOCUMENTATION.

Cet outil créé un fichier HTML décrivant les classes et méthodes utilisés.

3.2.5.) LE DEBOGUEUR JAVA.

Le débogueur peut être utilisé pour déboguer les programmes Java.

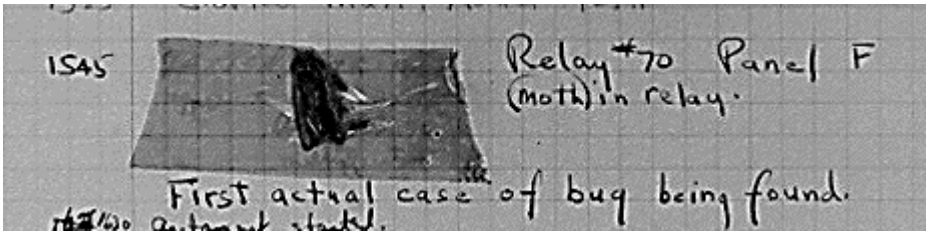


Figure 2 : Le premier bug de l'histoire de l'informatique.

3.2.6.) LES JIT COMPILER

Il y a une dizaine d'années L. Peter Deutsch inventa un procédé appelé " traduction dynamique " alors qu'il cherchait à accélérer l'interprétation SmallTalk.Ce truc a été repris depuis par plusieurs développeurs (Borland, Symantec, SunSoft, …) pour créer des " Just In Time Compilers " Java. Car à l'instar de SmallTalk il souffre de lenteur d'exécution qui nuisent à son succès.

L'astuce consiste à remarquer qu'un interpréteur rapide classique - écrit en C par exemple - possède déjà une séquence de code binaire correspondant à un bytecode Machine Virtuelle Java : le code machine que l'interpréteur lui même exécute. Parce que l'interpréteur a déjà été compilé du C en langage machine, pour chaque bytecode, il passe à travers une séquence de code natif au CPU sur lequel il s'exécute. Si l'on arrive à sauvegarder une copie de chaque instruction binaire qu'il traverse, l'interpréteur pourrait journaliser son cheminement du code binaire qu'il a exécuté afin d'interpréter le bytecode étudié. De la même façon il peut conserver un journal entier mémorisant les procédures assembleur qu'il a exécuté afin d'interpréter une méthode entière. En prenant ce journal et en optimisant par à-coups (de la même façon qu'un compilateur), on pourra éliminer les redondances et les instructions inutiles. Et c'est la que le " truc " opère : en effet, lors de la prochaine exécution de la méthode (de la même façon), l'interpréteur pourra exécuter le code binaire natif stocké dans le journal tel quel. Ce faisant, il évite la surcharge que constitue l'interprétation

que chacun des bytecode de la méthode et permet ainsi une accélération d'un facteur allant expérimentalement jusqu'à dix (10).
Pour certaines sections du code, on obtiendra des performances égalant un code C compilé. La technique est nouvelle et de nombreuses améliorations peuvent encore êtres ajoutées.
Afin de qualifier ces améliorations, un test à été effectué. Il est limité à quelques procédures benchmark classiques sous les environnements Windows 95 et Windows NT 4.0b . Le benchmark suivant est donc à lire avec prudence :

	Pentium Pro	Cyrix 6x86				i486
	Symantec Cafe JIT Compiler NT 4.0 β	Symantec Cafe JIT Compiler	Netscape 2.01	Symantec Cafe Debugger	Netscape Gold 2.0β	Sun JDK 1.0
Sieve	5762	2116	99	100	27	64
Loop	6526	3706	95	100	28	53
Logic	4707	2128	96	100	32	65
String	522	394	83	100	26	55
Floating-Point	3733	804	93	100	27	59
Method	2076	1652	90	100	26	43
Image	268	149	80	100	13	22
Graphics	484	149	88	100	15	18
Dialog	240	100	92	100	29	12
Allocation And Garbage Collection (Local Only)	3000	1702	50	1117	N/A	N/A
Dynamic Compilation (Local Only)	371	113 (max 180)	89	100	N/A	N/A
CaffeineMark	1993	910	89	100	23	39
		JIT	**Navigateur**	**Normal**		

3.2.7.) LES API.

L'un des avantages principaux du paradigme (j'adore ce mot !) orienté objet est la réutilisabilité obtenu en sous-classant des classes existantes, ce qui permet de ne programmer que les différences entre vos nouvelles classes et celles que vous sous-classez.

Mais cela ne serait qu'un avantage mineur s'il était impossible de travailler avec une librairie de classes externes.

Une partie essentielle du JDK de Sun et de toutes les implémentation de la plate-forme Java est l'API Sun. Cette API se présente sous la forme de quinze packages et sous-packages dont huit principaux. Cette structure de package est propre à Java : les packages sont des rassemblements de classes, regroupées par fonctionnalité. Ces packages contiennent à la foi des classes bien sûr mais aussi des interfaces, des définitions d'exceptions et d 'erreurs.
Ces API contiennent les classes que Sun s'est engagé à porter et a maintenir sur une grande variété de plates-formes.

Les classes Sun Java API (1.0x) comprennent les packages suivant :

Package	Classes présentes
java.applet	Classes de base pour les applets
java.awt	Classes d'interface graphique AWT
java.awt.image	Classes de gestion des images AWT
java.awt.peer	Classes d'interfaçages aux environnements natifs
java.io	Classes d'entrées/sorties (flux, fichiers…)
java.lang	Classes de support du langage
java.net	Classes de support réseau (URL, *sockets*…)
java.util	Classes d'utilitaires (vecteurs, *hashtable* …)

Le package java.lang est le plus intéressant puisqu'il comprend notamment :

- la classe Object (superclasse, ancêtre de toutes les classes) fournit entre autres des méthodes permettant de copier des objets, de tester l'égalité et de convertir les objets en chaînes ;
- les wrappers de types simples. A l'instar du C++ et à l'opposé du SmallTalk, pour des soucis de performances, les types simples existent, en dehors des hiérarchies objets. Ces wrappers servent de version en classes des types fondamentaux. Il trouvent notamment leurs utilité avec les classes de java.util qui prennent comme paramètres des objets ;
- la classe Math qui permet l'accès aux constantes mathématiques (pi, e) et aux méthodes statiques (équivalentes aux fonctions du C) implementants les classiques fonctions mathématiques (telles sqrt, sin…) ;
- la classe String ;
- les classes System et Runtime sont des classes finales (qui sont constituées de méthodes statiques a.k.a. fonctions) et permettent d'accéder aux ressources systèmes (notamment les stdout et stdin) ainsi qu'a l'environnement runtime ;
- les classes et interfaces de gestions de threads, dont la classe Thread et l'interface Runnable ;
- les classes Class et ClassLoader pour travailler avec les classes. C'est la base du dynamisme Java ;
- les classes et interfaces de gestion d'erreurs et d'exceptions ;
- les classes de gestion de processus.

Il faut ajouter que Java fourni un bon nombre de classes intéressantes pour gérer les objets telles : une classe tableau multidimensionnelle, une classe chaîne (String), une classe vecteur (Vector, sorte de tableau dynamique), une classe HashTable (qui associe un objet avec une valeur unique, le hashcode) et bien d'autres.

3.2.8.) EXEMPLE.

Voici un exemple de programme Java élémentaire :

```
/**
 * @(#)Hello.java   1.0 2000/01/28
 * Création d'une classe Hello
 * @see
 * @author     CAMOS-CNAM / LWH
 * @version    1.0
 */

class Hello {

public static void main(String[] argv)
{

  System.out.println("Hello World !");

  }

}
```

Code 1 La classe Hello.java[6]

La compilation d'un programme Java se fait à l'aide du compilateur javac. Le pseudo-code obtenu (bytecode) peut ensuite être exécuté sur la machine virtuelle java.

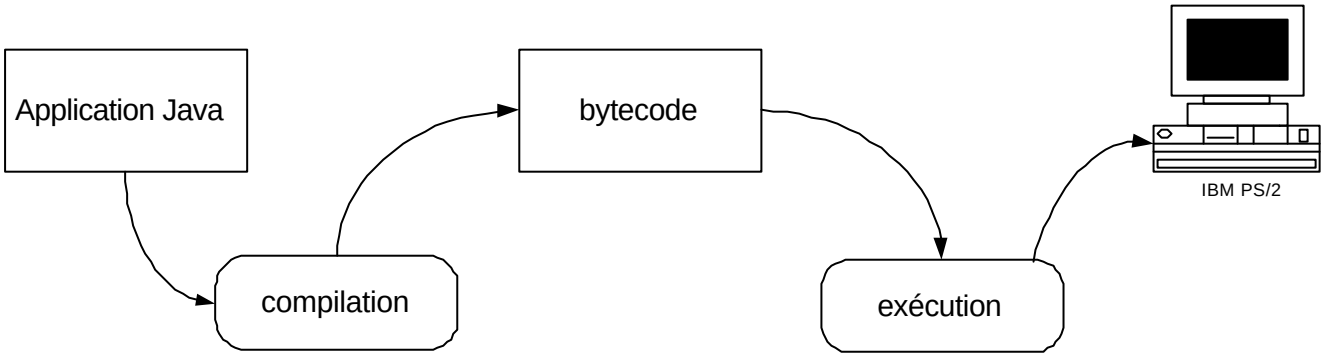


Figure 3 Compilation et exécution d'un programme Java.

Java est donc à la fois un langage compilé et interprété. Le compilateur Java, appelé javac, traduit un code source en un code de niveau intermédiaire que l'on appelle pseudo-code ou bytecode. Les pseudo-code n'est pas directement exécutable sur toutes les plates-formes, mais les codes sont interprétés par l'interpréteur Java qui peut opérer soit de façon isolée soit par le biais d'un browser Web comme Netscape. Le fait que le programme soit à la fois compilé et interprété donne les atouts des deux univers :

- Le programme sera efficace car il est compilé.
- Il est portable sur une multitude de plates-formes car il est interprété.

[6] Fichier "Hello.java".

4.) Concepts de base de la POO.

4.1.) Introduction.

Par définition, un langage de programmation est une abstraction du langage machine. Les langages d'assemblage ne fournissent qu'une faible abstraction de langage machine sous-jacent. Les langages impératifs ou procéduraux comme Pascal ou C sont eux-mêmes des abstractions des langages d'assemblage. Ces langages apportent déjà un réel progrès en terme de qualité de programmation mais ils obligent encore le programmeur à penser en termes de structures de contrôle plutôt qu'à la structure du problème à résoudre. Le programmeur doit en effet établir une correspondance entre le modèle de la machine et sa propre vision du problème. L'effort nécessaire à établir cette correspondance est intrinsèque aux langages de programmation impératifs.

Les concepteurs de langages ont toujours cherché à effacer cette frontière entre le mode de pensée "humain" et la logique des machines. Des langages comme LISP ou APL choisissent des visions différentes. Pour LISP, tous les problèmes peuvent se résumer à des manipulations de listes. Pour APL, tous les problèmes sont algorithmiques. PROLOG raisonne en chaînage et en décisions. Caml se base sur la récursivité et la pleine fonctionnalité. Chacune de ces approches est une bonne solution pour une classe particulière de problème.

L'approche orientée objet propose une nouvelle vision du problème. L'idée de base est qu'un programme peut se réduire à un assemblage d'objets (de briques) et que la résolution d'un problème peut se faire par créations de nouveaux objets et assemblage d'objets existants, de la même façon qu'on construit une maison avec du "sur mesure" et du "standard". Dans un tel programme, chaque objet se comporte comme un mini-ordinateur capable d'effectuer quelques tâches élémentaires. C'est l'assemblage de ces morceaux de programmes, de ces micros compétences qui permet d'effectuer des tâches complexes.

En pratique, cette approche se révèle être la plus flexible et la plus puissante existant actuellement[7].

On peut définir les langages objets par 5 caractéristiques de base :

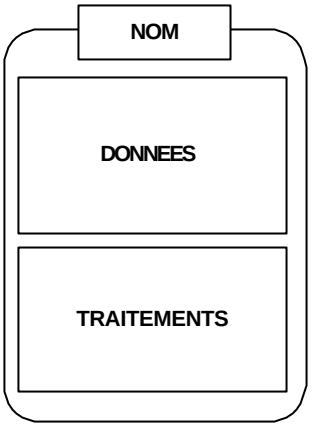
1. **Tout est objet**. On peut voir un objet comme une simple variable. Il stocke des informations mais, en plus, on peut lui demander d'exécuter des actions ou de répondre à des messages. En théorie, on peut modéliser chaque composant du problème à résoudre (compte, balance, composant électronique) comme un objet.
2. **Un programme est un assemblage d'objets qui communiquent entre eux en s'envoyant des messages**. Pour demander à un objet de faire une action, on lui "envoie un message". Concrètement, chaque action sur un objet s'appelle un message.
3. **Chaque objet gère son propre espace mémoire composé d'autres objets**. Pour créer un objet, on procède donc par assemblage d'autres objets.
4. **Chaque objet possède un type**. Dans le vocabulaire de la POO, on parle de "classe".
5. **Tous les objets d'une même type peuvent recevoir et traiter les mêmes messages**.

[7] Bien que les recherches actuelles s'orientent vers les langages "multi-paradigme", à la fois objets, fonctionnels, logiques et procéduraux...

4.2.) Concepts de base.

4.2.1.) OBJET :.

L'unité de base d'un programme écrit dans un langage orienté objet est l'objet. Cette unité de base est identifiée par un nom et contient notamment des données et une série d'actions effectuables sur ces données.

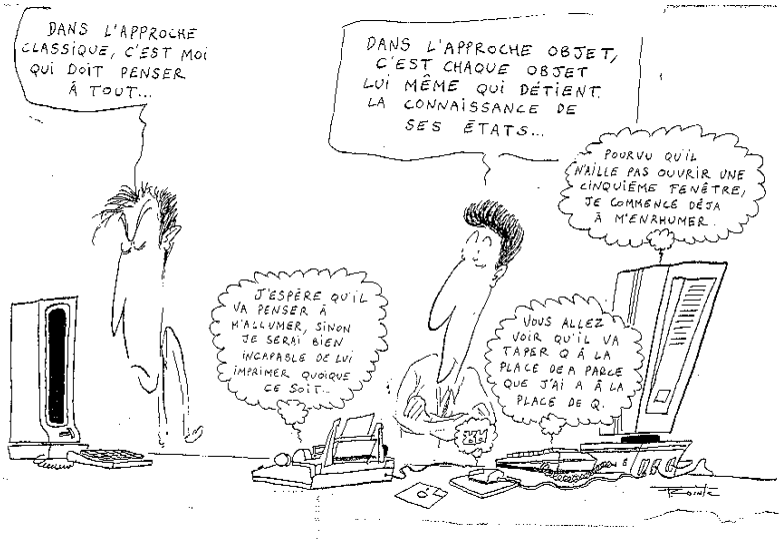


Un objet regroupe donc les deux points de vue complémentaires de la programmation traditionnelle :

- Statique : les données.
- Dynamique : les méthodes.

On peut également voir un objet comme un mini-programme ou comme une machine indépendante.

Il faut bien comprendre que chaque donnée, ce à un certain moment, a une valeur (d'un certain type). L'ensemble des valeurs des attributs de l'objet détermine l'état courant de l'objet. Cet état ne doit changer que si une méthode de l'objet est invoquée. Ce n'est pas une obligation, mais cela est fortement conseillé (nous y reviendrons lorsque nous parlerons de la programmation orientée composant).

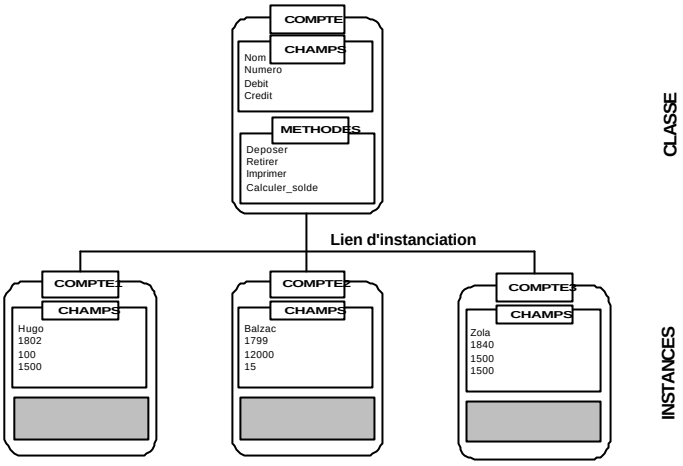


4.2.2.) CLASSE :

En Java (et dans la plupart des langages objets) on ne définit par directement l'objet. On en définit la classe.

La classe est un modèle décrivant les caractéristiques d'une collection d'objets.

Tout objet a un modèle : sa classe.



La classe correspond à la définition d'un type abstrait.

La classe est un moule servant à la création d'objets.

C'est Simula-67, le premier langage orienté objet qui a introduit le mot clé "**class**". Comme son nom l'indique, Simula est un langage qui fut créé pour développer des simulations comme celle du "problème du banquier". Dans cette simulation, on a des clients, des comptes, des transactions, plusieurs monnaies, bref plusieurs objets différents. Les objets qui présentent les mêmes caractéristiques (excepté à un moment précis de l'exécution du programme) sont regroupés en classes d'objets. Les membres d'une même classe partagent les mêmes caractéristiques : chaque compte à un solde, chaque client a un nom, une adresse, chaque monnaie a un taux de change etc...

Le concept de classe se rapproche donc de celui de type de donné abstrait. On peut donc voir un TDA comme la pile ou la liste comme une classe d'objet : Il possède des propriétés et des méthodes.

4.2.3.) INSTANCIATION :

Une instance est un objet particulier :

- Créé à partir d'une classe.
- Ayant la même structure que la classe dont il est issu.
- Les données ont des valeurs propres à l'instance.
- C'est une occurrence d'objet.

L'instanciation est un mécanisme de création d'objets à partir d'une classe qui joue le rôle de modèle.

De façon pratique, dans Java, l'instanciation d'un objet se fera de la façon suivante :

```
String s = new String("L'aurore grelottante en robe rose et verte...")[8]
```

L'identificateur *s* est appelé un *handle* (ou un pointeur) vers un objet de classe *String*.

[8] Baudelaire.

4.2.4.) HANDLE :

Pour pouvoir manipuler un objet, il faut disposer d'une référence ou d'un pointeur vers cet objet. Cela s'appelle un handle.

Il faut voir le handle comme la télécommande d'un téléviseur. Le handle n'est pas l'objet mais seulement un outil pour le manipuler.

4.2.5.) PROPRIETES :

Aussi appelé champs ou attributs, c'est un composant statique d'un objet auquel on associe une valeur.

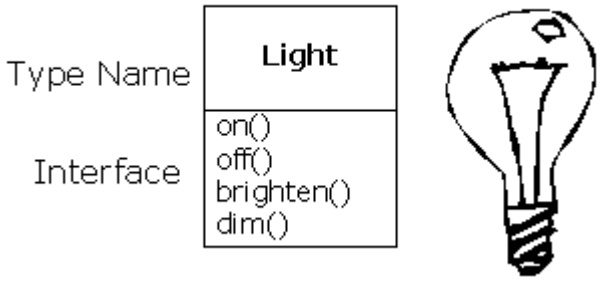
4.2.6.) METHODES :

Une méthode est un programme associé à une classe.
L'exécution d'une méthode s'applique à un objet de la classe.
Les méthodes d'une classe sont communes à tous les objets potentiels de cette classe.
Une méthode accepte des arguments et renvoie des valeurs.

4.2.7.) INTERFACE :

Chaque objet possède une interface. L'interface précise quelles sont les actions que l'on peut effecteur sur un objet.

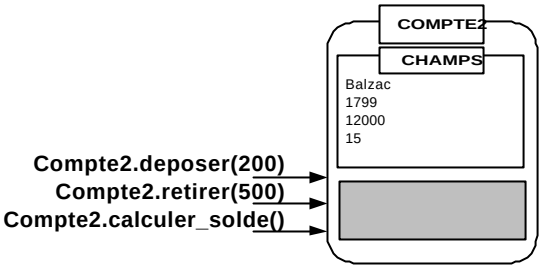
L'objet "Light" pourrait, par exemple se modéliser ainsi :



Ainsi, sur un objet de classe "Light", on peut effectuer les actions suivantes : l'allumer (on), l'éteindre (off), augmenter la puissance de l'éclairage (brighten) ou la réduire (dimmer).

4.2.8.) MESSAGE :

L'objet est une boite noire dont la structure interne n'est connue que de lui-même.
Le message est le moyen de formuler une requête à un objet : lui dire ce qu'il doit faire.
L'objet réagit au message en exécutant la méthode désignée.
Le message doit contenir l'identifiant de l'objet, un sélecteur de méthode et des arguments éventuels.



4.2.9.) COMPOSITION :

Un objet peut être composé de plusieurs objets. Par exemple, un objet « datetime » pourrait être composé d'un objet « date » et d'un objet « time ». Le lien qui unit ces objets est un lien de composition. On dira :

Un objet « datetime » contient un objet « date »
ou
Un objet « voiture » contient un objet « moteur ».

4.2.10.) HERITAGE :

L'héritage est un concept fondamental de l'orientation objet.

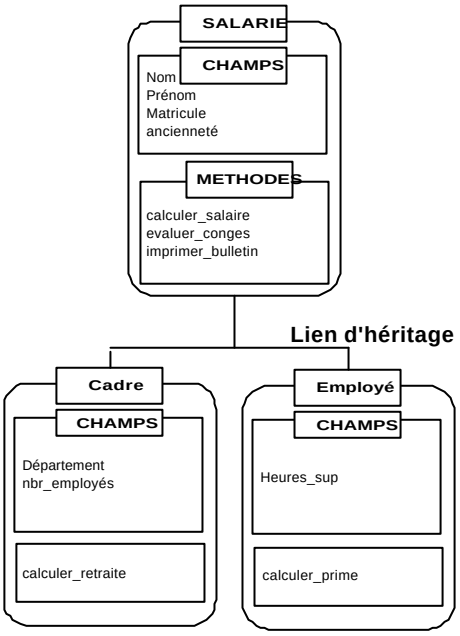
Il permet de définir de nouvelles classes par affinage de classes plus générales.

On ne précise dans les sous-classes que les différences avec la classe de niveau supérieur (super classe).

La sous-classe hérite des méthodes et des propriétés de sa super classe.

Certaines propriétés et méthodes de la classe supérieure peuvent être redéfinies dans les sous-classes. Ce mécanisme s'appelle la surcharge.

On pourra utiliser l'héritage quand on peut dire « l'objet héritant **est une sorte de** objet hérité ». Par exemple, « objet voiture **est une sorte de** objet moyen_de_transport »



4.3.) Concepts avancés.

4.3.1.) CONSTRUCTEURS ET DESTRUCTEURS.

Les constructeurs sont des méthodes spéciales qui servent à initialiser l'objet. Un objet peut avoir plusieurs constructeurs.

Les destructeurs sont des méthodes qui sont appelées juste avant la destruction de l'objet. Un objet ne peut avoir qu'un seul destructeur.

4.3.2.) SIGNATURE D'UNE METHODE.

On appelle *signature* d'une méthode le nom de la méthode et la liste des arguments.

4.3.3.) SURCHARGE.

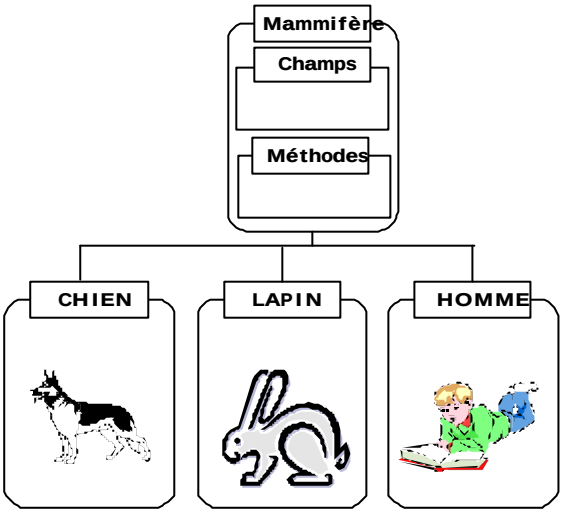
Réécriture d'une méthode ayant le même nom mais une liste d'arguments différents.

4.3.4.) CLASSES ET METHODES ABSTRAITES.

Une Classe abstraite est une classe qui ne peut avoir d'instance.

- Elle a toujours des spécialisations.
- Elle sert à factoriser les propriétés de ses spécialisations.

Dans l'exemple suivant, la classe « Mammifère » est une classe abstraite.



4.3.5.) ATTRIBUTS ET METHODES CONSTANTS..

4.3.6.) L'ENCAPSULATION.

4.3.7.) LE POLYMORPHISME.

Le polymorphisme est un mécanisme qui permet d'adresser le même message à des objets de types différents qui l'interpréteront à leur manière.

Le polymorphisme signifie que l'émetteur d'un stimulus n'a pas besoin de connaître la classe de l'instance réceptrice. Cette dernière peut appartenir à n'importe quelle classe.

4.3.8.) LES METACLASSES.

Une métaclasse permet de créer des classes.

- en SmallTalk : une pollution ≈ 1 par catégorie
- en KOOL : *Class, AbstractClass, RockBottom,* etc.
- la classe *MetaClass* permet d'en créer d'autres
  - en Java : *interface* et *class*,
  - pas de création

4.3.9.) BIBLIOTHEQUES DE CLASSES..

Les bibliothèques de classes permettent de regrouper des classes ayant une même destination. En Java, on utilise le terme de « package ».

5.) Stratégies de développement.

5.1.) L'approche « programmer-corriger »

L'approche « programmer-corriger » est très simple : il suffit de se faire une vague idée du résultat souhaité, puis d'utiliser, au gré de ses intuitions, toutes les combinaisons d'analyse, de programmation, de correction et de procédures de test possibles jusqu'à obtention d'un produit présentable. Ou jusqu'à épuisement des programmeurs.

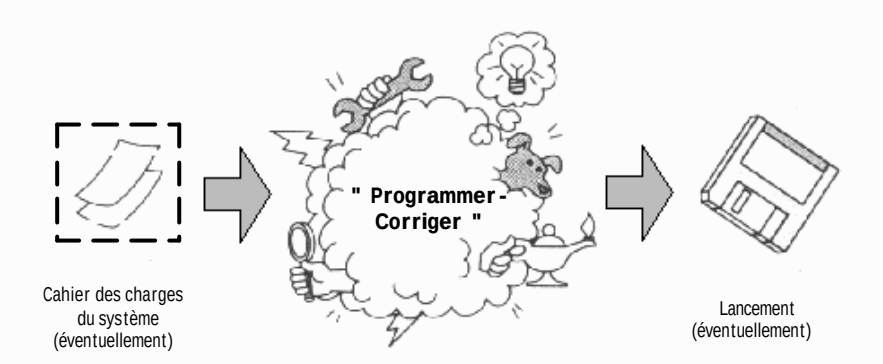


Figure 4 : Approche "Programmer-Corriger".

Même si cette méthode se fonde sur un joyeux désordre et se caractérise par son manque total de rigueur, il ne faut pas s'en haussebecquer bêtement. Si elle conduit souvent au pire, elle est parfois capable du meilleur : des produits comme Visicalc ou Microsoft NT 3.0 ont été développés ainsi..

5.2.) Le modèle en cascade.

Le modèle en cascade est infiniment plus rigoureux. Il consiste à faire progresser un projet étape par étape, de la conception initiale aux tests, chaque phase faisant l'objet d'un examen final avant de passer à la suivante.

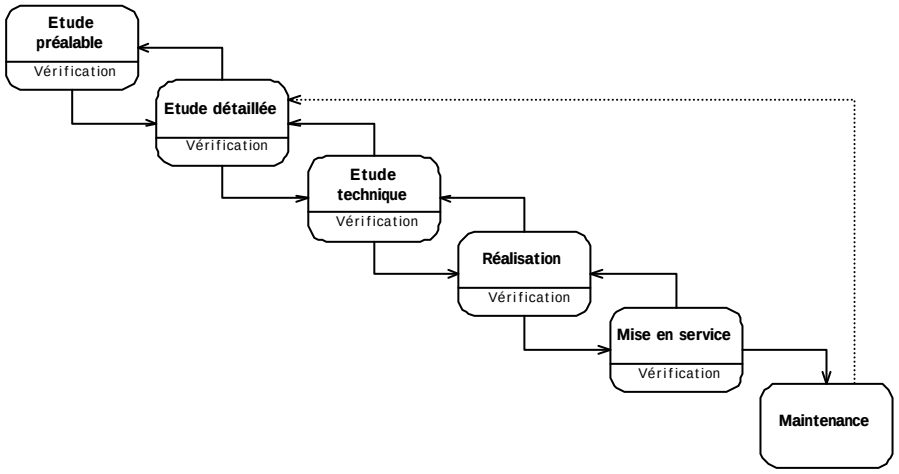


Figure 5 : Le modèle de méthodologie de développement en cascade.

Il existe de nombreuses variantes du modèle en cascade. Le modèle « Sashimi[9] », par exemple, prévoit une superposition des différentes phases, l'étude technique pouvant ainsi commencer avant que l'étude détaillée ne soit entièrement terminée ; le modèle « en cascade avec sous-projets » procède à un découpage du projet en plusieurs sous-projets qui seront traités de manière indépendante ; le modèle « en cascade avec réduction des risques » intègre une spirale au début pour examiner et réduire les risques liés aux spécifications. Le « cycle en V » met en correspondance les phases de conception et de réalisation pour vérifier leur adéquation.

Le principal inconvénient de ces modèles réside dans le fait que les besoins ne sont pas correctement testés tant que le système n'est pas opérationnel. Les erreurs figurant dans le document des spécifications des besoins seront les dernières à être découvertes et leur coût de correction sera donc très élevé. Souvent, pour se protéger, le concepteur s'abrite derrière la procédure contractuelle : l'utilisateur est le maître d'ouvrage, c'est lui qui passe la commande et c'est donc lui qui est responsable du contenu de cette commande.

5.3.) Le modèle en spirale.

L'idée fondamentale de cette méthode est de débuter à petite échelle, d'explorer les risques éventuels, d'élaborer un plan pour les résoudre puis de passer à l'itération suivante, chaque itération permettant d'élargir l'échelle de travail.

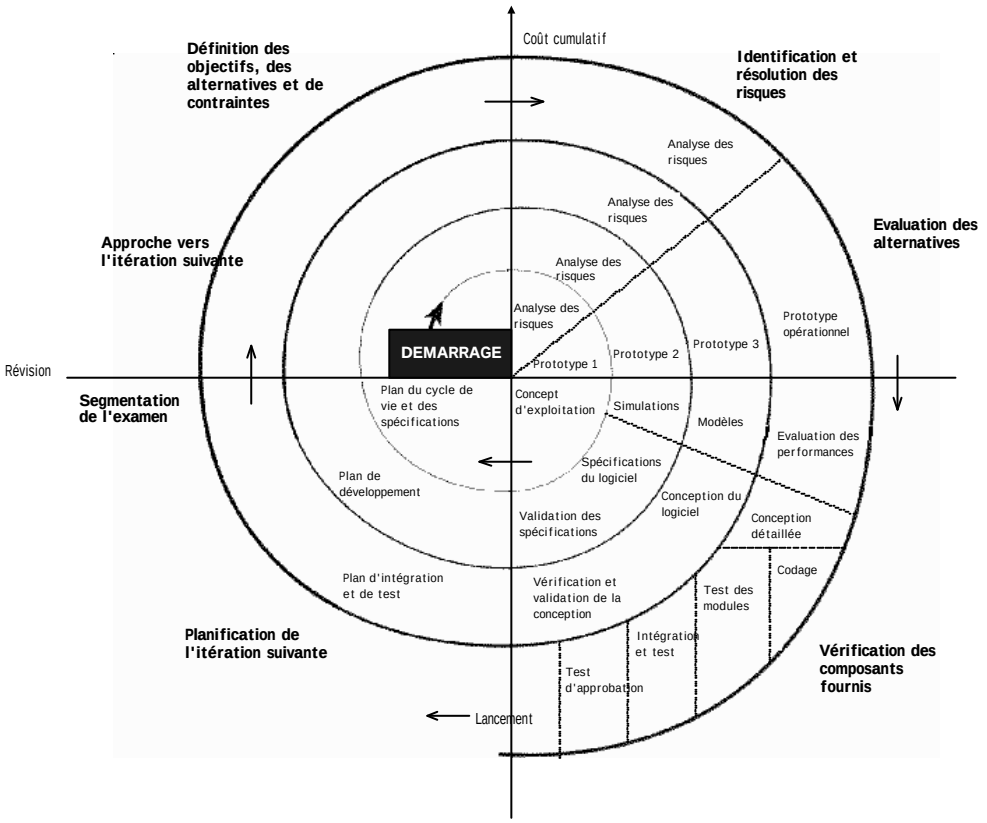


Figure 6 : Modèle en spirale.

Ce modèle présente un avantage majeur : à mesure que les coûts augmentent, les risques diminuent. Plus on investit de temps et de moyens financiers, et moins les risques sont importants. Malheureusement , il est complexe à mettre en œuvre. Il est en effet difficile de définir de manière

[9] Le nom de ce modèle (mis au point par Fuji et Xerox) est inspiré de la présentation du poisson cru, découpé en fines lamelles qui se chevauchent les unes les autres.

objective les étapes permettant de passer à la boucle suivante. De plus, il requiert une grande attention et d'excellentes qualités de gestionnaire.

5.4.) Les techniques de développement rapide.

Contrairement à ce que leur nom laisse supposer, l'objectif principal de ces techniques n'est pas tant d'accélérer que d'améliorer les qualité des développements. Partant du principe qu'il vaut mieux marcher dans la bonne direction que courir dans la mauvaise, ces techniques se basent toutes sur des mises au point fréquentes avec les utilisateurs et l'utilisation intensive du prototypage itératif. Cette manière de faire s'apparente à ce qu'on appelle - dans le domaine scientifique - le protocole essais-erreurs : on lance des ballons d'essai et on profite des erreurs commises pour améliorer son expérience.

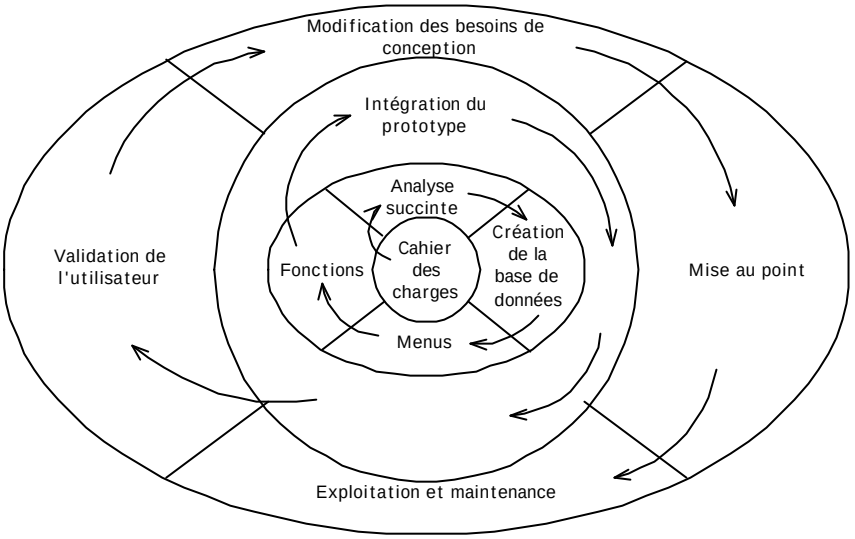


Figure 7 : Prototypage itératif d'applications.

Le « JAD » (développement mixte d'application) est une forme étendue de prototypage itératif dans laquelle l'utilisateur final participe de manière active aux phases d'analyse et de conception. Cette technique regroupe les membres des équipes de développement et du client pour les faire travailler ensemble afin de produire un document contenant les besoins et spécifications.

Le « RAD » (développement rapide d'application) intègre cinq composantes avec l'objectif d'accélérer le développement de systèmes de qualité supérieure. Ce sont :

- La planification du projet.
- La planification et développement de ressources regroupées.
- Le prototypage.
- Les outils CASE.
- La réutilisation du code.

Ces techniques présentent deux risques majeurs. Le premier est que, tout occupé à la réalisation de sa maquette, le concepteur risque d'oublier que l'application doit s'inscrire de façon cohérente dans un système d'information. Le second est que l'utilisateur peut être tenté d'abuser de la liberté qui lui est accordé en changeant indéfiniment d'avis ou en demandant fréquemment des évolutions impliquant un retour en arrière majeur dans la conception.

AUTRES MODELES.

Il existe bien sûr d'autres modèles. Pêle-mêle, on peut citer :

- La « livraison évolutive ». Ce modèle se situe entre le prototypage évolutif et la livraison par étapes. On développe une version de base qui est soumise au client. Ensuite, on améliore ses performances en fonction des exigences du client et on lui soumet la nouvelle version.
- La « conception selon outil ». L'idée fondamentale de cette technique est de ne développer le produit que s'il peut l'être à l'aide d'outils logiciels disponibles. Par « outils », j'entends les bibliothèques de classes, les générateurs de codes et tous les outils logiciels qui permettent de réduite le temps de développement.
- La « livraison par étapes ». Ce modèle - également appelé « implémentation incrémentielle » -suppose une connaissance préalable exacte des caractéristiques du produit à développer. Le logiciel est livré en plusieurs fois, chaque version étant complétée par rapport à la précédente.
- La « conception selon planning » ressemble à la livraison évolutive car, dans les deux cas, on planifie de développer le produit par étapes successives. Mais, avec ce modèle, on ne peut savoir, en démarrant le projet, s'il sera mené à terme. Ce modèle donne en effet la priorité aux caractéristiques les plus importantes du produit : elles seront développées en premier. Les autres sont classées par priorité et seront développées s'il n'y a pas dépassement du planning.

6.) Exercices.

6.1.) Composition / héritage.

Voiture		Moteur
Renault		Constructeur_automobile
Camion		Véhicule
2		Entier
Entier		Nombre
Fraction		Entier
Chaîne_de_caractères		Caractère

6.2.) Conception d'une classe "Date".

6.3.) Conception d'une classe "Arbre binaire".

7.) Etude de la classe String.

La classe String se trouve dans le package java.lang.

Elle hérite de la classe java.lang.Object

Il s'agit d'une classe publique.

Il s'agit d'une classe finale qui n'est donc pas héritable.

Cette classe procure une représentation des chaînes de caractères. Tous les littéraux dans les programmes Java (comme "il était une fois") sont implémentés comme des instances de cette classe.

Constructeurs
String() Initialise un nouvel objet String représentant une chaîne vide.
String(byte[] bytes) Construit un nouvel objet String en convertissant un tableau d'octets.
String(char[] value) Construit un nouvel objet String en convertissant un tableau de caractères.
String(String value) Construit un nouvel objet String en copiant celui passé en paramètre.

Methodes	
`char`	**charAt(int index)** Retourne le caractère à l'index spécifié.
`int`	**compareTo(String autre)** Renvoie une valeur négative si la chaîne se trouve avant autre (dans l'ordre lexicographique), une valeur positive si elle se trouve après ou 0 si les deux chaînes sont identiques.
`String`	**concat(String str)** Ajoute str à la fin de la chaîne.
`static String`	**copyValueOf(char[] data)** Retourne une chaine construite à partir d'un tableau de caractères.
`static String`	**copyValueOf(char[] data, int offset, int count)** Retourne une chaine construite à partir d'une partie d'un tableau de caractères.
`boolean`	**endsWith(String suffix)** Teste si la chaîne se termine par suffix.
`boolean`	**equals(Object anObject)** Compare la chaîne à un objet.
`boolean`	**equalsIgnoreCase(String anotherString)** Comparaison sans prendre en compte les majuscules.
`byte[]`	**getBytes()** Conversion en un tableau d'entiers.
`void`	**getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)** Conversion en un tableau de caractères.
`int`	**hashCode()** Renvoie un « hashcode » pour cette chaîne.
`int`	**indexOf(String str)** Retourne la position de la première occurrence de str.

`int`	**indexOf(String str, int fromIndex)** Retourne la position de la première occurrence de str après fromindex
`int`	**length()** Renvoie la longueur de la chaîne
`String`	**replace(char oldChar, char newChar)** Retourne une chaîne dans laquelle toutes les occurences de oldChar ont été remplacées par newChar.
`boolean`	**startsWith(String prefix)** Test si la chaîne commence par prefix.
`String`	**substring(int beginIndex, int endIndex)** Retourne une sous-chaîne.
`char[]`	**toCharArray()** Converti la chaîne en un tableau de caractères.
`String`	**toLowerCase()** Conversion en minuscules.
`String`	**toLowerCase(Locale locale)** Conversion en minuscules en utilisant les règles de Locale
`String`	**trim()** Enlève les blancs aux extrémités de la chaîne.
`static String`	**valueOf(boolean b)** Retourne une représentation sous forme de chaîne d'un booléen.
`static String`	**valueOf(char c)** Retourne une représentation sous forme de chaîne d'un caractère.
`static String`	**valueOf(char[] data)** Retourne une représentation sous forme de chaîne d'un tableau de caractères
`static String`	**valueOf(double d)** Retourne une représentation sous forme de chaîne d'un double.
`static String`	**valueOf(float f)** Retourne une représentation sous forme de chaîne d'un flottant.
`static String`	**valueOf(int i)** Retourne une représentation sous forme de chaîne d'un entier.
`static String`	**valueOf(long l)** Retourne une représentation sous forme de chaîne d'un entier long.
`static String`	**valueOf(**Object **obj)** Retourne une représentation sous forme de chaîne d'un objet.

8.) Etude de la classe Math.

La classe Math se trouve dans le package java.lang.

Elle hérite de la classe java.lang.Object

Il s'agit d'une classe publique, donc accessible à tous.

Il s'agit d'une classe finale qui n'est donc pas héritable.

Cette classe procure des méthodes permettant les opérations mathématiques de base.

Champs	
`static double`	**E** Base des logarithmes naturels.
`static double`	**PI** 3,1415...

Methodes	
`static double`	**abs(double a)** Valeur absolue.
`static float`	**abs(float a)** Valeur absolue.
`static int`	**abs(int a)** Valeur absolue.
`static long`	**abs(long a)** Valeur absolue.
`static double`	**acos(double a)** Arc cosinus.
`static double`	**asin(double a)** Arc sinus.
`static double`	**atan(double a)** Arc tangente.
`static double`	**ceil(double a)** Renvoie la plus petite valeur égale à un entier et qui n'est pas plus petite que a.
`static double`	**cos(double a)** Cosinus.
`static double`	**exp(double a)** Exponentielle.
`static double`	**floor(double a)** Renvoie la plus grande valeur égale à un entier et qui n'est pas plus grande que a.
`static double`	**log(double a)** Logarithme naturel.
`static double`	**max(double a, double b)** Le plus grand de a et b.
`static float`	**max(float a, float b)** Le plus grand de a et b.
`static int`	**max(int a, int b)** Le plus grand de a et b.

`static long`	**max(long a, long b)** Le plus grand de a et b.
`static double`	**min(double a, double b)** Le plus petit de a et b.
`static float`	**min(float a, float b)** Le plus petit de a et b.
`static int`	**min(int a, int b)** Le plus petit de a et b.
`static long`	**min(long a, long b)** Le plus petit de a et b.
`static double`	**pow(double a, double b)** a puissance b.
`static double`	**random()** Retourne un nombre aléatoire compris entre 0.0 et 1.0
`static long`	**round(double a)** Arrondi de a.
`static int`	**round(float a)** Arrondi de a.
`static double`	**sin(double a)** Sinus.
`static double`	**sqrt(double a)** Racine carrée.
`static double`	**tan(double a)** Tangente.
`static double`	**toDegrees(double angrad)** Conversion radians - degrés
`static double`	**toRadians(double angdeg)** Conversion degrés - radians..

9.) Glossaire.

Abstraction de données.	Principe permettant de définir complétement un objet par son interface, la réalisation des opérations restant cachée et inaccessible du monde extérieur.
Affinage.	Procédé permettant de partir d'une application générale pour en produire une autre plus spécifique à un domaine particulier.
Attribut (ou champ)	Composant statique d'un objet auquel on associe une valeur.
Classe	Description d'une famille d'objets possédant des champs et des comportements similaires. Elle sert de modèle pour créer ses représentants, les Instances.
Classe abstraite	Mécanisme de mise en facteurs des caractéristiques de classes, elle n'a pas pour vocation d'être instanciée.
Encapsulation	Regroupement des données et des traitements dans une même entité. Elle permet de réaliser l'abstraction des données.
Généricité	Modèle de programme dans lequel les types ou les opérations ne sont pas déterminés à priori mais au moment de l'instanciation.
Héritage	Relation de généralisation / spécialisation, qui organise les Classes en une structure hiérarchique.
Instance	Objet, représentant d'une classe.
Instanciation	Mécanisme de création d'objets à partir d'une classe qui joue le rôle de modèle.
Interface	Opération, connue du monde extérieur, applicable à un objet.
Liaison	Mécanisme permettant d'associer un envoi de message à la méthode à appliquer. La liaison peut être statique ou dynamique selon que la recherche de la méthode est effectué à la compilation ou à l'exécution.
Message	Requête adressée à un objet demandant l'exécution d'une méthode de cet objet.
Métaclasse	Modèle de classe dont les instances sont des classes.
Méthode	Procédure décrite dans une classe, appartenant à son interface, exécutée à la réception d'un message.
Objet	Entité regroupant données et procédures, terme générique pour désigner une instance.
Polymorphisme	Mécanisme permettant d'adresser le même message à des objets de types différents qui l'interpréteront à leur manière.
Super-méthode	Désignation d'une méthode masquée par une méthode donnée.
Surcharge	Mécanisme permettant d'appliquer des opérations de même nom à des arguments de types différents.

10.) Exercice.

Exercice 1.1 :

Soit le programme suivant :

```
import java.util.*;
/**
 Manipulation de l'environnement de développement
 * @(#)EX01.java    1.0 2000/01/28
 * @author     CAMOS-CNAM / LWH
 * @version    1.0
 */
public class EX01 {

/** <EM>Point d'<B>entrée</B> de la classe et de <B>l'application</B></EM>
* @param parametres aucun paramètre attendu
* @return pas de valeur de retour
* @exception exceptions Aucune exception traitée
*/
public static void main(String[] argv)
  {
       String v1 = new String("J'aime le jeu, l'amour, les livres, la musique");
       String v2 = new String("La ville, la campagne, enfin tout; Il n'est rien");
       String v3,v4;
       v3 = new String("Qui ne me soit souverain bien");
       v4 = new String("Jusqu'au sombre plaisir d'un coeur mélancolique...");
       String auteur = new String ("Jean de la Fontaine");

       System.out.println(v1);
       System.out.println(v2);
       System.out.println(v3.toUpperCase());
       System.out.println(v4.toLowerCase());
       System.out.println(auteur.toUpperCase());

       v1=v1+v2+v3+v4+" "+auteur;
       System.out.println(v1);

       for (int i = 0 ; i<10;i++)
       {
             System.out.println((int)(Math.random()*10));
       }

       System.out.println("Date du jour : "+new Date());
       System.out.println("Mémoire totale : "+Runtime.getRuntime().totalMemory());
       System.out.println("Mémoire libre  : "+Runtime.getRuntime().freeMemory());

  }
}
```

- En utilisant un éditeur de texte quelconque (Notepad par exemple), créez le fichier "EX01.java".
- Compilez le programme en utilisant javac. Vous devez obtenir un fichier "EX01.class".
- Exécutez le programme avec la machine virtuelle java.
- Générez la documentation avec javadoc.
  (Utilisez la commande **`javadoc -version -author EX01.java`**)
- Désassemblez le programme "EX01.class" à l'aide du désassembleur Java.
  (Utilisez la commande **`javap -c EX01`**)