**JAVA OBJET**

Table des matières

[Chapitre 1 INTRODUCTION : 2](#_Toc99212651)

[Chapitre 2 : DECLARATION D’UNE CLASSE : 2](#_Toc99212652)

[Chapitre 3 : L’INTERET DE PLUSIEURS CONSTRUCTEURS POUR UNE CLASSE : 4](#_Toc99212653)

[Chapitre 4 : L’HERITAGE : 5](#_Toc99212654)

[Chapitre 5 CLASSE ABSTRAITE : 7](#_Toc99212655)

[Chapitre 6 : LE POLYMORPHISME : 7](#_Toc99212656)

[Chapitre 7 : INTERFACES : 8](#_Toc99212657)

# 

# INTRODUCTION :

A chaque fois qu’on crée une classe dans java, elle est considérée un type à part entière. Une classe contient des attributs et méthodes.

Dans la majorité des cas, on utilise des classes public ; sinon on ne peut pas appeler cette classe ni ses attributs et méthodes dans d’autres classes.

C’est plus pratique de créer une classe par fichier car dans tous les cas (même si on crée plusieurs classes dans le même fichier) quand on exécute le programme la commande javac génère automatiquement les fichiers .class pour chaque classe.

Attribut : c’est une variable.

Méthode : c’est à l’intérieur des méthodes qu’on écrit du code. (le code est non valable à l’extérieur des méthodes.

Quand on met un point après une instance d’une classe cela veut dire (après le point) on peut écrire toutes les variables (attributs) et méthodes de la classe de l’instance. (Membres = attributs et méthodes).

On dit qu’un programme est de bonne qualité s’il répond à toutes les solutions. Comme par exemple on dit un programme qui solutionne une fonction du 2eme degré est bon si et seulement s’il prend en considérations tous les cas de figures (gestions des exceptions).

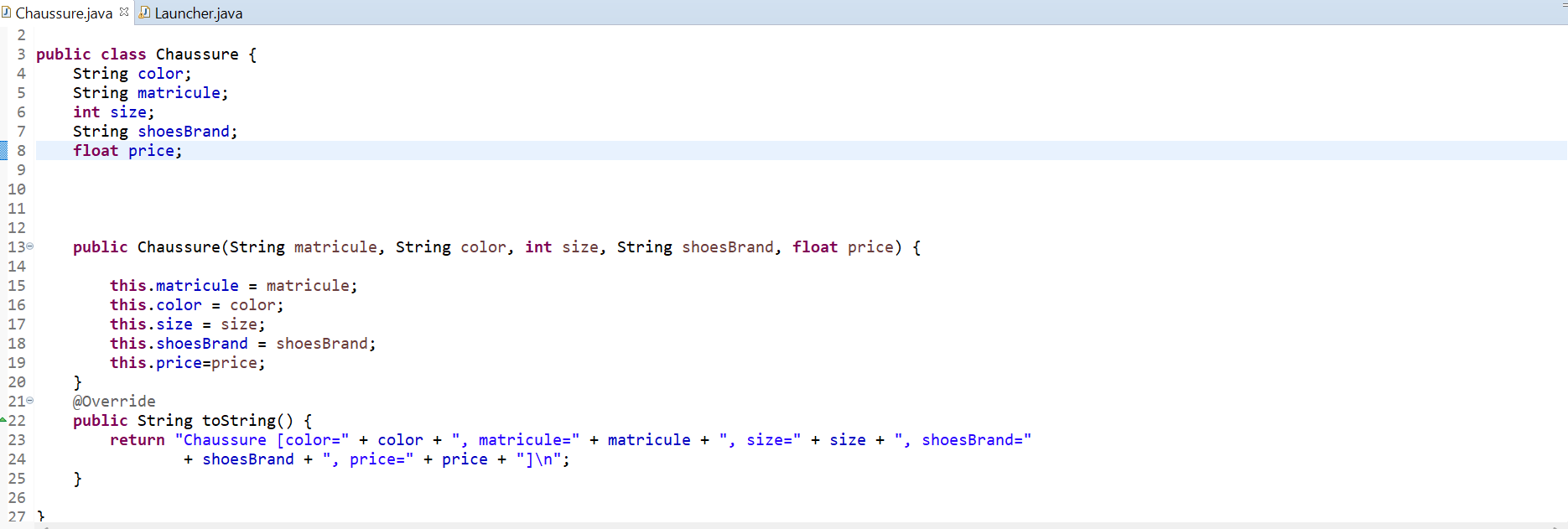
La classe c’est une abstraction de la réalité en gros c’est un modèle. Comme l’exemple évoqué en cours avec la lettre écrite par hakim qui a donné à Merouane pour s’en inspirer mais Fortunat a fait

# : DECLARATION D’UNE CLASSE :

Une classe comprte un corps et un nom

Nom = référence mémoire. Corps = {c’est ce quil y a entre les accolades}

Nom de la classe (Chaussure)



Corps de la classe

Chausssure

Une méthode de la classe Chausssure

Le constructeur de la classe Chausssure

ATTRIBUTS de la classe Chausssure

Si on a besoin d’appeler la classe Chaussure dans une autre classe alors on crée une instanciation de la classe de la façon suivante.

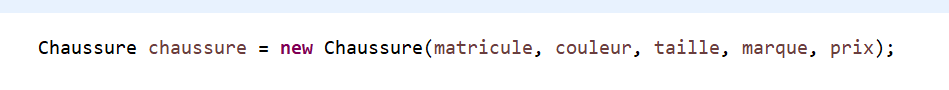


Figure  : instanciation qu’on appelle chaussure de la classe chaussure dans une autre classe

L’instance chaussure de la classe Chaussure est un Objet de la classe Chaussure. L’instance est plus précise que l’objet.

This on ne peut pas l’utiliser dans un contexte static.

Un paramètre implicite c’est un paramètre qui est caché dans l’instance.

Pour calculer par exemple la surface d’un cercle on utilise la méthode area() mais cette méthode est définie dans la classe Circle et cette classe c’est elle qui contient le paramètre r (rayon).

Donc on écrit c.area() (c étant une instance de classe Circle).

This. 🡺 nous propose la liste des choix à l’intérieur de la même classe.

Instance. 🡺 nous propose la liste des choix à l’extérieur de la classe.

Garbage collector (System.gc()) permet de supprimer toutes la variables dans la mémoire qui ne sont plus utilisables .

Pour comparer deux objets de type String il faut toujours utiliser .equals

A chaque fois qu’on crée une variable dans un programme, au moment de l’initialisation faut penser à son utilité et à ses différentes valeurs dans le programme (gestion des exceptions).

La création de package est une façon de regrouper les classes qui font les mêmes fonction :

Connexion à la base de données.

Interception de l’information par l’IHM.

Relais des informations entre l’IHM et la base de données.

On dit qu’une classe 1 est indépendante de la classe2 s’il existe une méthode de la classe 2 qui est utilisée dans la classe1.

Quand on appelle une méthode une méthode non public dans un projet 🡺 cette méthode on ne pourra l’appeler que dans le package auquel elle appartient.

# : L’INTERET DE PLUSIEURS CONSTRUCTEURS POUR UNE CLASSE :

Exemple : On a la classe suivante :

Public Person, définie par deux constructeurs :

1. un avec des paramètres : Person (String nom, String prenom, int id) ;
2. un autre constructeur sans paramètre : Person() ;

On a une méthode suivante qui se connecte à la base de données pour récupérer l’identité de la personne en connaissant son identifiant (retourne Person) :

Public Person selectPersonById (int id) {

String query = “select \* from tabPerson where id “ = +id;

RS rs= DB.getConnection.execute(query); // (synthaxe pour se connecter à la base de données et récupérer un objet rs de type RS (RS est un curseur en langage sql).

While (rs.hasNext()) { // quand il n’y a plus de lignes la boucle s’arrête)

Rs.next() ;

Person p = new Person() ; //ON appelle le 1er constructeur à vide

p.setId(rs.getInt(« id”) //on change la valeur de Id grâce au setteur set id

ou p.setId(rs.getInt(0)) / / colonne 0 du tableau correspond à la colonne id

p.setNom(rs.getString(« nom”);

ou

p.setNom(rs.getString( 1 ) ;

p.setPrenom(rs.getString(« prenom”);

ou

p.setprenom(rs.getString(2); // maintenant p il a des paramètres p(nom, prénom, id), donc si on fait retour p on retourne p avec des paramètres d’où la nécessité d’avoir plusieurs paramètres pour une classe ; }return p }

A chaque fois qu’on crée une classe dans java derrière c’est l’Object qui réserve la mémoire pour cette classe.

# : L’HERITAGE :

On dit qu’une classe B hérite d’une classe A si la classe B contient les membres (attributs et méthodes) de classe B.

-nom

-prenom

-age

-etc..

On gagne dans la précision

-grade

-filière enseignée

-planning

-etc..

General

Person

On perd dans la précision

Prof

Etudiant

Administrateur

particulier

-description de poste

-lieu d’intervention

-Ecole d’inscription

-notes

-modules

-etc..

L’étudiant le professeur et l’administrateur avant qu’ils y soient ils étaient d’abord des Person. Car ils partagent tous l’âge, un nom, un prénom.

Dans la réalité on dit que l’étudiant EST une Person. Pour le compilateur ce verbe être est remplacé par le mots clé **extends.**

On écrit alors Etudiant extends Person ; et à partir de ce moment on peut appeler tous les membres de la classe qui ne sont pas privé.

Implicitement toutes les classes qu’on crée dans java héritent de Object (objet).

En java il n’existe pas de l’héritage multiple (une classe peut hériter uniquement et seulement d’une seule classe)

A chaque fois qu’on crée un object « a » = new object Administrateur() ; on crée derrière un object de type Person (mais on ne le voit pas).

Dans le jargon on dit que Etudiant est une classe fille (sous classe) de Person

Et Person est la classe (mère) superclasse de Etudiant.

Il existe des classes abstraites et des méthodes abstraites. Quand une classe contient au moins une méthode abstraite on dit que la classe est abstraite

Méthode abstraite = méthode qui n’est pas implémentée.

**NB :** la différence majeure entre un constructeur et une méthode c’est que grâce au constructeur on peut créer une instance de la classe avec laquelle on peut appeler la méthode de classe.

Propriétés d’une classe = ses membres = attributs et méthodes.

Final = pour les attributs dont on ne peut modifier la valeur

Final = pour les classes veut dire qu’on ne peut plus créer des classes à partir de cette final classe

N’importe quel objet de type supérieur peut recevoir un objet de type inférieur.

Dans la diapo exemple :

e = c (on perd la valeur initiale de e qui vient de la classe sup, et c vient de la classe inf)

e devient c mais on ne peut pas appeler les nouvelles méthodes de c par l’instance e (même s’il prend sa forme) car e à la base ne prend que les méthode de E.

par contre si on fait c = e (erreur d’incompatibilité et on doit encastrer.

**Surcharge de méthode (overloading) :**

Classe A {

Public method (){

Syso (« salut ») }

}

Classe B {

Public method (String nom ){

Syso (« salut » + nom)

}

}

La surcharge est de créer une autre méthode avec une autre signature.

**ECRASEMENT OVERRIDING :**

On ne peut pas appeler la méthode de la haut par les instances de la classe fille si la méthode a la même signature et qui a été redéfinie dans la classe fille.

# CLASSE ABSTRAITE :

Tous les éléments (attributs) & comportements (méthodes vb,n) d’une classe non connus.

Dans une classe abstraite il existe au moins une méthode dont le comportement est inconnu est une méthode abstraite

Elle n’est donc pas instanciable, mais sert avant tout à **factoriser du code**.

Une classe qui hérite d’une classe abstraite doit obligatoirement implémenter les méthodes manquantes (qui ont été elles-mêmes déclarées « abstraites » dans la classe parente). En revanche, elle n’est pas obligée de réimplémenter les méthodes déjà implémentées dans la classe parente (d’où une maintenance du code plus facile).

Méthode abstraite 🡺 classe abstraite ;

On crée une classe abstraite en précèdent le mots clé **abstract** au nom de classe.

On crée des classes abstraites pour qu’elles soient héritées.

**La différence entre une classe abstraite et une interface :**

Classes abstraites et interfaces ont chacune une fonction bien distincte : les classes abstraites servent à **factoriser du code**, tandis que les interfaces **servent à définir des contrats de service.**

Et pourquoi ne pas utiliser des classes abstraites (dans lesquelles aucune méthode ne serait implémentée) en lieu et place des interfaces ? :  La réponse est simple : dans la plupart des langages actuels (c’est notamment le cas de Java, C#, PHP), il n’est possible pour une classe d’hériter que d’une seule classe parente (abstraite ou non), mais d’implémenter plusieurs interfaces.

# : LE POLYMORPHISME :

Poly : plusieurs, morphisme : formes,

C’est la capacité d’un objet de prendre plusieurs formes.

**Super** est une référence (instance) dynamique. Si on met un membre (méthode ou attribut) en statut **protected** cela signifie que ce membre ne peut être appelé que par les classes fille de cette classes.

La possibilité de redéfinir une méthode dans des classes héritant d'une classe de base s'appelle la **spécialisation**. Il est alors possible d'appeler la méthode d'un objet sans se soucier de son type intrinsèque : il s'agit du **polymorphisme d'héritage**. Ceci permet de faire abstraction des détails des classes spécialisées d'une famille d'objet, en les masquant par une interface commune (qui est la classe de base).

Imaginons un jeu d'échec comportant des **objets *roi*, *reine*, *fou*, *cavalier*, *tour* et *pion***, héritant chacun de l'objet ***pièce*.** La méthode ***mouvement()*** pourra, grâce au polymorphisme d'héritage, effectuer le mouvement approprié en fonction de la classe de l'objet référencé au moment de l'appel. Cela permettra notamment au programme de dire ***Piece.Mouvement*** sans avoir à se préoccuper de la classe de la pièce

# Chapitre 7 : INTERFACES :

Une interface (contrat) permet de définir un ensemble de services qu’un client peut obtenir d’un objet. Une interface introduit une abstraction pure qui permet un découplage maximal entre un service et son implémentation. On retrouve ainsi les interfaces au cœur de l’implémentation de beaucoup de bibliothèques et de frameworks. Le mécanisme des interfaces permet d’introduire également une forme simplifiée d’héritage multiple.

Une interface c’est une classe mais ce n’est pas une classe

Toutes ses méthodes sont abstraites. A partir de java 8 on peut mettre des méthodes non abstraites dans les interfaces mais il faut les procéder de défault.

Une classe en java peut hériter d’une seule classe mais elle peut implémenter plusieurs interfaces.

L’un des avantages phares des interfaces est de se débarrasser des indépendances transitives fortes qui tuent le code. Une classe ne dépend plus d’une autre classe mais implémente des méthodes de l’interface.

**TP BATCH ALLIMENTATION DE BASE DE DONNEES PAR LECTURE DE DIFFERENTS TYPES DE FICHIERS (csv, xml, jason,) :**

**EAI =**

Enterprise Application Integration (EAI) pour le terme consacré en anglais, est aussi appelé Echanges Inter-applicatifs de Données en français. Ce terme désigne une architecture SI qui permet à plusieurs logiciels de communiquer entre eux et de gérer les différents échanges. On parle aussi de gestion des flux inter-applicatifs. Par raccourci, les outils logiciels middleware utilisés pour réaliser ces architectures EAI sont eux aussi appelés « logiciels EAI ».

Un logiciel EAI permet de se connecter à tout type de sources de données, d’en extraire des données, de manipuler ces données, de structurer ces données puis de les déposer dans toutes autres structures de données. L’exécution de ces manipulations et la synchronisation de données est programmable et déclenchée par tout type d’évènements.

A l’aire du BigData, la donnée est devenue la matière première des Systèmes d’Information dans de nombreuses entreprises, il paraît évident de pouvoir l’utiliser dans plusieurs logiciels. En effet, la DSI d’aujourd’hui veut utiliser les données de l’ensemble de ses infrastructures logicielles dans différents outils en parallèle. Tout en assurant une qualité et une fiabilité à toutes épreuves.

Finalement, les EAI ont pour objectifs de rassembler, de router et de synchroniser les données issues des différentes applications. De fait, cela évite d’interfacer directement les applicatifs les uns avec les autres et créer ainsi une architecture complexe et emmêlée (spaghetti).

**ESB=**

L'ESB (entreprise service bus) peut être considérée comme une nouvelle génération d'[intégration d'applications d'entreprise](https://fr.wikipedia.org/wiki/Int%C3%A9gration_d%27applications_d%27entreprise) (EAI) construite sur des standards comme [XML](https://fr.wikipedia.org/wiki/XML), [Java Message Service](https://fr.wikipedia.org/wiki/Java_Message_Service) (JMS) ou encore les services web. La différence majeure avec l'[EAI](https://fr.wikipedia.org/wiki/Int%C3%A9gration_d%27applications_d%27entreprise) est que l'ESB propose une intégration complètement distribuée grâce à l'utilisation des conteneurs de services[1](https://fr.wikipedia.org/wiki/Enterprise_service_bus#cite_note-1). Ces « mini-serveurs » contiennent la logique d'intégration et peuvent être déposés n'importe où sur le réseau.

**(FACTORY)Fabrique d’objet =**

La fabrique permet de créer un objet dont le type dépend du contexte : cet objet fait partie d'un ensemble de sous-classes. L'objet retourné par la fabrique est donc toujours du type de la classe mère mais grâce au **polymorphisme** les traitements exécutés sont ceux de l'instance créée.

L'utilisation d'une fabrique permet de rendre l'instanciation d'objets plus flexible que l'utilisation de l'opérateur d'instanciation new.

**Singleton =**

Ce motif de conception propose de n'avoir qu'une seule et unique instance d'une classe dans une application. (le cas du scanner par exemple dans le projet my-library)

**POJO =**

Plain Old Java Object. Fondamentalement, une classe avec des attributs et des getters et les setters.