

# Formation Java

Introduction au langage Java et à WPILib

Étienne Beaulac  
Ultime FRC 5528

Dernière modification  
5 octobre 2017





# Table des matières

<b>Table des extraits de code</b>	<b>ii</b>
<b>Table des figures</b>	<b>iii</b>
<b>I Les bases de Java</b>	<b>1</b>
<b>1 Introduction au Java</b>	<b>2</b>
1.1 Les langages de programmation . . . . .	2
1.2 Qu'est-ce que le Java ? . . . . .	2
<b>2 Votre premier programme</b>	<b>4</b>
2.1 L'IDE Eclipse . . . . .	4
2.2 Création du projet . . . . .	5
2.3 Les instructions . . . . .	7
2.4 Les chaînes de caractères . . . . .	7
2.5 La méthode <code>println</code> . . . . .	7
2.6 L'indentation . . . . .	7
2.7 Les commentaires . . . . .	8
2.7.1 Les commentaires standards . . . . .	8
2.7.2 Les commentaires Javadoc . . . . .	9
<b>3 Variables et opérateurs</b>	<b>11</b>
3.1 La déclaration de variables . . . . .	11
3.2 Lire la console . . . . .	13
3.3 Les variables de type primitif . . . . .	14

## Table des extraits de code

2.1	Programme de base . . . . .	6
2.2	Programme de base avec commentaires . . . . .	8
2.3	Ajout de commentaires Javadoc . . . . .	9
3.1	Utilisation d'une variable String . . . . .	12
3.2	DemanderNom.java . . . . .	13

# Table des figures

1.1	Le processus de compilation. . . . .	2
2.1	L'interface principale de Eclipse. . . . .	5
2.2	Compiler, exécuter et déboguer un programme avec Eclipse. . . . .	6
2.3	Écriture dans la console. . . . .	6
2.4	Visualisation de la Javadoc dans Eclipse . . . . .	9
3.1	Une variable contenant l'âge de l'utilisateur en mémoire. . . . .	11

# **Première partie**

## **Les bases de Java**

# Chapitre 1

## Introduction au Java

### 1.1 Les langages de programmation

La programmation, en somme, est l'art de formuler ses algorithmes de manière à les faire comprendre à un ordinateur (Ada Lovelace, vers 1840<sup>1</sup>). Cependant, à la base, les ordinateurs ne comprennent que le binaire (Alan Turing, 1936<sup>2</sup>). Pour se simplifier la vie, les informaticiens ont créé des langages intermédiaires qui font le pont entre nous et les ordinateurs (Grace Hopper, 1951<sup>3</sup>). Tous les langages de programmation ont le même but : vous permettre de parler à un ordinateur plus simplement qu'en binaire.



FIGURE 1.1 – Le processus de compilation.

### 1.2 Qu'est-ce que le Java ?

Le langage Java a été créé , entre autres, par James Gosling, Patrick Naughton et Mike Sheridan, tous les trois employés chez *Sun Microsystems* dans les années 1990. Sa première version parut en 1995. Java est maintenant propriété de *Oracle Corporation*.

---

1. On attribue à Ada Lovelace, mathématicienne britannique, la création des premiers programmes informatiques. Ils furent conçus pour être exécutés sur la machine analytique de William Babbage, entièrement mécanique.

2. Alan Turing, mathématicien, cryptologue et logicien britannique, formalisa en 1936 le concept mathématique de *machine de Turing*.

3. Grace Hopper, informaticienne et *rear admiral (lower half)* de l'armée américaine, conçut en 1951 *A-0 System*, le premier compilateur pour ordinateur.

Java est un langage presque entièrement **orienté objet**. Il reprend une grande partie de la syntaxe du C/C++, tout en y ajoutant certaines fonctionnalités : une librairie standard très complète, la réflexivité, les expressions lambdas, l'*autoboxing* et l'*unboxing*, les interfaces, et plusieurs autres. Toutefois, les pointeurs et l'héritage multiple ne sont pas supportées. Ils ajouteraient une trop grande complexité au langage, alors que le but de Java est d'être simple, sécuritaire et robuste.

Le Java compte un nombre impressionnant d'utilisateurs. Une de ses forces est d'ailleurs sa portabilité. Tout programme Java, une fois compilé en *bytecode*, peut fonctionner sur n'importe quelle machine, tant qu'une machine virtuelle Java (JRE, ou *Java Runtime Environment*) y est installée.



# Chapitre 2

## Votre premier programme

### 2.1 L'IDE Eclipse

Pour programmer, il est préférable d'utiliser un bon environnement de développement (**IDE**, ou *Integrated Development Environment*). De tels logiciels comprennent un **éditeur de texte**, un **compilateur** et un **débogueur**. Nous utiliserons l'IDE Eclipse<sup>1</sup> avec l'extension WPILib fournie par FIRST.

Eclipse est disponible gratuitement sur [eclipse.org](http://eclipse.org). Vous devrez également vous assurer d'avoir installé une version récente du **JDK** (*Java Development Kit*). Les étapes d'installation sont également détaillées [ici](#).

Eclipse est un logiciel ayant plusieurs fonctionnalités. On peut d'ailleurs lui en ajouter à l'aide d'extensions (*plugins*), comme celle que nous utiliserons pour développer sur le roboRIO. Voici les fenêtres qui nous intéresseront le plus :

<b>Package Explorer</b>	Cette fenêtre regroupe tous vos projets, subdivisés en dossiers et paquetages ( <i>packages</i> ), jusqu'aux fichiers Java.
<b>Fenêtre d'édition</b>	Cette fenêtre affiche tous les fichiers que vous êtes en train d'éditer, vous permettant facilement de naviguer entre différents documents.
<b>Problems</b>	Comme son nom l'indique, on y retrouve une liste de tous les avertissement et erreurs concernant votre code. Chaque item précise la nature de l'erreur et où elle se trouve.
<b>Console</b>	La console est un outil essentiel, c'est le premier lien entre vous et l'exécution de votre programme. Vous pourrez y afficher du texte et en insérer.
<b>Javadoc</b>	Java a l'avantage de fournir son propre outil de documentation. Il suffit de cliquer sur un mot (classe, variable, méthode, etc.) et sa description y apparaîtra. Nous verrons plus loin comment créer ses propres entrées pour Javadoc.

Évidemment, toute l'interface est entièrement personnalisable. À vous de l'adapter comme il vous plaira !

---

1. Pour plus d'information concernant Eclipse, consultez ...

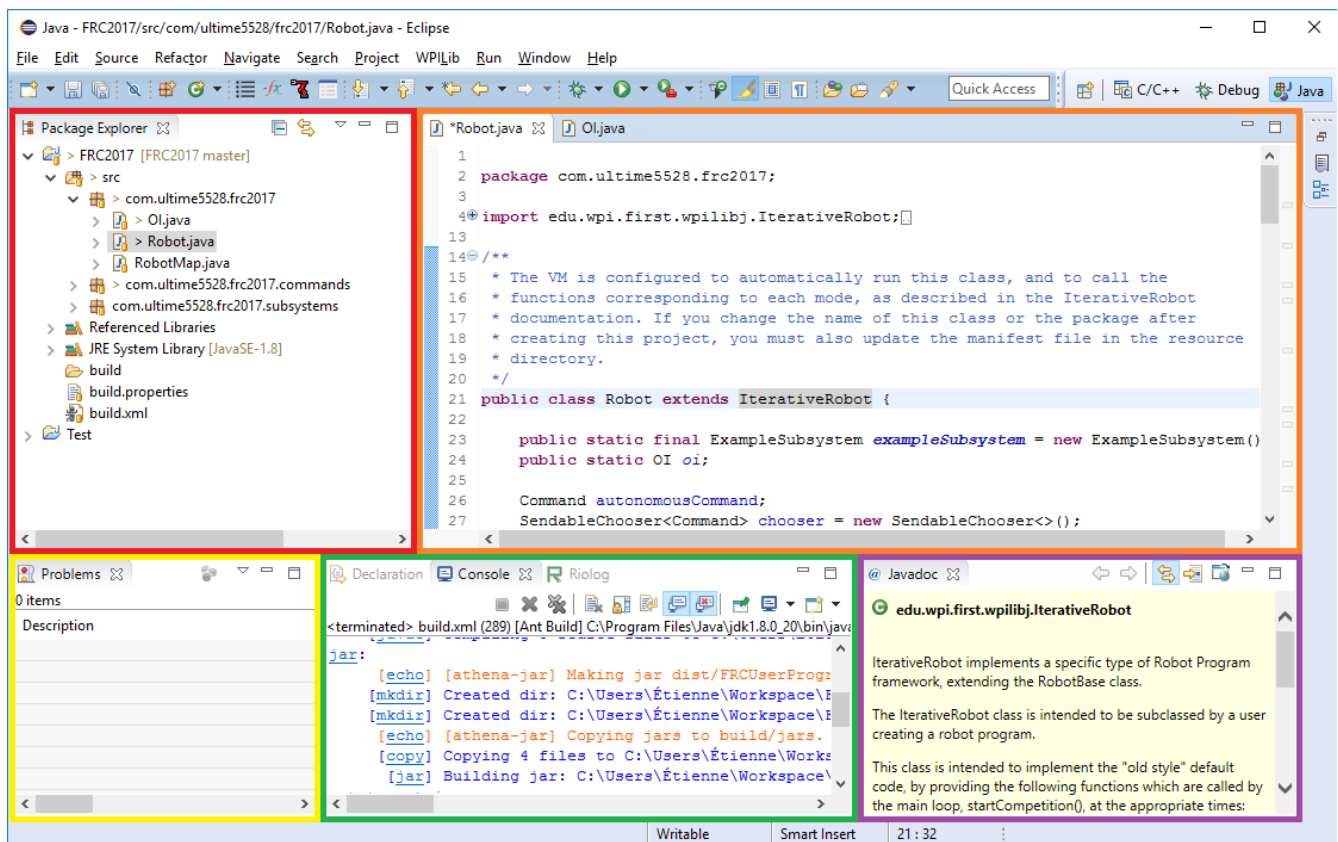


FIGURE 2.1 – L'interface principale de Eclipse.

## 2.2 Création du projet

1. Dans Eclipse, créez un nouveau projet avec **File > New > Java Project** . Donnez un nom à votre projet, puis cliquez sur **Finish** .
2. Ajoutez une classe à votre projet : **Clic droit sur votre projet > New > Class** . Donnez un nom à votre classe, cochez l'ajout de la méthode **main** , puis cliquez sur **Finish** .
3. Complétez le corps de la méthode avec l'exemple suivant, puis compilez et exécutez votre programme.

## CODE 2.1 — Programme de base

```
1 public class MonPremierProgramme {  
2  
3     public static void main(String[] args) {  
4  
5         System.out.println("Hello, world");  
6  
7     }  
8  
9 }
```

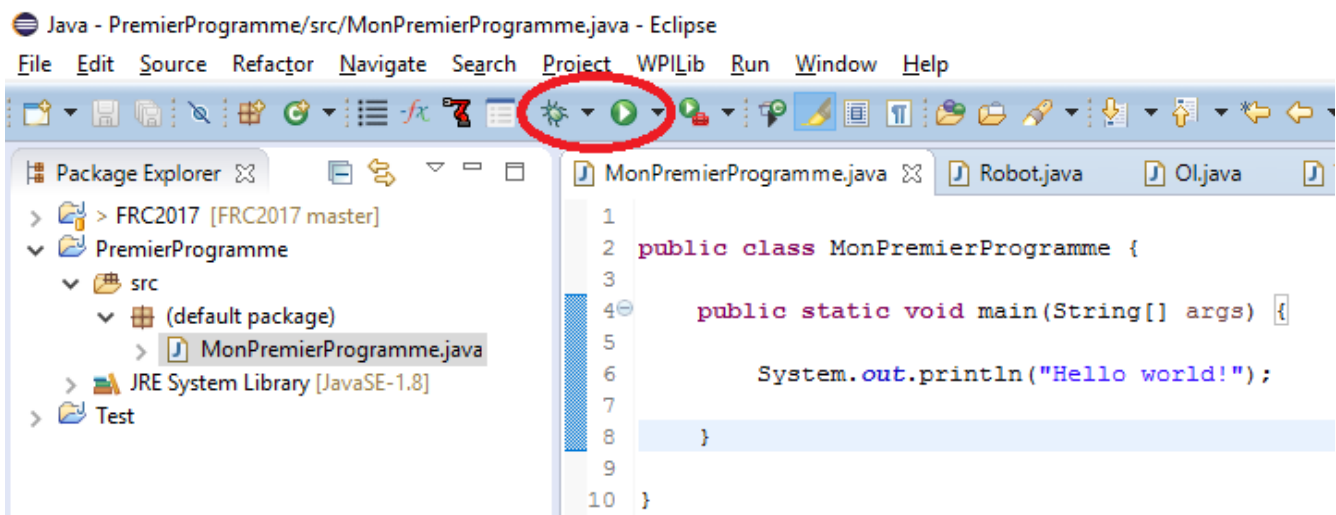




FIGURE 2.2 – Compiler, exécuter et déboguer un programme avec Eclipse.

Le bouton  vous permet de lancer votre programme en mode débogage. La flèche verte , quant à elle, compile et exécute. Après avoir cliqué dessus, vous devriez voir apparaître du texte dans votre console.

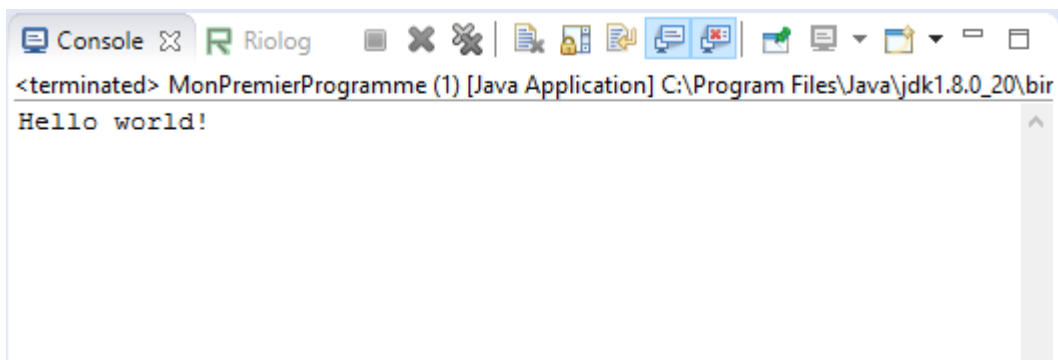


FIGURE 2.3 – Écriture dans la console.

Félicitations, vous venez d'exécuter votre premier programme ! Décortiquons en détails ce qu'il se passe à l'intérieur.

## 2.3 Les instructions

En Java, une **instruction** est une commande effectuant une certaine action. On écrit une instruction par ligne, et chacune se termine toujours par un **point-virgule** (;). Pour l'instant, votre programme ne contient qu'une instruction :

```
System.out.println("Hello, world!");
```

Vos instructions sont écrites dans la méthode `main`. En Java, tous les programmes ont une méthode `main`. Il s'agit, en quelque sorte, du point d'entrée du programme.

## 2.4 Les chaînes de caractères

Le rôle de votre programme est d'afficher du texte dans la console. Vous avez sûrement remarqué que le texte à afficher est encadré de guillemets anglais ("..."), mais qu'ils n'apparaissent pas dans la console. Ils sont essentiels pour que le compilateur fasse la différence entre du code et du texte. On les appelle des **chaînes de caractères**, ou ***String*** en anglais. Essayer de modifier le texte entre les guillemets et d'exécuter votre programme : vous constaterez que la chaîne de caractères affichée dans la console s'est modifiée !

On peut joindre plusieurs chaînes de caractères ensemble avec l'opérateur `+`. Cette opération s'appelle la **concaténation**. On peut donc écrire :

```
System.out.println("Bonjour " + "à tous" + " et à toutes" + "!");
```

## 2.5 La méthode `println`

En Java, une **méthode** est une instruction qui réalise une opération prédéfinie. On utilise une méthode en écrivant son nom suivi d'une paire de parenthèses. Certaines méthodes ont besoin de paramètres pour effectuer leur travail. C'est le cas de la méthode `System.out.println`, qui demande un *String* en paramètre. Elle s'occupe ensuite de l'afficher sur la console.

## 2.6 L'indentation

Dans l'exemple précédent, vous pouvez constater qu'à chaque fois que des accolades (...) sont ouvertes, on ajoute de l'espace au code qui se situe à l'intérieur. C'est ce que l'on appelle l'**indentation**

du code. C'est essentiel pour rendre le code clair et facile à modifier. Pour indenter son code, on ajoute une tabulation (touche `Tab ⇌`) pour chaque paire ouverte d'accolades. Eclipse s'en occupe automatiquement la plupart du temps.

## 2.7 Les commentaires

### 2.7.1 Les commentaires standards

Lors de l'écriture, il est possible de spécifier au compilateur de ne pas compiler certaines parties du code. C'est ce qu'on appelle les **commentaires**. Ils permettent de spécifier l'utilité des variables, des méthodes, des classes, etc. Il est crucial d'en ajouter, surtout lors d'un projet en collaboration avec plusieurs personnes !

CODE 2.2 — Programme de base avec commentaires

```
1  /*
2   * La classe suivante affiche un message
3   * dans la console.
4   */
5  public class MonPremierProgramme {
6
7      /* Fonction principale
8       du programme.          */
9      public static void main(String[] args) {
10
11          //Début du programme
12
13          System.out.println("Hello, world"); //Affichage du message
14
15      }
16
17 }
```

Les plus courants sont les **commentaires en fin de ligne**. Ils débutent par deux barres obliques `//`. Ils informent le compilateur d'ignorer tout le reste de la ligne. Ils sont souvent courts et précis. On les utilise pour mettre en contexte une instruction ou en début de section.

Pour de longs commentaires, on utilise les **commentaires en blocs**. Ils débutent par `/*` et se terminent par `*/`. Le compilateur ignore alors tout ce qui se trouve entre ces deux balises, un peu comme des parenthèses. On les utilise, entre autres, en entête de fichier, pour spécifier le rôle du fichier (ou de la classe), les noms des auteurs et les dates de création et de modification.

Il est important de mettre des commentaires, mais il ne faut pas en abuser (comme dans l'exemple précédent). Il suffit de trouver le juste équilibre entre clarté et concision. Il est également essentiel

de mettre en contexte l’instruction.

Bon commentaire :

```
age += 1; // L'utilisateur vieillit d'un an.
```

Mauvais commentaire :

```
age += 1; // Ajout de 1 à la variable age.
```

## 2.7.2 Les commentaires Javadoc

Ces commentaires spéciaux sont propres au Java. Ils permettent de créer une documentation accessible pour votre projet. Ils sont très semblables aux commentaires en blocs : il suffit de les faire débuter avec deux étoiles `/**` . Vous aurez donc accès au contenu de votre commentaire partout dans votre projet, sans devoir ouvrir à nouveau le fichier d’origine !

### CODE 2.3 — Ajout de commentaires Javadoc

```
1  /**
2   * Ceci est un commentaire Javadoc!
3   * @author Etienne
4   *
5   */
6  public class MonPremierProgramme { ... }
```

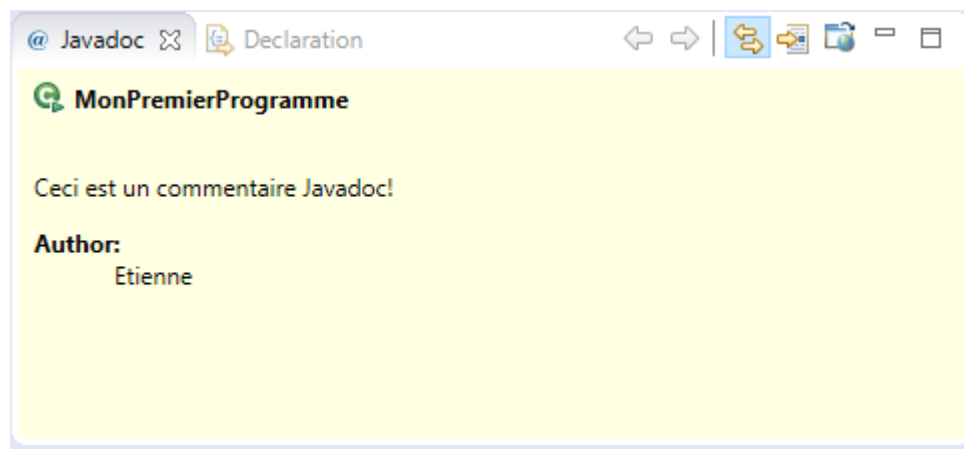


FIGURE 2.4 – Visualisation de la Javadoc dans Eclipse

La Javadoc possède plusieurs attributs spéciaux débutant par un arrobre `@` . Les exemples de ce guide feront appel aux trois attributs suivants.

**@author** On l’utilise dans l’entête d’une classe pour en spécifier l’auteur.

**@param** Dans l'entête de méthodes, il précise le rôle de chaque paramètre.

**@return** Également dans l'entête de méthodes, il précise la valeur de retour.

# Chapitre 3

## Variables et opérateurs

### 3.1 La déclaration de variables

Une **variable** est une case mémoire pouvant contenir un certain type de données. Comme son nom l'indique, sa valeur est *variable* : elle peut changer au cours l'exécution.

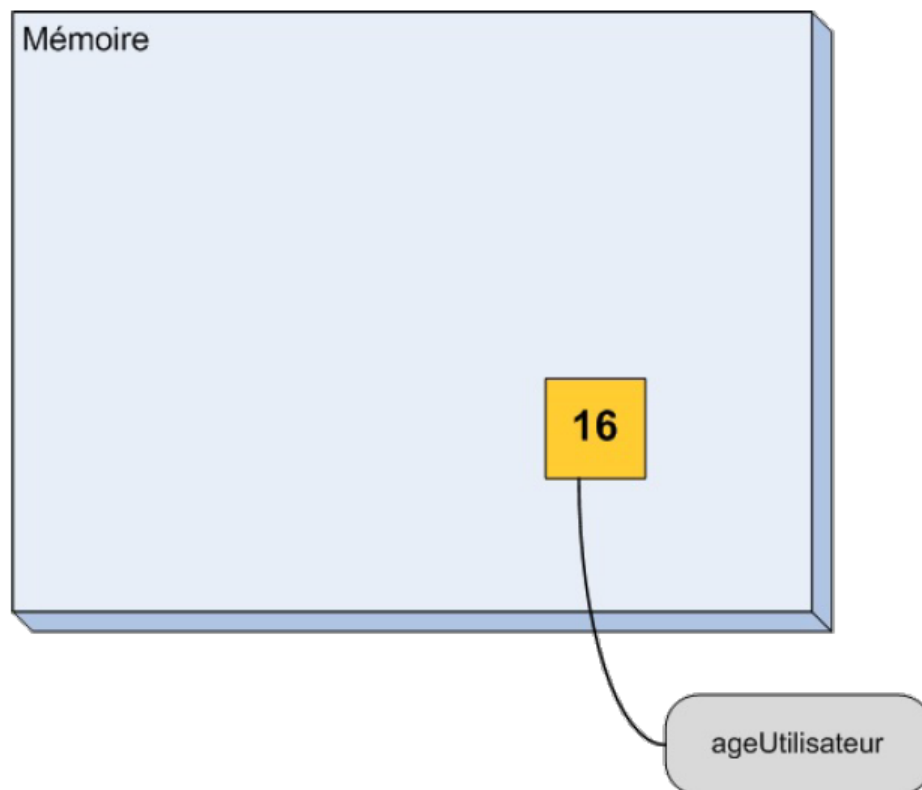


FIGURE 3.1 – Une variable contenant l'âge de l'utilisateur en mémoire.

Pour commencer, regardons un programme utilisant une variable de type *String*.



### CODE 3.1 — Utilisation d'une variable String

```
1  /**
2   * Affiche des noms dans la console.
3   *
4   * @author Etienne
5   */
6  public class AffichageNom {
7
8      public static void main(String[] args) {
9
10         String nom = "Étienne"; // Nom de l'utilisateur
11
12         System.out.println("Je m'appelle " + nom + "!"); //Affichage
13
14         nom = "Alexandre"; //Nouvelle valeur
15
16         System.out.println("Je m'appelle maintenant " + nom + "!");
17         ↪ //Affichage de la nouvelle valeur
18     }
19
20 }
```

#### Sortie console

```
Je m'appelle Étienne!
Je m'appelle maintenant Alexandre!
```

On commence par créer la variable `nom` de type *String* et on lui donne la valeur "Étienne". On affiche ensuite la valeur de `nom` dans la console. À la troisième instruction, on met la valeur "Alexandre" dans `nom`. L'ancienne valeur est alors **écrasée** par la nouvelle. La dernière affiche alors la nouvelle valeur de `nom` dans la console.

#### Déclaration et initialisation de variables

Déclaration et initialisation dans la même instruction

```
type nomVariable = valeur;
```

Déclaration, puis initialisation plus tard

```
type nomVariable;
```

```
...
```

```
nomVariable = valeur;
```

Lorsque c'est possible, on déclare et on initialise une variable en même temps. C'est ce qui a été

fait dans l'exemple précédent. Lorsqu'on ne connaît pas quelle valeur lui donner, on peut la déclarer et lui donner une valeur plus tard.

On peut donner n'importe quel nom à une variable, tant qu'il respecte les conditions suivantes :

- pas d'espace ni d'accent ;
- ne commence pas par un chiffre ;
- commence par une minuscule ;
- si son nom est composé de plusieurs mots, les autres mots peuvent commencer par une majuscule.

Par exemple, les identificateurs `prix`, `ageUtilisateur`, `vitesseMoteurGauche1` et `estOuvert` respectent cette convention.

## 3.2 Lire la console

Vous savez déjà comment afficher du texte dans la console avec la méthode `System.out.println()`. Par contre, il pourrait être pratique de lire ce qui est écrit dans la console. Pour effectuer cette tâche, nous utiliserons la class `Scanner` de la manière suivante.

CODE 3.2 — DemanderNom.java

```
1  import java.util.Scanner;
2
3  /**
4   * Demande le nom de l'utilisateur, puis l'affiche.
5   *
6   * @author Etienne
7   */
8  public class DemanderNom {
9
10     public static void main(String[] args) {
11
12         String nom;
13         Scanner scanner = new Scanner(System.in);
14
15         //Demander le nom
16         System.out.print("Saisissez votre nom : ");
17         nom = scanner.nextLine();
18
19         //Affichage
20         System.out.println("Votre nom est " + nom + "!");
21
22     }
23
24 }
```

Ici, on déclare une variable sans l'initialiser. C'est tout à fait logique, car on ne connaît pas encore le nom à afficher. On déclare ensuite une variable spéciale : la variable `scanner` de type `Scanner`. C'est elle qui va nous permettre de lire les entrées dans la console. Remarquez son initialisation : on utilise le `new` suivi de `Scanner`, le type de notre variable. Nous verrons plus loin que c'est parce que `scanner` est un **objet**, une sorte de « super-variable ».

Par la suite, on utilise une variante de `println()` : la méthode `print()`. Elles agissent presque de la même façon, sauf que `print()` n'ajoute pas de saut de ligne après avoir affiché le texte. Essayez les deux et constatez la différence.

Ensuite, on utilise notre `scanner` et on appelle sa méthode `nextLine()`. Cela indique au programme de faire une pause jusqu'à ce qu'on écrive un mot dans la console et qu'on appuie sur la touche `Entrée`. Le texte saisi est ensuite stocké dans la variable `nom` grâce à l'opérateur `=`.

Finalement, on affiche la valeur de `nom` par concaténation avec d'autres chaînes de caractères.

### 3.3 Les variables de type primitif

Jusqu'à présent, nous avons rencontré