**Java Orienté Objet**

Sommaire

[Introduction : 2](#_Toc529385053)

[Chapitre 1 : **Programmation Orienté Objet** 2](#_Toc529385054)

[1- La classe 2](#_Toc529385055)

[Méthode constructeur : 3](#_Toc529385056)

[on doit explicitement le déclarer le constructeur et on peut l’utiliser par défaut 3](#_Toc529385057)

[le nombre de constructeur selon le nombre des attributs. 3](#_Toc529385058)

[Le GC (Garbage collector ) : 4](#_Toc529385059)

[2- L’objet : 4](#_Toc529385060)

[Chapitre 2 : **L’héritage** 4](#_Toc529385061)

[Principe de l’héritage 4](#_Toc529385062)

[Polymorphisme : 5](#_Toc529385063)

[Chapitre 5: **Les interfaces** 6](#_Toc529385064)

[Chapitre 6 : **La classe abstraites.** 6](#_Toc529385065)

[Classes abstraites : 6](#_Toc529385066)

## Introduction :

Chaque langage de programmation appartient à une “famille” de langages définissant une approche ou une méthodologie générale de programmation, la programmation orientée-objet, propose une méthodologie centrée sur les données. Le programmeur Java va d’abord identifier un ensemble d’objets, tel que chaque objet représente un élément qui doit être utilisé ou manipulé par le programme, sous la forme d’ensembles de données. Ce n’est que dans un deuxième temps, que le programmeur va écrire les traitements, en associant chaque traitement à un objet donné.

## Chapitre 1 : **Programmation Orienté Objet**

1. La classe:

Un objet est une variable (presque) comme les autres. Il faut notamment qu’il soit déclaré avec son type. Le type d’un objet est un type complexe (par opposition aux types primitifs entier, caractère, …) qu’on appelle une classe.

Une classe peut regrouper trois types de membre :

* Classe qui stocke les données ;
* Classe qui regroupes des méthodes et constructeur ;
* Classe qui contient elle-même une sous-classe interne.

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

**Encapsulation** Lors de la conception d’un programme orienté-objet, le programmeur doit identifier les objets et les données appartenant à chaque objet mais aussi des droits d’accès qu’ont les autres objets sur ces données. L’encapsulation de données dans un objet permet de cacher ou non leur existence aux autres objets du programme. Une donnée peut être déclarée en accès :

* public : les autres objets peuvent accéder à la valeur de cette donnée ainsi que la modifier ;
* privé : les autres objets n’ont pas le droit d’accéder directement à la valeur de cette donnée (ni de la modifier), dans ce cas pour afficher la valeur de l’attribue on a besoin des Guetteurs et Setteur

Guetteur : méthodes public return Int

Exemple : public int get Id( ){

Setteur : méthodes public qui return void

Exemple : public void set Id( ){

Méthode constructeur : Chaque classe doit définir une ou plusieurs méthodes particulières appelées des constructeurs. Un constructeur est une méthode invoquée lors de la création d’un objet. Cette méthode, qui peut être vide, effectue les opérations nécessaires à l’initialisation d’un objet. Chaque constructeur doit avoir le même nom que la classe où il est défini et n’a aucune valeur de retour (c’est l’objet créé qui est renvoyé). Dans l’exemple précédent de la classe rectangle, le constructeur initialise la valeur des données encapsulées :

class Rectangle {

Rectangle(int lon, int lar) {

this.longueur = lon ;

this.largeur = lar ;

this.origine\_x = 0 ;

this.origine\_y = 0 ;

}

}

on doit explicitement le déclarer le constructeur et on peut l’utiliser par défaut

le nombre de constructeur selon le nombre des attributs.

## Le GC (Garbage collector ) :

Ou le collecte des miettes son rôle est récupérer Les objets qui ne sont plus référencés pour les recycler pour ne pas prendre place dans la mémoire.

1. L’objet :

Un objet est une instance (anglicisme signifiant « cas » ou « exemple ») d’une classe et est référencé par une variable ayant un état (ou valeur). Pour créer un objet, il est nécessaire de déclarer une variable dont le type est la classe à instancier, puis de faire appel à un constructeur de cette classe.

Type objet valeur par défaut :nul

Type variable par défaut :1

Type boolean par défaut : False

## Chapitre 2 : **L’héritage**

Principe de l’héritage :

L’idée principale de l’héritage est d’organiser les classes de manière hiérarchique. La relation d’héritage est unidirectionnelle et, si une classe B hérite d’une classe A, on dira que B est une sous-classe de A. Cette notion de sous-classe signifie que la classe B est un cas particulier de la classe A et donc que les objets instanciant la classe B instancient également la classe A. Prenons comme exemple des classes Carre, Rectangle et Cercle. propose une organisation hiérarchique de ces classes telle que Carre hérite de Rectangle qui hérite, ainsi que Cercle, d’une classe Forme

La classe Rectangle héritant d’une classe vide, elle ne peut profiter d’aucun de ses attributs et doit définir toutes ses variables et méthodes. Une relation d’héritage se définit en Java par le mot-clé extends utilisé comme dans l’exemple suivant :

public class Rectangle extends Forme {

private int largeur ;

private int longueur ;

public Rectangle(int x, int y) {

this.largeur = x ;

this.longueur = y ;

}

public int getLargeur() {

return this.largeur ;

}

public int getLongueur() {

return this.longueur ;

}

public int surface() {

return this.longueur \* this.largeur ;

}

public void affiche() {

System.out.println(”rectangle ” + longueur + ”x” + largeur);

}

}

# Polymorphisme :

Le polymorphisme est la faculté attribuée à un objet d’être une instance de plusieurs classes. Il a une seule classe “réelle” qui est celle dont le constructeur a été appelé en premier peut aussi être déclaré avec une classe supérieure à sa classe réelle. Cette propriété est très utile pour la création d’ensembles regroupant des objets de classes différentes comme dans

## Chapitre 5: **Les interfaces**

Une interface est un type, au même titre qu’une classe, mais abstrait et qui donc ne peut être instancié (par appel à new plus constructeur). Une interface décrit un ensemble de signatures de méthodes, sans implémentation, qui doivent être implémentées dans toutes les classes qui implémentent l’interface. L’utilité du concept d’interface réside dans le regroupement de plusieurs classes, tel que chacune implémente un ensemble commun de méthodes, sous un même type. Une interface possède les caractéristiques suivantes :

● elle contient des signatures de méthodes ;

● elle ne peut pas contenir de variables ;

● une interface peut hériter d’une autre interface (avec le mot-clé extends)

● une classe (abstraite ou non) peut implémenter plusieurs interfaces.

La liste des interfaces implémentées doit alors figurer après le mot-clé implements placé dans la déclaration de classe, en séparant chaque interface par une virgule.

## Chapitre 6 : **La classe abstraites.**

# Classes abstraites :

Le concept de classe abstraite se situe entre celui de classe et celui d’interface. C’est une classe qu’on ne peut pas directement instancier car certaines de ses méthodes ne sont pas implémentées. Une classe abstraite peut donc contenir des variables, des méthodes implémentées et des signatures de méthode à implémenter. Une classe abstraite peut implémenter (partiellement ou totalement) des interfaces et peut hériter d’une classe ou d’une classe abstraite. Le mot-clé abstract est utilisé devant le mot-clé class pour déclarer une classe abstraite, ainsi que pour la déclaration de signatures de méthodes à implémenter. Imaginons que l’on souhaite attribuer deux variables, origine\_x et origine\_y, à tout objet représentant une forme.