Chapitre 1

Introduction

Nos objectifs pédagogiques sont :

1. Étudier les concepts de base de la programmation dans les langages de haut-niveau, de manière à :
   – les appliquer en Java,
   – comprendre des concepts présents dans beaucoup d'autre langages de programmation.
2. S'initier à l'analyse et resolution de problèmes, via la programmation,
3. Acquerir certaines méthodes de resolution des problèmes classiques en informatique.

1.1 Pourquoi Java ?

Tout langage de programmation de haut niveau renferme *grosso modo* les mêmes concepts. Chacun devient en quelque sorte, un *dialecte* différent pour un *pouvoir d'expression* (ce que l'on peut faire avec) très proche. Notre but est, qu'une fois les concepts de base de la programmation acquis via ce cours, et via la la pratique de Java, vous soyez capables de :

– vous former par vous même dans l'apprentissage d'un autre langage de programmation (dialecte)
– de programmer rapidement (parler ce dialecte) en gardant les mêmes techniques et reflexes déjà appris avec l'étude de Java.

Les avantages de Java :

– Il existe des environnements de développement gratuits que vous pouvez installer par vous mêmes,
– Java est fortement typé, ce qui signifie, que beaucoup d'erreurs sont détectée s automatiquement (et c'est essentiel pour un débutant)
– Java incorpore des nombreux traits de programmation de haut niveau : orienté objet, exceptions, polymorphisme, gestion de la mémoire, transparence des pointeurs.
– Java possède une sémantique précise.
– Les programmes Java sont portables. Leur exécution est indépendante de la plateforme d'installation (type de machine). Il suffit de disposer d'un environnement d'exécution Java pour l'exécuter.
– Java est robuste et sécurisé.

Principaux domaines d'application :

– Internet et le Web,

- Programmation distribuée,
- Programmation embarquée.

1.2 Les programmes

Les programmes servent à décrire les solutions d'un problème.

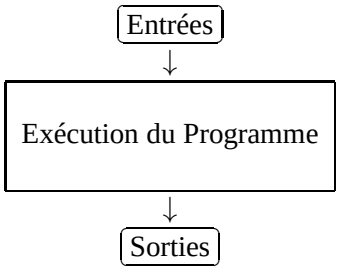
Qu'est-ce qu'un programme ?

- C'est la **description d'une méthode** mécanique (i.e, appliquable par une machine) de **resolution** d'un problème donné,
- Cette description est donnée par une **suite d'instructions** d'un langage de programmation.
- Ces instructions ont pour but de **traiter et transformer les données** du problème à resoudre, jusqu'à aboutir à une solution.

Un programme est en quelque sorte une méthode à suivre pour trouver les solutions, mais n'est pas une solution en soi.

Trouver les solutions : **exécuter** le programme sur les données du problème.

- Les traitements décrits par les instructions sont appliqués aux données (*entrées*),
- Les données ainsi transformées permettent d'aboutir aux solutions recherchées (*sorties*).



1.3 Les ordinateurs

Composants principaux :

- **Unité centrale (CPU)** ou processeur, pour le traitement des données et instructions logées en mémoire centrale. Capable d'exécuter un ensemble d'instructions dites *instructions machine*. Il s'agit d'opérations simples telles que la lecture/écriture en mémoire centrale, opérations arithmétiques et logiques, comparaison des valeurs, branchement (saut) vers une adresse afin de poursuivre l'exécution, etc. Ces instructions sont codées en binaire (s'équences de 0 et de 1) et diffèrent dans chaque plateforme matérielle. Ainsi, une instruction qui s'exécute sur un processeur Intel/PC ne peut pas s'exécuter sur un processeur SPARC/Sun.
- **Mémoire Centrale (RAM)**, très rapide (accès directe), mais volatile. Cette mémoire est organisé comment une suite d'emplacements appelés *mots* et munis chacun d'une adresse. On doit y stocker les données à traiter, ainsi que les instructions machine à exécuter. Tout programme à exécuter et toute donnée à traiter doit être chargé en mémoire centrale au préalable.
- **Périphériques**. Dispositifs de communication de l'ordinateur pour l'aquisition, production, communication des données : clavier, écran, enceintes, ports de communication ; et pour le stockage persistent tels que disques durs, disquettes, etc.

Fonctionnalités principales :

- Exécution des instructions des programmes chargés en mémoire centrale.
- Commande des périphériques.
- Acquisition, stockage, communication et production des données : saisie au clavier, affichage à l'écran, etc.

1.4 Les langages de programmation

Il en existe deux groupes principaux :

- **Langages de haut niveau**. (Ex : C, Ada, Pascal, Cobol, Java, OCaml, Python). Ils fournissent des nombreuses constructions sophistiquées qui facilitent l'écriture des programmes. Ils sont comprehensibles par les humains, mais pas directement exécutables par les machines. Un programme écrit en langage de haut niveau devra être traduit en langage machine avant son exécution.
- **Langages de bas niveau**. Ce sont les différents ensembles d'instructions propres à chaque machine (SPARC/Sun, Intel/PC, etc). Appelés également *langages cibles* ou *natifs*. Ils sont codés en binaire et directement exécutables par chaque machine.

Langages de haut niveau

Un langage de programmation est composé de trois ensembles : un ensemble de *types de données*, un ensemble de règles de *construction syntaxiques des instructions* et un ensemble de *règles sémantiques*. Ces trois composants décrivent toutes les possibilités des données et instructions dans un programme, ainsi que leur comportément à l'exécution.

- **Les types des données** : utilisées pour décrire et modéliser les données. *Exemple :* Le type `int` en Java sert à modéliser les nombres entiers.

- **La syntaxe** : règles de *formation textuelle* des instructions et des programmes. *Exemple :* pour écrire l'expression mathématique $1 \leq x \leq 7$, une syntaxe possible en Java est : `1 <= x && x <= 7`.

- **La sémantique** : règles qui précisent, d'une part le *comportement* ou le *sens* des contructions syntaxiques lorsqu'elles sont executées par une machine et d'autre part, les *contraintes de cohérence entre types*. *Exemples* : `4+3*2` équivaut en Java à la valeur entière 10. L'expression `"bonjour" * 2` est bien formée du point de vue de la syntaxe, mais pas du point de vue sémantique : `*` est un opérateur numérique, et ici il a pour opérande la chaîne de caractères `"bonjour"`. Cette expression est incohérente vis-à-vis des types.

1.5 La production des programmes

La fabrication d'un programme suit les phases suivantes :

1. **Analyse et Conception** : On établit précisement le problème à resoudre, les données de départ et ce que l'on souhaite obtenir en résultat, puis, on raisonne sur la manière de représenter les données du problème, et sur une méthode de resolution. Cette méthode est ensuite énoncée sous

forme de suite de pas à accomplir pour aboutir aux solutions : c'est *l'algorithme de resolution du problème*.

Représentation données $\Rightarrow$ *Modèle des données*

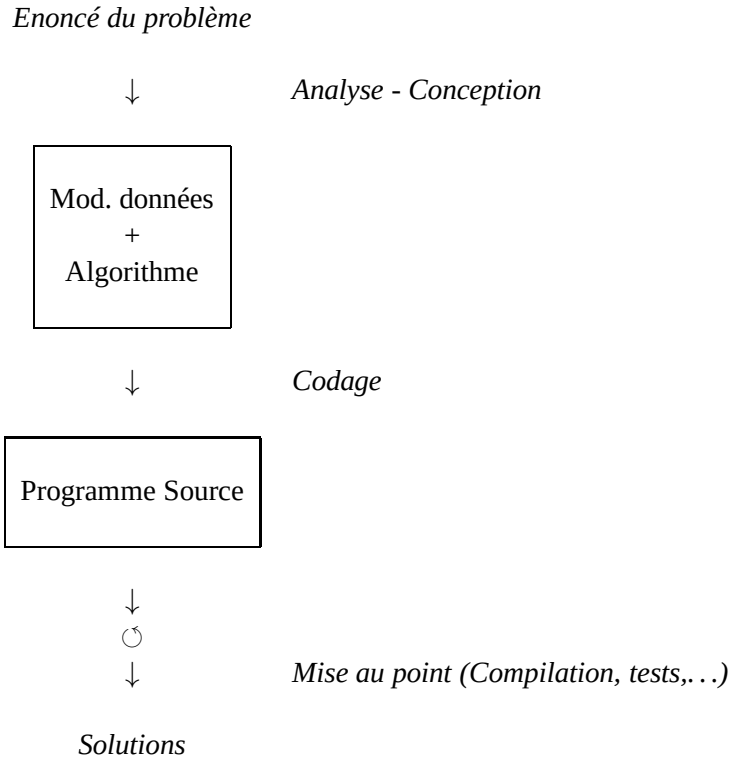
+

Enoncé pas à pas des traitements $\Rightarrow$ *Algorithme*

2. **Codage** : il s'agit de traduire l'algorithme en langage de programmation, et sous forme de fichier texte (c.a.d., suite de caractères sans formattage ni mise en page). Le résultat du codage est un fichier appelé *code source* du programme.

3. **Mise au point** : comprend le plus souvent plusieurs étapes repétées jusqu'à ce que le programme semble satisfaisant :
   - Compilation : un langage de programmation de haut niveau n'est pas directement comprehensible par la machine. Il faut donc traduire le *code source* du programme vers le *langage natif* de la machine (ou parfois vers du code intermédiaire). Le résultat est un nouveau fichier écrit en langage machine, et appelé *code objet*. Cette étape comprend souvent également *l'édition de liens*, qui est la préparation du *code objet* pour l'exécution.
   - Tests : exécution du code objet avec divers cas typiques des entrées, ou *jeu de tests*. C'est le moment où la plupart des erreurs apparaisent.
   - Correction d'erreurs : on modifie le code de manière à corriger les erreurs au fil des tests, et l'on recommence la compilation, exécution et tests, etc.

4. **Maintenance** : il s'agit le plus souvent de la correction des erreurs apparues après la mise en service, mais aussi de la modification du programme pour l'adapter à de nouvelles spécifications du problème.



1.6 Traducteurs de programmes

Chaque langage de programmation est équipé d'un *logiciel de traduction* des programmes écrits dans ce langage. Entre autres traitements, il traduit chaque instruction de haut niveau en plusieurs instructions machine équivalentes.

- code source : fichier texte avec les instructions à traduire.
- code cible : fichier (binaire ou texte, selon le type de traduction) résultat de la traduction.
- code objet : fichier binaire avec des instructions machine.
- pseudo-code ou byte-code : fichier texte contenant du code intermédiare d'assez bas niveau, mais non exécutable directement par la machine.

Il existe deux sortes de traducteurs : les **interprètes** et les **compilateurs**.

Compilateurs

Réalisent les traitements suivants sur le code source :

1. *Analyse syntaxique* : vérifie que le programme est correct d'après les règles de syntaxe.
2. *Typage* : vérifie la correction d'après les règles (sémantiques) de cohérence entre types.
3. *Traduction* : S'il n'y pas d'erreurs de syntaxe ni de typage, chaque instruction du programme source est traduite vers plusieurs instructions du langage natif ou de *pseudo-code* (code intermédiare). Le résultat de la compilation est un nouveau fichier dit de *code cible* qui est pour la plupart des compilateurs (Ada, Pascal, etc.) du code machine pour la plupart des compilateurs mais ou du pseudo-code. Dans ce dernier cas, il sera à traduire avant exécution. S'il y a des

erreurs de syntaxe ou de typage, il faut les corriger et relancer la compilation jusqu'à ce qu'il n'y ait plus d'erreurs. Seulement alors, le code cible est généré.

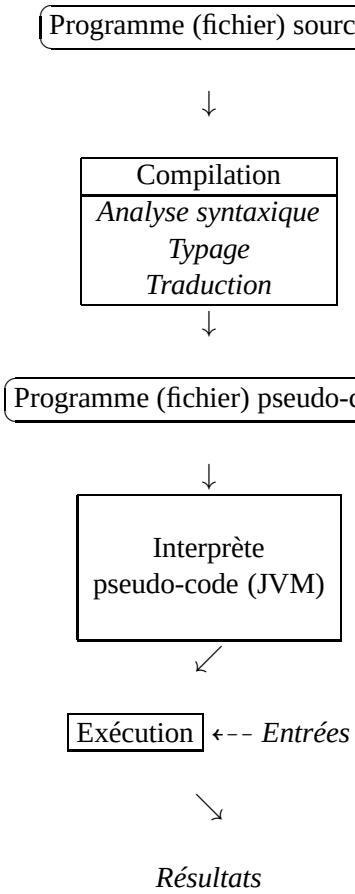
C, Pascal, Ada, Ocaml, Java sont des langages compilés. Mais Ocaml et Java sont compilés vers du pseudo-code, plutôt que vers du code natif.

Interpréteurs

Pour chaque instruction du programme, ils réalisent, l'une après l'autre, traduction en code puis exécution. Ce mécanisme a pour inconvénient de limiter le type d'analyses réalisées sur le programme source, mais surtout, il autorise l'exécution d'un programme même s'il présente des erreurs de syntaxe ou de sémantique. Cependant, ce mécanisme est bien adapté aux programmes pre-compilés vers du pseudo-code. C'est le cas de Java.

Compilation en Java

Un des objectifs de Java, est de permettre l'écriture des programmes independants des plateformes matérielles d'exécution. C'est le defi de la programmation distribuée que d'envoyer des programmes à travers le réseau pour leur exécution à distance. Dans ce contexte, la compilation vers du code natif n'a pas grand sens. C'est pourquoi en Java, on compile vers un code intermédiaire ou *byte-code* identique pour tous les ordinateurs. L'environnement d'exécution Java est équipé, dans chaque machine, d'un interprète qui lui est propre. Il traduit le byte-code vers du code natif et l'exécute, avec un minimum d'analyses. L'ensemble des interprètes est c'est que l'on appelle la Machine Virtuelle Java (JVM). Ainsi, un programme traduit en byte-code, pourra être exécutée par n'importe quelle machine munie d'une JVM. Les programmes Java développés sur une machine particulière sont *portables* sur n'importe quelle installation sans avoir à réecrire du code.



Le programme `Bonjour`

Ce texte correspond à un programme Java syntaxiquement et sémantiquement correct.

```
/* Premier programme
   -a enregistrer dans le fichier Bonjour.java
   -a compilar par javac Bonjour.java
   -a executer par java Bonjour
*/

public class Bonjour {
   public static void main (String[] args) {
      Terminal.ecrireStringln("Bonjour tout le monde !");
   }
}
```

- Le texte entre `/*` et `*/` est un commentaire.
- Le nom après `public class` est le nom du programme. Dans cet exemple, c'est `Bonjour`.
- Ce programme ne manipule qu'une donnée : le message

  `Bonjour tout le monde !`

qui est affichée à l'écran lors de l'exécution.
- Nous le composons dans un fichier du même nom que le programme, et complété de l'extension `java`, à savoir, dans le fichier `Bonjour.java`

Ce programme n'est pas directement exécutable par l'ordinateur. Nous devons donc le traduire en langage machine.

Compilation et exécution de Bonjour

- Création du fichier source : Le code Java est mis dans un fichier nommé `Bonjour.java`. C'est le programme source.
- Compilation : Nous lançons la commande de compilation sur le fichier `Bonjour.java`,
  ```
  Java/Essais> javac Bonjour.java
  Java/Essais>
  ```
  La compilation reussit (pas d'erreurs). Le pseudo-code obtenu se trouve dans le fichier `Bonjour.class` du repertoire courant, mais il n'est pas directement exécutable.

- Interprétation + Exécution : elle se fait en appelant l'interprète de byte-code de la machine, avec la commande `java` suivie du nom du programme. On voit s'afficher le message attendu :
  ```
  Java/Essais> java Bonjour
  Bonjour tout le monde !
  ```

L'exemple avec des erreurs

Nous changeons le code source afin d'y introduire quelques erreurs et voir les messages de diagnostique donnés par le compilateur. Le code est sauvegardé dans le fichier `Bonjour2.java` :

```
public classe Bonjour2 {
    public static void main (String[] args) {
        Terminal.ecrireStringln("Bonjour tout le monde !"*2);
    }
}
```

Ce code contient deux erreurs :
- *erreur de syntaxe* : le mot-clé `class` qui introduit le nom du programme a été changé en `classe`.
- *erreur de typage (sémantique)* : le message `"Bonjour tout le monde !"` est une chaîne de caractères (type `String`) et en tant que tel ne peut être multiplié par 2.

```
Java/Essais> javac Bonjour2.java
Bonjour2.java:1: 'class' or 'interface' expected
public classe Bonjour2 {
       ^
1 error
```

Le compilateur s'arrête à la première erreur trouvée (ligne 1). Nous corrigeons et relançons la compilation :

```
Java/Essais> javac Bonjour2.java
Bonjour2.java:3: operator * cannot be applied to java.lang.String,int
        Terminal.ecrireStringln("Bonjour tout le monde !"*2);
                                                         ^
1 error
```

Ce message est plus difficile à comprendre. Il nous dit que à la ligne 3, l'opérateur `*` n'est pas applicable sur le type `String`.

1.7 Analyse des problèmes

Voici une démarche posible pour développer un programme à partir de l'énoncé d'un problème :

1. *Déterminer le problème à résoudre :*
   (a) quelles sont les données d'entrées et leur nature ?
   (b) quelles sont les données attendues en sorties et leur nature ?

2. *Déterminer la méthode :*
   (a) comment modéliser les données (d'entrée, de sortie, intermédiaires) ?
   (b) quels sont les différents cas des entrées à traiter, et les cas écheant, quels sont les traitement associés.
   (c) exprimer sous-forme d'algorithme la manière d'obtenir le résultat final à partir des traitements sur les données d'entrée.
3. *Correction, complétude et lisibilite*
   (a) A-t-on bien prevu tous les cas des entrées et sorties ?
   (b) Obtient-on dans chaque cas ce que l'on voulait calculer ?
   (c) Notre algorithme est-il comprehensible par quelqu'un d'autre ?
4. *Tests*
   (a) Elaborer un jeu de tests representatif de tous les cas possibles (univers) des entrées. Un *jeu de tests* consiste en une suite de "valeurs typiques" pour chacune des entrées, accompagnées des valeurs attendus comme résultat dans ce cas. Par exemple, si on veut tester un programme qui calcule le carre d'un nombre entier, avec entrée x (entier) et sortie z (entier) ; un jeu de tests (non exhaustif) peut être $\{(x \leftarrow 0, z \mapsto 0); (x \leftarrow 1, z \mapsto 1); (x \leftarrow 3, z \mapsto 9)\}$
   (b) Confronter le fonctionnement de l'algorithme avec le jeu de tests proposé.

Les algorithmes

Un algorithme est un énoncé détaillé sous forme de pas à suivre dans l'application des traitements ou calculs sur les données. Il s'agit d'une *méthode systématique* de résolution d'un problème.

Exemple 1 : la feuille de déclaration d'impôts sur le revenu est accompagnée d'une description des opérations pour calculer le montant de l'impôt :

- le *problème* : le calcul du montant de votre impôt
- les *données* : les chiffres de vos salaires, charges, abattements, etc.
- *l'algorithme* : ou méthode de résolution du problème, c'est la suite des calcule à réaliser sur les données.
- le *résultat ou sortie* : le montant de votre impôt.

Il peut y avoir plusieurs méthodes pour résoudre un problème. Par exemple, on peut commencer par calculer le quotient familial, plutôt que par le calcul des abattements. □

Exemple 2 : Une recette de cuisine est un exemple classique d'algorithme.

Problème : Préparation d'une omelette

Données : Ce sont les ingredients,
- 2 oeufs,
- sel,
- un peu de matière grasse

Algorithme : C'est la méthode de préparation,

1. Casser les oeufs dans un bol,
2. Y ajouter du sel, puis les battre,
3. Faire chauffer la matière grasse dans une pôele,
4. Verser le mélange des oeufs dans la pôele et faire cuire doucement jusqu'à la consistance souhaitée.

□

En pratique, un problème est rarement aussi simple (à comprendre, à modéliser, à resoudre). Par exemple, considérez le problème :

Etant donné le réseau routier d'une ville et la situation de la circulation, calculer le plus court chemin pour aller d'une adresse à une autre.

Un exemple

Problème : Calculer et afficher la conversion en francs d'une somme en euros saisie au clavier.

Analyse :

1. Déterminer le problème à résoudre :
   (a) les entrées et leur nature ⇒ un nombre réel eu
   (b) les sorties attendues ⇒ un réel fr tel que $fr = eu * 6.559$

2. Déterminer la méthode :
   (a) modélisation des données : fr, eu modélisés par des nombres flottants (type `double` de Java).
   (b) l'algorithme

1. lire la valeur saisie pour eu,
2. calculer $fr = eu \times 6.559$
4. afficher le résultat final fr

(c) un jeu de tests :

$(eu \leftarrow 0.0, fr \mapsto 0.0)$;
$(eu \leftarrow 15.0, fr \mapsto 98.35)$;
$(eu \leftarrow -3.5, fr \mapsto -22.9565)$;
$(eu \leftarrow 10, fr \mapsto 65.59)$;

Résultats de l'analyse

Le résultat principal de l'analyse est l'algorithme, que l'on doit accompagner des données sur lequel il agit, afin de le rendre comprehensible. Il devient ainsi l'analogue d'une recette de cuisine : on donne la liste des ingredients, puis la procédure à suivre pour les mélanger.

Données + algorithme :

- *Données* :
  entrées : eu nombre réel
  sortie : fr nombre réel
- *Algorithme* :
  1. lire eu
  2. calculer $fr = eu \times 6.559$
  3. afficher le résultat se trouvant dans fr

Un codage en Java

```
/* Un premier programme:
   -son nom est Conversion
   -il est sauvegarde dans le fichier Conversion.java
   -il est compile par javac Conversion.java
   -il est execute par java Conversion
*/

public class Conversion {
    public static void main (String[] args) {
        // Variables du programme
        double euros, francs;
        // Message pour la saisie
        Terminal.ecrireStringln("Entrez la somme en euros: ");
        // Lecture dans la variable euros
        euros = Terminal.lireDouble();
        // Calcul de la conversion
        francs = euros *6.559;
        // Affichage du resultat
        Terminal.ecrireStringln("La somme convertie en francs: "+ francs);
```

```
    }
}
```

Suite et fin de l'exemple

Le code Java est mis dans un fichier nommé `Conversion.java`. Nous lançons la compilation puis l'exécution :

```
Java/Essais> javac Conversion.java
Java/Essais> java Conversion
Entrez la somme en euros:
10
La somme convertie en francs: 65.59
```

Tests : Nous répetons plusieurs fois l'exécution avec différents valeurs pour les entrées :

```
Java/Essais> java Conversion
Entrez la somme en euros:
-3
La somme convertie en francs: -19.677
Java/Essais> java Conversion
Entrez la somme en euros:
15.5
La somme convertie en francs: 101.6645
```

1.8 Résumé du chapitre 1

Les *ordinateurs* sont composés principalement d'un *processeur* (CPU), d'une *mémoire centrale*, et de dispositifs *périphériques*. Le processeur est chargé d'exécuter les programmes. La mémoire centrale contient les instructions du programme et ses données. Elle est organisée comment une suite d'emplacements appelés *mots*, et munis chacun d'une adresse pour un accès direct. Les périphériques sont les dispositifs de communication de l'ordinateur : clavier, écran, enceintes, ports de communication, disques durs.

Un *programme* sert à décrire une méthode mécanique, autrement dit, applicable par une machine, de resolution d'un problème. Il traite *des données d'entrée* et calcule et produit des solutions, qu'on appele aussi ses *sorties*. Pour obtenir les sorties, une machine *exécute* le programme sur ses données d'entrée.

Un programme est exprimé sous forme de *suite d'instructions* dans un *langage de programmation*. Il existe deux grandes familles de langages : les *langages de haut niveau*. (ex : C, Ada, Pascal, Cobol, Java, OCaml, Python), et les *langages de bas niveau* ou *langages machine*, qui sont propres à chaque plateforme matérielle (SPARC/Sun, Intel/PC, etc). Les langages de haut niveau sont puissants et compréhensibles par les humains, mais ils ne sont pas directement exécutables par une machine. Ils sont à *traduire au préalable en langage machine*.

Un *langage de haut niveau* est décrit par ses types, sa syntaxe et sa sémantique. *Les types des données* servent à décrire les données du programme, p.e., le type `int` en Java sert à modéliser les

nombres entiers. *La syntaxe* décrit les règles de *formation textuelle* des instructions et des programmes, p.e, pour écrire l'expression mathématique $1 \leq x \leq 7$, une syntaxe possible en Java est : `1 <= x && x <= 7`. *La sémantique* précise, d'une part le *comportement* ou le *sens* des contructions syntaxiques lorsqu'elles sont executées par une machine et d'autre part, les *contraintes de cohérence entre types*.

Un *algorithme* pour la résolution d'un problème est un énoncé détaillé sous forme de pas à suivre pour calculer les solutions ou sorties pour le problème à partir de ses données d'entrée.

La *production d'un programme* suit les phases suivantes :

1. *Analyse et Conception* : On précise le problème à resoudre et les données (*entrées* et *sorties*) du problème. Puis, on énonce une méthode de resolution sous forme d'algorithme.
2. *Codage* : traduction de l'algorithme en langage de programmation, et sous forme de fichier texte : c'est le *code source* du programme.
3. *Mise au point* : comprend la compilation, tests et correction d'erreurs.

Un *compilateur* est un traducteur de programmes : il traduit chaque instruction de haut niveau se trouvant dans le code source en plusieurs instructions machine ou de pseudo-code équivalentes. Le résultat de la compilation du code source est un nouveau fichier, nommé *code objet*, s'il est composé d'instructions machine (fichier binaire), ou pseudo-code, si c'est un fichier texte composé d'instructions en pseudo-code. Un compilateur réalise aussi de traitements *d'analyse syntaxique* et de *typage* du code source. Il ne réalise la traduction du code source que si aucune erreur n'est détectée lors de ces analyses.

Il existe une autre sorte de traducteur : les *interprètes*. Pour chaque instruction du programme, ils réalisent, l'une après l'autre, traduction en code machine, puis exécution. En Java, on compile vers un code intermédiaire ou *byte-code* identique pour tous les ordinateurs. Ensuite, un interprète propre à chaque machine, traduit le byte-code vers du code machine et l'exécute. L'ensemble des interprètes est c'est que l'on appelle la Machine Virtuelle Java (JVM).