

FORMATION

Algorithmique et programmation
structurée - 35h

Romain DINEL - 2024

Présentation

Développeur Full-Stack - Banque Populaire et
Caisse d'Epargne depuis 2023

Formateur depuis 2022

Développeur PHP / JS de 2020 à 2022

Horaire et fonctionnement

- 9h - 17h
- 15 mn de pause le matin
- 1h de pause déjeuner
- 15 mn de pause l'après midi

- Utilisation de Google Form

Table des matières

- Chapitre 1 : Introduction à l'algorithmique
- Chapitre 2 : Les variables
- Chapitre 3 : Expressions et opérateurs
- Chapitre 4 : Les instructions conditionnelles
- TP de validation des acquis

- Chapitre 5 : Les boucles
- Chapitre 6 : Les fonctions
- Chapitre 7 : Les tableaux
- TP de validation des acquis

- Chapitre 8 : Programmation procédurale et POO
- Chapitre 9 : Introduction à l'UML
- TP de validation des acquis

- Chapitre 11 : Introduction à Java
- TP de validation des acquis

Chapitre 1 : Introduction à l'algorithmique

Un algorithme c'est quoi ?

[Selon la CNIL](#)

Un algorithme est la description d'une suite d'étapes permettant d'obtenir un résultat à partir d'éléments fournis en entrée.

Un algorithme c'est quoi ?

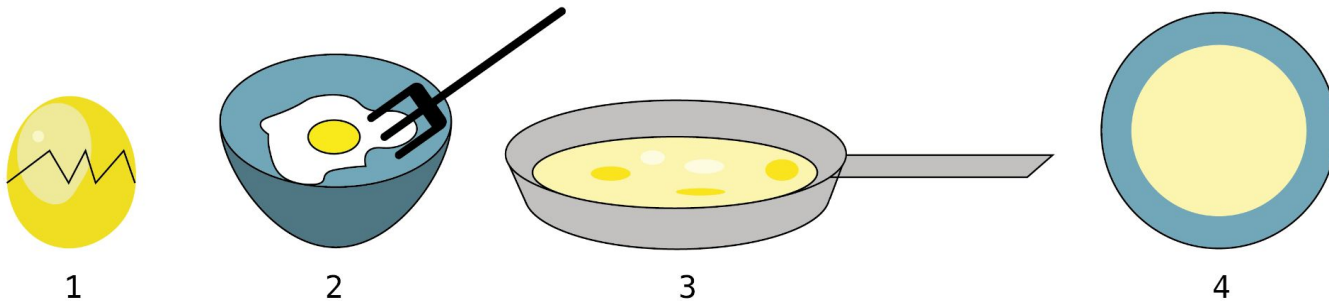
Procédure de calcul bien définie qui prend en entrée une valeur, ou un ensemble de valeurs, et qui donne en sortie une valeur, ou un ensemble de valeurs.

- Une séquence **d'étapes de calcul** qui transforment **l'entrée** en **sortie**.
- Une “recette” à suivre pour résoudre un problème donné
- Décomposer un problème complexe en sous-problèmes plus simples

Un premier algorithme

1. Casser les œufs dans un bol.
2. Mélanger les œufs jusqu'à obtenir un mélange homogène.
3. Cuire le mélange d'œufs dans une poêle à température moyenne.
4. Lorsque cuite, glisser l'omelette dans une assiette.

A votre avis quelle est l'entrée et la sortie de cet algorithme ?

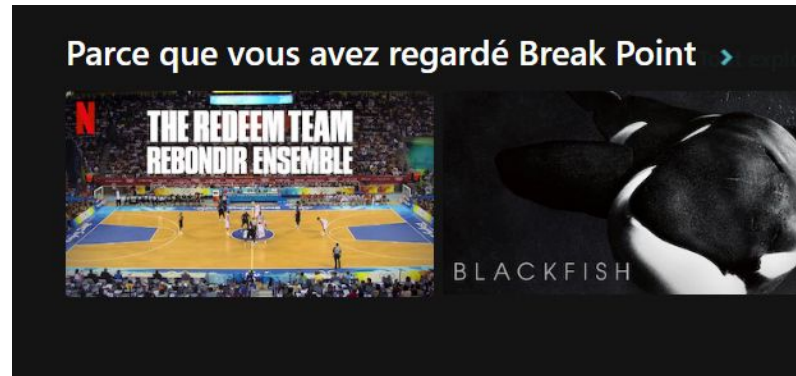


Les problèmes d'algorithmique

1. Quelle est la **complexité** de l'algorithme, en combien de temps va t-il atteindre le résultat attendu ?
2. L'algorithme répond t-il au problème posé, dans tous les cas de figure possible de ce problème ? (**Correction**)

Cas d'utilisations d'algorithmes

- Chemin le plus court pour aller d'un point A à B
- Recommandations Netflix
- Programmation d'un robot cuiseur
- Recommandation d'achats sur Amazon en fonction des choix de l'utilisateur
- ...



Algorithmique et langage de programmation

[Selon Wikipedia](#)

- Un langage de programmation est un langage informatique destiné à **formuler des algorithmes et produire des programmes informatiques** qui les appliquent
- Un programme informatique est un ensemble d'instructions et d'opérations destinées à être exécutées par un ordinateur.

NO CODE

```
Algorithme EcrireAlgorithme
    afficher "A"
    afficher "\n"
    afficher "g"
    afficher "o"
    afficher "r"
    afficher "i"
    afficher "t"
    afficher "h"
    afficher "m"
    afficher "e"
```

Fin

JAVA

```
public void afficherAlgorithme() {
    System.out.print("A");
    System.out.print("\n");
    System.out.print("g");
    System.out.print("o");
    System.out.print("r");
    System.out.print("i");
    System.out.print("t");
    System.out.print("h");
    System.out.print("m");
    System.out.print("e");
}
```

PYTHON

```
def afficher_algorithme():  
    print("A", end="")  
    print("l", end="")  
    print("g", end="")  
    print("o", end="")  
    print("r", end="")  
    print("i", end="")  
    print("t", end="")  
    print("h", end="")  
    print("m", end="")  
    print("e", end="")
```

JAVASCRIPT

```
function afficherAlgorithme() {  
    process.stdout.write("A");  
    process.stdout.write("l");  
    process.stdout.write("g");  
    process.stdout.write("o");  
    process.stdout.write("r");  
    process.stdout.write("i");  
    process.stdout.write("t");  
    process.stdout.write("h");  
    process.stdout.write("m");  
    process.stdout.write("e");  
}
```

Compilation et interprétation

- La compilation d'un programme consiste à transformer toutes les instructions en langage machine avant que le programme puisse être exécuté. Par conséquent il sera nécessaire de refaire la compilation après chaque modification du code source.
- Si un langage n'est pas compilé, il est nécessaire d'utiliser un interprète qui traduit les instructions en temps réel (on run time). Dans ce cas le code source est lu à chaque exécution et par conséquent les changements apportés au code seront pris en compte directement.

Langage de programmation

L'exécution d'un programme passe par :

- Un compilateur
- Un Interpréteur

On écrit les instructions avec un langage de programmation.

```
public void Test {  
    System.out.print('Hello World');  
}
```

Le compilateur
traduit vers le
langage machine

Le processeur comprend ce qu'on lui
demande et le traduit dans son
propre langage

```
100111011101010100100001110  
010010001111010101010101010  
101010
```


Langage de programmation

1. La compilation vérifie la “syntaxe” du programme
2. L'exécution déroule les instructions du programme

Un programme qui n'est syntaxiquement pas correct n'est pas exécuté

Un programme syntaxiquement correct n'est pas un programme correct.

Un programme correct est un programme qui produit le résultat attendu.

Un exemple : Le fichier source Chaine.java

```
public class Chaine {  
    private String name;  
    public Chaine(String name) {  
        this.name = name;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName (String name) {  
        this.name = name;  
    }  
    public static void main (String[] args) {  
        Chaine ch;  
        ch = new Chaine("Bonjour");  
        System.out.println(ch.getName());  
        ch.setName("Au revoir !");  
        System.out.println("ch.getName()");  
    }  
}
```

La compilation :

> javac Chaine.java

ls -l

... Chaine.class

... Chaine.java

L'exécution :

> java Chaîne

Bonjour !

Au revoir !

Différents types de langages

Langage de bas niveau

- Plus proche du langage machine (binaire)
- Généralement plus difficile à apprendre
- Possède plus de possibilité d'interactions avec la machine



Langage de haut niveau

- Plus proche du langage humain
- Généralement plus simple à apprendre
- Interactions plus limitées avec les fonctionnalités que met le langage à disposition

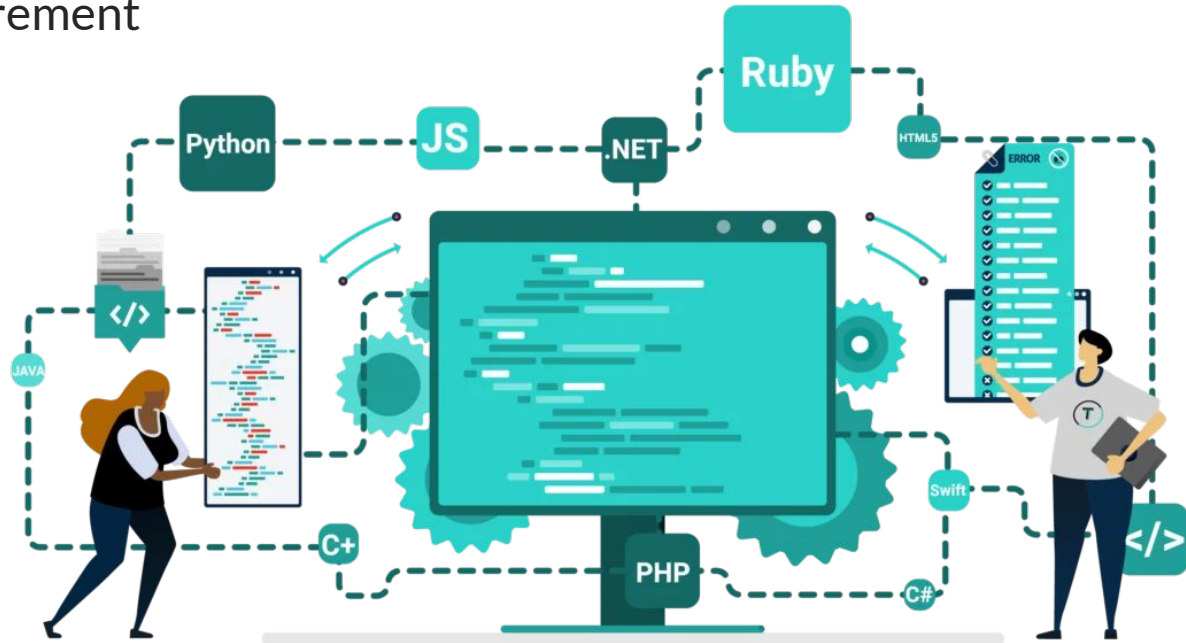


Les éléments fondamentaux

- Les variables
- Les opérateurs
- Les instructions conditionnelles
- Les boucles
- Les fonctions
- Les tableaux (ou array)
- Les objets

Apprentissage d'un langage : les bases

- Comprendre les bases du langage
- Apprendre à apprendre
- [Consulter la documentation](#)
- Pratiquer régulièrement



Exercices

Pour chaque problème, énoncer les données d'entrée et de sortie.

1. Addition de deux nombres
2. Calcul d'une moyenne d'un élève selon ses notes
3. Recherche de la valeur maximale parmi plusieurs nombres
4. Vérifier si un nombre est premier
5. Convertir une température d'une unité de mesure à une autre
6. Maximiser la valeur totale des objets que l'on peut mettre dans un sac à dos, en respectant sa capacité maximale.
7. Planifier la production pour maximiser le profit d'une entreprise, en tenant compte des contraintes, de la ressources et du temps.

Exercices - Correction

- Addition de deux nombres

Entrée : Deux nombres

Sortie : La somme de ces deux nombres

Exemple d'entrée : 5,9 - Exemple de sortie : 14

Exercices - Correction

- Calcul d'une moyenne

Entrée : Une liste de nombres

Sortie : La moyenne des nombres

Exemple d'entrée : 50, 23, 17, 18 - Exemple de sortie : 27

Exercices - Correction

- Recherche de la valeur maximale parmi plusieurs nombres

Entrée : Une liste de nombres

Sortie : La valeur maximale d'un ou de plusieurs des nombres

Exemple d'entrée : 50, 23, 17, 18 - Exemple de sortie : 50

Exercices - Correction

- Vérifier si un nombre est premier

Entrée : Un nombre à tester

Sortie : Vrai si le nombre est premier, Faux si le nombre ne l'est pas

Exemple d'entrée : 2 - Exemple de sortie : Vrai

Exercices - Correction

- Convertir une température d'une de mesure à une autre

Entrée : Un nombre, l'unité de mesure correspondant, l'unité de mesure ciblée

Sortie : Un nombre avec sa température convertie

Exemple d'entrée : 5, "°C ", "°F" - Exemple de sortie : 41

Exercices - Correction

- Maximiser la valeur totale des objets que l'on peut mettre dans un sac à dos, en respectant sa capacité maximale.

Entrée : Une liste d'objets avec leur poids, ainsi qu'un nombre représentant la capacité maximale du sac à dos

Sortie : Une liste d'objets à mettre dans le sac à dos en ayant respecté sa capacité maximale

Exemple d'entrée : Objet1(Poids:50kg), Objet2(Poids:25kg), Objet3(Poids:10kg),
Capacité du sac : 75kg

Exemple de sortie : Objet1(Poids:50kg), Objet2(Poids:25kg)

Exercices - Correction

- Planifier la production pour maximiser le profit d'une entreprise, en tenant compte des contraintes, de la ressources et du temps.

Entrée : Une liste de contraintes, une liste de ressources, une liste de coûts, les prix de vente, une liste de délais, etc.

Sortie : Un plan de production optimale qui maximise le profit de l'entreprise en respectant les contraintes et les ressources disponibles, sous forme de liste par exemple

Introduction à l'algorithmique : ce qu'il faut retenir

- Un algorithme est un ensemble d'instructions à suivre pour résoudre un problème
- Les langages de programmation permettent la formulation d'algorithmes sous différentes formes
- L'apprentissage se fait par la pratique, et l'utilisation de la documentation est essentielle.



Commit chapitre 1 : Introduction à l'algorithmique

- Lien vers mon github
- Le commentaire du commit est “chapitre 1 introduction à l’algorithmique”

Installation d'Algobox



Installation d'Algobox

- Lien d'installation : <https://www.xm1math.net/algobox/download.html>

Algobox - Instructions

- **Déclaration** : Il faut toujours commencer par déterminer les variables nécessaires à l'algorithme. Une variable ne pourra être utilisée dans Algobox que si elle est déclarée au préalable. On utilise pour cela le bouton "Déclarer une nouvelle variable". Les variables peuvent être de trois types : nombres (réels ou entiers), chaîne (mots) ou liste ; pas de booléen !
- **Affectation** : Pour affecter une valeur à une variable, on utilise le bouton "AFFECTER valeur à variable".
- **Entrée d'une valeur** : Pour que l'utilisateur puisse entrer une valeur au cours de l'algorithme (et l'affecter à une variable), on utilise le bouton "Ajouter LIRE variable".
- **Affichage d'une valeur** : Pour afficher à l'écran la valeur d'une variable, on utilise le bouton "Ajouter AFFICHER variable". S'il s'agit d'afficher un message (par exemple une phrase expliquant ce que l'utilisateur doit faire), on utilise le bouton "Ajouter AFFICHER message".

Chapitre 2 : Les variables

Une définition d'une variable

- Une variable est un espace de stockage nommé utilisé pour contenir des données dans un programme.
- Les variables permettent de manipuler et de modifier des données tout au long de l'exécution du programme.
- Une variable possède un **nom** et un **type**

Exemple : Age \leftarrow 36

Variables et manipulation des données

- Les variables sont essentielles pour stocker des informations telles que des nombres, des chaînes de caractères, des résultats de calculs, etc.
- Elles permettent de rendre les programmes flexibles et génériques en manipulant des données de manière dynamique.
- Les variables sont utilisées à différentes étapes de l'algorithme pour stocker des données temporaires ou permanentes.
- Elles peuvent être déclarées au début de l'algorithme ou à mesure que le besoin se fait sentir lors de l'exécution.

Variables et manipulation des données

Algorithme MonAlgorithme

afficher("Donnez-moi le prix hors taxe :")

saisir(**prixHT**)

TVA \leftarrow 20 {taux de TVA en pourcentage}

prixTTC \leftarrow **prixHT** * (1 + **TVA** / 100) {calcul du prix TTC}

montantTVA \leftarrow **prixTTC** - **prixHT**

Titre \leftarrow "Résultat du calcul de TVA"

afficher(**Titre**) {présentation du résultat}

afficher(**prixHT**, " euros H.T. + **TVA** ", **TVA**, " devient ",
prixTTC, " euros T.T.C.")

Fin

Variables et manipulation des données

Cet algorithme permet à l'utilisateur de saisir un prix hors taxe, calcule le prix toutes taxes comprises en utilisant un taux de TVA fixé, puis affiche le résultat avec une description appropriée.

- **prixHT** : Variable pour stocker le prix hors taxe saisi par l'utilisateur.
- **TVA** : Variable pour stocker le taux de TVA en pourcentage (ici, 20%).
- **prixTTC** : Variable pour stocker le prix TTC
- **montantTVA** : Variable pour stocker le montant de la TVA.
- **Titre** : Variable pour stocker le titre de la section de présentation du résultat.

Quelles sont les variables ?

Algorithme CalculSomme

afficher("Entrez le premier nombre :")

saisir(nombre1)

afficher("Entrez le deuxième nombre :")

saisir(nombre2)

somme \leftarrow nombre1 + nombre2

Titre \leftarrow "Calcul de la somme de deux nombres"

afficher(Titre)

afficher("La somme de ", nombre1, " et ", nombre2, " est ", somme)

Fin

Quelles sont les variables ?

Algorithme CalculSomme

// Étape 1 : Demander à l'utilisateur d'entrer deux nombres

afficher("Entrez le premier nombre :")

saisir(**nombre1**)

afficher("Entrez le deuxième nombre :")

saisir(**nombre2**)

// Étape 2 : Calculer la somme des deux nombres

somme ← **nombre1** + **nombre2**

// Étape 3 : Afficher le résultat

Titre ← "Calcul de la somme de deux nombres"

afficher(**Titre**)

afficher("La somme de ", **nombre1**, " et ", **nombre2**, " est ", **somme**)

Fin

Types de Variables

“Entier”

- Stock des nombres entiers sans décimale
- Exemple : `age = 25`

Types de Variables “Flottant”

- Stock des nombres à virgule, incluant des décimales
- Exemple : $\pi = 3.14$

Types de Variables “Boolean”

- Stock des valeurs de “vérité”, soit vrai soit faux (True / False)
- Exemple : `est_majeur = True`

Types de Variables

“Chaîne de caractères”

- Stock des séquences de caractères comme des mots et phrases
- Exemple : nom = “Dupont”

Types de Variables “Tableaux” et “Listes”

- Stock des collections de données.
- Permettent d’accéder à des éléments individuels en utilisant un index.
- Exemple : nombres = [1, 12, 15, 8, 41]

Les variables : ce qu'il faut retenir

- Les variables sont des outils indispensables dans un algorithme et permettent de stocker des données
- Il existe différents types de variables : Entiers, Flottants, Boolean, Chaînes de caractères, Tableaux.
- Elles sont utilisées à toutes étapes d'un algorithme



Commit chapitre 2 : variables

- Lien vers mon github
- Le commentaire du commit est “chapitre 2 Variables”

Chapitre 3 : Expressions et opérateurs

Expression

Une expression est une combinaison de valeurs, de variables, et/ou d'opérateurs qui peut être évaluée pour produire un résultat.

Exemple : $\text{totalTTC} \leftarrow \text{prixHT} * 10$

Affectation

L'affectation est une opération qui consiste à attribuer une valeur à une variable. En algorithmique, elle est généralement représentée par le symbole « \leftarrow » ou « $:=$ ».

Exemple : $\text{prixHT} \leftarrow 100$

Les opérateurs

- Les opérateurs sont des éléments essentiels en algorithmique et en programmation. Ils permettent de manipuler des données en effectuant diverses opérations, telles que l'addition, la soustraction, la multiplication, la division, la comparaison, la concaténation, etc.

- Différents types d'opérateurs :
 - a. Opérateurs arithmétiques
 - b. Opérateurs de comparaison
 - c. Opérateurs logiques
 - d. Opérateurs de chaîne de caractères
 - e. Opérateurs d'affectation
 - f. Opérateurs d'incrément

Opérateurs arithmétiques

1. **+** (addition) : Utilisé pour additionner deux valeurs.
2. **-** (soustraction) : Utilisé pour soustraire la deuxième valeur de la première.
3. ***** (multiplication) : Utilisé pour multiplier deux valeurs.
4. **/** (division) : Utilisé pour diviser la première valeur par la deuxième.
5. **%** (modulo) : Renvoie le reste de la division de la première valeur par la deuxième.
6. ****** (puissance) : Utilisé dans certain langage comme python pour calculer une puissance

Opérateurs arithmétiques - Exemples

- `int a = 8;`
- `int b = 3;`

- `int addition = a + b; // addition vaut 11`
- `int soustraction = a - b; // soustraction vaut 5`
- `int multiplication = a * b; // multiplication vaut 24`
- `int division = a / b; // division vaut 2 (la division entière)`
- `int modulo = a % b; // modulo vaut 2`

Opérateurs de comparaison

1. **==** (égalité) : Vérifie si deux valeurs sont égales.
2. **!=** (différent) : Vérifie si deux valeurs sont différentes.
3. **<** (inférieur à) : Vérifie si la première valeur est inférieure à la deuxième.
4. **>** (supérieur à) : Vérifie si la première valeur est supérieure à la deuxième.
5. **<=** (inférieur ou égal à) : Vérifie si la première valeur est inférieure ou égale à la deuxième.
6. **>=** (supérieur ou égal à) : Vérifie si la première valeur est supérieure ou égale à la deuxième.

Opérateurs de comparaison - Exemple

- `int y = 10;`
- `int x = 5;`

- `boolean egalite = x == y; # egalite vaut False`
- `boolean difference = x != y; # difference vaut True`
- `boolean inferieur = x < y; # inferieur vaut True`
- `boolean superieur = x > y; # superieur vaut False`
- `boolean inferieur_ou_egal = x <= y; # inferieur_ou_egal vaut True`
- `superieur_ou_egal = x >= y # superieur_ou_egal vaut False`

Opérateur logiques

1. **and** (et / &&) : Renvoie True si les deux expressions sont vraies.
2. **or** (ou / ||) : Renvoie True si au moins l'une des expressions est vraie.
3. **not** (non / !) : Inverse la valeur de l'expression.

Opérateurs logiques - Exemple

Python

- `p = True`
- `q = False`

- `et = p and q` # et vaut False
- `ou = p or q` # ou vaut True
- `non_p = not p` # non_p vaut False

Java

- `boolean p = true;`
- `boolean q = false;`

- `boolean et = p && q; // et vaut false`
- `boolean ou = p || q; // ou vaut true`
- `boolean non_p = !p; // non_p vaut false`

Opérateurs de chaîne de caractères

1. **+** (concaténation) : Utilisé pour joindre deux chaînes de caractères.
2. ***** (répétition) : Utilisé pour répéter une chaîne de caractères plusieurs fois.

Exemple

- String chaine1 = "Bonjour";
- String chaine2 = "monde!";
- String concatenation = chaine1 + " " + chaine2; # concatenation vaut "Bonjour monde!"
- String repetition = chaine1 * 3; # repetition vaut "BonjourBonjourBonjour"

Opérateurs d'affectation

1. `=` (affectation) : Utilisé pour assigner une valeur à une variable.
2. `*=`, `+=`, `-=`, `=`, `/=`, `%=` (affectation avec opération) : Utilisé pour effectuer une opération et mettre à jour la variable en une seule étape.

Exemple

`x = 5;`

`x += 3;` # x vaut maintenant 8 (équivalent à `x = x + 3`)

Opérateurs d'incrémentation (ou décrémentation)

Les opérateurs d'incrémentation sont

- Utilisés pour augmenter la valeur d'une variable numérique (comme un entier ou un flottant) d'une certaine quantité.
- Couramment utilisés dans les langages de programmation pour simplifier l'augmentation d'une variable d'une unité (incrément)

Exemple

```
int x = 1;
```

```
x++; // La variable x est maintenant égale à 2
```

```
x+=2 // La variable x est maintenant égale à 4
```

Exercices - Variables et opérateurs arithmétiques

1. Algorithme qui affiche le prénom de l'utilisateur

Écrivez un algorithme qui demande à l'utilisateur de saisir son prénom puis affiche à l'écran le nom de l'utilisateur

2. Algorithme qui ajoute deux nombres et affiche leur résultat

Écrivez un algorithme qui demande à l'utilisateur de saisir deux nombres, les additionne, puis affiche le résultat de l'addition."

3. Algorithme qui calcule la moyenne d'un élève

Écrivez un algorithme qui demande à l'utilisateur de saisir les notes de trois matières d'un élève, calcule la moyenne de ces notes, puis affiche la moyenne.

4. Algorithme qui convertit une valeur en euro en dollars

Écrivez un algorithme qui demande à l'utilisateur de saisir son prix en €. L'algorithme convertit alors le prix en dollar et l'affiche.
 $1\text{€} = 1.3\text{\$}$

5. Algorithme qui salue l'utilisateur

Écrivez un algorithme qui demande à l'utilisateur de saisir son prénom, son nom puis affiche à l'écran "Bonjour [Nom] [Prénom]"

Les opérateurs : ce qu'il faut retenir

Types d'opérateurs	Liste	Exemple (Java)
Opérateurs arithmétiques	+, -, *, /, %, **	int addition = a + b; int division = a / b;
Opérateurs de comparaison	==, !=, >, <, <=, >=	boolean egalite = x == y; boolean difference = x != y;
Opérateurs logiques	and, or, not &&, , ! et / ou / non	boolean et = true && false; boolean et = true false;
Opérateurs de chaîne de caractères	+, *	String variable = "Hello" + " World"
Opérateurs d'affectation	=, *=, +=, -=, /=, %=	int variable = 2

Chapitre 4 : Les instructions conditionnelles

Les instructions conditionnelles

Les structures de contrôle permettent d'exécuter seulement certaines instructions d'un programme selon la vérification d'une ou plusieurs conditions.

La version la plus répandue des structures de contrôle est « **si... alors...** ».

Le type de mécanisme de contrôle peut concerner différents niveaux de granularité.

Par exemple :

- Si la note d'un examen est mineur de 4, alors la note est insuffisante
- Si le joueur n'a pas débloqué une porte, alors il ne peut pas accéder au niveau supérieur
- Si le mot de passe choisi contient moins de 6 caractères, alors il est trop court
- Si l'extension du fichier n'est pas .pdf, alors ne le téléverse pas sur le serveur

Les instructions conditionnelles

En général elles sont utilisées pour déterminer si certaines instructions doivent s'exécuter ou non :

- **Si cette condition s'avère, alors...**

Exécute l'instruction #1

Exécute l'instruction #2

Exécute l'instruction #9

- **Si non...**

Exécute l'instruction #7

Exécute l'instruction #10

```
if(isSucces == True) {  
    System.out.println("win");  
else {  
    System.out.println("Lose");  
}
```

Les instructions conditionnelles : Exercice

1. Demandez à l'utilisateur de saisir son âge. En fonction de l'âge saisi par l'utilisateur, affichez l'un des 3 messages suivants : "Vous êtes majeur", "Vous êtes majeur et vous avez exactement 18 ans", "Vous êtes mineur".
2. Demandez à l'utilisateur de saisir 2 nombres entiers. Affichez le résultat de l'addition de ces 2 nombres - Affichez le résultat de la soustraction de ces 2 nombres - Affichez le résultat de la multiplication de ces 2 nombres - Affichez le résultat de la division de ces 2 nombres. Uniquement pour ce dernier point (division): lorsque le dénominateur (deuxième nombre saisi par l'utilisateur) est égale à 0, n'effectuez pas l'opération, affichez le message suivant "Division par 0 impossible"
Pour chaque résultat, afficher le message correspondant.
Exemple : "Le résultat de l'addition est x"
3. Demandez à l'utilisateur de saisir un nombre compris entre 0 et 100. Affichez à l'utilisateur l'un des messages suivants : "Nombre compris entre 0 et 50" ou "Nombre compris entre 51 et 75" "Nombre supérieur à 75 ou inférieur à 0"

Les instructions : ce qu'il faut retenir

- Les structures de contrôle permettent d'exécuter seulement certaines instructions d'un programme selon la vérification d'une ou plusieurs conditions.
- La version la plus répandue des structures de contrôle est « **si... alors...** ».
- Elle est utilisée pour déterminer si une ou plusieurs instructions doivent être exécutées

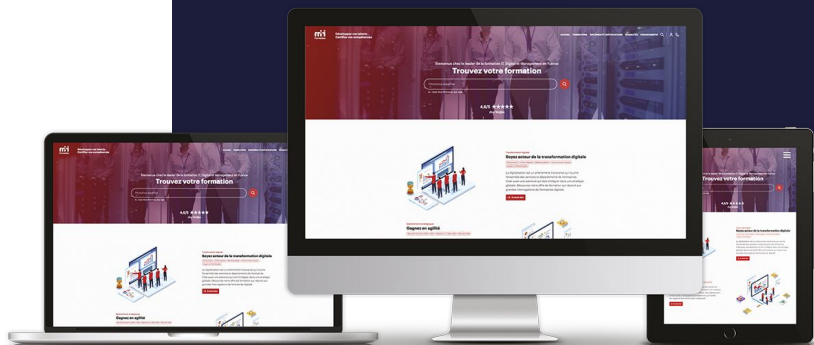


Commit chapitre 3 : Les instructions

- Lien vers mon github
- Le commentaire du commit est “chapitre 3 Instructions”



TP de validation des acquis Introduction, Variables et Instructions

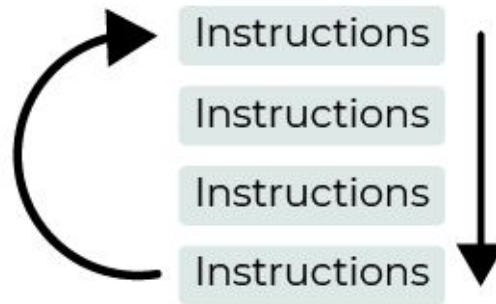


m2iinformation.fr

Chapitre 5 : Les boucles

Les boucles

- Une boucle permet d'exécuter une séquence d'opérations de manière répétée jusqu'à ce qu'une condition spécifiée soit satisfaite ou qu'un certain nombre d'itérations ait été atteint.
- En informatique une itération désigne l'action de répéter un processus
- Les boucles sont utilisées pour automatiser des tâches répétitives et rendre les programmes plus efficaces.



La boucle “Tant que”

- La boucle "Tant que" est une structure de contrôle qui répète un bloc d'instructions tant qu'une condition est vraie. Le flux d'exécution reste à l'intérieur de la boucle tant que la condition est vraie, et il sort de la boucle une fois que la condition devient fausse.
- Il est crucial de s'assurer que la condition finira par devenir fausse pour éviter une boucle infinie.

```
Tant que (condition)  
    # Bloc d'instruction à répéter  
Fin Tant Que
```

```
i = 1  
Tant que (i < 10)  
    i = i + 1  
Fin Tant Que  
Afficher (i)
```


La boucle “Pour”

- La boucle "Pour" est utilisée lorsque le nombre d'itérations est connu à l'avance. Elle se compose d'une initialisation, d'une condition de continuation et d'une expression d'itération.
- Cette boucle est souvent utilisée pour itérer sur une séquence d'éléments, comme une liste ou un intervalle de nombres.

```
Pour (initialisation; condition; expression d'itération)  
    # Bloc d'instructions à répéter  
Fin Pour
```

```
Pour i de 1 à 9 pas 1  
    Afficher(i)  
Fin Pour
```

La boucle “Répéter”

- La boucle "Répéter" (ou "Faire... Jusqu'à ce que" dans certaines langues) est similaire à la boucle "Tant que", mais elle teste la condition à la fin de la boucle. Cela garantit que le bloc d'instructions est exécuté au moins une fois.

```
Répéter  
    # Bloc d'instructions à  
    répéter  
Jusqu'à ce que (condition)
```

```
i = 1  
Répéter  
i = i + 1  
Jusqu'à ce que (i > 10)  
Afficher(i)
```

Les boucles imbriquées

- Les boucles imbriquées sont des boucles placées à l'intérieur d'autres boucles. Cela permet de **gérer des structures plus complexes** et de traiter des problèmes plus élaborés.
- Les boucles imbriquées sont souvent utilisées pour parcourir des tableaux bidimensionnels, créer des combinaisons, ou effectuer d'autres opérations nécessitant plusieurs niveaux d'itérations.
- Il est important de gérer soigneusement les indices et les conditions pour éviter des erreurs ou des comportements inattendus dans les boucles imbriquées.

```
Pour i de 1 à n  
    Pour j de 1 à m  
        # Bloc d'instructions à répéter  
    Fin Pour  
Fin Pour
```

```
Pour i de 1 à n  
    Pour j de 1 à m  
        Afficher(i, j)  
    Fin Pour  
Fin Pour
```

Les boucles : Exercice

1. Demandez à l'utilisateur de saisir un nombre compris entre 0 et 10. Affichez à l'utilisateur le " Votre nombre est [nombre saisi]" uniquement lorsque le nombre est compris en 0 et 10. Affichez le message "Saisissez un nombre compris en 0 et 10" tant que le nombre n'est pas compris entre 0 et 10
2. Demandez à l'utilisateur de saisir un nombre inférieur à 100. Forcer l'utilisateur à saisir un nombre inférieur à 100. Affichez tous les nombres à partir du nombre saisi jusqu'à 100

Exemple du résultat attendu :

L'utilisateur saisit 95 Le programme doit afficher "95 96 97 98 99 100"

Les boucles : Exercice

1. Écrivez un programme qui affiche un motif triangulaire à base d'étoiles. Le programme doit demander à l'utilisateur la hauteur du triangle et ensuite afficher le triangle correspondant

Exemple du résultat attendu si l'utilisateur saisit 5 :

*

**

Les boucles : ce qu'il faut retenir

- Les boucles sont des outils puissants en programmation, offrant la flexibilité nécessaire pour automatiser des tâches répétitives et résoudre des problèmes complexes.
- Choisir la bonne boucle dépend du contexte spécifique du problème à résoudre et des conditions d'itération nécessaires :
 1. Boucles “Tant que”
 2. Boucles “Répéter”
 3. Boucles “Pour”

Chapitre 6 : Les fonctions

Les fonctions

- Les fonctions représentent une sorte de "programme dans le programme", car elles sont la première forme d'organisation du code.
- On utilise des fonctions pour regrouper des instructions et les appeler sur demande. Chaque fois que l'on a besoin de ces instructions, il suffira d'appeler la fonction au lieu de répéter toutes les instructions.
- Une fonction permet de **réaliser une tâche complexe en écrivant un seul mot.** Pour accomplir ce rôle, le cycle de vie d'une fonction se divise en deux :
 - a. **Une phase unique dans laquelle la fonction est déclarée.** On définit à ce stade toutes les instructions qui doivent être groupées pour obtenir le résultat souhaité
 - b. **Une phase qui peut être répétée une ou plusieurs fois dans laquelle la fonction est exécutée.** On demande à la fonction de mener à bien toutes les instructions dont elle se compose à un moment donné dans la logique de notre application

Les fonctions

- Une fonction reçoit aucun, un ou plusieurs arguments, que l'on transmet à la fonction. Un **argument** est une variable ou une valeur
- Une fonction renvoie généralement un résultat, qui est une valeur, mais cela n'est pas obligatoire

Les fonctions : Exemple 1

```
fonction somme() # Déclaration de la fonction  
    résultat = 5 + 2 # Début des instructions  
    afficher résultat
```

```
... # Autres instructions dans mon code
```

```
somme() # Appel de la fonction
```

Les fonctions : Exemple 2

```
fonction somme(a,b) # Deux arguments : a et b  
    résultat = a + b # Début des instructions  
    afficher résultat
```

```
... # Autres instructions dans mon code
```

```
somme(5, 3) # Appel de la fonction
```

Les fonctions : Exemple 3

```
fonction somme(a,b) # Deux arguments : a et b
    résultat = a + b # Début des instructions
    retourner résultat # résultat est retourné par la
fonction
```

```
... # Autres instructions dans mon code
```

```
total = somme(5, 3) # Appel de la fonction sur une variable
afficher total
```

Les fonctions : Exercice

1. Écrivez la fonction **somme(a,b)** d'exemple du cours : elle prend en paramètre a et b, et retourne la somme, puis testez la fonction.
2. Ecrivez une fonction **disMonAge(age)** qui prend en paramètre l'âge et qui renvoie "Vous avez x **ans**". Si la personne a 1 an : "Vous avez 1 **an**".
Demandez ensuite à l'utilisateur de saisir son âge et appelez la fonction

Les fonctions : ce qu'il faut retenir

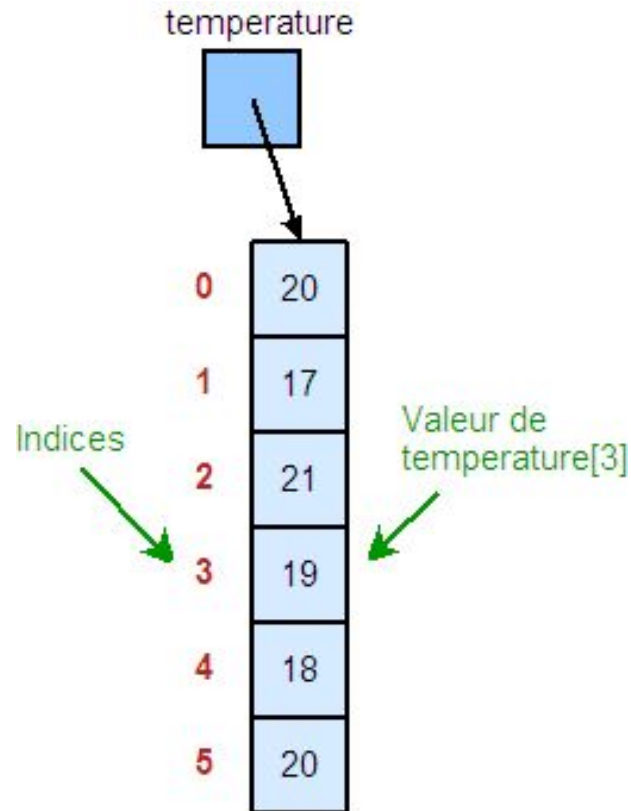
1. Les fonctions sont en réalité des morceaux de code déjà écrits, parfois longs et complexes, qu'on utilise pour faciliter l'écriture d'algorithmes plus élaborés.
2. Les fonctions peuvent prendre (ou pas) prendre des arguments (aussi appelés paramètres) et retourner (ou pas) une valeur

Chapitre 7 : Les tableaux

Les tableaux

- Les tableaux (ou “array”) sont des listes indexées d'éléments qui partagent une certaine relation sémantique pour appartenir à la liste.
- Un exemple tout simple d'array est la liste des courses. On peut indexer cette liste en fonction de l'ordre des articles à acheter :
 - a. Lait
 - b. Farine
 - c. Pommes
 - d. Fromage
- **Attention : l'index d'un tableau commence à 0.** En prenant l'exemple précédent on dira que l'élément “Lait” est dans la première case du tableau qui est à l'index 0. De la même façon l'élément “Fromage” est l'élément dans la 4ème case du tableau qui se trouve à l'index 3.

Les tableaux - Représentation



Les tableaux : Exemple

- **Java :**

```
String[] listeCourses = {"Lait", "Farine", "Pommes", "Fromage"};  
String course = "lait";
```

- **Python :**

```
liste_courses = ["Lait", "Farine", "Pommes", "Fromage"]
```

Les tableaux : Exercice

- Saisissez les notes un par un, lorsque l'utilisateur saisit la note -1, la saisie des notes prend fin et votre programme affiche le nombre total de notes (ie nombre d'étudiants) et la moyenne de la classe.

- Utilisez **obligatoirement** une fonction

- Utilisez **obligatoirement** un tableau

Rappel : le premier indice d'un tableau commence à 0

- **Attention :** le logiciel atteint ses limites pour l'utilisation des fonctions. Ecrivez votre algorithme sans vous soucier de l'exécution du programme et du résultat, en respectant les principes de l'algorithmique.



TP de validation des acquis Chapitres 1 à 7



m2information.fr

Chapitre 8 : Programmation procédurale et programmation orientée objet

Programmation procédurale - Principes

- Consiste à écrire son code en suivant une procédure logique, en suivant de façon séquentielle une suite de fonctions qui sont appelées dans le code avec parfois des fonctions qui s'appellent entre elles.
- Ces fonctions sont appelées dans le code source en leur passant ou non des paramètres. Ces fonctions renvoies parfois des résultats exploitables dans la suite de la procédure.
- Cette façon de coder est plus abordable aux débutants et demande moins d'analyse avant le développement.
- Cependant le code source sera souvent plus long, difficilement réutilisable et plus difficile à maintenir.
- Certains langages comme le C sont prévus pour écrire du code en procédural.

Programmation orientée objet (POO) - Principes

- Cette façon de coder s'approche un peu plus proche du raisonnement humain. Dans cette approche on crée des objets qu'on appelle des classes. Chaque classe va avoir ses propres attributs et ses propres méthodes.
- L'héritage et le polymorphisme vont nous permettre de factoriser certaines opérations entre les différents modèles de données similaires.
- De plus nous allons pouvoir aller du général vers le particulier en créant des classes parents et des classes enfants.
- La POO donne un code plus lisible car il est constitué de petites entités qui vont interagir.
- Les classes enfants héritent des modifications apportées aux classes mères ce qui limite les erreurs et oublis plus fréquents en procédural.

Programmation procédurale versus POO

Programmation procédurale

- A base de fonctions qui ne contiennent pas les données.
- Code difficilement réutilisable tel quel.
- Chaque fonction ou procédure est associée à un traitement particulier.
- Chaque fonction peut être utilisée à différents emplacements du programme.
- Plus simple à conceptualiser.

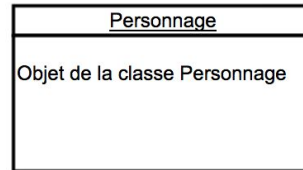
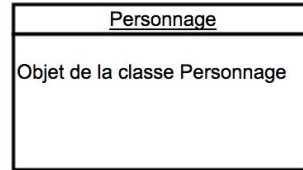
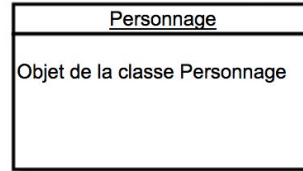
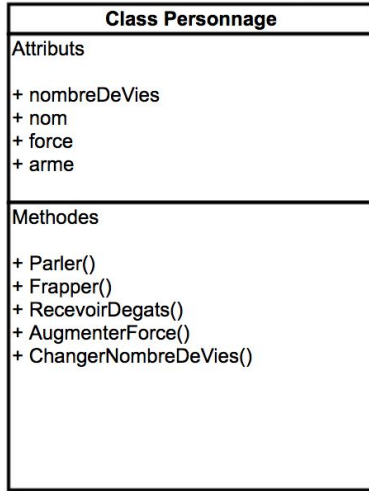
POO

- A base d'objets qui contiennent les données
- Code réutilisable.
- Code scindé en entités autonomes (classes)
- Les données sont protégées par encapsulation.

Zoom sur la Programmation Orientée Objet

Notion de classe

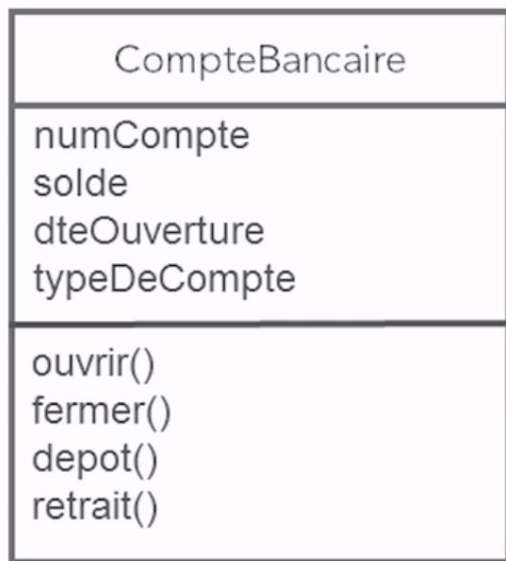
- Une classe est un plan qui nous permet de créer plusieurs objets à partir de ce plan. Exemple : Plusieurs personnages d'un jeu vidéo. On parle alors d'instances de la classe Personnage.



Comment créer une classe

- **Nom** : c'est quoi ?
Employé, compte bancaire, joueur, document, album...
- **Attributs** : ce qui la décrit.
Largeur, hauteur, couleur, type de fichier, score...
On les appelle aussi des propriétés.
- **Comportements** : que peut-elle faire ?
Jouer, ouvrir, chercher, enregistrer, imprimer...
On les appelle le plus souvent des méthodes.

Classe / Objet



Classe



compteJean



compteFlorence



comptePierre

Objet (instance)

Exemple en Java

CompteBancaire
numCompte solde dteOuverture typeDeCompte
ouvrir() fermer() depot() retrait()

Classe

```
// Déclaration de la classe
Class CompteBancaire {

    // Déclaration des attributs
    int numCompte;
    int solde;
    int dateOuverture;
    String typeDeCompte;

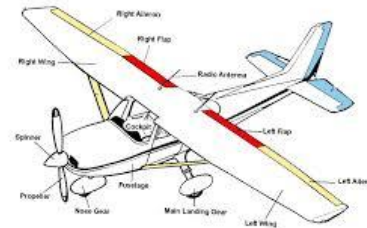
    // Déclaration des méthodes de ma classe
    public void ouvrir() {}
    public void fermer() {}
    public void depot() {}
    public void retrait() {}
}
```

Les class natives d'un langage

- La plupart des langages aujourd'hui proposent des classes pré-conçues prêtes à l'emploi.
- On peut utiliser ces classes et créer les nôtres.
- String(), Date(), Array()....

Notion d'objets en programmation

- Chaque objet à une vie qui lui est propre (si je casse une tasse, l'autre reste intacte).
- Chaque objet peut avoir des propriétés différentes (une tasse peut être vide et l'autre à moitié pleine, une pomme peut avoir un nombre de feuilles différent ou une couleur différente).
- Chaque objet à ses propres comportements (un avion vole alors qu'un téléphone sonne)



Exercice d'initiation

- On imagine un jeu avec 3 personnages : Un guerrier, une fée et un magicien.
- Chaque personnage a sa propre classe
- Imaginer 3 objets qui seraient des instances de ces classes, avec pour chacun une liste d'attributs et fonctions

ABSTRACTION

4 notions fondamentales en POO

- **L'Abstraction.**
- Encapsulation.
- Héritage.
- Polymorphisme.

La notion d'abstraction fait référence à un objet que l'on visualise sans pour autant que celui-ci soit parfaitement décrit. Exemple : une table est un objet que l'on visualise, pourtant il reste abstrait car l'on ne connaît pas ses propriétés. Le principe de l'abstraction en programmation est de se focaliser sur l'essentiel en ignorant tous les détails.



4 notions fondamentales en POO

- **L'Abstraction.**
- Encapsulation.
- Héritage.
- Polymorphisme.

Par exemple, dans le cas d'un lecteur de musique, l'utilisateur n'a pas besoin de connaître les détails de lecture de chaque morceau (comme la manipulation des bits audio), mais seulement d'interagir avec des méthodes abstraites comme `play()`, `pause()` ou `stop()`.

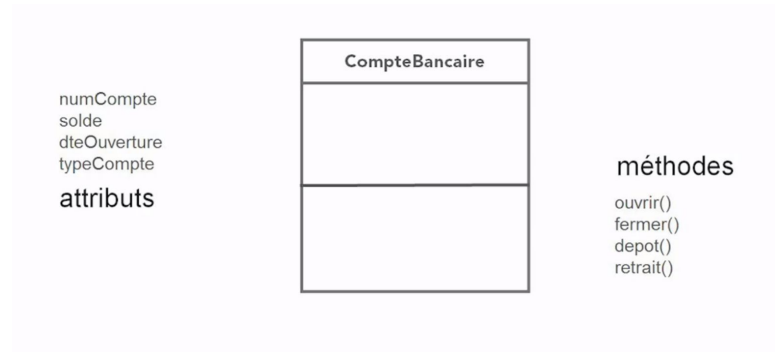


ENCAPSULATION

4 notions fondamentales en POO

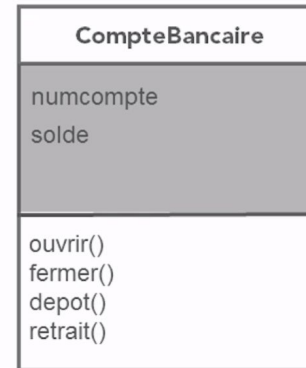
- L'Abstraction.
- **Encapsulation.**
- Héritage.
- Polymorphisme.

Consiste en premier lieu à encapsuler nos méthodes et nos attributs au sein d'une même unité : la classe.



Respect de l'encapsulation

- Un objet ne doit révéler de lui-même que le strict nécessaire aux autres parties du programme. Exemple du compte bancaire : le solde n'est accessible que via une méthode de la classe. Sans utiliser cette méthode, on ne peut pas accéder au solde.
- On peut modifier les données d'une classe sans pour autant modifier les données de tous les objets.
- Les attributs sont privés.



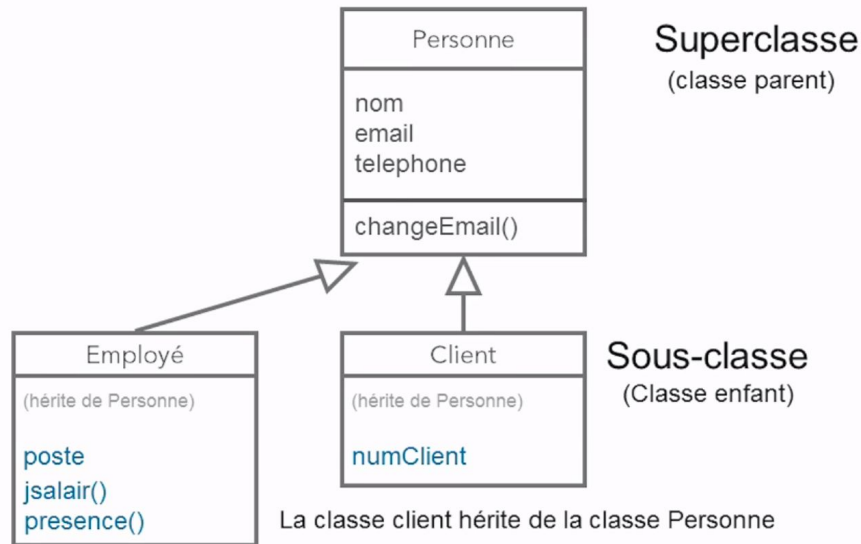
HERITAGE

4 notions fondamentales en POO

- L'Abstraction.
- Encapsulation.
- **Héritage.**
- Polymorphisme.

L'héritage est une manière très pratique de réutiliser du code. Les classes enfants héritent des attributs et méthodes de la classe parent.

Si on modifie la classe parent, toutes les classes enfants bénéficient de ces modifications.



Si une classe parente n'est jamais instanciée (Exemple de "Personne"), on parle de classe "Abstraite". On ajoute alors le mot clé `Abstract` devant en Java

Identifier les situations d'héritage

Une voiture **est un** véhicule.

Un bus **est un** véhicule.

~~Une voiture est un bus.~~

Un employé **est une** personne.

Un client **est une** personne.

~~Un client est un panier.~~

← Héritage

Un compte courant **est un type de** compte en banque.

Un compte d'épargne **est un type de** compte en banque.

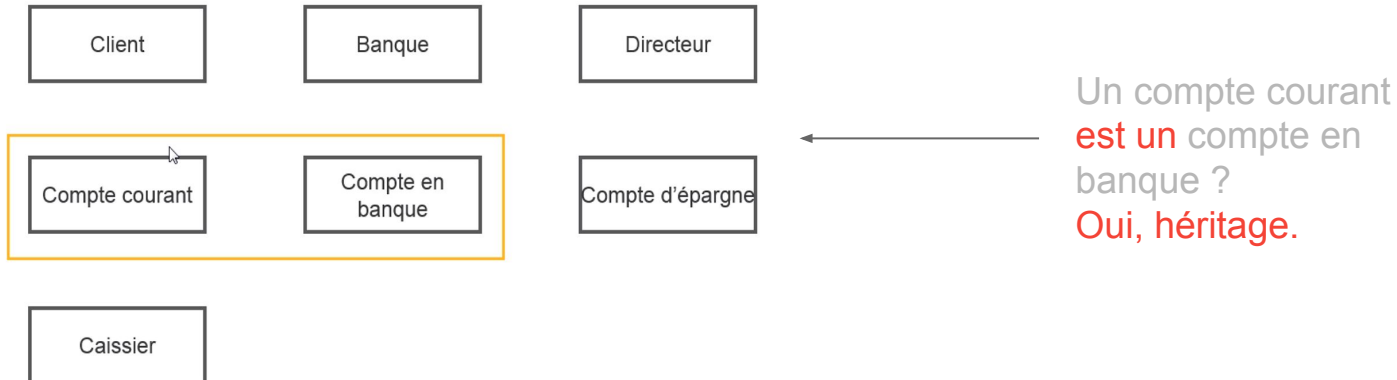
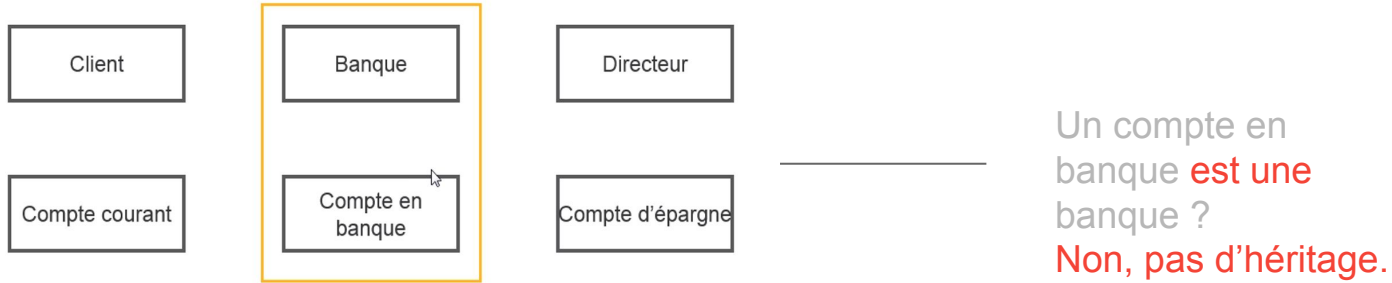
← Double
Héritage

Une Peugeot 108 **est une** voiture **et un** véhicule.

Un caniche **est un** chien **et un** mammifère **et un** animal.

← Triple
Héritage

Identifier les situations d'héritage



POLYMORPHISME

4 notions fondamentales en POO

- L'Abstraction.
- Encapsulation.
- Héritage.
- **Polymorphisme.**

Polymorphisme : qui peut prendre plusieurs formes.

$a + b$
5 7

addition
 $a + b$
12

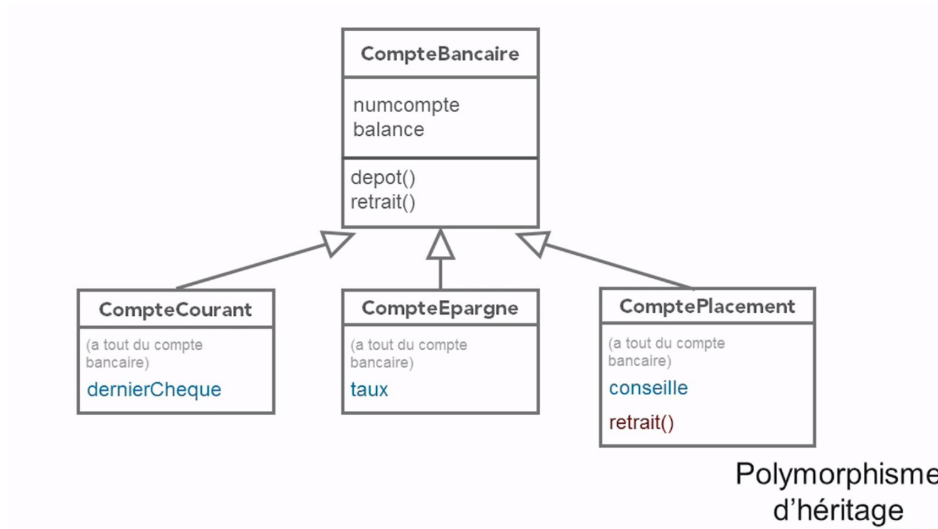
Le + ne fait pas la même chose.

$a + b$
"Pierre" "Dubois"

concaténation
 $a + b$
"PierreDubois"

4 notions fondamentales en POO

- Le polymorphisme consiste à concevoir des objets qui partagent des comportements, ce qui permet de traiter les objets de différentes manières



- La méthode retrait() de la classe ComptePlacement se comporte différemment.
- Le polymorphisme est rarement utilisé.

Exercice initiation

- Reprendre le travail sur les personnages précédemment.
- S'interroger sur les fonctions, les paramètres et les retours de ces fonctions.
- S'interroger sur la notion d'héritage dans notre exemple et proposer une manière de l'utiliser
- Imaginez les interactions entre les objets (par des flèches par exemple).
Qu'est ce qu'un guerrier peut faire à un magicien, inversement ... Essayez de modéliser les relations entre les objets

INTERFACES

Utiliser les interfaces

Une notion très voisine de l'héritage

- Existe dans de nombreux langage de POO
- Communément appelée interface mais n'a rien à voir avec les fenêtres ni les IHM.
- C'est un peu comme une classe sans fonctionnalités ni code réel à l'intérieur.
- Ne contient que des signatures de méthodes que je veux rendre disponible pour plusieurs classes (qui doit implémenter toutes les méthodes)

```
Interface Impression {  
    // signatures des méthodes  
    void imprime();  
    void imprimePDF(String nomFichier);  
}
```


Utiliser les interfaces : implémentation

- Créer des classes qui vont implémenter les signatures d'une interface.
- Utilisation du mot-clé "implements" pour créer une classe qui va détailler le fonctionnement des méthodes.

```
Interface Impression {  
    // signatures des méthodes  
    void imprime();  
    void imprimePDF(String nomFichier);  
}
```

```
class UneClasse implements Impression{  
  
    // méthodes de l'interface  
    public void imprime() {  
        // la suite du code  
    }  
    public void imprimePDF(String nomFichier){  
        // la suite du code  
    }  
  
    // autres fonctionnalités  
}
```

Utiliser les interfaces dans le code

- Dans ma boucle while, si l'objetGenerique est une instance d'impression, j'ai le droit d'utiliser sa méthode "imprime()".

Intérêts :

- Nous pouvons utiliser des méthodes appartenant à une classe sans en connaître l'organisation du code.
- J'ai simplement une interface qui met à ma disposition des méthodes que je peux utiliser à n'importe quel moment.

```
Interface Impression {  
    // signatures des méthodes  
    void imprime();  
    void imprimePDF(String nomFichier);  
}
```

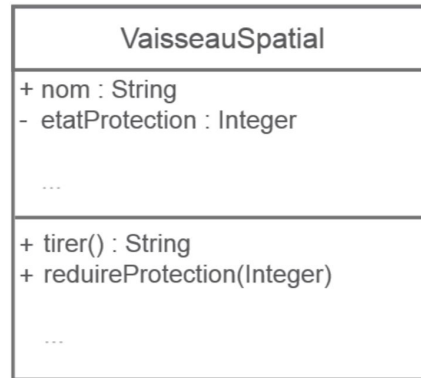
```
// plus loin dans le code  
while (objetGenerique in ListeObjets) {  
    if (objetGenerique instanceof Impression){  
        //  
        objetGenerique.imprime();  
    }  
}
```

Durée de vie d'un objet

La durée de vie d'un objet

Instancier un objet et lui attribuer des valeurs avec des paramètres est possible grâce à la notion de constructeur.

- Que se passe t-il quand on crée un objet (le constructeur) ?
- Que se passe t-il quand on a plus besoin de l'objet (le destructeur) ?



Instanciación d'un objet

Dans la plupart des langages, on utilise le mot réservé “New”

Que se passe t-il à la création d'un objet ?

- Allocation mémoire pour l'objet
- Initialisation des variables de l'instance.
- Renvoi une référence vers cet objet.
- Dès lors on peut commencer à utiliser l'objet.

Java `Client jean = new Client();`

C# `Client jean = new Client();`

VB.NET `Dim jean As New Client`

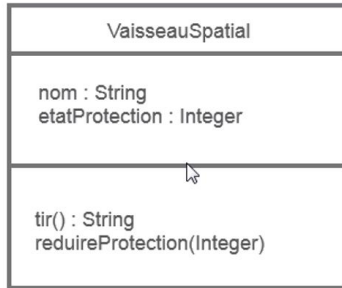
Ruby `jean = Client.new`

C++ `Client *jean = new Client();`

Objective-C `Client *jean= [[Client alloc] init];`

Le constructeur

- Une méthode qui est utilisée pour construire l'objet.



VaisseauSpatial **ariane** = new VaisseauSpatial();

object : ariane

nom : null etatProtection : 0

Résultat

Le constructeur

- Créer une instance avec des paramètres pour le constructeur.

```
public class VaisseauSpatial {

    // variables d'instance
    public String nom;
    private int etatProtection;

    // constructeur de méthode
    public VaisseauSpatial() {
        nom = "Vaisseau sans nom";
        etatProtection = 100;
    }

    // surcharge de méthode
    public VaisseauSpatial(String n) {
        nom = n;
        etatProtection = 200;
    }

    // autres méthodes

}
```

```
VaisseauSpatial ariane =
    new VaisseauSpatial("ariane 2");
```

object : ariane

nom : ariane 2
etatProtection : 200

Résultat

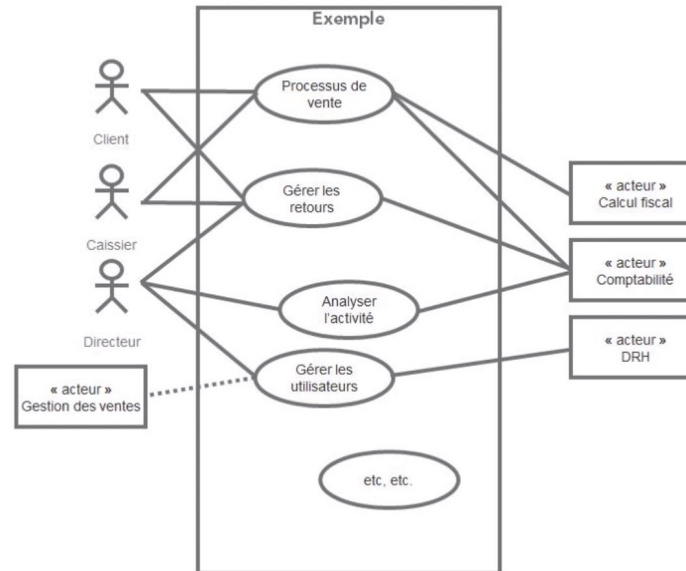
Le destructeur

- le destructeur d'une classe est une méthode spéciale lancée lors de la destruction d'un objet afin de récupérer les ressources (principalement la mémoire vive) réservée dynamiquement lors de l'instanciation de l'objet
- Certains langages gèrent la destruction de l'objet eux même, ils ont une gestion automatique de la mémoire (Java, Python..., aller voir du côté du “garbage collector” pour en savoir plus)

CHAPITRE 9 - Introduction à l'UML

Qu'est ce que l'UML

- UML (Unified Modeling Language)
- Langage de modélisation unifié.
- Utilisé pour modéliser des projets informatiques
Particulièrement utilisé en orienté objet.
- Exemple d'outil en ligne pour UML (et autres...)
 - <https://www.gliffy.com/>

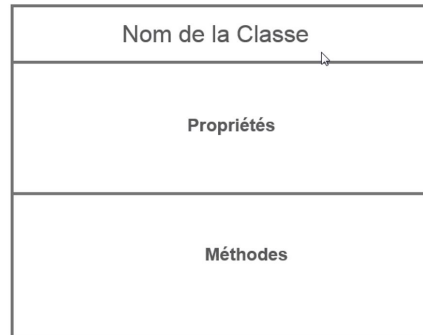


De nombreux types de diagrammes

- **Diagramme de classe** : Représenter la structure d'un système
- **Diagramme de séquence** : Représenter les interactions entre les objets d'une manière temporelle
- **Diagramme d'état** : Représenter les transitions d'état entre divers objets

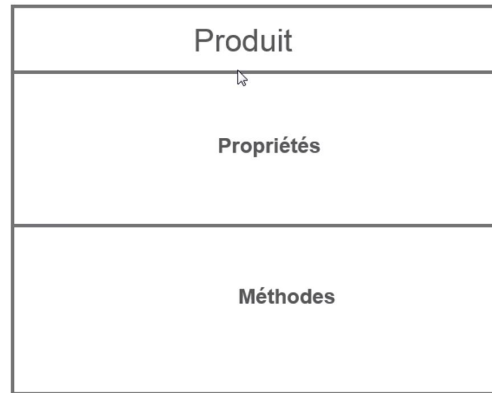
Le diagramme de classe

- C'est le diagramme le plus couramment rencontré.
- Il permet de définir les classes avec leurs attributs et méthodes.



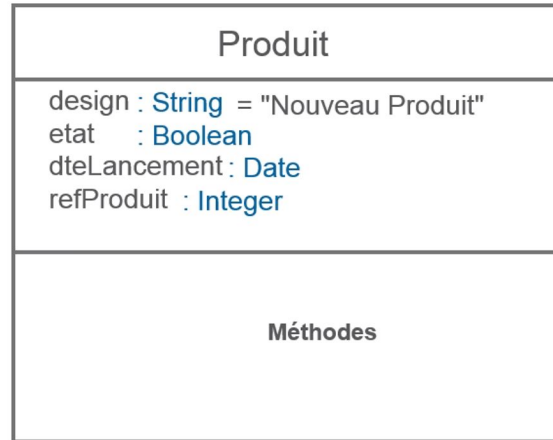
Le diagramme de classe : conventions

- Respecter les conventions de nommage.
- Les noms de classes commencent par une majuscule.



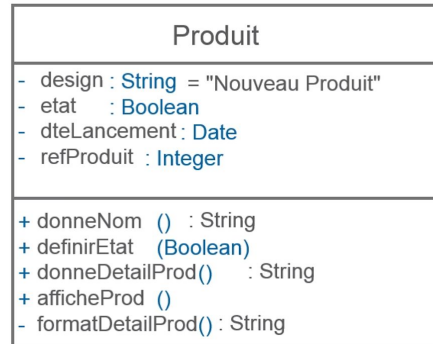
Le diagramme de classe : les attributs

- Respecter les conventions de nommage.
- Commencent par une minuscule puis chaque mot par une majuscule, ni espace ni caractères spéciaux.
- On peut en désigner le type sans tenir compte du langage utilisé par la suite.
- On peut l'initialiser.



Le diagramme de classe : les méthodes

- Commencent par une minuscule puis chaque mot par une majuscule, ni espace ni caractères spéciaux.
- Faire suivre des parenthèses.
- On peut désigner les paramètres en réception par leur type.
- On peut désigner les types de valeurs retournées (type return).
- Autant que possible nommer les méthodes getters et setters (setNom, getDate).
- Désigner les visibilités (+ et -,...), ne rendre visible que le stricte minimum.
- Respecter les règles d'écriture en UML

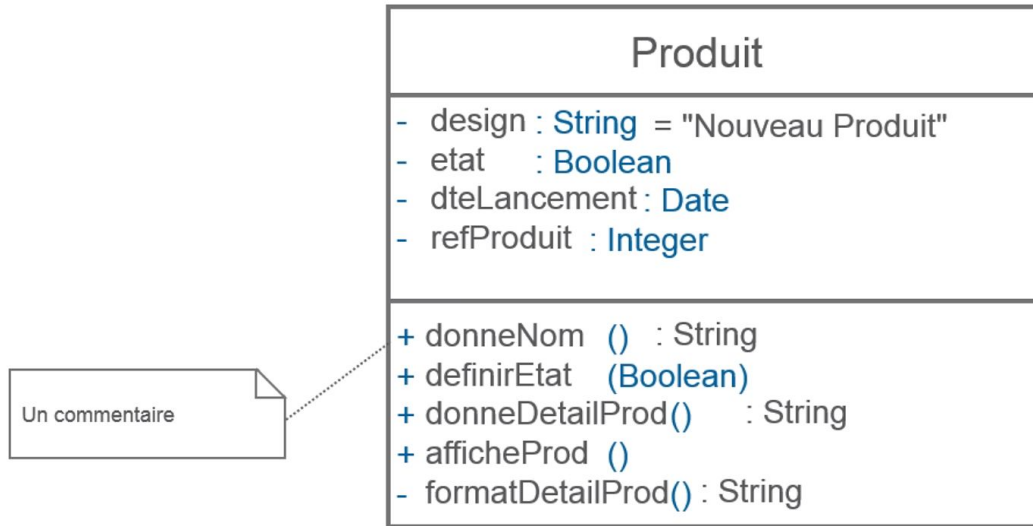


Le diagramme de classe : visibilité ?

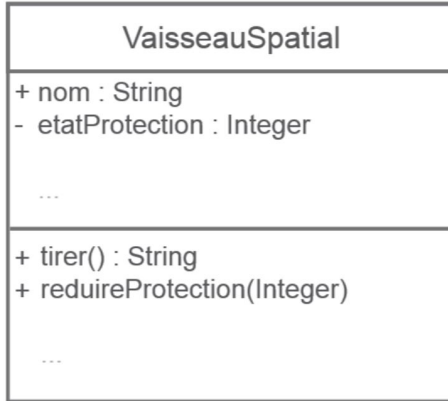
- Dans les diagrammes de classes UML, la visibilité définit si les attributs et les opérations de classes spécifiques peuvent être vus et utilisés par d'autres classes.
- Par exemple, les attributs et opérations d'une classe avec visibilité **publique** **peuvent être vus et utilisés par d'autres classes**, tandis que ceux d'une classe avec visibilité **privée** ne peuvent l'être que par la classe qui les contient.
- 4 niveaux de visibilité : **public (+)**, **private (-)**, package et protégé

Le diagramme de classe : commentaires

- Représenté par un cadre dont l'angle supérieur droit est replié.
- Relié à la méthode par des pointillés.



Convertir le diagramme de classe en Java



```

public class VaisseauSpatial {

    // variables d'instance
    public String nom;
    private int etatProtection;

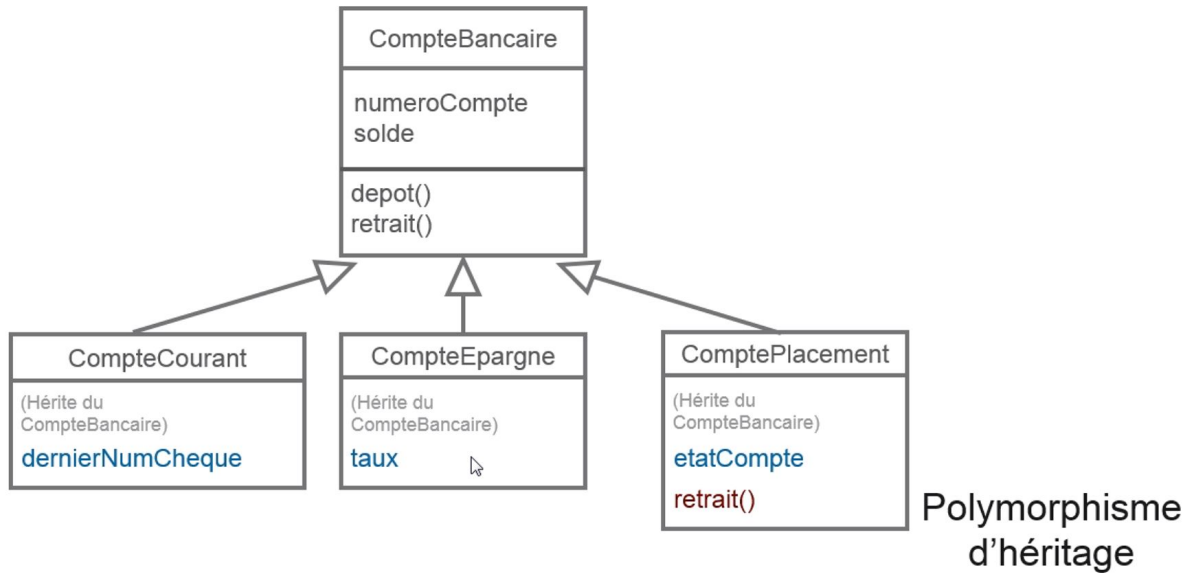
    // méthodes
    public String tirer() {
        return "Booum!";
    }

    public void reduireProtection (int quantite) {
        etatProtection -= quantite;
    }

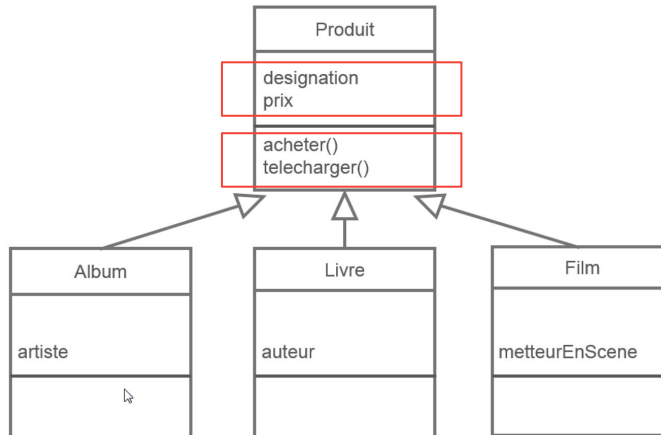
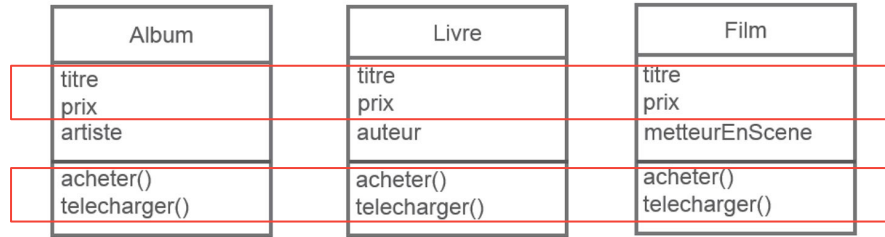
}

```

Héritage en UML

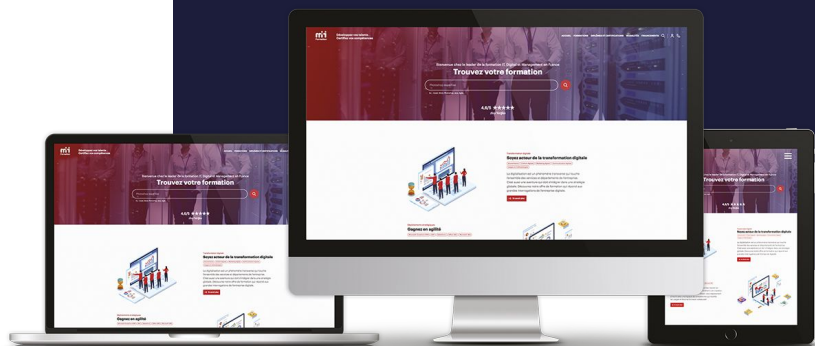


Comment déterminer qui hérite de qui ?



En partant du bas vers le haut
Je constate des points commun
Je crée une classe parent

TP de validation des acquis Chapitres 8 et 9



CHAPITRE 10 - Initiation JAVA

L'histoire de JAVA

- Développé par James Gosling chez Sun Microsystems à la fin des années 1980.
- Initialement nommé "Oak", conçu pour les appareils électroniques grand public.
- Lancé officiellement en 1995, Java devient crucial pour l'interactivité sur le Web.
- Répond à la nécessité de portabilité et de sécurité sur Internet.

Caractéristiques principales

- **Portabilité** : Indépendant de la plateforme, fonctionne sur tout appareil compatible Java.
- **Sécurité** : Modèle de sécurité robuste pour prévenir les attaques malveillantes.
- **Robustesse** : Gestion automatisée de la mémoire pour réduire les erreurs de programmation
- **Toujours en évolution pour rester pertinent**
- **Java est Orienté Objet** : On est obligé de passer par des classes.

Que faut-il pour faire du java : Installation

- Un outil pour code : <https://code.visualstudio.com/>
- Le JDK de Java :
<https://www.oracle.com/fr/java/technologies/downloads/#jdk17-windows>

Créer mon premier programme

1. Créer un nouveau fichier Test.java
2. Copier coller le code fournis
3. Ouvrir le terminal de commande et se placer à l'endroit où mon programme est sauvegardé dans mon PC.
4. Lancer la commande **javac Test.java** pour compiler. S'il n'y a pas de message, la compilation s'est bien passée
5. Lancer la commande **java Test** pour exécuter le programme.

PARTIE 1 : Reprise des concepts

ALGOBOX

Caractère d'instruction

- Le caractère de fin de l'instruction est “;”

`a = b + c;`

Oublier ce caractère provoquera une erreur à la compilation.

```
PS C:\Users\rddin\Desktop> javac .\Exemple.java
.\Exemple.java:6: error: ';' expected
    int a = 1 + 2
                ^
1 error
```

Les types en Java

- On utilisera la classe **String** pour les chaînes de caractère (Équivalent à Chaîne sur algobox) → `String prenom = "Michel";`
- Dans la plupart des cas, les types "int" et "float" suffisent pour manipuler des entiers ou des nombres à virgule

Type	Classe éq.	Valeurs	Portée	Défaut
boolean	Boolean	true ou false	N/A	false
byte	Byte	entier signé	{-128..128}	0
char	Character	caractère	{\u0000..\uFFFF}	\u0000
short	Short	entier signé	{-32768..32767}	0
int	Integer	entier signé	{-2147483648..2147483647}	0
long	Long	entier signé	{-2 ³¹ ..2 ³¹ - 1}	0
float	Float	réel signé	{-3,4028234 ³⁸ ..3,4028234 ³⁸ } {-1,40239846 ⁻⁴⁵ ..1,40239846 ⁻⁴⁵ }	0.0
double	Double	réel signé	{-1,797693134 ³⁰⁸ ..1,797693134 ³⁰⁸ } {-4,94065645 ⁻³²⁴ ..4,94065645 ⁻³²⁴ }	0.0

Les types : exemples

- La chaîne de caractères String est toujours entre guillemets pour délimiter le départ et la fin de ma chaîne

```
boolean estVrai = true; // boolean
int nombreEntier = 42; // int
float nombreFlottant = 3.14f; // float
String message = "Bonjour le monde !"; // String
```

Le terme void

- En plus de ces types primitifs, le terme “void” est utilisé pour spécifier le retour vide

```
// Méthode qui ne retourne rien (void) et ne prend aucun paramètre
static void afficherMessage() {
    System.out.println("Ceci est un message.");
}
```

Concaténation d'une chaîne de caractère

- Voici un exemple de concaténation en Java. s3 contiendra "Hello world"

```
String s1 = "hello";  
String s2 = "world";  
String s3 = s1 + " " + s2;
```


Les tableaux

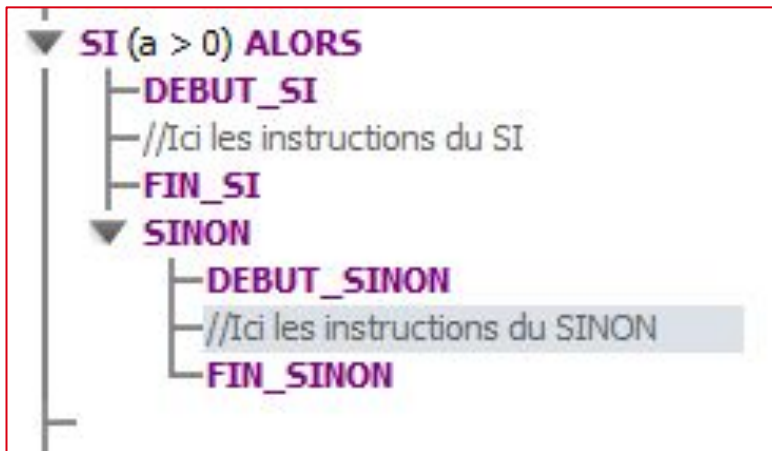
- Un tableau est déclaré de la manière suivante :
`monType[] nomTableau = new int[tailleTableau];`
- Exemple, créer un tableau de 10 entiers : `int[] tab = new int[10];`
- Créer un tableau avec des valeurs connues : `int[] tab`
- En java (contrairement à algobox), un tableau a toujours **une taille fixe**, qui doit être précisée avant l'affectation des valeurs à mon tableau.
- La taille de mon tableau est toujours disponible dans une variable "length" appartenant au tableau.
- Exemple avec mon tableau d'entier : `tab.length` contiendra 10
- Comme sur Algobox, le premier indice de mon tableau est 0. Pour parcourir un tableau, on ira donc de 0 à taille-1.

Les opérateurs

- La plupart des opérateurs sont identiques à Algobox
- `variable = variable + 1;` peut s'agréger `variable++;`
- L'opérateur ET devient `&&`
- L'opérateur OU devient `||`

Instructions conditionnelles

Algobox



Java

```

if(a > 0) {
    // Ici les instructions du si
}
else {
    // Ici les instructions du sinon
}
  
```

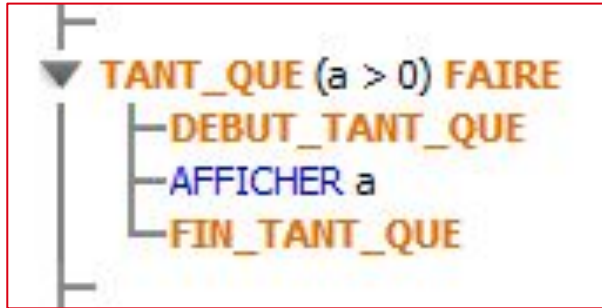
Instructions conditionnelles

- Si on veut ajouter un if on peut, on peut faire des else if {} à l'infini.

```
public Test()
{
    if(variable == 20)
    {
        System.out.println("La variable 'variable' est bien égale à 20");
    }
    else if(variable == 15)
    {
        System.out.println("La variable 'variable' n'est pas égale à 20 et est égale à 15");
    }
    else if(variable == 10)
    {
        System.out.println("La variable 'variable' n'est pas égale à 20 ni à 15 et est égale à 10");
    }
    else
    {
        System.out.println("La variable 'variable' n'est ni égale à 20 ni à 15 ni à 10");
    }
}
```

Boucle Tant Que

Algobox



Java

```
while(a > 0) {  
    System.out.println(x:"Bonjour");  
}
```

Boucle Pour

AlgoBox



Java

```
// Initialisation de i = 0 puis ";"  
// Condition de sortie : tant que i < 10 puis ";"  
// Ajout de 1 à la valeur i à chaque fois que je parcours ma boucle  
for(int i = 0; i < 10; i++) {  
    // Instructions à exectuer  
}
```

Visibilité en java

- public : Les membres public sont accessibles depuis n'importe quelle classe (Les méthodes principales et les classes principales sont déclarées publiques par convention)
- private : Les membres private sont accessibles uniquement à l'intérieur de la même classe.
- **Pour la partie programmation (et non pas UML), par soucis de simplicité, vous pourrez laisser vos attributs et méthodes en public**
- Il existe d'autres visibilitées qui prennent en compte des notions pas encore vues (default, protected)

Structure d'une fonction

- Déclarer une fonction :

```
visibilité typeDeRetour nomFonction(typeArgument1 argument1, typeArgument2  
argument2...){  
    // Liste d'instructions  
}
```

- Exemples :

```
public int somme(int a, int b){  
    // Liste d'instructions  
}  
private void maFonctionSansParametres(){  
    // Liste d'instructions  
}
```


Structure d'une fonction

- Le mot clé “return” permet d’annoncer le retour d’une variable dans une fonction. **Cette instruction est toujours la dernière réalisée par le programme**
1. Ma fonction somme retourne la somme de nombre1 et nombre2.

```
public int somme(int nombre1, int nombre2) {  
    int resultat = nombre1 + nombre2;  
    return resultat;  
}
```


Instructions utiles : Affichage de Texte

- Afficher du texte : `System.out.print("montexte")` Ou `System.out.print(maVariable)`
- On peut ajouter `\n` à la fin pour créer un saut de ligne après l'affichage : `System.out.println`

```
java

System.out.print("Bonjour ");
System.out.print("monde");
```

Sortie :


 Copy code

Bonjour monde

```
java

System.out.println("Bonjour");
System.out.println("monde");
```

Sortie :

 Copy code

Bonjour
monde

Instructions utiles : Interaction utilisateur

- Pour demander à l'utilisateur de saisir du texte, il faut importer une classe existante java qui n'est pas directement dans votre fichier :
 - a. On importe la classe avec l'instruction "**import java.util.Scanner;**" que l'on place en TOUTE première ligne de notre fichier.
 - b. Lorsque l'on veut demander à l'utilisateur de saisir du texte, on crée un objet Scanner : "**Scanner scanner = new Scanner(System.in);**"
 - c. On demande à l'utilisateur de saisir du texte de cette manière :
System.out.print("Veuillez saisir du texte : ");
String texteSaisi = scanner.nextLine(); // Lire la ligne saisie par l'utilisateur
scanner.close();
- Dans cet exemple, la variable `texteSaisi` contient l'entrée de l'utilisateur

TP de validation des acquis JAVA



PARTIE 2 : Reprise des concepts de POO

Structure d'une classe en Java

- Création de ma classe

```
public class VaisseauSpatial {
```

- Création de mes attributs

```
// variables d'instance
```

```
public String nom;  
private int etatProtection;
```

- Création de mes méthodes

```
// méthodes
```

```
public String tirer() {  
    return "Booum!";  
}
```

```
public void reduireProtection (int quantite) {  
    etatProtection -= quantite;  
}
```

```
}
```

Structure d'une classe en java

- Création de ma classe

- Création de mes attributs

- Création de mes méthodes

```
public class VaisseauSpatial {  
  
    public String nom;  
    private int etatProtection;  
  
    public String tirer() {  
        return "Boom!";  
    }  
  
    public void reduireProtection(int quantite) {  
        etatProtection = etatProtection - quantite;  
    }  
}
```

Le constructeur en Java

- Constructeur VaisseauSpatial
- Prend en paramètre le nom
- l'attribut nom prend le paramètre pNom

```
public class VaisseauSpatial {  
  
    public String nom;  
    private int etatProtection;  
  
    public VaisseauSpatial(String nom) {  
        this.nom = nom;  
    }  
  
    public String tirer() {  
        return "Boom!";  
    }  
  
    public void reduireProtection(int quantite) {  
        etatProtection = etatProtection - quantite;  
    }  
}
```


Création et manipulation de mon objet

- On utilise le mot clé “New” pour créer un objet
- Pour manipuler un objet, on appelle le `nomObjet.monAttribut` ou `nomObjet.maFonction()`

```
public static void main(String[] args) {  
  
    // Création de monVaisseau avec le mot clé "new".  
    // On passe en paramètre une chaine de caractère. Ici "jeremy"  
    VaisseauSpatial monVaisseau = new VaisseauSpatial(pNom:"Jeremy");  
  
    // Mon objet est crée, je veux voir ce que contient mon attribut nom  
    // Donc je l'affiche, il doit normalement contenir Jeremy  
    System.out.println(monVaisseau.nom);  
  
}
```

Héritage en Java

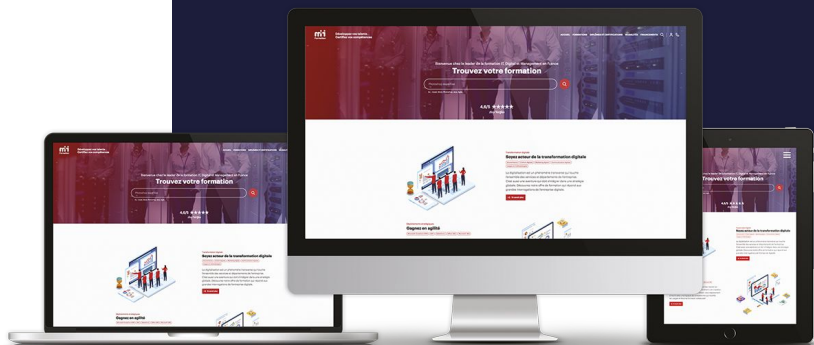
- Pour qu'une classe hérite d'une autre classe (et donc de ses méthodes et attributs) on utilise le mot clé **extends** dans la déclaration

```
// Classe de base (classe parent)
public class Animal {
    public void manger() {
        System.out.println(x:"L'animal mange.");
    }
}

// Sous-classe (classe enfant)
public class Chien extends Animal {
    public void aboyer() {
        System.out.println(x:"Le chien aboie.");
    }
}

Run | Debug
public static void main(String[] args) {
    Chien monChien = new Chien();
    monChien.manger(); // Utilisation de la méthode héritée de la classe parent
    monChien.aboyer(); // Utilisation de la méthode spécifique à la sous-classe
}
```

TP de validation des acquis JAVA



Bonus

- Livre numérique : Coder proprement
- Monter en compétence en programmation : [codewars](#), [condingame](#)

Dossier pédagogique

- Feuilles d'émargement signées pour chaque journée.
- Évaluations formateur
- Évaluations stagiaires

Retour pause déjeuner

**Merci d'avoir suivi cette formation
M2I et à très bientôt !**



Votre formateur

Romain Dinel

Formateur externe M2I

romain.dinel.pro@gmail.com

Linkedin : Romain Dinel

Et encore merci !

UML - Analyse et conception - 25/29 mars

SQL et Bases de Données relationnelles - 4/8 avril