

Chapitre1:**Introduction à la Programmation orienté objet****1 Les différents styles de programmation****1.1 Style applicatif**

Proche des notations mathématiques, utilise beaucoup la récursivité

Exemple: Lisp, Caml, ML, Haskell

1.2 Style impératif

- Utilise beaucoup les itérations et autres structures de contrôle

- Les structures de données sont fondamentales

Exemple: Fortran, C, Pascal

Style objet

Propose une méthodologie de programmation centrée sur les objets. Un objet peut être vu comme une entité regroupant un ensemble de données exprimant son **état** et de méthodes exprimant son **comportement**.

Java n'est pas le seul langage objet, d'autres langages existent comme : Simula, Smalltalk, C++, OCaml...

2 La programmation orientée objet (POO)

Pour écrire un programme orienté objet:

- Identifier les objets utiles
- Définir les attributs de ces objets (leurs états)
- Définir les méthodes que ces objets peuvent faire (comportement)

Les objets pouvant avoir le même état (même attributs) et le même comportement (méthodes) doivent appartenir à une même **classe**.

2.1 Intérêts de la POO

- Programmation modulaire: Faciliter de la réutilisation de code.
- Encapsulation (Principe de la POO) et abstraction (proche du monde réel):
 - Cacher les membres (attribut, méthodes) d'une classe: choix dans les niveaux de confidentialité
- L'objectif est de produire du code:
 - facile à développer, à maintenir, à faire évoluer
 - réutilisable, tout ou en partie, sans avoir besoin de le dupliquer
 - générique, et dont les spécialisations sont transparentes

2.2 Principes de base de la POO

- L'encapsulation
- L'héritage
- Le polymorphisme

2.3 Généralités sur les objets

- L'objet est l'unité logique de la programmation orientée objet. C'est une entité rassemblant des données et le code travaillant sur ses données.
- Il est définie par:
 - un état (attributs- Fields)
 - un comportement (méthodes - fonctions)

2.3.1 L'état

Il est défini à l'aide des attributs (Fields, appelés aussi données membres) qui stockent des valeurs (des informations sur l'état de l'objet).

Exemple: L'état d'un objet de la classe **Voiture** est caractérisé par les attributs : **couleur et marque**

L'état de l'objet **maVoiture** de la classe Voiture est caractérisé rouge, fiat

2.3.2 Le comportement

Caractérisé par des méthodes (appelées parfois fonctions) qui permettent de modifier l'état de l'objet.

- Une méthode rattachée à l'objet permet de déclencher un des comportements associés à l'objet.
- Il n'est pas possible d'agir directement sur les données d'un objet pour modifier son état; il est nécessaire de passer par ses méthodes. On dit qu'on « envoie un message » (on fait une requête) à un objet. Un objet peut recevoir un message qui déclenche:
- une méthode qui modifie son état.

Ex: peindre(mavoiture)

et / ou

- une méthode qui envoie un message à un autre objet
démarrer

Ex: mavoiture —————> moteur

En conclusion, Un objet :

- stocke les données
- on peut effectuer des requêtes sur cet objet qui est équivalent à un appel de fonction (méthode)
- chaque objet a son espace mémoire: la simplicité des objets mis en œuvre, cache la complexité des programmes.
- chaque objet a un type précis : c'est-à-dire chaque objet est une instance d'une classe (type)
- un programme est ensemble d'objets qui s'envoient des messages (méthodes)

3 Le langage Java

- créé en 1995 par Sun Microsystems, la version actuelle est Java 21 (9/2023) édité par Oracle.
- orienté objet
- fortement typé :
 - Toute variable doit être déclarée avec un type
 - Le compilateur vérifie que les utilisations des variables sont compatibles avec leur type (notamment via un sous-typage correct)
 - Les types sont d'une part fournis par le langage, mais également par la définition des classes
 - Compilé : La compilation d'un programme Java ne traduit pas directement le code source en fichier exécutable. Elle traduit d'abord le code source en un code intermédiaire appelé «bytecode». Code intermédiaire indépendant de la machine ce qui permet de rendre le code source indépendant de la machine qui va exécuter le programme.
 - Interprété : le bytecode est exécuté par une machine virtuelle Java (JVM ; Java Virtual Machine).
 - Il est fourni avec le JDK (Java Development Kit)
 - Outils de développement
 - Ensemble de paquetages très riches et très variés
 - Sun fournit le compilateur javac avec le JDK.
 - En Java, tout se trouve dans une classe. Il ne peut pas y avoir de déclarations ou de code en dehors du corps d'une classe.
 - La classe elle-même ne contient pas directement du code. Elle contient
 - des attributs.
 - et des méthodes (équivalents à des fonctions)
 - Le code se trouve exclusivement dans le corps des méthodes

3.1 Premier exemple

Dans un fichier de nom **HelloWorld.java**

```
public class HelloWorld {
/* Un style de commentaire
sur plusieurs lignes. */
public static void main(String[] args) {
// Un commentaire sur une seule ligne
System.out.println("Bonjour à vous les LSI ADBD!");
/**
*Commentaire javadoc
*/
}
}
```

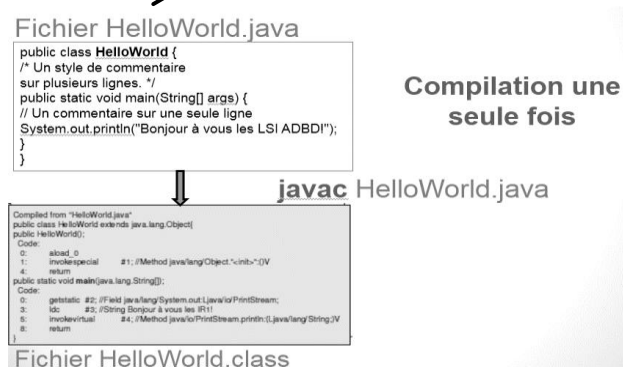
→ Règle: toute classe publique doit être dans un fichier qui a le même nom que la classe

- ça définit une classe, qui est une unité de compilation
- comme il y a une méthode **main**, alors cette classe est « exécutable »

Compilation

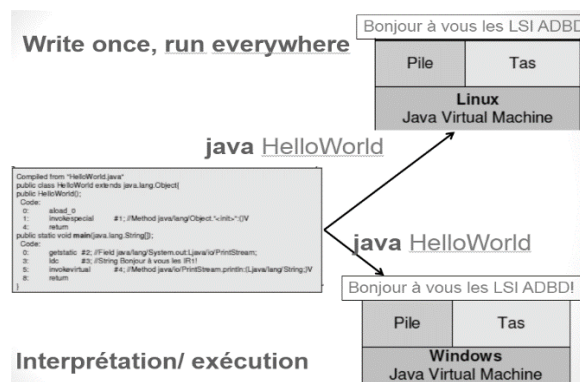
Compilation du code source produit un fichier d'extension .class qui est le bytecode

javac HelloWorld.java → HelloWorld.class



Exécution

Java HelloWorld



3.2 Compilation

La compilation se fait une fois pour obtenir le byte code indépendant de la machine. L'exécution se fait sur n'importe quelle plateforme (Windows, linux, etc.)

- Si le fichier HelloWorld.java fait référence, par exemple, à des classes situées dans les répertoires **/prog/exemple** et **/cours**, alors la compilation se fait de la façon suivante:
 - sous windows:


```
javac -classpath \prog\exemple ; \cours HelloWorld.java
```
 - sous Linux:


```
javac -classpath /prog/exemple : /cours HelloWorld.java
```
- On peut désigner le fichier à compiler par un chemin absolu ou relatif :

```
javac chemin\HelloWorld.java
```

```
Example: javac c:\user2\code\ HelloWorld.java (chemin absolu)
```

```
javac code\HelloWorld.java
```

3.3 Le bytecode

- Le **langage source Java** est défini par la JLS (*Java Language Specification*) éditée par SunOracle
- La compilation du code source effectué avec **javac** produit le bytecode, qui est le «langage machine» de la machine virtuelle Java
- Le **bytecode** d'une classe est destiné à être **chargé** par une **machine virtuelle** qui doit l'exécuter avec la commande **java** suivi du nom du fichier sans extension

3.4 La machine virtuelle (JVM)

- Son rôle est d'**abstraire le comportement d'une machine** pour le rendre le plus possible indépendant de la plateforme
- Son comportement est défini par la JVM Spec éditée par SunOracle
- Peut être adaptée à une plateforme d'accueil (Windows, Linux, Mac...)
- Une JVM traduit le bytecode dans le langage machine de la plateforme d'accueil

3.5 Java: un langage et une plateforme

Pour développer en Java, on a besoin:

- du **langage de programmation** et du compilateur et d'autres commandes utiles: jar, javadoc, etc
- de la JVM et des APIs (Application Programming Interfaces) regroupées dans une « plateforme »:
 - Java SE (Java Platform, Standard Edition) applications classiques, desktop
 - Java EE (Java Platform, Enterprise Edition): pour développer et déployer des applications serveur, Web services, etc.
 - Java ME (Java Platform, Micro Edition): J2ME pour les applications embarquées, PDA, téléphones, etc.
- Pour le développement il faut le JDK (Java Development Kit)
- Si on veut juste exécuter, il suffit du JRE (Java Runtime Execution)

4 Éléments d'un programme JAVA

- **classe (class):** un modèle à partir duquel les objets sont créés
- **Objet (object):** L'objet aura ses propres variables et l'accès à toutes les méthodes définies par sa classe.
- **Etat (variables d'instance):** Les valeurs assignées à une instance de variables de l'objet créent l'état de l'objet
- **Comportement (méthodes):** Les méthodes instaurent la logique de la classe. C'est où les algorithmes sont programmés et les données sont manipulées pour déterminer le comportement de l'objet
- **Héritage:** Une sous-classe qui hérite d'une superclasse (eg. **class** ClassA **extends** ClassB{ }) aura automatiquement l'accès aux variables d'instance et aux méthodes définies par la superclasse. Mais aussi il est possible de redéfinir ou surcharger les méthodes de la superclasse pour définir un comportement plus spécifique.

Exemple:

```
class ClassB {  
void m1(){int y=0;}  
int m2(){int x=3; return x ;}  
}  
class ClassA extends classB{  
int m2(){int x=3; x=x+5; return x;}  
}
```

⇒ La classe ClassA hérite de la classe ClassB la méthode m1() et redéfinit la méthode m2().

Interfaces: Une classe peut implémenter une interface (eg. `class ClassA implements InterfaceI{}`). Ces dernières définissent les méthodes qu'une classe doit supporter mais non comment elles doivent être supportées.

Exemple: une interface Alimentation peut déclarer une méthode manger() sans lui fournir une logique :

```
interface Alimentation{  
void manger();}
```

La classe Lion implémente l'interface Alimentation comme suit:

```
class Lion implements Alimentation {  
public void manger(){  
String nourriture=" viande " ;  
}}}
```

⇒ En implémentant l'interface Animal, la classe Lion doit définir la méthode manger()

- **Identificateurs :** Les identificateurs légaux doivent se composer des caractères unicodes, nombres, symboles (\$, £) et caractères de connexion « _ » .

- **Classes et interfaces :** La première lettre doit être majuscule. Si l'identificateur se compose de plus d'un mot, les mots doivent être collés et la première lettre de chaque mot doit être majuscule.

Exemples de noms de classes: Voiture, Imprimante, CompteBancaire

Exemples de noms d'interfaces: Runnable, Serializable, Executable

- **Méthodes :** La première lettre du premier mot doit être minuscule, les mots doivent être typiquement des paires de verbes-nom, le deuxième mot doit commencer par majuscule comme : getName, setBalance, faireClacul

- **Variables :** même principe que les méthodes
Exemple: largeurBouton, compteBalance, maChaine

- **Constantes :** Les constantes sont créées en les marquant **static** et **final** tout en majuscule, les mots sont séparés par _ ;

Exemple: static final int MIN__HAUTEUR=10;

Exercice: mettre une croix devant les identificateurs qui ne sont pas légaux :

```
int .f;      int _a;      int $c;   int :b;      int -d;      int e#;      int _____2_w;  
int 7g;      int _$;
```

5 Classes et objets

Une classe, disons, Toto représente plusieurs choses:

- Une **unité de compilation**
 - La compilation d'un programme qui contient une classe Toto produira un fichier Toto.class
 - La définition du type Toto qui peut servir à déclarer des variables comme Toto t;
 - Un moule pour la création d'objets de type Toto

5.1 Structure d'une classe

- Une classe est définie par son nom et son package d'appartenance. ex: **java.lang.String**
 - En l'absence de directive, les classes sont dans un package dit « par défaut » (i.e., pas de package)
- Une classe peut contenir trois sortes de **membres**
 - **champs** ou attributs
 - **méthodes** et **constructeurs**
 - **classes internes**

- Les membres statiques (static) sont dits **membres de classe**
 - Ils sont définis sur la classe et non sur les objets
- Les membres non statiques (ou d'instance) ne peuvent exister sans un objet

Exemple:

```
public class Pixel {
    public final static int ORIGIN = 0; ← .....
    private int x;
    private int y; } ← .....
    private static int z; ← .....

    public Pixel(int x1, int y1) { ← .....
        x = x1;
        y = y1; .....
    }
    public void printOnScreen() {
        System.out.println("(" + x + ", " + y + ")");
    }
    public static boolean same(Pixel p1, Pixel p2) {
        return (p1.x == p2.x) && (p1.y == p2.y);
    }
    public void reset() {
        x = ORIGIN;
        y = ORIGIN;
    }
    public static void main(String[] args) {
        Pixel p1 = new Pixel(2, 5); } ← .....
        Pixel p2 = new Pixel(1, 3); } ← .....
        p1.printOnScreen();
        System.out.println(same(p1, p2));
        p1.reset();
    }
}
```

5.2 Manipulation des Objets

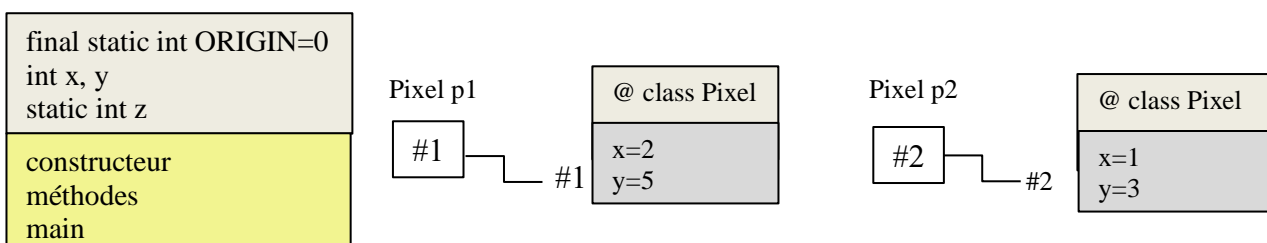
Une fois la classe est définie, on peut créer des objets (variables).

- Chaque objet est une variable d'une classe. Il a son espace mémoire. Il admet une valeur propre à chaque attribut.
- La création de l'objet est appelé instantiation. Ainsi, un objet est aussi appelé une instance d'une classe.
- On parle indifféremment d'instance, de référence ou d'objet

Une classe permet d'instancier plusieurs objets.

- Les attributs et les méthodes d'un objet (d'une instance d'une classe) sont les mêmes pour toutes les instances de la classe.
- Les valeurs des attributs sont propres à chaque objet.

class Pixel



5.3 Les variables

Les variables peuvent être déclarées :

- des variables de classe: précédées par **static**,
- des variables d'instance
- des paramètres de méthode
- des variables locales à une méthode ou à un bloc



Les **variables** ne peuvent être que de deux natures:

5.3.1 type « primitif »

char, boolean, byte, short, int, long, double, ou float

Dans ce cas, la déclaration de la variable réserve la place mémoire (octets) pour stocker sa valeur (qui dépend de son type).

Exemple :

int (entier)  long (entierLong) 

Le tableau 1 ci-dessous montre tous les types de primitives et leurs intervalles de valeurs.

Tableau 1: Intervalles de Primitives Numériques					
Type	Bits	Bytes	Valeur Minimale	Valeur Maximale	Valeur par défaut
byte	8	1	-2^7	2^7-1	0
short	16	2	-2^{15}	$2^{15}-1$	0
int	32	4	-2^{31}	$2^{31}-1$	0
long	64	8	-2^{63}	$2^{63}-1$	0
float	32	4	n/a	n/a	0.0f (f est facultatif)
double	64	8	n/a	n/a	0.0

- Type boolean: une variable de type boolean peut avoir l'une des valeurs **true** ou **false** avec valeur par défaut **false**
- Type char : contient un caractère unicode 16 représenté par bits non signés, c'est à dire 2^{16} valeurs possibles de 0 à 65535 (2^{16})-1. un type caractère est réellement un entier et peut être assigné à n'importe quel type nombre qui peut recevoir 65535, c'est à dire n'importe quel type supérieur à short. Valeur par défaut '\0'

Exemple: `int a =12; double k=15.5; float b= 1.2f; b=5; char t='b';`

- La création d'un objet entraîne toujours une initialisation par défaut de tous les attributs de l'objet même si on ne les initialise pas
- Cette garanti d'initialisation par défaut ne s'applique pas aux variables locales (variables déclarées dans un bloc ou locales d'une méthode)

5.3.2 Variables de référence

Servent à référer (accéder) un objet d'un type déclaré ou d'un sous-type du type déclaré.

Exemples de déclaration de variable de référence:

- `Object o; Dog myNewDogReferenceVariable;`
- `String s1, s2, s3; // déclare 3 variables String`

Dans ce cas, la déclaration de la variable ne fait que réserver la place d'une **référence** (une sorte de pointeur) qui permettra d'accéder à l'endroit en mémoire où est effectivement stocké l'objet en lui-même (vaut **null** si référence inconnue, i.e. pas d'objet).

Exemple : l'instruction: `Pixel p1;`

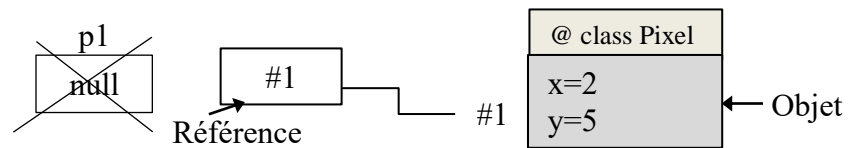
- déclare p1 comme objet (variable) de type: Pixel
- Aucun objet n'est créé: un objet seulement déclaré dont la référence vaut « null ».

p1

null

L'instruction: `p1= new Pixel (2,5);`

- crée un objet => crée un emplacement en mémoire pour stocker un objet de type **Pixel**.
- initialise ses attributs.



La déclaration et la création de l'objet peuvent se faire en une même instruction comme suit:

```
Pixel p1= new Pixel (2,5);
```

5.4 Les méthodes

Chaque méthode **est définie dans une classe** par :

- un type de retour
- un nom
- une liste (éventuellement vide, auquel cas mettre que les parenthèses) de paramètres **typés en entrée**
- une suite d'instructions (un bloc d'instructions) qui constitue le corps de la méthode

Syntaxe :

```
typeRetour nomMethode (type arg1, type arg2, type argi) {
/*
corps de la méthode qui peut contenir l'instruction
*/
}
```

- Si la méthode ne fournit aucun résultat, alors **typeRetour** est remplacé par le mot clé **void**.
- La valeur retournée par la méthode est spécifiée comme dernière instruction de la méthode par:
return expression;

5.5 Accès aux membres d'un objet

5.5.1 Accès aux attributs

Considérons la classe ClasseTest suivante:

```
class ClasseTest {
double x;
char y;
void nomMethode1(int arg1, int arg2) { }
}
```

Pour accéder aux membres d'un objet, il faut que l'objet soit créé (non uniquement déclaré)

```
ClasseTest obj = new ClasseTest();
```

accéder aux attributs de obj en utilisant l'opérateur point (.) Comme suit: `obj.x` `obj.y`

De la même façon, accéder aux méthodes de obj en utilisant l'opérateur point (.) Comme suit:

```
Obj.nomMethode1 (arg1, arg2);
```

Exemple 1: soit la classe suivante:

```
class Toto {
float x;
boolean b;
Toto h;
int f(double x) {
return x*2;
}
}
```

Est-ce que l'instruction suivante est correcte?

```
h.b= true;
```

.....

.....