CÉGEP
MARIE-VICTORIN

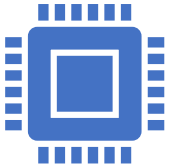
420-210-MV
Programmation orientée objet

Samuel Fostiné

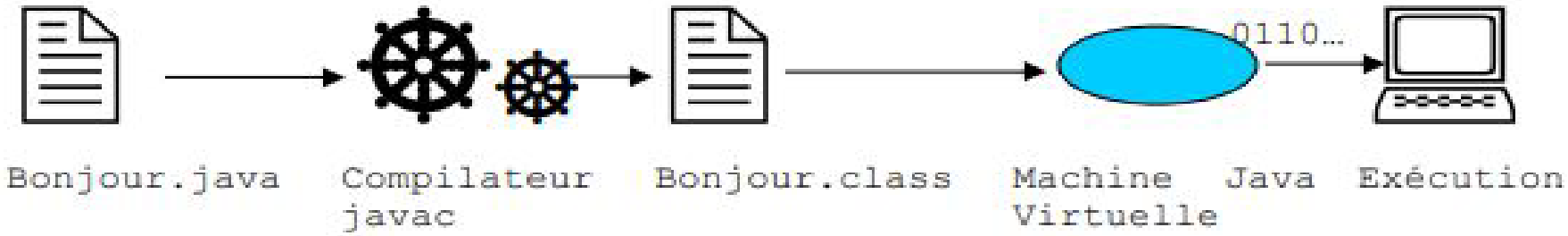
Définition de la portabilité en Java

Une des caractéristiques du langage Java est sa portabilité, c'est-à-dire, qu'un programme développé avec le langage Java fonctionnera automatiquement sous Windows, Mac, Linux, etc.

En programmation Java, le code source est d'abord écrit en texte brut et enregistré avec l'extension .java. Les fichiers sources sont ensuite compilés par le compilateur « javac » pour produire des fichiers portant l'extension « .class ».



Ces fichiers .class peuvent être ensuite exécutés sur votre processeur s'il dispose d'une JVM (Java Virtual Machine - Machine Virtuelle Java), le programme qui interprète le code Java et le convertit en code natif. Illustrons ces étapes de production :



Le JDK, le JRE et le JVM

JDK

- Le JDK (Java Development Kit) est un ensemble d'outils de développement logiciel pour la programmation Java. Il contient tout ce dont un développeur a besoin pour créer, compiler, déboguer et exécuter des applications Java. Le JDK est distribué par Oracle Corporation, qui est la principale organisation qui maintient et développe Java

Composition du JDK

- Java Compiler (javac): Le compilateur Java qui traduit le code source Java en bytecode, un code intermédiaire exécutable par la machine virtuelle Java (JVM). Le compilateur vérifie la syntaxe, les types et effectue des optimisations lors de la compilation.
- Java Virtual Machine (JVM): La JVM est un composant essentiel du JDK. Elle est responsable de l'exécution du bytecode généré par le compilateur Java. La JVM est une machine virtuelle capable de s'exécuter sur différentes plates-formes matérielles.
- Java Runtime Environment (JRE): Le JRE est inclus dans le JDK. Il fournit la JVM et d'autres bibliothèques de classes Java nécessaires à l'exécution des applications Java.
- Bibliothèques Java (Java API): Le JDK inclut une vaste collection de bibliothèques standard, connue sous le nom d'API Java (Application Programming Interface). Ces bibliothèques fournissent des fonctionnalités prêtes à l'emploi pour des tâches courantes telles que la manipulation de chaînes, l'entrée/sortie, la gestion des collections, etc.
- Outils de Développement (javap, jconsole, jvisualvm, etc.): Le JDK contient divers outils de développement, tels que le désassembleur (javap), les outils de surveillance de la performance (jconsole, jvisualvm), et d'autres utilitaires utiles pour le développement et le débogage.
- Outils de Développement (javap, jconsole, jvisualvm, etc.): Le JDK contient divers outils de développement, tels que le désassembleur (javap), les outils de surveillance de la performance (jconsole, jvisualvm), et d'autres utilitaires utiles pour le développement et le débogage.
- JavaFX: Le JDK inclut également JavaFX, une plate-forme pour la création d'applications riches en interface utilisateur (UI) et pour le développement d'applications client riches (RIA).JavaFX: Le JDK inclut également JavaFX, une plate-forme pour la création d'applications riches en interface utilisateur (UI) et pour le développement d'applications client riches (RIA).
- Le JDK propose une gamme d’outils de développement pour faciliter le processus de création d’applications Java. Parmi ces outils figurent le débogueur (jdb) pour identifier et corriger les erreurs ou encore le générateur de documentation (javadoc) pour créer une documentation lisible par les développeurs.

JRE

- Le JRE (Java Runtime Environment) est un environnement d'exécution pour les applications Java. Il fournit les bibliothèques, la machine virtuelle Java (JVM), et d'autres composants nécessaires à l'exécution des programmes Java. Contrairement au JDK (Java Development Kit) qui est utilisé pour le développement, le JRE est destiné à l'exécution des applications Java sur une machine.
- Les utilisateurs finaux, qui souhaitent simplement exécuter des applications Java, ont besoin du JRE sur leur système, tandis que les développeurs, pour créer des applications Java, utilisent le JDK qui inclut le JRE ainsi que des outils de développement supplémentaires.

JVM

- La JVM (Java Virtual Machine) est une machine virtuelle qui fournit l'environnement d'exécution pour les applications Java. Elle joue un rôle essentiel dans l'exécution des programmes Java en permettant la portabilité des applications sur différentes plates-formes matérielles. La JVM interprète ou compile le bytecode Java en code machine spécifique à la plate-forme lors de l'exécution.

Quelques éléments que peut contenir la JVM

Class Loader (Chargeur de classes) : La JVM utilise des class loaders pour charger les classes Java nécessaires à l'exécution d'une application. Les class loaders peuvent charger des classes à partir du système de fichiers, du réseau, ou d'autres sources.

Java Interpreter (Interpréteur Java) : La JVM peut utiliser un interpréteur pour exécuter directement le bytecode Java. L'interpréteur convertit chaque instruction bytecode en code machine au fur et à mesure de l'exécution.

Garbage Collector (Collecteur de déchets) : La JVM inclut un collecteur de déchets qui gère la gestion automatique de la mémoire en libérant les objets qui ne sont plus référencés. Cela simplifie la gestion de la mémoire pour les développeurs Java.

Règles de base en Java

- Java est sensible à la casse, c'est-à-dire qu'il fait la distinction entre les majuscules et les minuscules. Une variable nommée « nom » est différente d'une variable nommée « Nom ».
- Les blocs de code en Java sont encadrés par des accolades { }
- Chaque instruction Java se termine par un point-virgule ;
- Il est utile pour expliquer ce que fait le code de placer des commentaires, en Java on utilise les caractères « // » et « /*…*/ » pour insérer des commentaires. Un commentaire d'une seule ligne commence par une double barre oblique (« // ») et se termine au retour à la ligne.
- Un commentaire sur plusieurs lignes est encadré par une barre oblique suivie d'une étoile (« /* ») et se termine par une étoile et une barre oblique (« */ »).

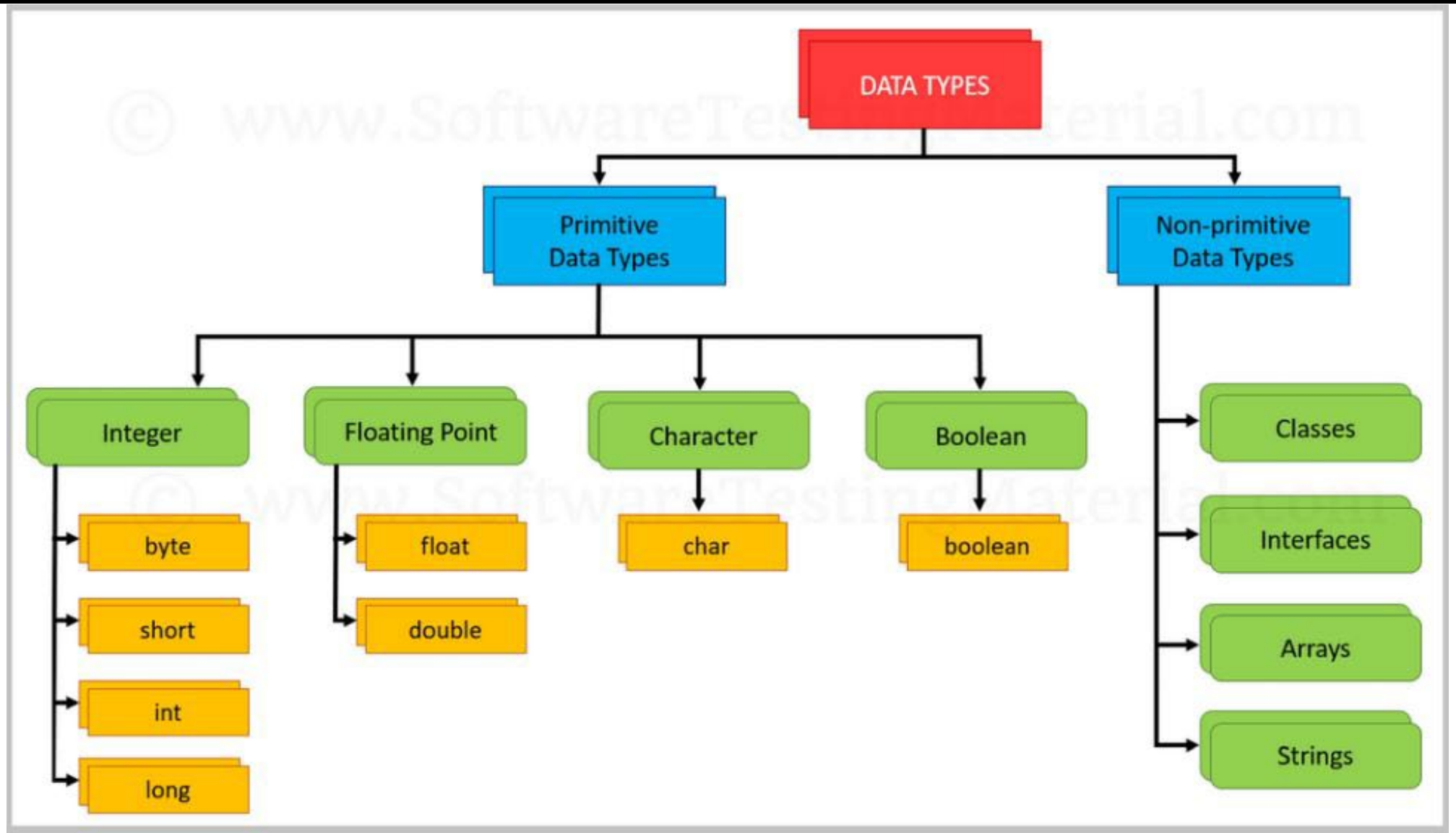
Convention de programmation dans le cours

- Les noms des classes commencent par une majuscule.
- Si le nom de la classe est composé de plusieurs mots, on utilise une majuscule pour la première lettre de chaque mot, par exemple afficherTotal. Cette technique se nomme « Camel Case ».
- Les noms des méthodes débutent par une minuscule. Si le nom de la méthode est composé de plusieurs mots, on utilise la technique du « Camel Case ».
- Les noms des variables débutent par une lettre minuscule. Si le nom de la variable est composé de plusieurs mots, on utilise la technique du « Camel Case ».
- Les variables à une seule lettre sont permises uniquement pour un usage local (dans une boucle for, par exemple) par exemple i ou j
- Les noms des constantes sont écrits en majuscules et l'on sépare les mots par le trait de soulignement _, par exemple TAUX_DE_TAXE

Java et les types primitifs

- La mémoire centrale est un ensemble de "positions binaires" nommées bits.
- Les bits sont regroupés en octets (8 bits), et chaque octet est repéré par ce qu'on nomme son adresse.
- Compte tenu de sa technologie (actuelle !), l'ordinateur ne sait représenter et traiter que des informations exprimées sous forme binaire. Toute information, quelle que soit sa nature, devra être codée sous cette forme.
- Dans ces conditions, on voit qu'il ne suffit pas de connaître le contenu d'un emplacement de la mémoire (d'un ou de plusieurs octets) pour pouvoir lui attribuer une signification. Il faut également connaître la manière dont l'information qu'il contient a été codée. Il en va de même en général pour traiter cette information.
- La notion de type, tel qu'elle existe dans les langages de programmation, sert (entre autres choses) à régler les problèmes d'interprétation binaire.

La liste des types primitifs



La liste des types primitifs - Suite

- **Les types primitifs de Java se répartissent en quatre grandes catégories selon la nature des informations qu'ils permettent de représenter :**
  - **nombres entiers,**
  - **nombres flottants,**
  - **caractères,**
  - **booléens.**

Les types entiers

- Ils servent à représenter des nombres entiers relatifs.
- Un bit est réservé au signe (0 pour positif et 1 pour négatif). Les autres servent à représenter :
  - la valeur absolue du nombre pour les positifs,
  - ce que l'on nomme le complément à deux du nombre, pour les négatifs.
  - Examinons cela plus en détail, dans le cas de nombres représentés sur 16 bits (ce qui correspondra finalement au type short de Java).

Cas d’un nombre positif

- Voici quelques exemples de codages de nombres positifs. À gauche, le nombre en décimale ; au centre, le codage binaire correspondant ; à droite, le même codage exprimé en hexadécimal :

1	00000000 00000001	0001
2	00000000 00000010	0002
3	0000000000000011	0003
16	0000000000010000	00F0
127	0000000001111111	007F
255	0000000011111111	00FF

Cas d’un nombre négatif

- Voici quelques exemples de codages de nombres négatifs. À gauche, le nombre en décimale ; au centre, le codage binaire correspondant ; à droite, le même codage exprimé en hexadécimal :

-1	1111111111111111	FFFF
-2	1111111111111110	FFFE
-3	1111111111111101	FFFD
-4	1111111111111100	FFFC
-16	1111111111110000	FFF0
-256	1111111100000000	FF00

Le nombre 0 est codé d’une seule manière (0000000000000000).

Les différents types d'entiers

- Java dispose de quatre types entiers, correspondant chacun à des emplacements mémoires de taille différente, donc à des limitations différentes.
- Le tableau suivant en récapitule leurs caractéristiques. Notez l'existence de constantes prédéfinies de la forme « *Integer.MAX_VALUE* » qui fournissent les différentes limites.

Type	Taille (octets)	Valeur minimale	Valeur maximale
byte	1	-128 (Byte.MIN_VALUE)	127 (Byte.MAX_VALUE)
short	2	-32 768 (Short.MIN_VALUE)	32 767 (Short.MAX_VALUE)
int	4	-2 147 483 648 (Integer.MIN_VALUE)	2 147 483 647 (Integer.MAX_VALUE)
long	8	-9 223 372 036 854 775 808 (Long.MIN_VALUE)	9 223 372 036 854 775 807 (Long.MAX_VALUE)

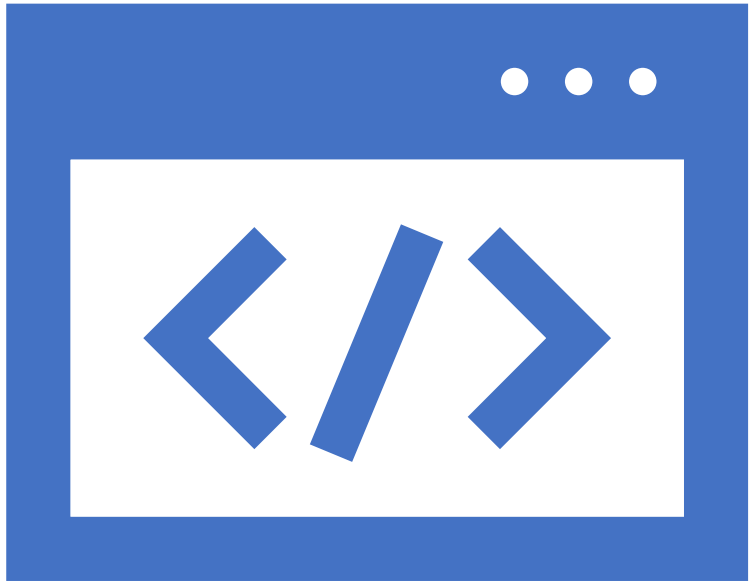
Les types flottants

- Java possède deux types de flottant correspondant chacun à des emplacements mémoires de tailles différentes : float et double.
- Le tableau suivant en récapitule leurs caractéristiques. Notez l'existence de constantes prédéfinies de la forme « *Float.MAX_VALUE* » et « *Double.MAX_VALUE* » qui fournissent les différentes limites.

Type	Taille (octets)	Valeur minimale absolue	Valeur maximale absolue
float	4	-1.40239846E-45 (Float.MIN_VALUE)	3.40282347E38 (Float.MAX_VALUE)
double	8	4.9406564584124654E-324 (Double.MIN_VALUE)	1.79769313486231 6E308 (Double.MAX_VALUE)

Les types caractères

- Comme la plupart des langages, Java permet de manipuler des caractères. Mais il offre l'originalité de les représenter en mémoire sur deux octets.



Les types booléens

Les types de données booléens sont utilisés pour représenter quelque chose qui n'a que deux réponses possibles, par exemple une question Oui/Non.

Booléen est utilisé pour représenter les valeurs vraies et fausses. Ainsi, le type de données booléen n'a que deux littéraux qui sont vrais et faux. Les types booléens sont par défaut à false s'ils ne sont pas initialisés.

Data Types in Java

1 byte = 8 bit

Data Type	Default Value	Default Size	Value Range	Example
boolean	false	1 bit (which is a special type for representing true/false values)	true/false	boolean b=true;
char	'\u0000'	2 byte (16 bit unsigned unicode character)	0 to 65,535	char c='a';
byte	0	1 byte (8 bit Integer data type)	-128 to 127	byte b=10;
short	0	2 byte (16 bit Integer data type)	-32768 to 32767	short s=11;
int	0	4 byte (32 bit Integer data type)	-2147483648 to 2147483647.	int i=10;
long	0L	8 byte (64 bit Integer data type)	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	long l=100012;
float	0.0f	4 byte (32 bit float data type)	1.40129846432481707e-45 to 3.40282346638528860e+38 (positive or negative).	float f=10.3f;
double	0.0d	8 byte (64 bit float data type)	4.94065645841246544e-324d to 1.79769313486231570e+308d (positive or negative)	double d=11.123;

Boxing et Unboxing des types primitifs

- En Java, chacun des 8 types primitifs possède son objet. Cet objet se nomme « enveloppe » « Wrapper ».
- Le processus d'encapsulation de la valeur primitive dans le type Wrapper est appelé boxing. D'autre part, obtenir une valeur primitive à partir du type Wrapper s'appelle unboxing
- Le compilateur Java effectue l'autoboxing et l'autounboxing chaque fois que cela est nécessaire automatiquement. Où l'autoboxing est la conversion du type primitif en type référence (objet de la classe Wrapper correspondante), par exemple : int en Integer. Le déballage automatique est la conversion du type de référence (objet de la classe Wrapper) en primitif, par exemple : conversion de Integer en int.

Les Types Wrappers (types de référence) fournis par java pour les types primitifs correspondants sont :

boolean	Boolean
byte	Byte
character	Character
short	Short
int	Integer
long	Long
float	Float
double	Double

L'API Java

Le langage Java possède un nombre impressionnant de classes (objets) dans sa librairie que l'on peut utiliser comme on veut. Il n'est pas nécessaire de les inclure, l'IDE (Eclipse) s'en chargera à notre place. Cela se nomme une API (Application Programming Interface).

Le rôle de toutes ces classes est de nous permettre de minimiser le code que l'on doit inventer.

Ces classes sont regroupées en « Package ».

Voici des liens vers quelques classes de l'API 19 Java

Classe	Lien vers l'API
Object	https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Object.html
System	https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/System.html
String	https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/String.html
Math	https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Math.html
Integer	https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Integer.html
Double	https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Double.html
Character	https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Character.html
Vector	https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/util/Vector.html