*Démarrez votre programme avec Java*

**Chapitre 1 : Débutez avec Java**

1. **Découvrez l’univers Java :**

Java a été développé par Sun Microsystems en 1990 dans le but d’avoir un langage beaucoup plus simple que le C++ mais surtout pouvant tourner sur n’importe quelle machine. Le développeur n’a, par exemple, plus besoin de gérer la mémoire de son programme ce qui en plus de faciliter les choses évite de nombreuses erreurs potentielles. Java doit principalement son succès à l’arrivée d’internet et du web pour lequel il proposé un développement facilité (les applets). Aujourd’hui, c’est Oracle qui est depuis 2009 responsable du développement de Java.

Java est toujours utilisé dans le web (avec d’autres outils que les applets qui ne sont eux plus utilisés) mais aussi dans d’autres secteurs tels que le big-data, le mobile ou même les serveurs de jeux en ligne et reste même le langage le plus rependu le plus rependu en entreprise pour plusieurs raisons :

* Son grand succès dans les années 90
* Les outils variés, riches et mature dont il dispose
* Sa très grande communauté
* Sa facilité de prise en main
* Il est destiné aux gros projets et pas aux petites applications ou simples scriptes

**/!\ Attention :** Quand on parle de Java on ne parle pas que du langage mais aussi de tout l’écosystème associé et notamment les technologies Microsoft .NET ou même d’autres langages tels que PHP, NodeJS, Ruby.

Lorsque l’on développe, dans n’importe quel langage, on écrit du code « humainement compréhensible » qu’il faudra rendre compréhensible pour la machine. Trois cas de figure existent à partir de là :

1. Dans le cas des langages compilés (C++) le code source est directement et en une seule fois transformé en code binaire. Cette étape peut être longue, répétitive (à chaque changement) et l’inconvénient majeure réside dans le fait qu’une compilation existe pour chaque configuration matérielle possible.
2. Dans le cas des langages interprétés c’est un programme qui va directement lire le code source ligne par ligne et le faire effectuer par la machine. On perd grandement en performances car l’ordinateur a beaucoup moins de mal à lire du code binaire directement.
3. Java fait partie des langages intermédiaires (ou pseudo compilés). C’est-à-dire que le code source est partiellement compilé pour donner ce qu’on appelle le « ByteCode ». C’est ce byte code que l’interpréteur (la Java Virtual Machine) va lire. On gagne en performances car une partie du code est déjà compréhensible par la machine.

La JVM est disponible sur Windows, Linux, Mac, Android et d’autres plateforme. Le code développé est ainsi utilisable sur la quasi majorité des machines existantes. La JVM se compose :

* D’un compilateur (passage du code source au ByteCode)
* D’un outil de gestion de mémoire
* D’un optimisateurs de codes
* D’autres fonctionnalités déjà codées
* D’un interprétateur (passage du ByteCode au code machine)

La JVM est tellement puissante est aboutie qu’il est courant de compiler d’autres langages vers du ByteCode Java. Cela rend compatible d’autres langage avec le Java (Jython pour Python par exemple).

1. **Démarrez avec Java :**

Comme dit plus haut, Java est un langage intermédiaire donc dont une partie est compilée (en ByteCode, interprété directement par la JVM) :

1. On écrit le code source dans un fichier.java
2. On le compile grace à la commande « javac fichier.java » (javac pour java compilator)
3. On lance le nouveau fichier crée avec la commande « java fichier.class »
4. **Démarrez avec un IDE :**

Les IDE (pour Integrated Development Environment) sont de puissants éditeurs de code censés aidés les développeurs à développer plus rapidement et simplement grâce à de puissants outils directement intégrés comme :

* Auto complétion
* Vérificateur d’erreur
* Outils de compilation
* Outils de debbuging
* Etc…

L’IDE Java utilisé dans ce cours s’appelle IntelliJ (dans sa version open source) et est développé par le très populaire et performant JetBrain. Les IDE proposent différents outils regroupés en « sous-fenêtres » aussi appelées « vues », par exemple la vue « Projet », ou la vue « Terminale », etc…

Les projets Java sont composés d’un grand nombre de fichier, chacun d’entre eux representant une partie logique de l’application (Home sert à décrire une maison, Calcul pourra contenir diverses opérations de calcul, etc…). Cela se traduit dans l’arborescence du projet par un classement des fichiers en différents sous dossiers, chaque sous dossier étant un package. Deux fichiers peuvent avoir le même nom de fichier sans risquer de conflit si et seulement s’ils appartiennent tous deux à deux packages différents. Cela permet également de faciliter l’import de code source en provenance d’un autre développeur dans notre propre projet.

* Par convention on nomme les package selon un nom de domaine que l’on possède. Par exemple, les packages Java utilisés par Google commencent par « com.google ». Dans notre cas, nous utiliserons la dénomination suivante : « OC.Wals »
* Le code source de notre projet, lui, se trouve dans le dossier « src »

Lorsqu’un programme Java s’exécute il fait abstraction de la notion de fichiers et ne fonctionne qu’avec des fonctions reparties dans des classes. Finalement, chaque fichier contient une classe portant le nom du fichier, qui elle-même peut contenir des fonctions. La fonction lancée en première dans un programme doit s’appeler « main ».

Décortiquons la commande suivante : « System.out.println(‘’Hello World !’’) ; »

* System est une classe (d’où la majuscule à son nom)
* out indique qu’on souhaite utiliser la sortie standard
* println est la fonction correspondant à l’affichage d’un texte donné en argument (entre parenthèses)

Pour exporter un programme Java, il faut le passer au format .jar qui est en fait un simple dossier d’archive contenant toutes les classes du programme et ses fichiers de configuration. Cela passe par la création d’Artifacts dont la procédure est détaillée dans l’actuel chapitre de ce cours (voir section « Comment exporter notre programme » sur le cours online).

1. **Récupérez des valeurs et utilisez-les :**

En Java, contrairement au PHP ou au Javascript, les variables sont typées. Il faut donc les définir elles et leur type avant de pouvoir les utiliser, par exemple : « int age = 26 ; ».

En plus des traditionnels int, double et boolean il existe aussi en Java ce qu’on appelle les variables dites de type « objet complexe ». Le type string en fait par exemple partie, parmi des centaines d’autres déjà intégrés à Java en plus de ceux que l’on peut créer soi-même. Les possibilités sont donc infinies, le type objet pouvant être imagé aux objets (ou concepts) de la vie réelle : Table, mot, voiture, etc…

L’opérateur de concaténation en Java est le symbole « + ».

**Chapitre 2 : Appréhendez les bases de la programmation**

1. **Importez votre premier objet :**

Voir le cours online pour la procédure détaillée d’importation.

La Java doc est une écriture normée servant à donner aux autres développeurs des informations sur le code (rôle d’une variable, fonctionnement d’une fonction, etc…). Elle est normée afin d’être automatiquement extraite du code si besoin est. Les blocs de Javadoc commencent par « /\*\* » et finissent par « \*/ ».

Pour tester des programmes en Java on peut utiliser Junit. Seulement, celui-ci n’est pas directement intégré et il faut donc importer les ce dont on a besoin, par exemple comme suit : « import org.junit.jupiter.api.Test; » ou encore « org.junit.jupiter.api.AfterEach ».

Les fonctions tests sont nommées selon le format Given/When/Then qui divise le test en trois parties :

1. Given : Quels sont les paramètres de fonctions utilisés ?
2. When  : Dans quel contexte ?
3. Then : Quel doit être le résultat ?

Il n’est pas utile de tout tester, le but étant de trouver un compromis entre temps passés sur mes tests et temps passé sur les fonctionnalités.

1. **Ajoutez de la logique à votre programme avec des conditions :**

Lorsque que l’on définit une condition Java va en réalité crée une variable de type boolean et y assigné « true » ou « false » selon le résultat de la condition. Dans le cas d’un « if », Java vérifie simplement si la variable crée vaut « true » ou « false » ce qui permet par exemple d’écrire le bloc « if (true) { code à exécuter ; } » dans lequel nous rentrerons systématiquement.

1. **Structurez votre code avec des fonctions et du TDD :**

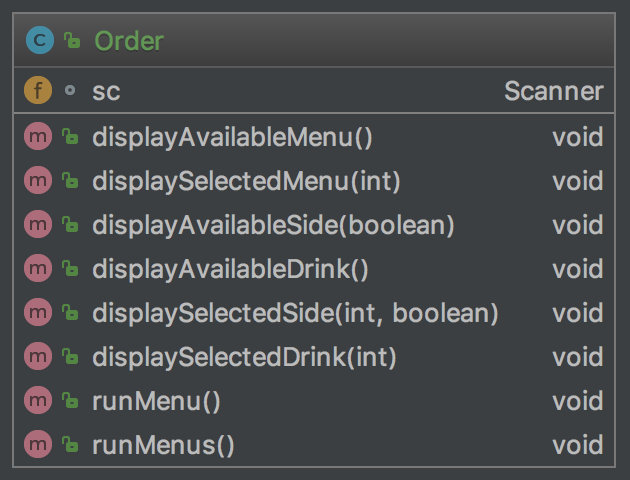
Le TDD pour Test Driven Development est une méthode de développement qui consiste à penser aux tests d’une fonction avant de la coder. Ainsi on s’assure, en théorie, de couvrir tous les cas de figures possibles :

1. Rédiger les tests
2. Exécuter les tests
3. Ecrire le code de la fonction
4. Exécuter à nouveau les tests

On exécute une première fois le test pour être sûr que celui-ci échoue. S’il passe c’est qu’il ne test finalement rien et il sera impossible de savoir, une fois la fonction codée, s’il est pertinent.

1. **Répétez une action avec des boucles :**

* La boucle while : « while (condition d’entrée) { … } »
* La boucle for : « for (int x = 0 ; x < 10 ; x++) { … } »
* La boucle do while : « do { … } while (condition de sortie);

Les variables déclarées à l’intérieur d’une boucle ne sont disponibles qu’à l’intérieur de celle-ci. Il est parfois plus judicieux de déclarer une variable comme attribut d’une classe, afin qu’elle soit partout accessible (par exemple le scanner). Pour savoir s’il faut déclarer une variable en tant qu’attributs, on peut se poser deux questions :

1. La variable représente elle une composante de mon objet ?
2. Vais-je avoir besoin de cette variable dans une seule méthode ou dans plusieurs ?

Les attributs sont aussi représentés dans le diagramme de classe UML.

**Chapitre 3 : Allez plus loin dans la programmation**

1. **Utilisez les tableaux pour organiser vos données :**