Chapitre 1: Introduction

(NFA031 - Jour)

V. Aponte

Cnam

30 décembre 2016

Objectifs

1. Démystifier la programmation :
   - les programmes sont partout !
   - comprendre leur fonctionnement,
   - savoir résoudre des problèmes simples avec des programmes,
   - mieux exploiter tous les dispositifs qui les utilisent !
2. Acquérir les éléments fondamentaux de la programmation : ils sont communs à tous les langages.
3. Savoir écrire des programmes simples en Java : avec boucles, tableaux et sous-programmes.

Pourquoi Java ?

- Nombreux environnements de développement gratuits,
- Java est fortement typé : beaucoup d'erreurs sont détectés à la compilation (essentiel pour un débutant)
- Langage multi-paradigme : impératif, objet, modules, polymorphisme.
- Java est un langage portable.

Partie I : Les programmes

A quoi sert un programme ?

- une machine trouve les solutions à un problème ;

Qu'est-ce qu'un programme ?

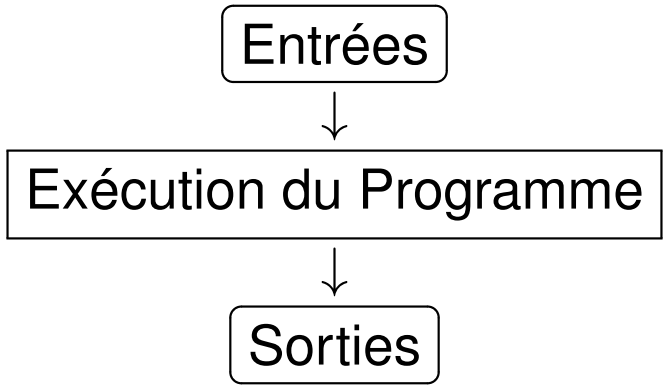
- méthode de résolution d'un problème *applicable par une machine* ;
- composé d'une suite d'instructions,
- écrit dans un langage de programmation,
- Exécution : application *séquentielle* des instructions.

Un programme traite des données

Programme : transforme des données (entrées) pour aboutir à une solution (sorties).

Entrées données initiales du problème à résoudre.

Sorties données obtenues après exécution du programme.



Un premier programme

A l'exécution, affiche «Bonjour à tous !»

```
/* Notre premier programme (dans fichier Prem.java) */

public class Prem {
  public static void main (String[] args) {
    System.out.println("Bonjour à tous!");
  }
}
```

- Le texte entre `/*` et `*/` est un commentaire.
- après `class` ⇒ nom du programme. (`Prem`)
- Utilise une donnée (toujours la même) : chaîne de caractères `"Bonjour à tous!"`

Les pas à suivre pour utiliser «son» programme

1. Création du programme :
   - taper son contenu dans un éditeur de texte ;
   - l'enregistrer dans le fichier `Prem.java`
2. Compilation du fichier :
   - avec la commande (linux) : `% javac Prem.java`
3. Exécution du programme :
   - avec la commande (linux) : `% java Prem`

```
% javac Prem.java
% java Prem
Bonjour à tous
```

Demo (1) : édition d'un fichier avec notre programme

1. Création du programme (avec JEdit)
   - taper son contenu dans l'éditeur JEdit ;
   - l'enregistrer dans le fichier `Prem.java`

Programme source : du texte en Java

Le fichier `Prem.java` :

- contient des instructions Java : **c'est du texte !**
- on appelle `Prem.java` $\Rightarrow$ programme source.
- Aucune machine ne sait exécuter du texte...

Alors, comment exécuter ce programme ?

Ce qu'un ordinateur sait exécuter

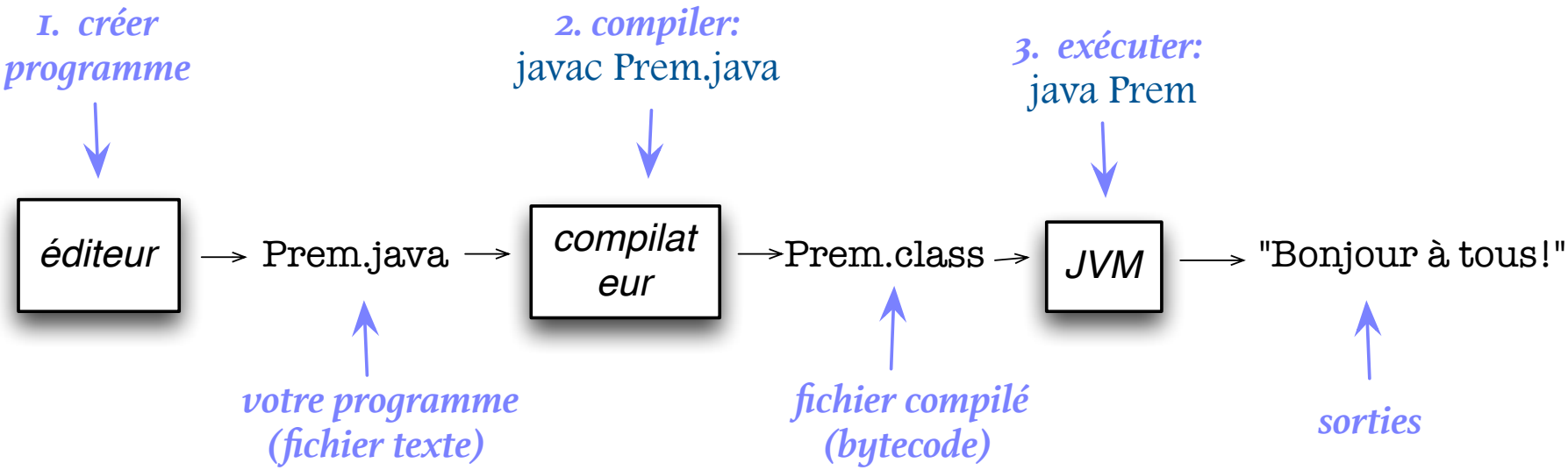
- Instructions en langage machine :
  - instructions bas niveau (binaire) «comprises» par le processeur,
  - taillées pour chaque plateforme matérielle (et non transposable vers une autre)
- fichier en langage machine : code objet ou binaire, exécutable,...
- exécutable uniquement par le type de processeur pour lequel il a été conçu :
  - un fichier `.exe` ne peut s'exécuter que sur une machine type PC sous Windows.

Avant exécution ⇒ Compiler

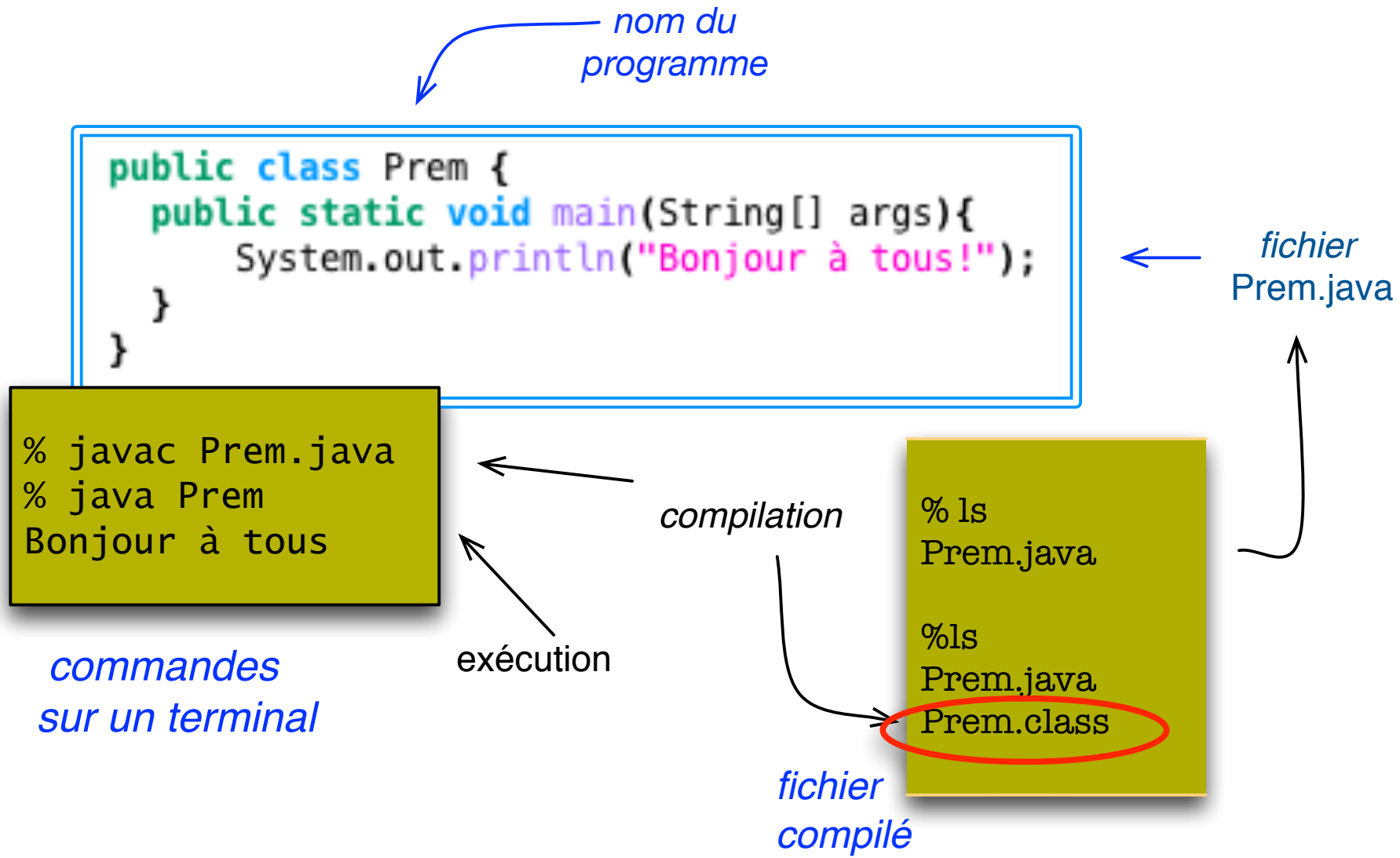
Traduire code source vers du code machine.

Du source à l'exécution

1. Création du programme source ⇒ fichier `Prem.java`
2. Traduction (compilation) du code source ⇒ fichier `Prem.class`
3. Exécution ⇒ sorties du programme.



Du source à l'exécution (suite)



Compilation (du code source)

Compilateur

Programme traduisant un fichier avec le code source vers un nouveau fichier contenant...

- du code exécutable : en binaire, exécutable par le processeur, **ou**
- du bytecode : instructions d'assez bas niveau, appelé aussi «code intermédiaire», non exécutable directement par le processeur, mais assez proche du langage machine.

Le compilateur Java (commande javac)

- la commande `javac` suivie du nom de fichier à compiler :
  - `javac Prem.java`
- produit un nouveau fichier contenant du *bytecode* :
  - fichier avec code compilé : `Prem.class`
  - il contient du texte (non exécutable)
- Ce fichier est recquis par la commande d'exécution (`java`)
  - l'environnement d'exécution JVM se chargera de le traduire en langage machine, puis de l'exécuter.

Java Virtual Machine (JVM)

JVM : environnement d'exécution des programmes Java.

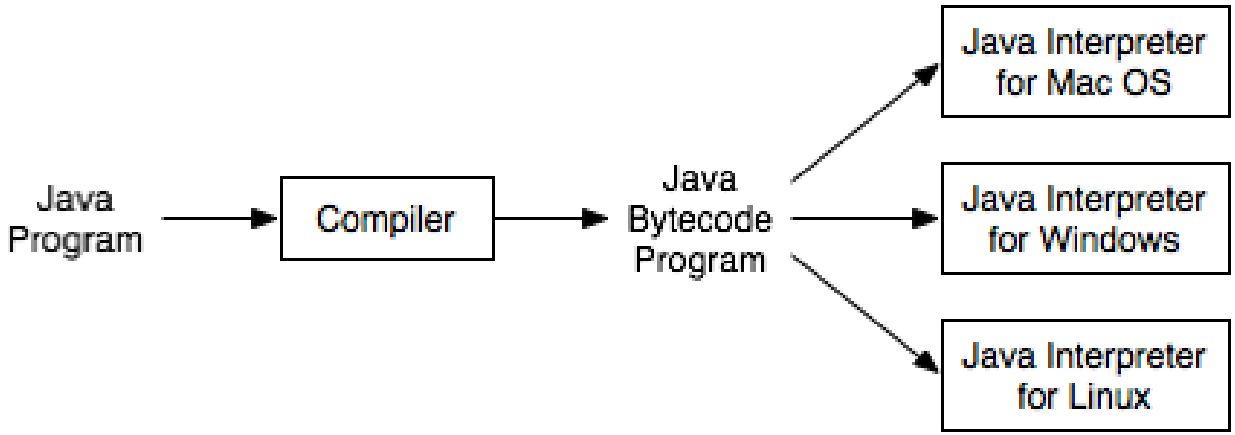
1. Invoqué via la commande `java Prem` (répertoire courant)
2. La comande utilise un fichier compilé (devant exister) de nom `Prem.class` :
   1. traduit chaque instruction (bytecode) dans ce fichier vers du langage machine,
   2. puis l'exécute inmédiatement (à la volée).

Portabilité de Java

- Un même fichier en bytecode est exécutable sur divers types de machines.
- Il suffit de le transmettre (p.e., à travers le réseau) sur une machine avec une JVM.
- La compilation est la phase la plus compliquée de la traduction. Les machines qui exécutent n'ont pas besoin de posséder un compilateur : juste un interprète, qui prend moins de place.
- Les programmes Java deviennent **portables** : exécutables partout, transmissibles par le réseau, faciles à embarquer.

Portabilité de Java (2)

Un programme Java compilé dans une machine X produit du bytecode exécutable par n'importe quelle machine Y qui possède une JVM.



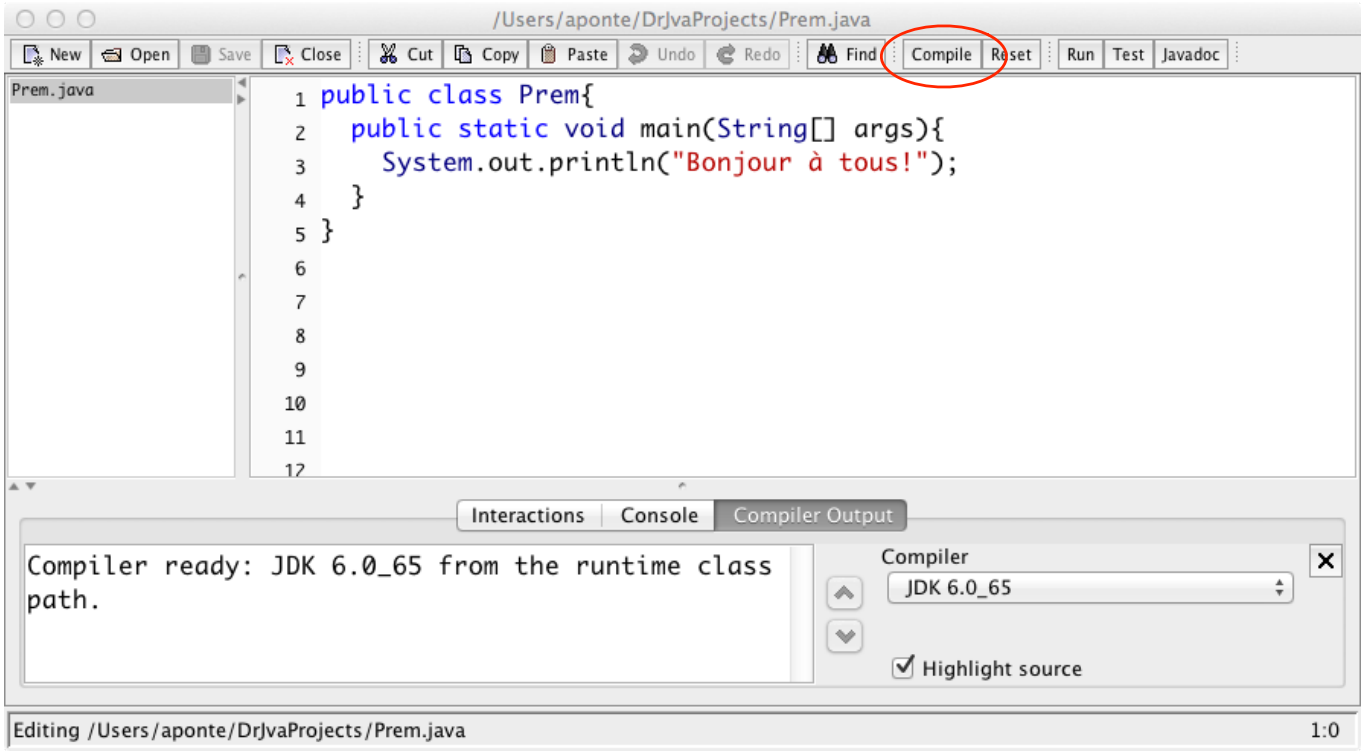
Partie II : utiliser DrJava

DrJava

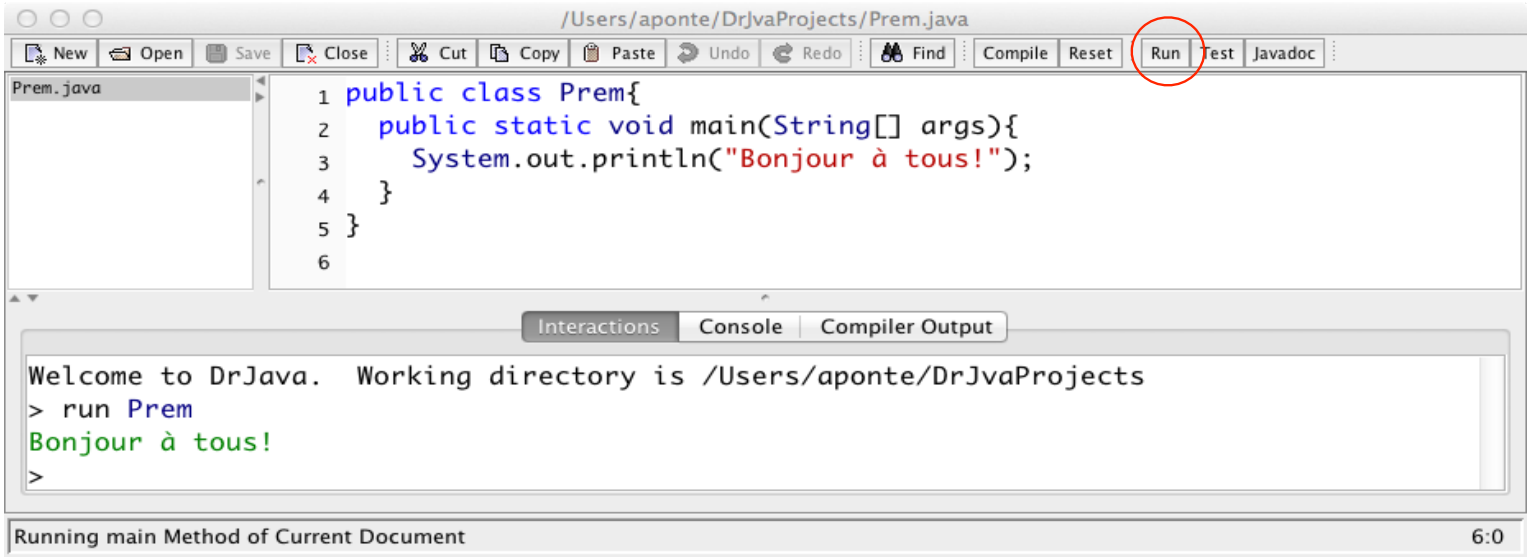
Environnement de développement (IDE) dédié à Java :

- Editeur avec reconnaissance de syntaxe ;
- Compilateur + JVM integrés ;
- Panneau d'interactions, travail par projet et par fichier.

Compilation Prem avec DrJava



Exécution Prem avec DrJava



Partie III : Erreurs dans les programmes

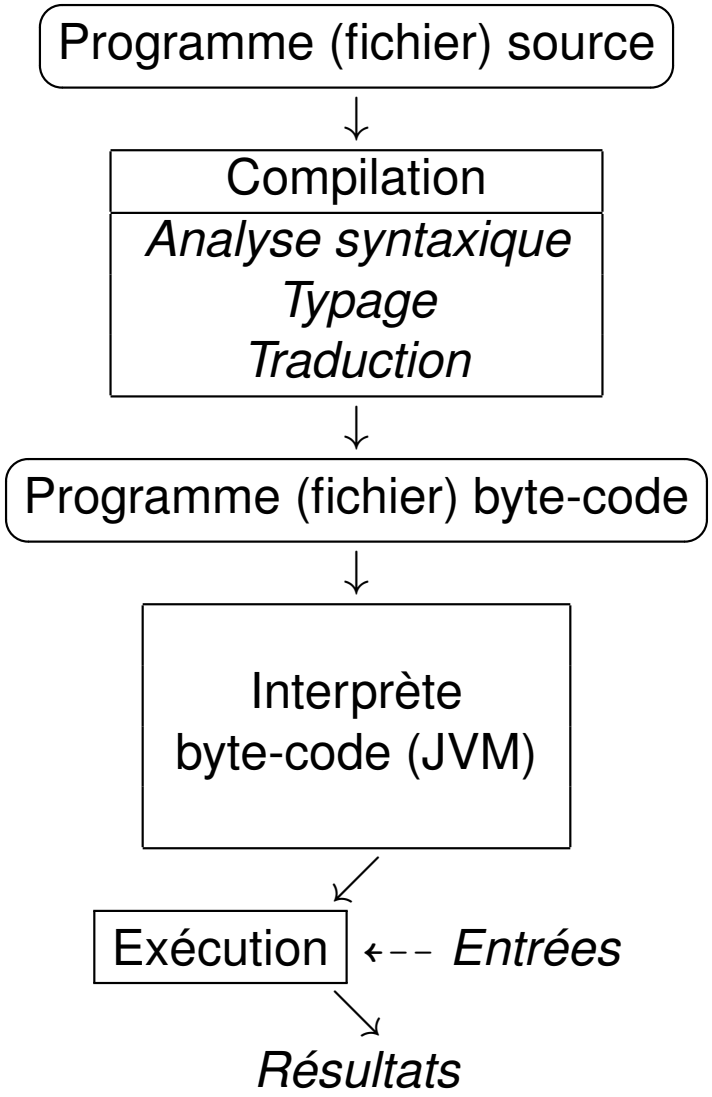
Trois sortes d'erreur, selon le moment où ils surviennent :

1. A la compilation : notre fichier ne correspond pas à du Java correcte : point-virgule mal placé, mot clé mal orthographié, déclaration oubliée... $\Rightarrow$ le compilateur refuse de produire un fichier compilé ;
2. D'exécution : le programme se termine anormalement avec une erreur (fatale) : il a exécuté une action invalide ;
3. De conception : le programme se termine sans erreur fatale, mais ne se comporte pas comme attendu.

Erreurs à la compilation

- de syntaxe :
  - mauvaise ortographe de symboles ou des mot-clés. Ex : `publique` au lieu de `public`
  - non respect des règles de grammaire. Ex : `(3 * / 7)` n'est pas une expression arithmétique correcte.
- de sémantique : la syntaxe est correcte, mais il est impossible de donner un sens à ce qui est formulé.
  - erreur de typage `3 * "bonjour"` : la multiplication s'applique bien sur deux opérandes, mais elles doivent être toutes les deux *de type* numérique ;
  - `3 * x` ne peut être calculé si la variable x n'a pas été définie.

Etapes de traitement de programmes (en détail)



Démonstration

On introduit des erreurs de compilation et d'exécution dans `Prem.java`.

Partie IV : Les algorithmes

algorithme description d'une méthode de résolution d'un problème par le calcul ou par la mise en oeuvre d'une solution.

langage en français, pseudo-java, ou pseudo-mathématique.

les données entrées initiales de l'algorithme + sorties ou solutions .

Les algorithmes (suite)

entrées les données sur lesquelles les opérations de l'algorithme vont s'appliquer ;

sorties c'est le résultat à calculer ou le comportement attendu.

formulation décrit *comment* (en partant des entrées)

- *(quoi)* calculer les sorties ou mettre en oeuvre les comportements
- *(comment)* par le bias d'une suite d'actions/calculs à exécuter *mécaniquement* par un calculateur (humain ou machine)

Exemple : calcul de l'impôt

La déclaration d'impôts sur le revenu vient avec une notice décrivant les opérations à réaliser pour calculer le montant de l'impôt :

- problème : calculer le montant de votre impôt
- entrées : montants des salaires, charges, abattements, etc.
- algorithme : la suite des calculs à réaliser sur les entrées.
- résultat ou sortie : le montant de impôt calculé.

Exemple : recette de cuisine

Problème : Préparation d'une omelette

Entrées : Ce sont les ingrédients,

- 2 oeufs,
- sel,
- un peu de matière grasse

Sorties : Une omelette pour 1 personne.

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.

Exemple : conversion euros/francs

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

Les données :

Entrées : un nombre réel x saisi au clavier,
Sorties : un réel z tel que $z = x * 6.559$

Algorithme :

1. lire la valeur saisie pour x,
2. calculer $z = x * 6.559$
4. afficher le résultat final z

Produire des programmes

1. **Analyse** : Énoncé détaillé du problème $\Rightarrow$ on obtient un cahier des charges.
2. **Conception** : Choix de représentation des données (nombres entiers, réels, tableaux, bases des données ?), puis conception d'une méthode de résolution
   on obtient $\Rightarrow$ Données + Algorithme.
3. **Codage** : L'algorithme est traduit dans un langage de programmation $\Rightarrow$ on obtient le programme source.
4. **Mise au point** : Compilation, tests, correction d'erreurs.
5. **Évolution** On fait évoluer le programme par des améliorations, corrections ou extensions.

Petite méthodologie de conception de programmes

1. Formaliser l'interface du futur programme :
   - Déterminer les *informations nécessaires* pour résoudre le problème. Ce sont les *données* ou encore les *entrées* du programme.
   - Déterminer ce que l'on veut calculer/comportement attendu. Ce sont les *résultats* ou *sorties* du programme
2. Produire plusieurs configurations qui illustrent le comportement attendu : *pour l'entrée x, le programme doit calculer y*. Faire une table avec.
3. Ecrire l'algorithme : suite de pas pour obtenir le résultat attendu en partant des entrées.
4. Coder cet algorithme en Java.

Conception pour conversion euros/francs

Énoncé du problème : Calculer et afficher la conversion en francs d'une somme en euros, entrée au clavier.

Interface

1. entrées : un nombre réel x saisi
2. sorties : un réel z tel que $z = x * 6.559$

Configurations d'exemples

somme en euros (x)	conversion en francs (z)
0.0	0.0
15.0	98.35
-3.5	-22.9565
10	65.59

Conception conversion euros/francs (suite)

Rappel *Interface*

1. entrées : un nombre réel x saisit
2. sorties : un réel z tel que $z = x * 6.559$

Algorithme

1. lire la valeur saisie pour x,
2. calculer $z = x * 6.559$
4. afficher le résultat final z

Codage en Java

```
public class Conversion {
 public static void main (String[] args) {
   double x, z;
   Terminal.ecrireStringln("Somme␣en␣euros?␣");
   x = Terminal.lire_double();
   z = x * 6.559;
   Terminal.ecrireStringln("En␣francs:␣"+ z);
  }
}
```

Tester un programme

1. Élaborer un jeu de tests représentatif de tous les cas possibles des entrées et sorties attendues.

   On pourra se servir de la table de configurations d'exemples élaborée pendant la conception !

2. Tester (à la main) le fonctionnement de l'algorithme à l'aide du jeu de tests.

Suite et fin de Conversion

Le programme est mis dans le fichier `Conversion.java`.

```
Java/Essais> javac Conversion.java
Java/Essais> java Conversion
Somme en euros?
10
En francs: 65.59
```

Tests : Nous répetons plusieurs fois l'exécution avec différents valeurs des test.

```
Java/Essais> java Conversion
Somme en euros?
-3
En francs: -19.677
Java/Essais> java Conversion
Somme en euros?
15.5
En francs: 101.6645
```