Chapitre 1

Introduction à la programmation

L'objectif de ce chapitre est d'introduire les langages de programmation.

A travers cette introduction un certain nombre de rappel est réalisé sur les concepts généraux de la programmation informatique.

1. La programmation

Pourquoi doit-on programmer ?

Le monde moderne des XXème et XXIème siècles utilise de nombreuses machines qui permettent de rendre automatisable de nombreux processus qui, sans cela, seraient pénibles à réaliser manuellement, plus couteux, moins sûr et même impossible à réaliser.

Ces machines sont par exemple des appareils industriels à commandes numériques utilisés dans les chaînes de montage, des systèmes complexes quasi autonomes comme une rame de métro sans chauffeur ou un satellite d'observation ou de communication, des systèmes de guidage de missile et bien sur des ordinateurs qui permettent de :

- stocker et gérer des masses d'information
- jouer
- simuler des systèmes complexes (physique, social, IA..)
- communiquer
- ou tout simplement calculer plus vite.

Or les machines et les ordinateurs ne sont pas intelligents (pas encore) et il est indispensable de programmer le comportement attendu avec une grande précision. Le concept fondamental de l'ordinateur n'a pas évolué depuis la création du 1er ordinateur (cf Machine de Turing).

De plus les ordinateurs sont de plus en plus utilisés en masse, communiquent entre eux, s'échangent des informations.

On parle alors de **Systèmes d'Information** (SI) qu'il est nécessaire de créer. Pour cela il est indispensable d'utiliser un ou plusieurs langages de programmation pour traduire en des termes compréhensibles par la machine des concepts de traitement que l'on est capable d'écrire sous une forme de plus en plus évoluée.

On parle alors de langages de programmation évolués.

Un langage de programmation évolué est un ensemble de termes syntaxiques et sémantiques compréhensible par un ordinateur. L'ordinateur exécute les programmes afin de commander les périphériques.

Sans périphériques un ordinateur n'a pas de sens car quelque soit le calcul ou le traitement réalisé il faut au moins afficher à l'écran le résultat, l'imprimer, le stocker pour l'exploiter plus tard ou le mettre à disposition d'autres programmes, transmettre (réseau) le résultat à un programme distant.

L'acte de programmer est d'écrire des programmes en utilisant un ou plusieurs langages informatiques qui doivent ensuite s'exécuter sans erreurs sur un ou plusieurs ordinateurs.

En résumé et par simplification, un **programme** informatique est constitué de très nombreux **traitements** informatiques qui s'enchainent séquentiellement un après l'autre.

Ces traitements sont répartis dans le programme sous forme de différentes **couches logicielles** (ou package).

Remarque : en programmation objet, ces couches logicielles sont des regroupements de classes (appelé package en Java).

On appelle un traitement : une méthode.

Un traitement sans données : cela n'existe pas.

Les données manipulées par un traitement sont de différentes sources :

- des **paramètres** d'entrée du traitement
- des **constantes** littérales

- des **données** appartenant à sa couche logicielle ou à une autre couche logicielle
- des données provenant d'un périphérique (base de données, disques, réseau, …).

On parle de données en **entrée** du traitement.

Généralement un traitement produit ou modifie des données.
Ces données peuvent être :
- des **paramètres** de sortie du traitement
- des **données** appartenant à sa couche logicielle ou à une autre couche logicielle
- des données envoyées à un périphérique (base de données, disques, réseau, …).

On parle de données en sortie du traitement.

Des données en sortie d'un traitement deviennent des données en entrée d'un autre traitement.
Nous verrons que en programmation objet que l'objet lui-même est une donnée qui est à la fois en entrée et en sortie des traitements de l'objet (ou méthode).

Toute la difficulté dans la création d'un programme informatique est le choix de la répartition de ces données et de ces traitements en vu d'optimiser et d'améliorer :
- la lecture de son programme par un tiers,
- la performance de ses traitements,
- la maintenabilité de ses données et de ses traitements,
- l'évolutivité des ses données et de ses traitements.

On parle d'architecture du programme.

2. La réalisation des programmes informatiques

La réalisation d'un programme informatique nécessite plusieurs étapes dont la programmation en est une.

On identifie les phases suivantes :
- l'expression d'un besoin
- la rédaction de spécification
- **la conception des programmes informatique**
- **la réalisation de chacun des programmes (programmation)**
- les tests unitaires des programmes
- les tests d'intégration des programmes
- les tests de validation des spécifications
- la recette des expressions des besoins

Chacune de ces phases nécessitent des connaissances spécifiques et sont réalisées avec des outillages diverses.

Dans le cadre de cette formation nous nous intéresserons aux deux phases en gras ci-dessus :
- **la conception des programmes informatique**
- **la réalisation de chacun des programmes**

La conception se fera par une définition des données et une identification des traitements.
La réalisation des programmes se fera en utilisant le langage Java.

Il est important qu'à ce titre vous puissiez réaliser les travaux dirigés chez vous sur un ordinateur personnel.
Si vous ne possédez pas d'ordinateur personnel, l'IPST-CNAM pourra vous donner un compte utilisateur utilisable en libre service dans les salles de TP.

Il est important qu'à ce titre vous puissiez réaliser les travaux dirigés.

Nous ne ferons pas de tests unitaires à proprement parler mais une simple exécution des programmes Java réalisés.

3. Les langages informatiques

3.1. Chronologie

820 : Al Khowarizmi (prononcez "algorizmi") introduit en Occident la numération décimale et établit des règles élémentaires de calcul numérique (ou **algorithme**)

1854 : **Boole** - tout processus logique peut être décomposé en une suite d'opérations logiques (ET, OU, NON) appliquées sur deux états (ZERO-UN)

1950 : Maurice V. Wilkes (Cambridge) invente **l'assembleur**.

1951 : Grace Murray Hopper - 1er **compilateur A0**

1957 : John Backus (IBM) : **FORTRAN** (FORmula TRANslator)

1958 : Idée d'un langage standard universel : **ALGOL** 58 (ALGOrithmic Language).

1958 : John Mc Carthy (MIT) crée **LISP** (LISt Processing)

1960 : **COBOL** (COmmon Business Oriented Language)

1962 : Kenneth Iverson crée **APL** (A Programming Language).

1964 : Thomas Kurtz et John Kemeny créent le **BASIC** (Beginner's All-purpose Symbolic Instruction Code)

1964 : IBM crée le **PL/I** (Programming Language I).

1964 : code **ASCII** (American Standard Code for Information Interchange), normalisé en 1966 par l'ISO

1966 : **LOGO** est crée par une équipe chez BBN

1968 : Création du langage **PASCAL** par Niklaus Wirth.

1970 : Ken Thompson (UNIX) crée en le langage **B**

1971-1973 : Dennis Ritchie reprend le langage B et défini le langage **C**. Les développements et les succès du langage C et d'UNIX sont intimement liés.

1972 : 1er langage orienté objet, **SmallTalk** par Alan Kay au Xerox PARC.

1979 : **ADA** développé par Jean Ichbiah (Honeywell Bull) choisi par le Pentagone Américain comme unique langage

1983 : Bjarn Stroustrup développe une extension orientée objet au langage C : le **C++**.

1994 : **HTML** développé au CERN

1995 : Delphi développé par Borland : **Pascal orienté objet**

1995 : Rasmus Lerdof crée le 1er "Web language" : PHP (Hypertext Preprocessor)

1995: **JAVA** par SUN (racheté par ORACLE)

1995 : ADA 95 (LOO)

1995 : JavaScript LOO de script pour le Web

1998 : C++98

2000 : C# LOO de Microsoft avec .NET

2009 : Go de Google

3.2. Evolutions

Le langage JAVA est un langage évolué appartenant à la famille des Langages Orientés Objet (LOO).

Deux mots importants : « **évolué »** et « **objet »**.

Evolué, car la syntaxe et la sémantique acceptées par le langage induit une souplesse et une richesse d'expression qui permet de traduire son acte de programmation le plus près de sa pensée et évite un certain nombre d'erreur de programmation.

Objet, car le principe de modélisation et d'architecture du langage est basé sur le modèle à classe d'objet (frame). A ce titre, la façon de programmer dans un LOO est particulière par rapport à la programmation dite « classique ».

La programmation dite classique est incluse dans la programmation objet.

On peut simplifier la situation en disant que la programmation classique permet l'écriture des traitements, et la programmation objet permet de créer les objets qui contiennent les traitements.

Il existe de nombreuses familles de langages :

- les langages orientés objet : Java, C++, C#
- les langages structurés : pascal, ada, C
- les langages fonctionnels : lisp, caml, scheme
- les premiers langages évolués (avant l'assembleur): Fortran, Basic
- les langages machines ou assembleur

- les langages de programmation logique : Prolog
- les langages de script : perl, php, javascript, ksh
- les langages de requête : SQL, Web sémantique
- les langages spécifiques : Postscript, awk,

Chacun de ces langages ont leurs propres mécanismes originaux qui justifient leur appartenance à une famille mais tous ces langages ont des points communs.

Le propre de la programmation est de réaliser le plus simplement possible mais le plus sûr possible un traitement plus ou moins complexe par un ordinateur.

La hiérarchie historique des langages informatiques est la suivante :

- la séquence d'instruction grâce à une bande perforée
  - codage d'une combinatoire d'ordres de commande
  - séquence d'instruction
- le langage machine
  - suite d'instructions très proche du jeu d'instruction du microprocesseur
  - interprétation
  - syntaxe pauvre et absence de sémantique
- le langage évolué
  - syntaxe plus riche, structures de contrôle, sous-programmes
  - gestion de la mémoire par le typage, sémantique
- le langage structuré
  - structuration des données et des méthodes
  - fort typage
  - grande richesse d'instruction et de bibliothèque
- le langage orienté objet
  - conception dirigée par les données
  - le langage devient un support de conception

4. Les moyens de programmation

Il n'y a pas si longtemps que cela, les moyens de programmation consistaient en deux éléments :

- un éditeur de texte plus ou moins sophistiqué permettant d'écrire le programme
- un compilateur (nous verrons plus loin le rôle exact de cet outil) permettant de traduire le programme en un code exécutable par l'ordinateur.

Depuis quelques années, il existe des environnements de programmation (exemple Eclipse , Netbean) qui assiste le développeur dans sa programmation :

- prise en charge automatique de la compilation

- complétion des fonctions et méthodes
- détection à la volée des erreurs d'écriture
- génération automatique de certains codes récurrents
- génération d'ihm (builder)
- génération d'architecture type
- prise en charge de « design patterns » (UML)
- calcul du niveau de qualité du code
- détection des mauvaises pratiques de programmation
- gestion de la configuration des sources
- …

L'utilisation de ces environnements demande une formation adaptée plus ou moins longue.

Le désavantage de ces environnements est qu'il cache un certain nombre de principes de programmation et la méthode de génération, ce qui n'aide pas à la compréhension du langage.

Dans un premier temps, nous faisons le choix de ne pas utiliser un tel environnement, pour les raisons suivantes :

- mieux comprendre les mécanismes de base du langage Java
- maitriser les principes de génération
- c'est par l'erreur que l'on comprend le mieux
- les programmes réalisés sont suffisamment simples pour ne pas nécessiter l'usage d'un tel environnement.

Nous verrons plus loin les moyens de programmation avec java :

- un éditeur (Wordpad, vi, emacs, JEdit, Notepad++)
- la commande de compilation dans une fenêtre de commande
- la commande d'exécution du programme dans une fenêtre de commande.

5. Un langage évolué de programmation

Un langage évolué de programmation est un langage de programmation qui permet une écriture dans une syntaxe « proche » du langage naturel :

- structure de contrôle
- identification libre des éléments (données, traitements)
- structure de données (tableau, structure, collection, indexation, liste)
- bibliothèque de traitements prédéfinis riche
- structure des sous-programmes
- gestion des cas d'erreur d'exécution.

Et permet par cela de minimiser l'effort de programmation.

Il est important de noter que l'usage d'un langage évolué permet de garantir une meilleure qualité de l'écriture du programme et donc un fonctionnement plus robuste.

La démarche industrielle de la réalisation d'un programme informatique est de créer un programme avec le moins de « bug » informatique.

L'apprentissage d'un langage de programmation suffisamment généraliste c'est acquérir les bases de la programmation afin de faciliter ensuite l'apprentissage d'un autre langage. Il est donc important dans la démarche d'assimiler les principes de base et d'appliquer ces principes dans le langage Java.

Le « graal » du langage évolué sera la programmation par la parole (vrai langage naturel). En attendant cela, on commence à exploiter des techniques de conceptions agiles qui aboutissent en la génération de code automatique à partir d'une démarche d'architecture et de designs patterns.

Mais cela est une autre histoire ….

En attendant cela, il nous reste donc à apprendre à programmer dans un langage évolué comme le Java.

6. L'algorithme

L'**algorithmique** est l'ensemble des règles et des techniques qui sont impliquées dans la définition et la conception d'algorithmes, c'est-à-dire de processus systématiques de résolution, par le calcul, d'un problème permettant de décrire les étapes qui conduit au résultat. En d'autres termes, un algorithme est une suite finie et non-ambiguë d'opérations permettant de donner la réponse à un problème.

L'avantage de l'algorithmique est la libre expression des ordres de traitements (ou instructions) avec une strict écriture des structures de contrôle

L'objectif de l'algorithmique est d'obtenir une précision de description de la solution suffisamment précise pour partager et éprouver la solution, mais aussi pour faciliter sa projection vers un langage informatique exécutable.

Exemple :
L 'algorithme de résolution de l'équation du second degre.

```
//Algorithme de résolution de l'équation du 2e degré
//    0 = ax2+bx+c
//
Debut
   Récupérer les paramètres a,b,c qui sont
      les coefficients de l'équation;
   Si a = 0 alors
      retourner qu'il n'y a pas de solution
   Sinon
      delta = b*b-4*a*c ;
      Si delta <0 alors
        retourner qu'il n'y a pas de solution
      Sinon
        si delta == 0 alors
           x = -b / (2*a);
           retourner la solution unique x;
        sinon
           x1 = (-b + racine(delta))/(2*a) ;
           x2 = (-b - racine(delta))/(2*a) ;
           retourner les deux solutions x1 et x2;
        Finsi
      Finsi
   Finsi
Fin
```

7. La création d'un programme exécutable

7.1. L'architecture d'un ordinateur et langage

Un ordinateur est essentiellement composé de :

- une unité centrale ou UAL (Unité Artithmétic Logique) ou microprocesseur (par simplification)
- un jeu d'instruction machine

- une mémoire centrale ou mémoire vive (ou RAM)
- un disque dur (ou mémoire non volatile)
- **des périphériques**

Tout programme informatique, quel qu'il soit, se ramène en la traduction d'un programme machine constitué par des instructions machines exécutables par le microprocesseur.

Ce programme est chargé dans la mémoire centrale constitué de mots mémoires. Cette mémoire contient donc les instructions du programme source ainsi que les zones mémoires (statique ou dynamique) des données.

Un programme machine est constitué de 3 parties :
- la zone des données statiques
- la zone des instructions
- la zone des données dynamiques

Seul langage que connaît un ordinateur est le langage machine de son microprocesseur.
Pour cela, il existe différentes solutions que nous allons aborder.

7.2. La génération d'un programme informatique

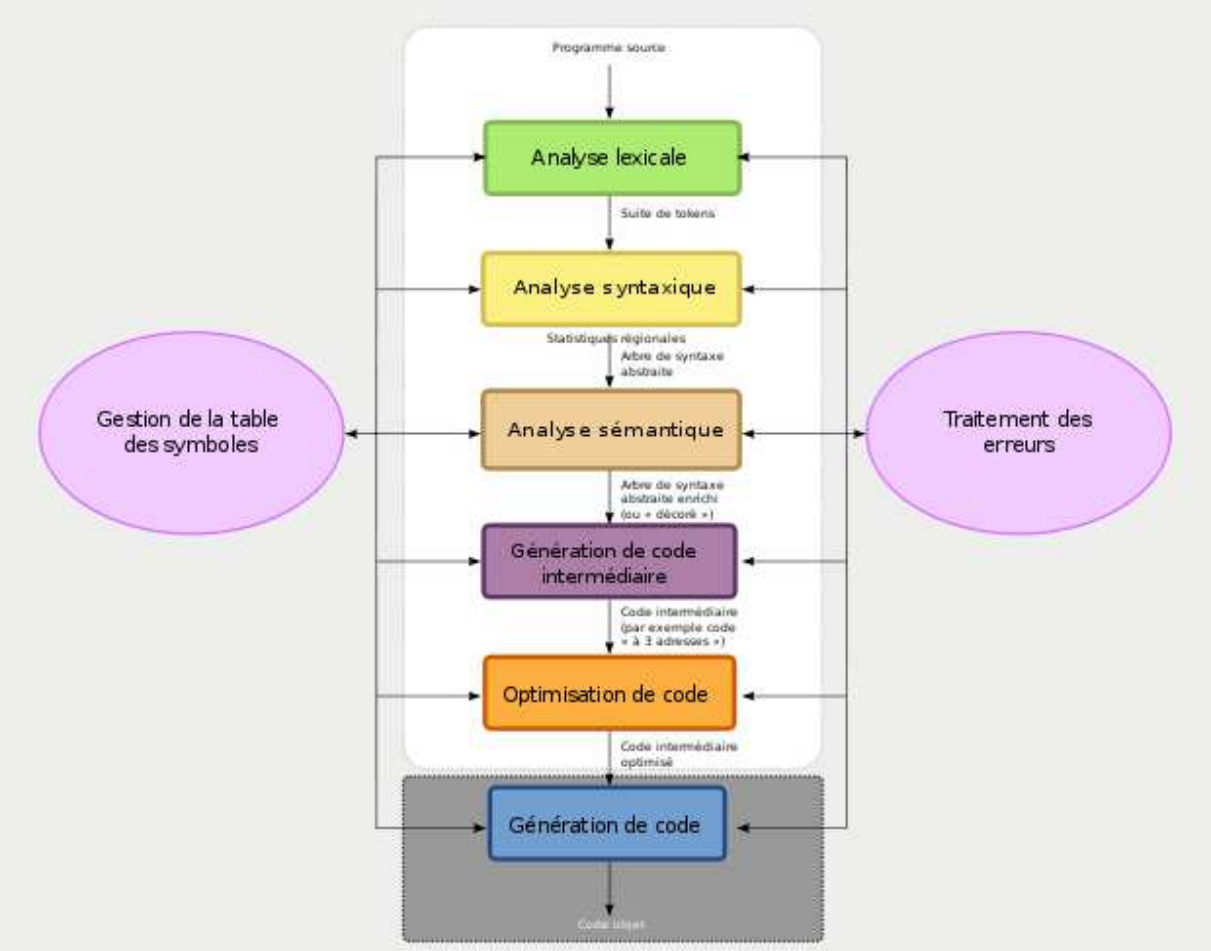
7.2.1. La compilation

Un compilateur est un logiciel dont le rôle est d'analyser le code source (écrit dans un langage évolué), de détecter les erreurs. S'il n'y a pas d'erreurs alors le compilateur traduit les instructions du langage évolué en des instructions binaires directement exécutables par le microprocesseur de l'ordinateur.

Remarque : dans le cas où le programme informatique est composé de plusieurs modules indépendants, il est indispensable de faire ce que l'on appelle une édition de lien permettant de résoudre les adresses des symboles non résolus.

Il existe autant de compilateur qu'il existe de langage de programmation, de fournisseur et de microprocesseur : Pascal, ADA, C, C++, Fortran, …

Le compilateur est donc un programme informatique comme un autre qui a été écrit en un autre langage. C'est ainsi que l'on rencontre des compilateurs ADA écrit en C, des compilateurs C++ écrit en C, des compilateurs Java écrit en C. Il existe aussi des langages auto-référencés (bootstrap) qui se définissent par eux-même à partir d'un noyau élémentaire écrit en langage machine.



7.2.2. L'interprétation

Un interpréteur est un programme informatique dont le rôle est de parcourir séquentiellement chaque ligne du programme, de traduite la ligne en du code machine directement exécutable par le microprocesseur et/ou directement exécutable par l'OS de l'ordinateur (exemple des scripts de commande (ou shell) comme sh, bsh dans le monde unix/linux).

Cela n'est possible que si le programme est écrit dans un langage pas trop complexe (script, perl, php[1], javascript, lisp, …).

7.2.3. Les langages intermédiaires

Cette voie est un mixe avec la compilation et l'interprétation (cas de JAVA).

Ces langages sont utilisés comme intermédiaire entre le langage évolué et le langage machine.

On traduit le langage évolué en un langage intermédiaire : phase de compilation. Le langage intermédiaire est écrit dans une syntaxe et sémantique plus simple et facilement portable sur les différents microprocesseurs du marché.

On exécute le programme en interprétant le langage intermédiaire qui s'exécute ligne par ligne sur le microprocesseur via une MACHINE.

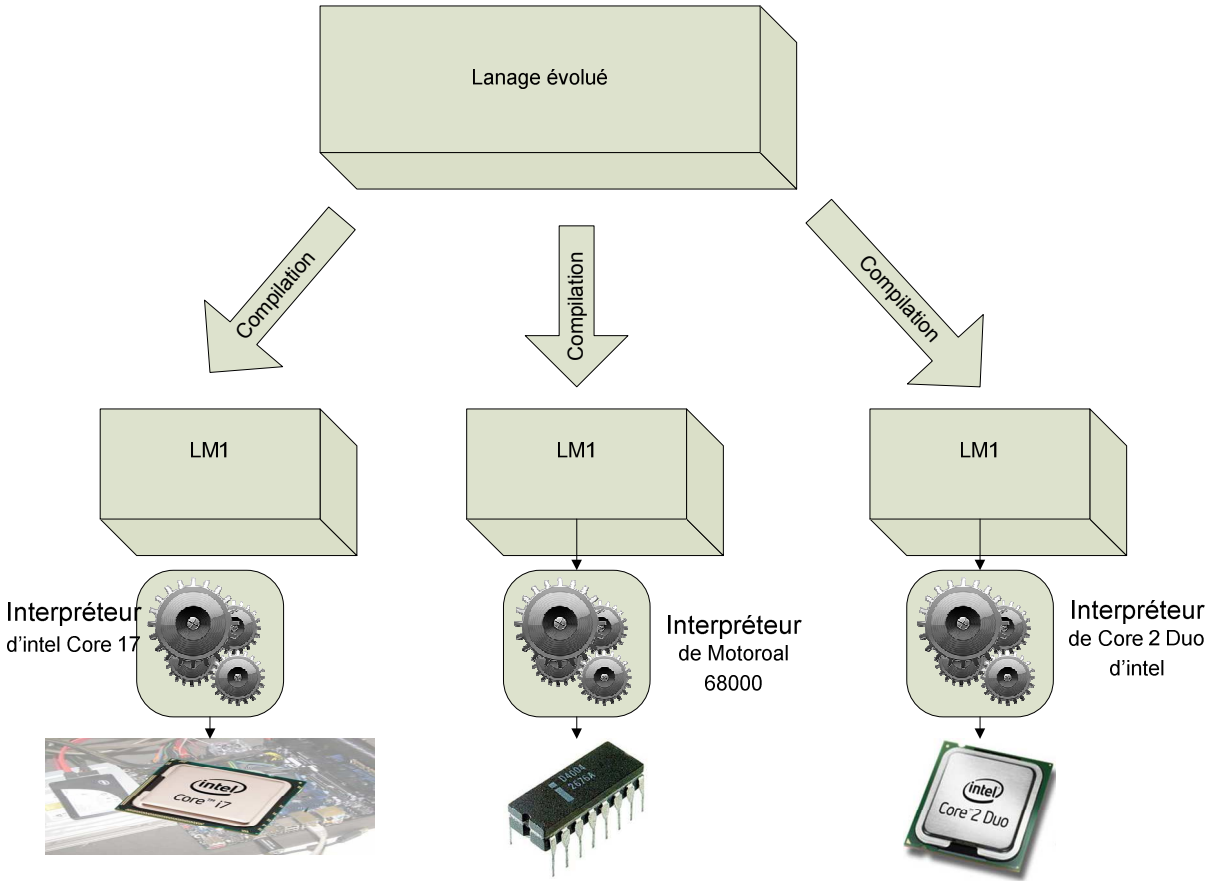
Cette MACHINE est un pré-requis. Il est indispensable qu'elle existe sur l'ordinateur cible. Heureusement comme cela est le cas sur la plupart des OS et des cartes embarquées, des JVM (Java Virtual Machine) sont créées afin de pouvoir exécuter le P-CODE (Pseudo-CODE ou bytecode) utilisé par Java.

[1] En faite, l'interprétation du PHP génère du bytecode qui est ensuite exécuté par une machine virtuelle.

Le bytecode est un code intermédiaire plus concret que le code source, il n'est pas directement exécutable. Il est contenu dans un fichier binaire qui représente un programme, tout comme un fichier objet produit par un compilateur.

Il est appelé bytecode du fait de son format où chaque instruction est codée en binaire.

Puisque c'est un code qui n'est pas exécutable directement par un processeur (à l'exception de certains processeurs gérant le bytecode Java nativement), il est utilisé par les créateurs de langages de programmation en guise de code intermédiaire réduisant la dépendance vis-à-vis du matériel et facilitant son interprétation sur plusieurs architectures.



Un programme à base de bytecode est exécuté par un interpréteur appelé machine virtuelle, du fait qu'il s'agit d'un programme qui exécute le code tout comme un microprocesseur. L'avantage est la portabilité : le même bytecode peut être exécuté sur diverses plates-formes ou architectures pour lesquelles un interpréteur existe.

Un programme sous forme de bytecode peut donc être transmis d'une machine à une autre et être exécuté sans modification par la machine exécutrice quelle qu'elle soit.

L'avantage est le même que pour les scripts, qui sont directement interprétés (et non compilés en bytecode). Cependant, le bytecode est plus abstrait, plus compact et plus facile à manipuler qu'un script, prévu, lui, pour être intelligible pour l'homme. De ce fait les performances des interpréteurs de bytecode sont généralement bien meilleures que celles des interpréteurs de scripts.

7.3. La conception d'un programme informatique

Nous n'aborderons pas dans le cadre de cette formation, les principes du Génie Logiciel ou Cycle de vie du Logiciel qui décrit précisément les différents cycles qui existent et les différentes étapes qu'il est nécessaire de faire dans le cadre d'un développement industrielle d'une application.

Nous simplifierons à notre niveau que pour réaliser un programme informatique et d'autant plus une application (somme de plusieurs programmes), il faut réaliser les étapes suivantes :

1. **lire** l'énoncé du problème (cahier des charges)
2. **identifier** les éléments informatiques (services, informations)
3. **modéliser** l'architecture logicielle (couches logicielles, classes, objets)
4. **détailler** le contenu des classes, identifier les traitements principaux et écrire sous forme algorithmique les traitements les plus complexes
5. **coder**
6. **compiler et exécuter** le programme

Cette simplification est adaptée à nos objectifs de programmation car les problèmes que nous allons traiter sont suffisamment simples (face à un vrai programme industriel) pour ne pas utiliser l'artillerie du génie logiciel.

L'étape 1 consiste à prendre connaissance du cahier des charges ou spécifications. Cette étape est essentielle car c'est à l'issu de celle-ci que nous avons compris ce qu'il faut faire.

L'étape 2 est primordiale dans la programmation objet car c'est elle qui identifie les noms des données à gérer et donc les classes à créer

L'étape 3 permet de dresser les liens de dépendances entre les classes, de créer l'architecture des classes

L'étape 4 consiste à détailler le contenu des classes (attributs et méthodes) et donc aussi de décrire sous une forme assez proche de la syntaxe d'un langage évolué la solution à certains traitements : on parle d'algorithme *de résolution du problème*.

Cette démarche peut s'adapter à beaucoup de domaine et pas uniquement à l'informatique (description de process, de façon de faire, d'existant, de démarche, de pratiques industrielles, …).

L'algorithme permet de vérifier (intellectuellement) et faire partager un process complexe détaillé.

L'étape 5 de codage consiste à transcrire en un langage évolué très formel, non ambigu et compréhensible par un ordinateur, un algorithme plus informel qui ne peut être compris par un ordinateur.

Il existe des ateliers de génie logiciel (framework) permettant de nous aider à réaliser 2, 3 et 4. Ils intègrent même des générateurs de code (étape 5) dans certains domaines (en objet (getteurs et setteurs), Ihm, Persistance avec une BD, …).

L'étape 6 consiste à exécuter le programme et **détecter les erreurs d'exécution**

La mise au point du programme consiste à itérer sur les étapes 5, 6 jusqu'à un fonctionnement satisfaisant et nominal du programme, avec de préférence, une mise à jour ou du retro-ingénierie sur les étapes précédentes.

8. Présentation du langage Java

Java est un vrai langage de programmation :

- JAVA est un langage de programmation à part entière

- L'objectif premier de ce langage était de définir un langage de programmation portable sur toutes les plates-formes existantes (1990)
- Il est devenu un langage moderne et incontournable
- A ne pas confondre avec Javascript

Historique de Java :

- 1993 : projet Oak
- Mai 95 : Netscape prend la licence
- Sept. 95 : JDK 1.0 b1
- Mai 96 : JDK 1.0.2, première version réellement utilisable
- Fin 96 : RMI, JDBC, JavaBeans, ...
- Fév. 97 : JDK 1.1
- Déc. 98 : Java 2 SDK 1.2, apparition du terme Plate-forme Java
- 1999 : Forte orientation de Sun vers le coté serveur plate-forme J2EE : Servlets, JSP et serveurs d 'application
- 2000 Java 2 SDK 1.3
- 2001 Java 2 SDK 1.4
- 2006 Java 2 EE (SDK 1.5)
- 2007 Java 2 EE (SDK 1.6)
- 2014 Version 1.8 (lambda expression)
- 2017 (21 Septembre !!) Version 1.9

JAVA est :

- simple,
- orienté objet,
- familier,
- distribué, (RMI Remote Method Invocation)
- robuste, (exception)
- sûre, (Security policy)
- portable,
- dynamique (ClassLoader)
- multithread
- et réflexif

JAVA est robuste :

- A l'origine, c'est un langage pour les applications embarquées.
- Gestion de la mémoire par un garbage collector.
- Pas d'accès direct à la mémoire.
- Mécanisme d'exception.
- Accès à une référence null exception.
- compilateur contraignant (erreur si exception non gérée, si utilisation d'une variable non affectée, ...).
- Tableaux = objets (taille connue, débordement exception).
- Seules les conversions sûres sont automatiques.
- Contrôle des cast à l'execution

JAVA est un langage orienté objet :

- Tout est classe (pas de fonctions) sauf les types primitifs (int, float, double, ...) et les tableaux
- Toutes les classes dérivent de java.lang.Object
- Héritage simple pour les classes
- Une classe peut implémenter plusieurs interfaces : héritage multiple des méthodes.
- Les objets se manipulent via des références (pointeurs)
- Une API objet standard est fournie
- La syntaxe est très proche de celle de C et C++
- Les exceptions sont des objets
- Pas de structure d'information (Record ou Structure)
-

JAVA est portable :

- Le compilateur Java génère du byte code (p-code) qui s'exécute ensuite dans une JVM
- La Java Virtual Machine (JVM) est présente sur Unix, Win32, Mac, OS/2, Linux, Netscape, IE, ...
- La syntaxe du langage Java est le même partout
- La taille des types primitifs est indépendante de la plate-forme.
- Les API de Java sont les mêmes dans tous les OS

Les domaines d'application de Java :

- La simple programmation de programme ou d'application standalone (eclipse)
- La programmation internet des IHM (applet)
- La programmation internet des services internet (Java Server Page)
- Le Web en général
- La programmation distribuée (RMI, CORBA)
- La programmation embarquée (JVM optimisé embarquée)