POO
Introduction Générale

SAGIM 2024/2025

Qu’est-ce que la programmation orientée objet ?

Définition de la POO

La programmation orientée objet (**POO**) est un concept essentiel en programmation informatique.

Dans le jargon, on dit de la **POO** qu’elle est un paradigme,c'est-à-dire une approche, une méthode.

Le but de la **POO** consiste à définir et faire interagir entre eux des “objets” pour faciliter l’écriture et la maintenance du code.

Qu’est-ce que la programmation orientée objet ?

La P.O.O peut se définir comme **l’art** de **décomposer** une application en un certain nombre **d’objets** qui **communiquent entre eux** afin de réaliser une ou plusieurs tâches.

Autrement dit, si dans la programmation fonctionnelle, l’unité de travail est la fonction, alors, la programmation orientée se base sur **les objets**.

Histoire & origines de la POO

Définie par les norvégiens **Ole-Johan Dahl** et **Kristen Nygaard** au début de la décennie 1960, la programmation orientée objet a été davantage développée dans les années 1970 par l'américain **Alan Kay** qui pose et précise ses grands principes que l'on utilise encore aujourd'hui.

les principaux langages orientés objet ?

Les langages de POO dits « purs » : Conçus spécifiquement pour faci liter les méthodes orientés objet : Ex: Ruby, Scala, Smalltalk, Eiffel, JADE, Self, Raku.

Les langages conçus pour la programmation orientée objet, mais qui appliquent quelques principes procéduraux : Ex : Java, Python, C++, C#, Delphi/Object Pascal, VB.NET.

Les langages procéduraux avec certaines caractéristiques orientées objets. Ex : PHP, Perl, Visual Basic (dérivé de BASIC), MATLAB, C++, C #, COBOL 2002, Fortran 2003, ABAP, Ada 95, Pascal.

LE BUT DE LA POO :

Le but de la programmation orientée objet est de r éduire la difficulté de la tâche à accomplir .

En effet, et selon le principe « Diviser pour régner » , on décompose le problème initial en un grand nombre de petits problèmes qui sont plus simples à comprendre et à résoudre.

les avantages de la POO ?

- Faciliter le développement des applications informatiques
- Rendre plus facile la maintenance de ces applications
- Modularité: (regrouper les éléments homogènes dans un seul module)
- Minimiser le coût
- Réduire le temps de réalisation
- La ré-utilisabilité
- L’extensibilité
- L’approche objet a été inventée pour faciliter l’évolution d’applications complexes.

les concepts fondamentaux de la POO : Classe et Objets

Une classe, est un type de données abstrait, caractérisé par des propriétés (ses attributs) et des méthodes communes à des objets**.**

La classe : elle défini un « moule », un « modèle », un ‘’plan’’ selon lequel se ront conçus les objets.

Une **Classe** est un **modèle informatique** représentant une famille d’objets ayant :

– la même **structure** de données (même liste d’attributs)

– les mêmes **méthodes** (mêmes comportements).

les concepts fondamentaux de la POO : Classe et Objets

Un Objet est une instance d'une classe, c'est la concrétisation d'une classe

Un **objet** informatique, au sens de l'orienté objet, est une **unité atomique** possédant :

– une **identité** ;

– un **état**, défini par un ensemble de données (attributs, ou données membres) ;

– un **comportement**, défini par un ensemble de fonctions (méthodes, ou fonctions membres).

Ainsi: Objet = identité + état (attributs) + comportement (méthodes)

Un **Objet** peut correspondre à : **objet concret** du monde réel, ayant une réalité physique, ou a un **concept abstrait**;

les concepts fondamentaux de la POO : Classe et Objets

L'opération d'instanciation

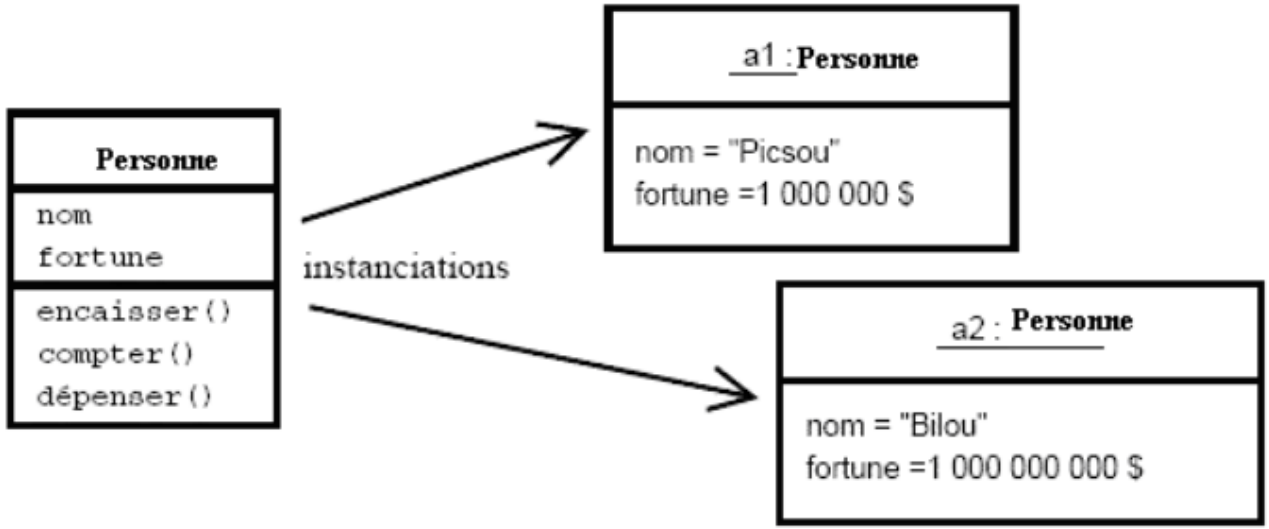
La Classe par elle-même ne contient pas les valeurs des données : c’est un type de données abstrait.

La création d’un objet en tant qu’exemplaire concret (contenant des données) d’une classe s’appelle une **INSTANCIATION**.

Chaque objet (instance d’une classe) donne des valeurs aux attrib uts de la classe.

les concepts fondamentaux de la POO : Classe et Objets

L'opération d'instanciation



Le langage Java

Est né en 1995 chez Sun Microsystems

- ➢ est orienté objet
- ➢ Toute variable doit être déclarée avec un type
- ➢ Le compilateur vérifie que les utilisations des variables sont compatibles avec leur type (notamment via un soustypage correct)
- ➢ est compilé
- ➢ En bytecode, i.e., code intermédiaire indépendant de la machine
- ➢ est interprété

  Le bytecode est interpété par une machine virtuelle Java

Le langage Java

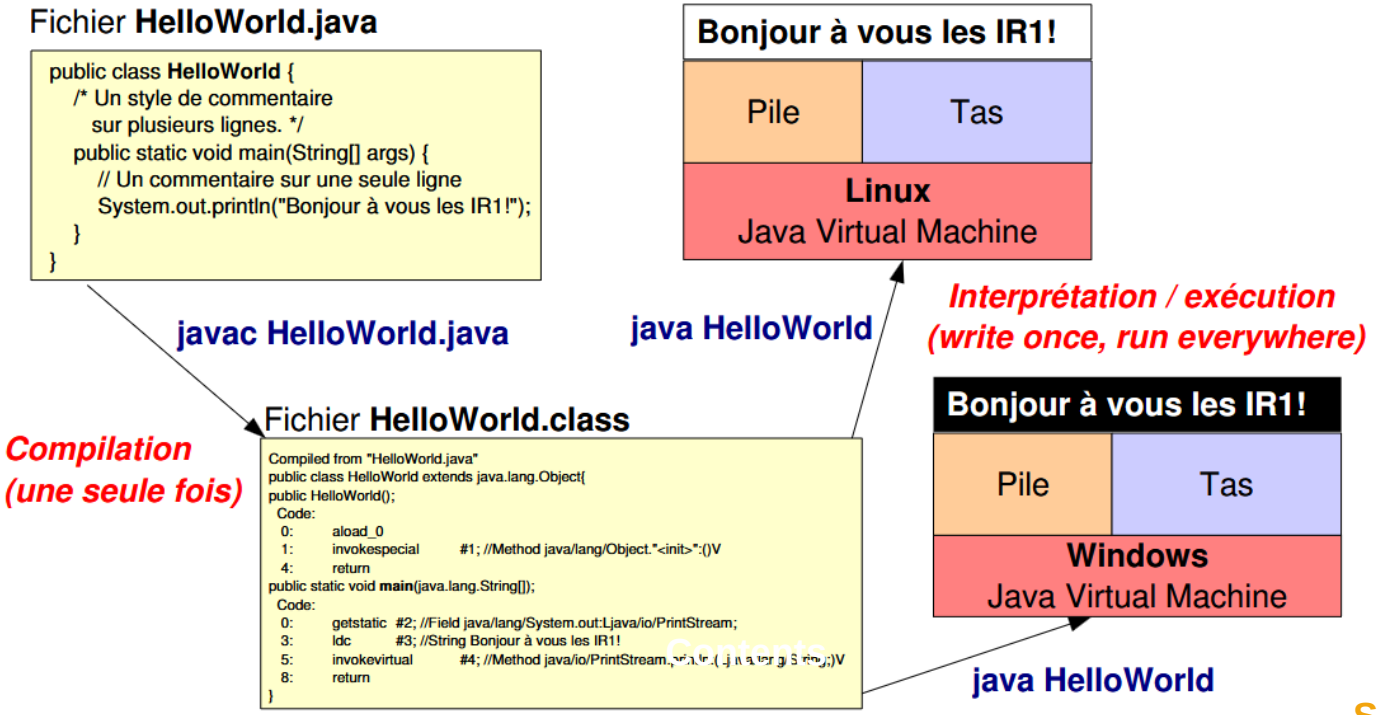
- Dans un fichier de nom `HelloWorld.java`
  - ***Règle: toute classe publique doit être dans un fichier qui a le même nom que la classe***
  - ***Règle: tout code doit être à l'intérieur d'une classe***

```
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 IR1!");
    }
}
```

  - Ça définit une classe, qui est une unité de compilation

Compilation, Bytecode et JVM

- Compilation du langage source -> exécution du bytecode



Bytecode

- Le ***langage source Java*** est défini par la JLS (*Java Language Specification*) éditée par Sun-Oracle
  - Dans sa syntaxe et sa sémantique
- Le code source d'une classe contenue dans un fichier est compilé avec la commande **javac**
  - Cela produit un code intermédiaire, appelé 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 comande **java**
  - Soit par ***interprétation***, soit par ***compilation*** « juste à temps » (*just-in-time* ou JIT)

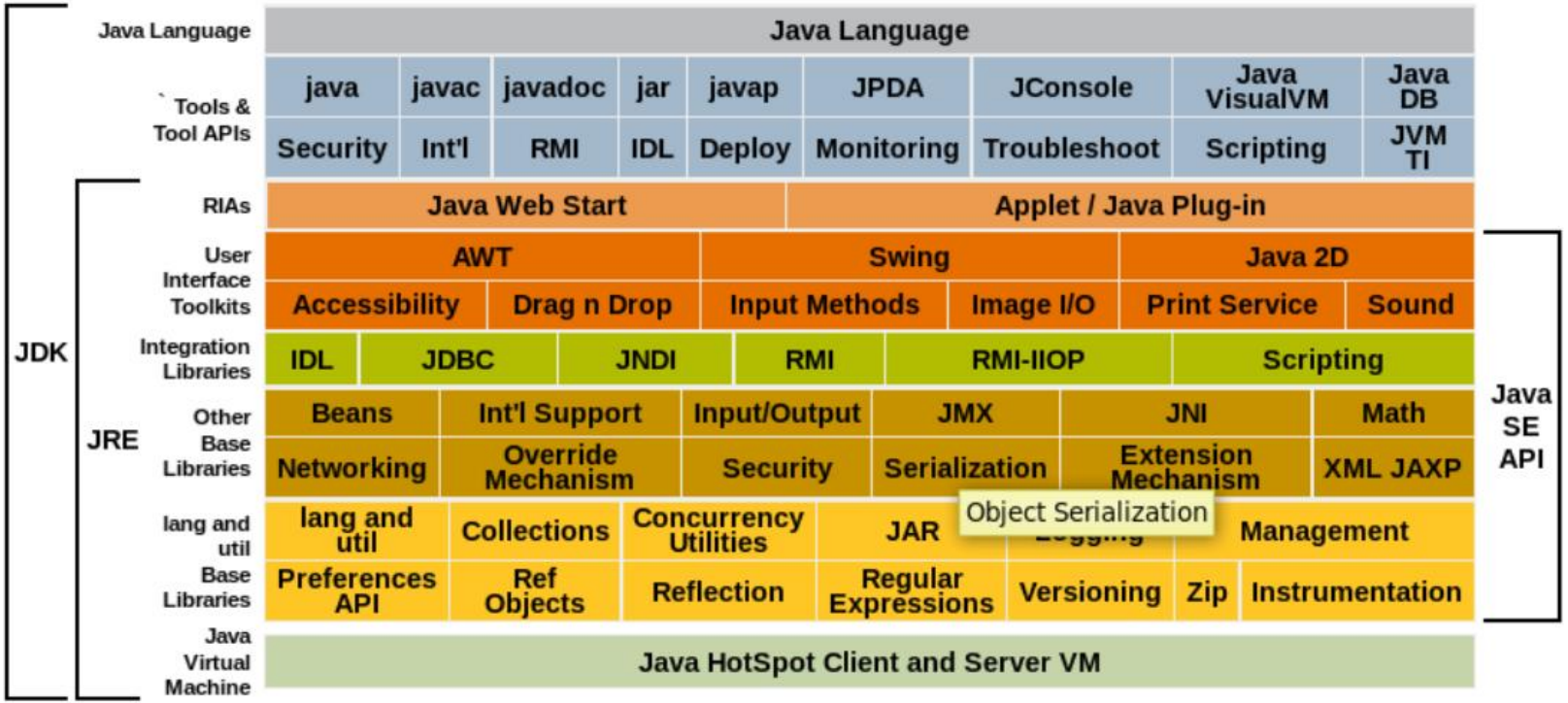
JVM

- Son rôle est d'**abstraire le comportement d'une machine**
  - Pour le rendre le + possible indépendant de la plateforme
  - Son comportement est défini par la JVM Spec édité par Sun-Oracle
- Une JVM est une implémentation de cette spec
  - Qui peut être adaptée à une plateforme d'accueil (Windows, Linux, Mac...)
  - Qui peut être développée par Sun (HotSpot: open source GPL depuis 2006) ou par d'autres: IBM, Jikes, etc.
- Une JVM **traduit** le **bytecode** dans le **langage machine** de la plateforme d'accueil

Java: un langage et une plateforme

- Dans la technologie Java, on a donc besoin
  - Du **langage de programmation** et du compilateur
    - Et plein de commandes bien utiles: jar, javap, javadoc, etc
  - De la **JVM et** des **APIs** (*Application Programming Interfaces*) regroupées dans une « plateforme »:
    - Java SE (Java Platform, Standard Edition): Java SE 6 pour applications classiques, desktop
    - Java EE (Java Platform, Enterprise Edition): Java EE 6 pour développer et déployer des applications serveur, Web services, etc.
    - Java ME (Java Platform, Micro Edition): J2ME pour les applications embarquées, PDA, téléphones, etc.
- Si on veut juste exécuter, il suffit du **JRE** (*Java Runtime Execution*) par opposition au **JDK** (*Java Developpement Kit*)

Java: un langage et une plateforme



Le langage Java

- Les variables, les opérateurs, les expressions, instructions, blocs, contrôle de flot sont très proches de ceux du C
  - Les exceptions sont une nouveauté
  - Les types primitifs ont une taille et une représentation normée
- S'y ajoutent des spécificités syntaxiques liées à la programmation objet, aux classes, à l'héritage...
- Un **style de nommage** (très fortement) conseillé
  - Style « chameau » (***CamelCase***) pour les indentificateurs
  - Première majuscule pour les classes (class HelloWorld)
  - Première minuscule pour les variables/champs et les fonctions/méthodes (radius, getRadius())
  - _Tout en majuscule pour les constantes (MAX_SIZE)

Le langage Java

- Les variables, les opérateurs, les expressions, instructions, blocs, contrôle de flot sont très proches de ceux du C
  - Les exceptions sont une nouveauté
  - Les types primitifs ont une taille et une représentation normée
- S'y ajoutent des spécificités syntaxiques liées à la programmation objet, aux classes, à l'héritage...
- Un **style de nommage** (très fortement) conseillé
  - Style « chameau » (***CamelCase***) pour les indentificateurs
  - Première majuscule pour les classes (class HelloWorld)
  - Première minuscule pour les variables/champs et les fonctions/méthodes (radius, getRadius())
  - _Tout en majuscule pour les constantes (MAX_SIZE)

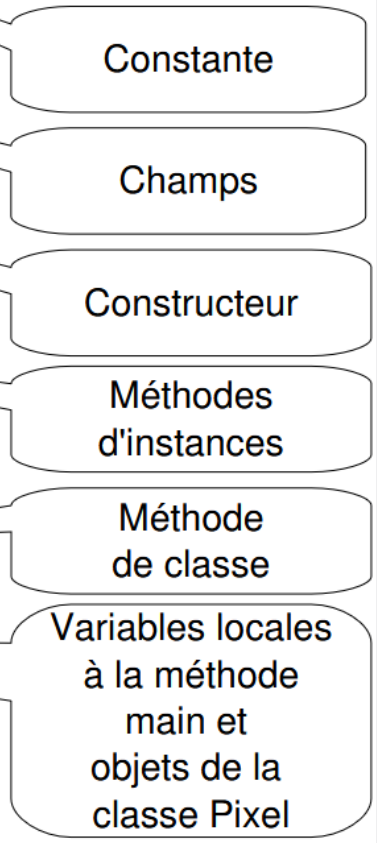
Classes et objets

- Une classe 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
    - Il peut servir à déclarer des variables comme Toto t;
  - Un **moule pour la création d'objets** de type Toto
    - Cela nécessite en général la définition d'un ensemble de **champs** (***fields***) décrivant **l'état** d'un objet de ce type et d'un ensemble de ***méthodes*** définissant son **comportement** ou ses focntionnalités
  - Chaque objet de la classe Toto
    - Dispose de son **propre état** (la valeur de ses champs)
    - Répond au **même comportement** (via les méthodes de la classe)

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 packge).
- Une classe peut contenir trois sortes de **membres**
  - Des **champs** (fields) ou attributs
  - Des **méthodes** (methods) et **constructeurs**
  - Des ***classes internes***
- Les membres statiques (static) sont dits **membres de classe**
  - Ils sont définis sur la classe et non sur les objets

```java
public class Pixel {
    public final static int ORIGIN = 0;
    private int x;
    private int y;
    public Pixel(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public void reset() {
        x = ORIGIN;
        y = ORIGIN;
    }
    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 static void main(String[] args) {
        Pixel p0 = new Pixel(0,0);
        Pixel p1 = new Pixel(1,3);
        p1.printOnScreen();                // (1,3)
        System.out.println(same(p0,p1)); // false
        p1.reset();
        System.out.println(same(p0,p1)); // true
    }
}
```



Écrire une fonction qui calcule la factorielle d’un entier positif donné.

Exercices & introduction

```
public class Factorielle {
    public static int factorielle(int n) {
        int resultat = 1;
        for (int i = 1; i <= n; i++) {
            resultat *= i;
        }
        return resultat;
    }
    public static void main(String[] args) {
        int nombre = 5;
        System.out.println("La factorielle de " + nombre + " est : " + factorielle(nombre));
    }
}
```

Écrire un programme Java qui demande à l'utilisateur de saisir un entier, puis affiche s’il est pair ou impair.

Exercices & introduction

```
import java.util.Scanner;
public class PairOuImpair {
    public static void main( String [ ] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Entrez un entier : ");
        int nombre = scanner.nextInt();

        if (nombre % 2 == 0) {
            System.out.println("Le nombre est pair.");
        } else {
            System.out.println("Le nombre est impair.");
        }
    }
}
```

Écrire une fonction qui prend une chaîne de caractères et retourne la chaîne inversée.

Exercices & introduction

```
public class InverserChaine {
    public static String inverser(String chaine) {
        String resultat = "";
        for (int i = chaine.length() - 1; i >= 0; i--) {
            resultat += chaine.charAt(i);
        }
        return resultat;
    }

    public static void main(String[ ] args) {
        String texte = "Bonjour";
        System.out.println("Chaîne inversée : " + inverser(texte));
    }
}
```

Écrire une méthode qui retourne le plus grand élément d’un tableau d’entiers.

Exercices & introduction

```
public class MaxTableau {
    public static int max(int[ ] tableau) {
        int max = tableau[0];
        for (int i = 1; i < tableau.length; i++) {
            if (tableau[i] > max) {
                max = tableau[i];    }   }
        return max;
    }
public static void main(String[ ] args) {
        int [ ] nombres = {3, 9, 12, 5, 1};
        System.out.println("Le plus grand nombre est : " + max(nombres));
    }
}
```