Programmation OOP/Java

Ali ZAINOUL

for FFU
March 28, 2023

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Find me

This course was made with ♥.
Find me in github: here.

Historique de Java

- Java a été créé par James Gosling, Patrick Naughton et Mike Sheridan à Sun Microsystems en 1991.
- Le langage a été initialement appelé "Oak".
- En 1995, Oak a été renommé Java.
- Java a été conçu pour être portable et pour fonctionner sur différentes plateformes.
- Java est maintenant l’un des langages de programmation les plus populaires au monde.

Caractéristiques de Java

- **Portabilité**: les programmes Java peuvent fonctionner sur différentes plateformes sans modification.
- **Sécurité**: Java inclut des mécanismes de sécurité pour protéger les utilisateurs contre les programmes malveillants.
- **Robustesse**: Java est conçu pour être robuste et résistant aux erreurs.
- **Facilité d'utilisation**: Java est un langage de programmation facile à apprendre et à utiliser.
- **Performance**: Java est performant grâce à son compilateur Just-in-Time (JIT).

Comparaison avec C++

- Java est plus facile à apprendre et à utiliser que C++.
- Java est plus portable que C++.
- Java inclut des mécanismes de sécurité qui manquent dans C++.
- Java est plus lent que C++ en termes de performances brutes, mais cela peut être compensé par son compilateur JIT.

Comparaison avec Python

- Java est plus performant que Python.
- Java est plus adapté aux applications d’entreprise que Python.
- Python est plus facile à apprendre et à utiliser que Java.
- Python est plus adapté aux tâches de traitement de données et d’analyse.

Installation du JDK

- Pour programmer en Java, vous devez installer le Java Development Kit (JDK). Voici les étapes pour l'installer:

  1. Téléchargez le JDK sur le site officiel de Java : `https://www.oracle.com/java/technologies/downloads/`
  2. Suivez les instructions d'installation pour votre système d'exploitation

Installation du JDK

- Une fois que le JDK est installé, vous devez ajouter le chemin vers le JDK à votre variable d'environnement PATH:
  3. Ouvrez une fenêtre de commande et tapez la commande suivante:

```
export PATH = $PATH:/chemin/vers/le/jdk
```

- Assurez-vous de remplacer /chemin/vers/le/jdk par le chemin d'installation réel du JDK sur votre système.
- C'est tout! Vous êtes maintenant prêt à programmer en Java avec le JDK installé sur votre système.

Installation du JDK

- Pour vérifier que l'installation a réussi, ouvrez une fenêtre de terminal et tapez la commande suivante: `java -version`. Si le JDK est correctement installé, vous devriez voir la version de Java installée affichée dans la fenêtre de terminal.

- Félicitations, vous avez maintenant installé avec succès le JDK sur votre système et vous êtes prêt à commencer à écrire des programmes Java!

Installation et configuration d’un IDE: Eclipse

Installation d’Eclipse

- Pour installer Eclipse, suivez les étapes suivantes :
  1. Tout d’abord, téléchargez Eclipse à partir du site web officiel : `https://www.eclipse.org/downloads/`.
  2. Choisissez la version qui convient à votre système d’exploitation et à vos besoins (par exemple, Eclipse IDE pour Java Developers).
  3. Une fois le téléchargement terminé, décompressez le fichier zip téléchargé dans un répertoire de votre choix.
  4. Lancez Eclipse en exécutant le fichier exécutable `eclipse.exe` (Windows) ou `eclipse` (Linux/MacOS).
  5. Eclipse vous demandera de choisir un répertoire pour stocker les données de workspace. Vous pouvez choisir le répertoire par défaut ou en créer un nouveau si vous le souhaitez.

Installation et configuration d’un IDE: Eclipse

Configuration d’Eclipse

- Une fois que vous avez installé Eclipse, vous pouvez configurer l’environnement pour vos besoins de développement. Voici quelques étapes courantes que vous pouvez suivre pour configurer Eclipse :
  1. Configurez les paramètres de la police et de la couleur pour rendre l’éditeur de code plus lisible.
  2. Importez les préférences d’un autre utilisateur pour utiliser les mêmes paramètres de configuration.
  3. Installez des plugins pour ajouter des fonctionnalités supplémentaires à Eclipse.
  4. Configurez les paramètres de build pour spécifier les versions de Java et les options de build.

Premier programme en Java

```
// Hello.java
public class Hello {
    public static void main(String[] args) {
       System.out.println("Hello, World!");
    }
}
```

Premier programme en Java

Compilation

- Afin de compiler ce programme, il faut suivre ces étapes:
  1. Ouvrir une fenêtre de commande (terminal) et naviguer au dossier (répertoire) contenant le fichier "Hello.java".
  2. Compiler avec cette commande:

     javac **Hello**.java

  3. Ceci créera un nouveau fichier appelé "**Hello**.class"
  4. Executer le programme en entrant la commande:

     java **Hello**

  5. Vous verrez à l'écran "Hello, World!" affiché.

Structure d’un programme Java

Les Paquets en Java

En Java, les paquets (ou packages) sont utilisés pour organiser les classes en groupes logiques. Les paquets sont définis en haut des fichiers source et doivent être nommés en utilisant la convention de nommage de package Java.

Exemple de définition d’un package :

```
package com.example.myapp;
```

Structure d’un programme Java

Les Classes en Java

Les classes sont les éléments de base d’un programme Java. Chaque classe représente une entité dans le monde réel et peut contenir des variables et des méthodes. **Les classes doivent être définies dans des fichiers sources portant le même nom que la classe**.

```
package com.example.myapp;

public class MyClass {
// variables et méthodes
}
```

Structure d’un programme Java

Les Classes en Java (suite)

- Voici les règles à respecter quant à l’appellation des fichiers et des classes:

  - Les noms de classe doivent commencer par une lettre majuscule et suivre la convention de la casse Camel. Cela signifie que chaque mot dans le nom doit commencer par une lettre majuscule, sauf pour le premier mot. Par exemple : MyClass, MyOtherClass, My_Class.

  - Les noms de fichiers doivent correspondre au nom de la classe publique définie dans le fichier et doivent se terminer par l’extension .java. Par exemple, si vous avez une classe nommée **MyClass**, le fichier doit être nommé **MyClass**.java. Si la classe fait partie d’un package, le fichier doit être situé dans une structure de répertoires qui correspond à la structure du package.

Structure d’un programme Java

Les Classes en Java (suite)

- Voici les règles à respecter quant à l’appellation des fichiers et des classes:
  - Le nom de la classe publique définie dans le fichier doit correspondre exactement au nom du fichier. Par exemple, si le fichier s’appelle MyClass.java, la classe publique définie dans ce fichier doit s’appeler MyClass.
  - Les noms de classe et de fichier ne doivent contenir aucun espace ni caractère spécial (tels que , #, %, etc.), à l’exception des tirets bas (_) et des signes dollar ($).
  - Les noms de classe et de fichier doivent être descriptifs et significatifs, et doivent refléter la finalité ou la fonctionnalité de la classe.

Structure d’un programme Java

Les Méthodes en Java

- Les méthodes sont des blocs de code qui effectuent une tâche spécifique. Chaque méthode doit être définie dans une classe et peut accepter des paramètres et renvoyer une valeur. Les méthodes doivent être appelées à partir d’une autre méthode ou d’un point d’entrée du programme.

```
package com.example.myapp;
public class MyClass {
public static void myMethod(int param) {
        // code pour la méthode
    }
}
```

Les commentaires en Java

En Java, les commentaires sont des annotations qui permettent de documenter le code. Les commentaires sont ignorés par le compilateur Java et n'affectent pas l'exécution du programme. Il existe deux types de commentaires en Java :

- Les commentaires en ligne : ils commencent par // et sont utilisés pour commenter une seule ligne de code.
- Les commentaires sur plusieurs lignes : ils commencent par /* et se terminent par */. Ils peuvent être utilisés pour commenter plusieurs lignes de code.

Exemple de commentaires en Java :

```
// Ceci est un commentaire en ligne
```

Les types de données en Java

Data Type	Size (bits)	Minimum Value	Maximum Value
`byte`	8	-128	127
`short`	16	-32,768	32,767
`int`	32	-2,147,483,648	2,147,483,647
`long`	64	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
`float`	32	1.4 x 10^-45	3.4 x 10^38
`double`	64	4.9 x 10^-324	1.8 x 10^308
`char`	16	0	65,535
`boolean`	-	`false`	`true`

Figure: Primitive Data Types

Les types de données en Java

- Les types primitifs sont les types de données les plus basiques que l'on peut retrouver dans les langages de programmation style Java, C, C++ ou Python. On en distingue 8: **boolean**, **byte**, **char**, **short**, **int**, **long**, **float**, **double**. Une combinaison de char**s** donne un **string**.
- **Types Void:** le type de retour **void** indique qu'aucune valeur n'est retournée.
- **Types Dérivés:** Les tableaux (arrays et arraylists), les classes.

Les types de données en Java

Type de données	Taille en octets (bytes)
byte	1
short	2
int	4
long	8
float	4
double	8
char	2
boolean	1

Table: Tailles des types de données en Java

Variables

Définition: Une **variable** est un nom donné à une zone mémoire qu'un programme informatique puisse manipuler.

- Chaque variable dans Java a:
  - **un type spécifique**, ce qui déterminera la taille et le rendu de la variable dans la mémoire;
  - **la gamme des valeurs** qui peuvent être stockées dans cette zone mémoire;
  - et **l'ensemble des opérations** que l'on peut appliquer sur notre variable.

Variables

- Le nom d’une variable doit impérativement suivre ces deux règles:
  - ***Cela doit commencer*** par une lettre ou un underscore "_".
  - La variable doit être composée de valeurs alphanumériques et/ou l’underscore "_". E.g: _1, _myvar, a_var, var, and var123_L sont toutes des appellations valides.

Remarque: Java est sensible à la casse, donc les lettres miniscules et majuscules sont distincts. e.g.: myvar ***n’est pas*** myVar.

Variables

- Dans les frames qui suivent, on accordera une attention particulière à la difference entre la **déclaration**, la **définition** et à **l'initialization** d'une variable, ***un concept super important*** dans les langages de programmation, et spécifiquement Java.

Variables
Déclaration de variables en Java

Définition: **La déclaration** d'une variable est généralement une introduction à une nouvelle mémoire allouée et mise en évidence par un identifiant d'un type donné (built-in, basic ou user-defined).

- Les trois propriétés de la déclaration sont:
  - La création de zone mémoire se fait au même moment que la déclaration elle-même.
  - Les variables déclarées ***mais pas définies ou initialisées*** peuvent avoir des valeurs au hasard. (garbage values)
  - Les variables ***ne peuvent pas être*** utilisées avant déclaration.

Variables
Déclaration de variables en Java

- C’est fait ainsi:

```
type variableList;
```

- Le **type *doit*** être un **type Java valide** incluant: char, int, float, double, bool etc. ou un objet user-defined (l’instance d’une classe par exemple).

- **variableList** consiste d’**une** ou **plusieurs** variables (identifiants) séparées par des virgules, quelques exemples:

Variables
Déclaration de variables en Java

```
// This is a simple comment and will be ignored by the compiler
int a; // Declaration of variable a with int type
char b, c; // Declaration of distinct variables b and c with char type
float d, e, f; // Declaration of distinct variables d, e and f with float type
double _d; // Declaration of the variable _d with double type.
```

Remarque: il en va de même pour les fonctions, on peut **déclarer** une fonction et **définir** celle-ci par la suite. Par exemple : void foo() ; est une déclaration de la fonction foo. Et sans lui donner de définition, le code peut se compiler à condition que la fonction ne soit pas utilisée ailleurs dans votre programme. (ce qui est assez inutile...).

Variables
Définition de variables en Java

Définition: **La Définition** d'une variable est le fait d'assigner une valeur à cette variable, typiquement avec l'**opérateur d'assignement =**. C'est le fait d'assigner une valeur **valide** à la variable préalablement déclarée.

```
int a, b; // Declaration of distinct variables a and b of type int
a = 1; // Definition of variable a
b = a + 1; // Definition of variable b as a + 1 (=2 here)
```

Variables
Initialisation de variables en Java

Définition: l'Initialisation d'une variable est simplement le fait de lui assigner (définir) une valeur au moment de la déclaration.

Par exemple:

```
float f = 0.3;
```

est la déclaration d'une variable avec l'identifiant (nom) f, de type float et définissant / assignant à elle la valeur 0.3.

Les constantes

```
public class Constantes {
	// Constante de type entier
	public static final int AGE_LEGAL = 18;
	// Constante de type double
	public static final double PI = 3.14159265359;

	// Constante de type chaîne de caractères
	 public static final String MSG_BIENVENUE = "Hola!";
	// Constante de type booléen
	public static final boolean EST_ADMIN = true;
}
```

Déclaration et définition de fonctions

Une **Déclaration de Fonction** a la signature:

```
return_type name_of_my_function(args if any);
```

Tandis que la **Définition d'une fonction** est donnée par:

```
return_type name_of_my_function(args if any) {
      Stuff_that_my_function_will_do;
}
```

Function Declaration and Definition

A complete example is as follows :

```
// Function Declaration
int my_function();
int main()
/*
Function call, the Definition may comes:
- just after the Declaration in the same source program
- be part of another source program that must be compiled before we use the function.
*/
// Function Call
my_function();

// Function Definition
int func() return (-123) ;
// The function simply returns the integer value -123, the parenthesis are here for the sake of exactness.
```

Opérateurs

Définition: En mathématiques et en programmation informatique, un **opérateur** est un symbole ou charactère spécifiant au compilateur de performer un process mathamétique ou logique spécifique.

- Chaque langage de programmation possède ces propres opérateurs, cependant les opérateurs que l'on va présenter sont communs à plusieurs langages de programmation. Java possède ces opérateurs : **Arithmétiques**, **Relationnels**, **Logiques**, **Bitwise**, **Assignement** and **Miscellaneous** operators (Misc).

Opérateurs
Table de vérité

- Un concept important à connaître avant de vouloir manipuler les opérateurs: la table de vérité. Supposons que l'on a deux propositions (P) and (Q). Alors:

P	Q	P AND Q	P OR Q	P XOR Q	NOT P	NOT Q
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	1
1	1	1	1	0	0	0

Les opérateurs **arithmétiques**

Opérateur	Signification
+	Ajoutes le membre à droite (RHS) au membre à gauche (LHS). (a+b)
−	Soustrais le membre à droite (RHS) du membre à gauche (LHS). (a-b)
*	Multiplies le membre à gauche (LHS) par le membre à droite (a*b)
/	Divises le membre à gauche (LHS) par le membre à droite (RHS) (a/b)
%	Modulo retourne le reste de la Division Euclidienne du membre à gauche par le membre à droite(a%b)
++	Incrémentes la valeur d'un entier par un (++a **preincrément**) or (a++ **postincrément**)
−−	Decrémentes la valeur d'un entier par un (–a **predécrement**) or (a– **postdecrément**)

Les opérateurs de **comparaison**

Opérateur	Signification
==	Évalué à vrai que si les deux opérandes sont égales. (a==b)
!=	Évalué à vrai que si les deux opérandes ne sont pas égales. (a!=b)
>	Évalué à vrai que si le membre à gauche est strictement plus grand que le membre à droite. (a>b)
<	Évalué à vrai que si le membre à droite est strictement plus grand que le membre à gauche. (a<b)
>=	Évalué à vrai que si le membre à gauche est strictement plus grand que le membre à droite OU égal. (a>=b)
<=	Évalué à vrai que si le membre à droite est strictement plus grand que le membre à gauche OU égal. (a<=b)

Les opérateurs **logiques**

Opérateur	Signification
&&	Opérateur ET logique. La condition n'est vraie que si les deux opérandes ne sont pas des zéros.
||	Opérateur OU logique. La condition n'est vraie que si l'un des opérandes est différent de zéro.
!	Opérateur NOT logique. Il inverse l'état logique.

Les opérateurs d'**affectation**

Opérateur	Signification
=	Opérateur d'affectation. Il affecte les valeurs des opérandes à droite aux opérandes
+=	Opérateur d'affectation et d'ajout. Il ajoute l'opérande à droite à l'opérande à gauche et affecte le rés
-=	Opérateur d'affectation et de soustraction. Il soustrait l'opérande à droite à l'opérande à gauche et affecte
*=	Opérateur d'affectation et multiplicateur. Il multiplie l'opérande à droite par l'opérande à gauche et affecte
/=	Opérateur d'affectation et de division. Il divise l'opérande à gauche avec l'opérande à droite et affecte l
%=	Opérateur d'affectation et de modulus. Affecte le reste de la division euclidienne de l'opérande à gauche par l'op
«=	Opérateur d'affectation et de shift à gauche des bits.
»=	Opérateur d'affectation et de shift à droite des bits.
&=	Opérateur d'affectation et de Bitwise ET (AND).
^=	Opérateur d'affectation et de Bitwise OU eXlusive. (XOR)
|=	Opérateur d'affectation et de Bitwise OU Inclusive. (OR)

Les opérateurs de **bits (Bitwise)**

Opérateur	Signification
&	Opérateur binaire ET par bit, évalue à VRAI que si les deux bits sont VRAIS.
|	Opérateur binaire OR par bit, évalue à VRAI que si l'un des bits est VRAI.
^	Opérateur XOR binaire par bit, évalue à VRAI si un bit est VRAI dans l'un des opérandes, mais pas dans les deux.
~	Opérateur NOT unaire dans le sens des bits. L'opérateur NOT unaire inverse les bits (même logique que NOT).
«	Opérateur binaire de décalage vers la gauche.
»	Opérateur binaire bitwise de décalage vers la droite.

Priorité des opérateurs en Java

Priorité de l'opérateur décide du regroupement des termes dans une expression. Elle affecte la manière dont l'expression est évaluée. Certains opérateurs ont une priorité plus élevée que d'autres.

- **La règle est** : dans une expression, les opérateurs ayant la priorité la plus élevée seront évalués en premier.
- Dans le tableau ci-dessous, les opérateurs ayant la priorité la plus élevée apparaissent en haut, tandis que ceux ayant la priorité la plus faible apparaissent en bas.

Priorité des opérateurs en Java

Operator	Classification	Associativity
() [] -> . ++ –	Postfix	Left to right
+ - ! ˜ ++ – (type)* & sizeof	Unary	Right to left
*/ %	Multiplicative	Left to right
+-	Additive	Left to right
« »	Shift	Left to right
< <= > >=	Relational	Left to right
== !=	Equality	Left to right
&	Bitwise AND	Left to right
ˆ	Bitwise XOR	Left to right
|	Bitwise OR	Left to right
&&	Logical AND	Left to right
||	Logical OR	Left to right
?:	Conditional	Right to left
= += -= *= /= %= »= «= &= ˆ= |=	Assignement	Right to left
,	Comma	Left to right

Structures de contrôle de flux: **if**

- En programmation informatique, on appelle une **structure de contrôle de flux**, ou une **structure conditionnelle** l'ensemble des instructions qui testent si une condition ou un ensemble de conditions est vrai ou pas. On en distingue trois structures de contrôle de flux:
  - La structure de contrôle de flux **if**:

```
if (condition) {
        my_instructions;
}
```

Structures de contrôle de flux: **if . . . else**

- **Suite:**
  - La structure de contrôle de flux **if . . . else** :

```
if (condition) {
        my_instructions;
}
else {
        my_other_instructions;
}
```

Structures de contrôle de flux: **if . . . elif . . . else**

- **Suite**:
  - La structure de contrôle de flux **if . . . elif . . . else** :

```
if (condition_1) {
        my_instructions_1;
}
elif (condition_2) {
        my_instructions_2;
}
else {
        my_other_instructions;
}
```

Structures de contrôle de flux: **switch . . . case**

- Suite:
  - La structure de contrôle de flux: **switch . . . case**

```
switch(expression) {
        case condition_1:
        // code_Instructions_1;
        break;
        case condition_2:
        // code_Instructions_2;
        break;
        default:
        // code_Instructions_default;
}
```

*Structures de contrôle de flux: boucle **while** ...*

- Une **structure conditiennelle itérative** permet d'avoir une série d'instructions exécutées plusieurs fois (iterations). La boucle **while** exécute les instructions du bloc tant que la condition est vraie. Le même principe est appliqué pour la boucle **for**, les deux structures sont détaillées comme suit:
  - La structure de contrôle de flux: boucle **while**:

```
while (condition) {
        my_instructions;
}
```

*Structures de contrôle de flux: boucle **for** ...*

- **Suite**:
  - La structure de contrôle de flux: boucle **for**:

```
for (my_initialization; my_condition; my_increment){
        my_instructions;
}
```

- **Remarque**: La majeur différence entre une boucle **for** et une boucle **while** est que la boucle **for** est utilisée quand le nombre d'itérations est connu à l'avance, tandis que l'exécution du bloc d'instructions dans la boucle **while** est fait jusqu'à ce que la condition ne soit plus vraie.

*Structures de contrôle de flux: boucle **do . . . while***

- **Suite**:
  - La structure de contrôle de flux: boucle **do . . . while**:

```
do {
        my_instructions;
}while(my_condition);
```

- **Note**: Une différence réside entre les boucles **while** et la boucle **do while**: dans la boucle **while** on vérifie la condition avant la première itération, tandis que dans la boucle **do while**, on vérifie la condition après la première itération (l'éxecution d'au moins une fois le block d'instructions.

Scopes

Définition: Dans n'importe quel langage de programmation, **la portée (scope)** d'une variable et/ou fonction est la zone où une fonction ou une variable est visible et accessible.

Il y a trois régions où une variable peut être déclarée en Java:

- À l'intérieur d'une fonction ou d'un bloc qui est appelé **variables locales**;
- En dehors de toute fonction, ce qu'on appelle des **variables globales** ;
- Dans la définition des paramètres d'une fonction qu'on appelle les **paramètres formels**.

Les tableaux (Arrays) en Java

Définition: Un **tableau (Array)** est une structure de données qui permet de stocker une collection d'éléments de même type.

- Exemple:

```
int[] numbers = { 1, 2, 3, 4, 5 };
```

Les tableaux (Arrays) en Java (suite)

- Les éléments d’un tableau peuvent être accédés par leur indice.

```
int firstNumber = numbers [0];          // outputs 1
int secondNumber = numbers [1];         // outputs 2
```

- La longueur d’un tableau peut être obtenue avec la propriété "length".

```
int length = numbers.length;
```

Les tableaux (Arrays) en Java: Initialisation et manipulation des tableaux en Java

Un tableau peut être initialisé avec une taille donnée. (**Allocation Statique**)

- Exemple:

```
int[] numbers = new int[5];
```

Les tableaux (Arrays) en Java: Initialisation et manipulation des tableaux en Java

- Les éléments d'un tableau peuvent être modifiés *via* leur indice. Exemple:

```
numbers[0] = 10; numbers[1] = 20;
```

- Les tableaux peuvent être parcourus avec une boucle for.

```
for (int i = 0; i < numbers.length; i++)
        System.out.println(numbers[i]);
```

/* Principes fondamentaux de la Programmation Orientée Objet (POO) */

Principes fondamentaux de la POO

Définition: La Programmation Orientée Objet (POO) est un des principes fondamentaux en programmation informatique, ses origines remontent aux années 1970 avec les langages Simula et Smalltalk, mais le principe a rapidement pris son envol grâce à la création du langage C++ qui est l'extension du langage C, avec en effet, cette quête de rendre les logiciels plus robustes.

Principes fondamentaux de la POO (suite)

Une mnémotechnique utile regroupant les **six principes** fondamentaux de la Programmation Orientée Objet est **"ACOPIE"**.

- **A***bstraction* (Abstraction)
- **C***lass* (Classe)
- **O***bject* (Objet)
- **P***olymorphism* (Polymorphisme)
- **I***nheritance* (Héritage)
- **E***ncapsulation* (Encapsulation)

Object

- En informatique, un objet est un conteneur (container) symbolique et autonome contenant des informations et des fonctions/méthodes concernant un sujet, manipulés dans un programme. Le sujet est souvent quelque chose de tangible appartenant au monde réel.

Class

- La classe est une structure informatique particulière dans le langage objet.
- Elle décrit la structure interne des données et elle définit les méthodes qui s'appliqueront aux objets de même famille (même classe) ou type.
- Elle propose des méthodes de création des objets dont la représentation sera donc celle donnée par la classe génératrice. Les objets sont dits alors instances de la classe. C'est pourquoi les attributs d'un objet sont aussi appelés variables d'instance et les messages opérations d'instance ou encore méthodes d'instance.

Encapsulation

- L’encapsulation est le fait de regrouper les données et les méthodes qui les manipulent en une seule entité appelée *capsule*.
- Elle permet d’avoir un code organisé, restreindre laccès à certaines portions depuis lextérieur de la capsule et avoir un code robuste.

Abstraction

- L’abstraction est l’un des concepts clés dans les langages de programmation orientée objet. Son objectif principal est de gérer la complexité (comprendre difficulté) en masquant les détails inutiles à l’utilisateur.
- L’abstraction est le processus consistant à représenter un objet dans la vie réelle en tant que modèle informatique.
- Cela consiste essentiellement à extraire des variables pertinentes, attachées aux objets que lon souhaite manipuler, et à les placer dans un modèle informatique convenable.

Inheritance

- C’est un mécanisme pour transmettre toutes les méthodes d’une classe dite "mère" vers une autre dite "fille" et ainsi de suite.

Polymorphism

- Le nom de polymorphisme vient du grec et signifie qui peut prendre plusieurs formes. Cette caractéristique est un des concepts essentiels de la programmation orientée objet. Alors que l'héritage concerne les classes (et leur hiérarchie), le polymorphisme est relatif aux méthodes des objets.

Définition et utilisation des classes en Java

Définition: Une **classe** est une structure de données qui définit les attributs et les méthodes qui définissent un objet.

- Exemple 1:

```
// Class MyClass
public class MyClass {
        private String name; // Member
        public myClass(String _name) { //Constructor
                this.name = _name;
        }
        public String getName() { // Getter Method
                return name;
        }
}
```

Définition et utilisation des classes en Java (suite)

- Exemple 2:

```
public class Person {
	private String name;
	private int age;
	public Person(String name, int age) {
		this.name = name;
		this.age = age;
	}
	public String getName() { return name; }
	public int getAge() { return age; }
}
```

Définition et utilisation des classes en Java (suite)

- Exemple 3:

```
public class Car {
	private String brand, model;
	private int year;
	public Car(String brand, String model, int year) {
		this.brand = brand;
		this.model = model;
		this.year = year;
	}
	public String getBrand() { return brand; }
	public String getModel() { return model; }
	public int getYear() { return year; }
}
```

Définition et instanciation des objets en Java

Définition: Un **objet** est une **instance** d'une classe.

- Exemples:

```
MyClass Myclass = new MyClass("myOwnClass");
```

```
Person person = new Person("Ali", 30);
```

```
Car car = new Car("Alfa Romeo", "Giulietta", 2020);
```

Définition et instanciation des objets en Java (suite)

Définition: une **méthode** est une fonction ou un bloc de code associée à une classe ou à un objet. Une méthode permet ainsi de définir le comportement de l'objet ou de la classe.

- Si elle est appelée par une instance de la classe (objet), c'est donc une **méthode régulière** ou simplement **méthode**.
- Si elle est appelée par la classe elle-même (classe), c'est donc une **méthode statique** ou **méthode de classe**.

Utilisation des objets en Java (suite)

- Les méthodes permettent d’interagir avec les membres (attributs) d’un objet (instance d’une classe).
- Exemples:

```
String name = myclass.getName();
```

```
String name = person.getName();
int age = person.getAge();
```

```
String brand = car.getBrand();
String model = car.getModel();
int year = car.getYear();
```

Les attributs

Définition: un **attribut**, également appelé **membre** de classe, est une variable déclarée à l'intérieur d'une classe qui a pour but de stocker des données spécifiques à cette classe. Les attributs sont ainsi des caractéristiques ou des propriétés d'un objet qui sont définies dans la classe et qui peuvent être utilisées pour décrire l'état ou les caractéristiques de l'objet.