Table des matières

[9. Classes et objets 2](#_Toc93307492)

[A/ Définition 2](#_Toc93307493)

[B/ Notes 5](#_Toc93307494)

[10. Propriétés des objets et des méthodes 7](#_Toc93307495)

[A/ Définition 7](#_Toc93307496)

[B/ Notes 8](#_Toc93307497)

# 9. Classes et objets

## A/ Définition

*Langages procéduraux :*

Disposant de la notion de fonction, outil qui permet de structurer un programme en le décomposant en des parties relativement indépendantes.

*Langages objet :*

Disposant en plus des notions de classe et d’objet ; comme nous le verrons, les classes permettent également de structurer un programme en le décomposant en des parties autonomes.

*Objet :*

Consiste à regrouper dans une même entité des données qu’on nomme des attributs (ou encore des champs) et des fonctions qu’on nomme méthodes (ou, parfois, fonctions membres).

*Attributs :*

Variables de la classe

*Attributs encapsulés :*

Ne sont plus visibles « de l’extérieur » de l’objet (ils sont privés)

*Classes :*

Une description (unique) pouvant donner naissance à différents objets disposant de la même structure de données (mêmes noms et types d’attributs) et des mêmes méthodes.

*Instanciation :*

Création de l’objet

*Méthodes d’accès :*

Obtenir la valeur d’un attribut donné

*Méthodes d’altération :*

Modifier la valeur d’un ou de plusieurs attributs

*Interface d’une classe :*

Informations dont on doit pouvoir disposer pour pouvoir l’utiliser :

• du nom de la classe

• de la signature (nom de la fonction et type des paramètres) et du type du résultat éventuel de chacune de ses méthodes

*Contrat d’une classe :*

Interface et définition du rôle de ses méthodes

*Implémentation d’une classe :*

Ensemble des instructions de la classe, écrites en vue de réaliser le contrat voulu

*Constructeur :*

Une méthode particulière de la classe, portant le nom de la classe elle-même qui a pour but d’initialiser les attributs de l’objet (appelé lorsque l’objet est créé)

• Un constructeur peut disposer de paramètres.

• Ce constructeur sera appelé au moment de la création de l’objet et il sera possible, le cas échéant, de lui fournir les paramètres souhaités.

*Surdéfinition :*

Définition une nouvelle (constructeur appelé une nouvelle fois) fois de l’objet

*Gestion par référence :*

L’objet voit son emplacement alloué dynamiquement au moment de sa création (manipule la référence à l’objet)

*Gestion par valeur :*

La seule déclaration d’un objet entraîne la réservation de l’emplacement mémoire correspondant (manipule directement l’objet)

## B/ Notes

Seules les méthodes sont habilitées à manipuler les données, qu’il s’agisse de les modifier ou plus simplement d’en utiliser la valeur

***Point*** *p* réserve simplement un emplacement pour une variable nommée p, destinée à recevoir la référence (adresse) d’un objet de type ***Point***.

*p* est une variable de type ***Point***, tandis que l’objet référencé par *p* est un objet de type ***Point***



Une classe peut tout à fait modifier son implémentation, sans que ceci n’ait de conséquences sur son utilisation (à condition, bien sûr de respecter le contrat !)

Il est très important de noter que l’instanciation d’un objet, réalisée par un appel tel que :

*Création Point (3,5)* réalise deux opérations :

– allocation d’un emplacement mémoire pour un objet de type Point

– appel éventuel du constructeur pour cet objet

Ces deux opérations sont indissociables. Le constructeur ne peut pas être appelé directement (sur un objet existant), en court-circuitant la première opération

Un constructeur peut très bien réaliser d’autres actions, par exemple : allocation d’emplacements dynamiques, vérification d’existence de fichier, ouverture d’une connexion Interne

Lorsqu’une classe dispose d’au moins un constructeur, il n’est plus possible d’instancier un objet, sans qu’il y ait appel de l’un des constructeurs

Pas de mécanisme permettant à un constructeur de fournir un résultat

Aucun mécanisme permettant d’initialiser les attributs d’un objet.

Par exemple, ceci est interdit : class X {

int n = 5 ; // interdit

…….

}

Tant qu’on en respecte l’interface, on peut modifier à volonté l’implémentation d’une classe

# 10. Propriétés des objets et des méthodes

## A/ Définition

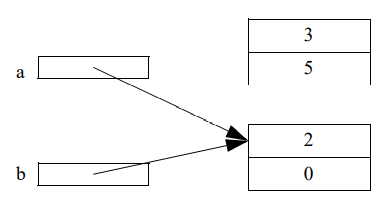
## B/ Notes

Objets gérés par référence

Instruction d’affectation peuvent s’appliquer à des variables d’un type objet

*a = b*

Elle va simplement recopier dans a la référence contenue dans b, ce qui nous conduit à cette situation :



Autrement dit, dorénavant, a et b désignent le même objet, et non pas deux objets de même valeur. Toute modification effectuée sur cet objet, par le biais de *a* se retrouvera sur *b*

La comparaison porte sur les références et non sur les valeurs des objets. L’égalité n’a donc lieu que si les références désignent un même objet et non pas deux objets de même valeur :

*Point p, q, r*

*p = Création Point (2, 5)*

*q = Création Point (2, 5)*

*si (p = q) alors ...... // ici, cette condition est fausse*

*r = p*

*si (p = r) alors ..... // en revanche, celle-ci est vraie*

Si langage gère objet par valeurs :

* Dans ce cas, une affectation telle que *a = b* (*a* et *b* étant deux objets du type Point) effectue une recopie de l’ensemble des valeurs des attributs de b dans ceux de *a*
* La comparaison porte sur les valeurs des attributs ; deux objets différents ayant les mêmes valeurs d’attributs apparaîtront égaux :

*Point a (3, 8), b (3, 8)*

*si (a = b)* ...... // dans un langage gérant les objets par valeur, cette relation sera vraie

Il est possible de définir des variables locales de type objet

Objet existe toujours alors que la référence peut disparaitre (créée dans une méthode)

Les objets transmis en paramètre à une fonction ou à une méthode sont une transmission par valeur. Mais, il faut bien réaliser que cette valeur est en fait une référence, de sorte que la méthode (ou la fonction) reçoit en fait la copie de la référence à l’objet correspondant qu’elle peut donc tout à fait modifier

*Point a*

*a = Création Point (3, 8)*

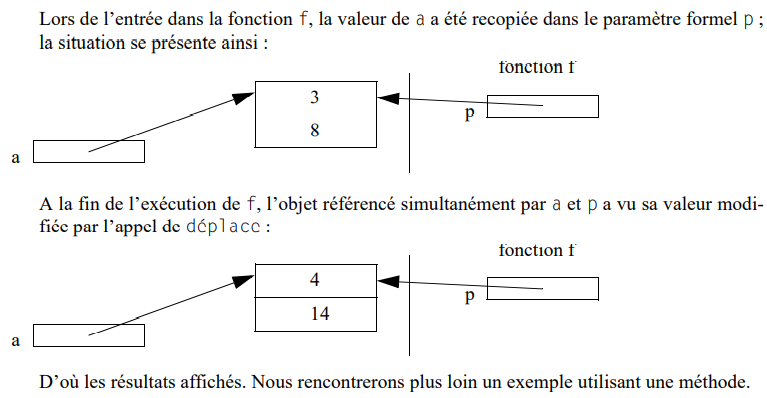
*f (a)*

*a.affiche*

*f (Point p)*

*{ p.deplace (1, 6)*

*}*



L’unité d’encapsulation est la classe et non l’objet