R2.01 - Développement Orienté Objets (DOO)

EX 2 : Objets et Classes

EXA : Objets, constructeurs

Lire le poly CH 02 de cours, notamment la partie sur les constructeurs.

Dans le cadre d'une application de gestion de meubles, on désire représenter des tables de différentes sortes.
Une table est caractérisée par un nombre de pieds, une couleur (de type String) et un prix.
On n'indiquera pour l'instant aucun modificateur d'accès devant les attributs (pas de public, private …) : nous verrons cela au chapitre 3 !
Nous allons commencer par la phase de conception en créant un schéma UML de la classe Table.

1) Créer un diagramme de classes UML pour une classe Table
   a) Lancer Visual Paradigm (accessible depuis le menu Windows)
   (vous pouvez aussi le télécharger sur le portail de l'IUT > Département Informatique > Visual Paradigm)
   b) Créez un nouveau projet.
   c) Sélectionnez pour "Data type set" : Java (cf le tutoriel https://www.visual-paradigm.com/tutorials/uml-class-diagram-in-diff-programming-languages.jsp) : cela permet d'utiliser ensuite des types de Java comme String (avec une majuscule ;-)
   d) A gauche, dans l'onglet vertical "Navigateur de diagrammes",
   Cliquez sur "Diagrammes UML"
   puis cliquez sur " Diagramme de classe",
   puis bouton droit de la souris "Nouveau Diagramme de Classe".
   Cliquez sur "Classe" puis cliquez dans l'espace de dessin => cela va ajouter un rectangle représentant une nouvelle classe.
   e) Dans le rectangle représentant la classe, tapez son nom (ici "Table").
   f) Cliquez sur le petit rond en haut à droite du rectangle de la classe "open
   g) specifications", puis onglet "Attributes", puis bouton "Add".
   h) Spécifiez pour chaque attribut, nom, visibility (unspecified) et son type.
   i) Dans le menu "Tools", "Code", "Générate Java". Puis Generate C++.
   j) Ouvrez le fichiers Table.h sous Visual Studio et regardez le code C++ qui a été généré (il n'y a pas de fichier .cpp généré car il n'y a pas encore de constructeur de méthodes)
   k) Ouvrez les fichiers Table.java (si besoin associer l'application Eclipse pour ouvrir les fichiers .java) et examinez le code Java qui a été généré.

2) Ouvrez Eclipse et créez un nouveau projet “EX-02”.
   Dans un explorateur Windows, dans votre dossier “VPProjects”, cliquez sur votre fichier Table.java et déposez le dans votre dossier src de votre projet Eclipse.

3) Ajouter un constructeur sans paramètre. Le constructeur doit avoir le même nom que la classe (donc commencer par une majuscule !). Ce n’est ni une procédure, ni une fonction : il n’est pas précédé de void et ne retourne pas de valeur. Dans ce constructeur, initialisez par défaut le nombre de pieds à 4.
4) Ajoutez une méthode main() dans la classe Table. Créez une table et affichez la valeur de ses attributs.
5) Ajoutez d’autres constructeurs avec paramètres de manière à pouvoir exécuter le main() suivant:

```
public static void main (String args []) {
     Table t = new Table ();
     System.out.println (t.nbPieds);
     System.out.println (t.couleur);
     System.out.println (t.prix);
     System.out.println ();

     Table t2 = new Table (3);
     System.out.println (t2.nbPieds);
     System.out.println (t2.couleur);
     System.out.println (t2.prix);
     System.out.println ();

     Table t3 = new Table (6, Table.NOIR);
     System.out.println (t3.nbPieds);
     System.out.println (t3.couleur);
     System.out.println (t3.prix);
     System.out.println ();
}
```

```
EXECUTION
     4
     Marron
     0.0

     3
     Marron
     0.0

     6
     Noir
     0.0
```

Utilisez toujours pour le nombre de pieds, un paramètre qui a exactement le même nom que l'attribut (sauf pour le constructeur sans paramètres ;-).
Vous devez lever cette ambiguïté en utilisant le mot clé « this. » devant cet attribut : vous devez faire référence à l'attribut de l'objet Table en cours de construction avec l'expression « this.nbPieds ».
Pour les autres paramètres et attributs, vous choisirez leurs identificateurs de manière à ne pas avoir à utiliser *this*.

6) Mettez en commentaires le constructeur qui n'a pas de paramètres. Essayez de créer un objet avec l'expression new Table(). Que remarquez-vous ?

- ……

Pour ajouter le constructeur dans Visual Paradigm : cliquer sur la classe puis bouton droit de la souris, menu "Add" > Constructor

EX B : Références et objets

1. Complétez le programme main en déclarant une référence à une table t4.
   Ne créez pas de nouvel objet.
2. Faites pointer la référence t4 vers le même objet que celui pointé par t3.
3. Affecter le prix de cette table à 80 en passant par la référence t3.
4. Afficher le nouveau prix via la référence t4.
5. Faites un schéma de la mémoire 1) avec ruban, 2) avec boîtes et flèches.

EX C : Tableaux d'objets

Nous allons maintenant gérer un ensemble de tables à l'aide des tableaux d'objets,
par exemple pour pouvoir représenter des magasins qui vendent plusieurs tables.

1) Version 1 (pas orienté objet … mais rapide ;-) tout dans la méthode main() :

Relisez l'exemple du Cabinet d'Assurances dans le poly CH 02 et inspirez-vous en pour la suite :

- Dans le main de Table déclarez un tableau de tables (on n'utilise pour l'instant pas la classe ArrayList : vous devez donc allouer le tableau en spécifiant une taille maximum et gérer un nombre de tables présentes dans le tableau)
- Allouez un tableau de 4 références vers des tables.
- Créez 3 tables et stockez leur adresses dans les 3 premières cases du tableau.
- Parcourez les *4* cases du tableau, affichez le prix de chaque table pointée par la case du tableau et calculez le total des prix.
- Est-ce que vous obtenez un message d'erreur ? Lequel ? pourquoi ?
- Eviter cette erreur en ajoutant un test avant d'accéder à l'attribut prix.

2) Version 2 : orientée objet : classe Magasin avec un tableau d'objets

La version du 1) avec tout dans la méthode main() n'est vraiment pas ré-utilisable : elle n'est pas orientée objet :-(

- Définissez dans Eclipse une nouvelle classe `Magasin` avec :
  - la déclaration d'attributs :
    - constante NB_MAX égale au nombre de "Table" maximal dans un "Magasin" à définir soi-même
    - un nom de magasin de type String,
    - un tableau de `Table`
    - un nombre de tables stockées dans le tableau
  - un constructeur : passez en paramètre au constructeur
    - le nom du magasin
    - le nombre de tables maximum que le magasin aura à gérer
  - une méthode add qui ajoute une Table t passée en paramètre
  - une méthode main (aussi dans la classe Magasin) pour tester votre classe : créez deux magasins ikeaEvry, ikea, ajouter des tables à chaque magasin, afficher le contenu de chaque magasin

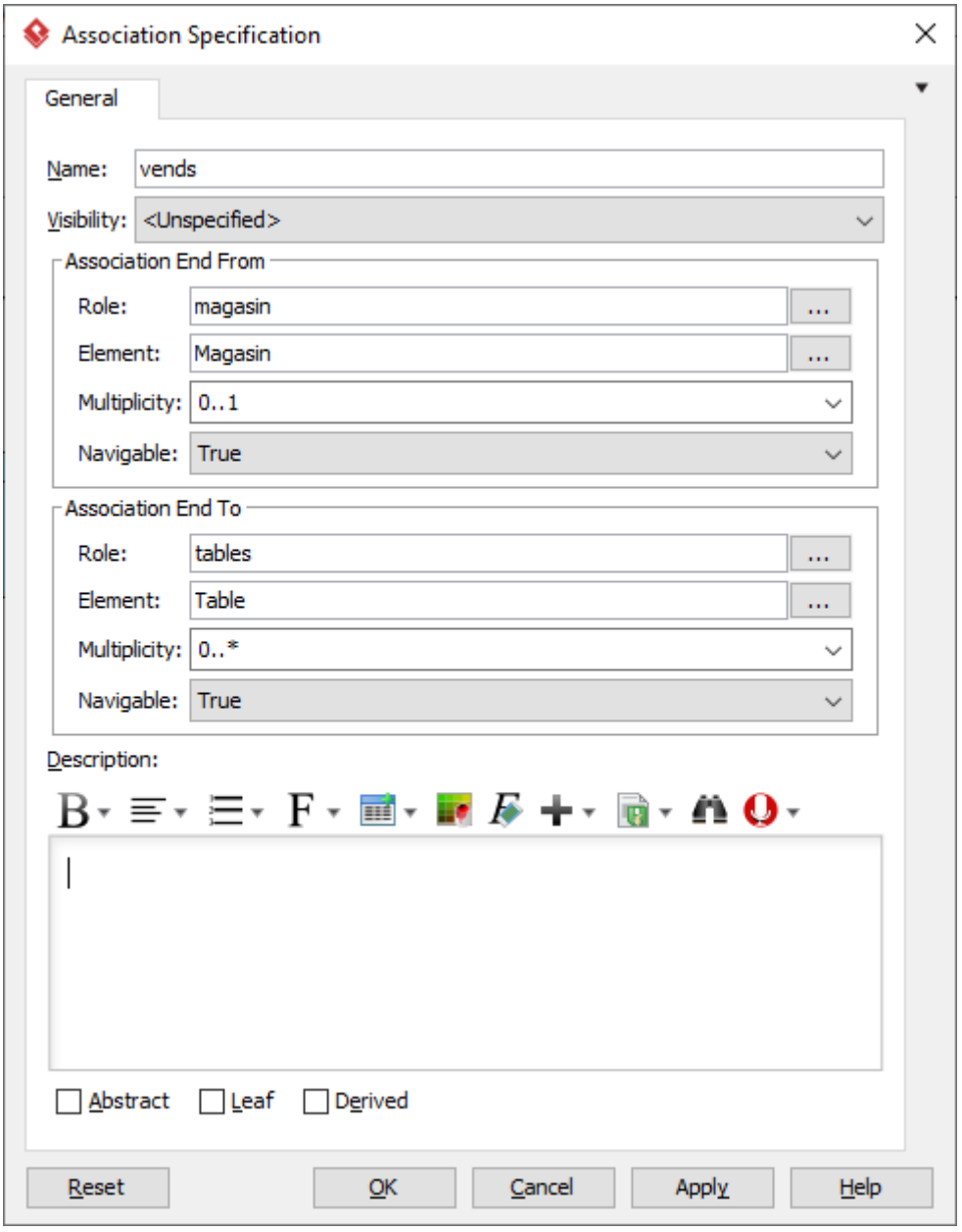
3) Conception avec Visual Paradigm et UML

Normalement on commence par une phase d'analyse et de conception avant de programmer en Java ! Nous allons utiliser le concept d'Association entre deux classes pour représenter le lien entre la classe `Magasin` et la classe `Table`.

Lancez Visual Paradigm et ouvrez votre projet dans lequel se trouve le diagramme de classe avec la classe `Table`.

Ajoutez dans le même schéma de classe une nouvelle classe `Magasin` avec juste un nom de type String comme attribut, un constructeur (même paramètres qu'à la question 2) et une méthode *add* qui ajoute une table passée en paramètre à la méthode

Dans le menu à gauche, cliquez sur "Association", puis tracez un trait entre vos deux classes Magasin et Table. Un curseur apparaît pour vous proposer de saisir un nom d'association. Tapez "vends". Cliquez sur le trait, puis sur le bouton droit de la souris et sélectionnez dans le menu "Open Specifications". Remplissez les éléments comme indiqué ci-dessous :



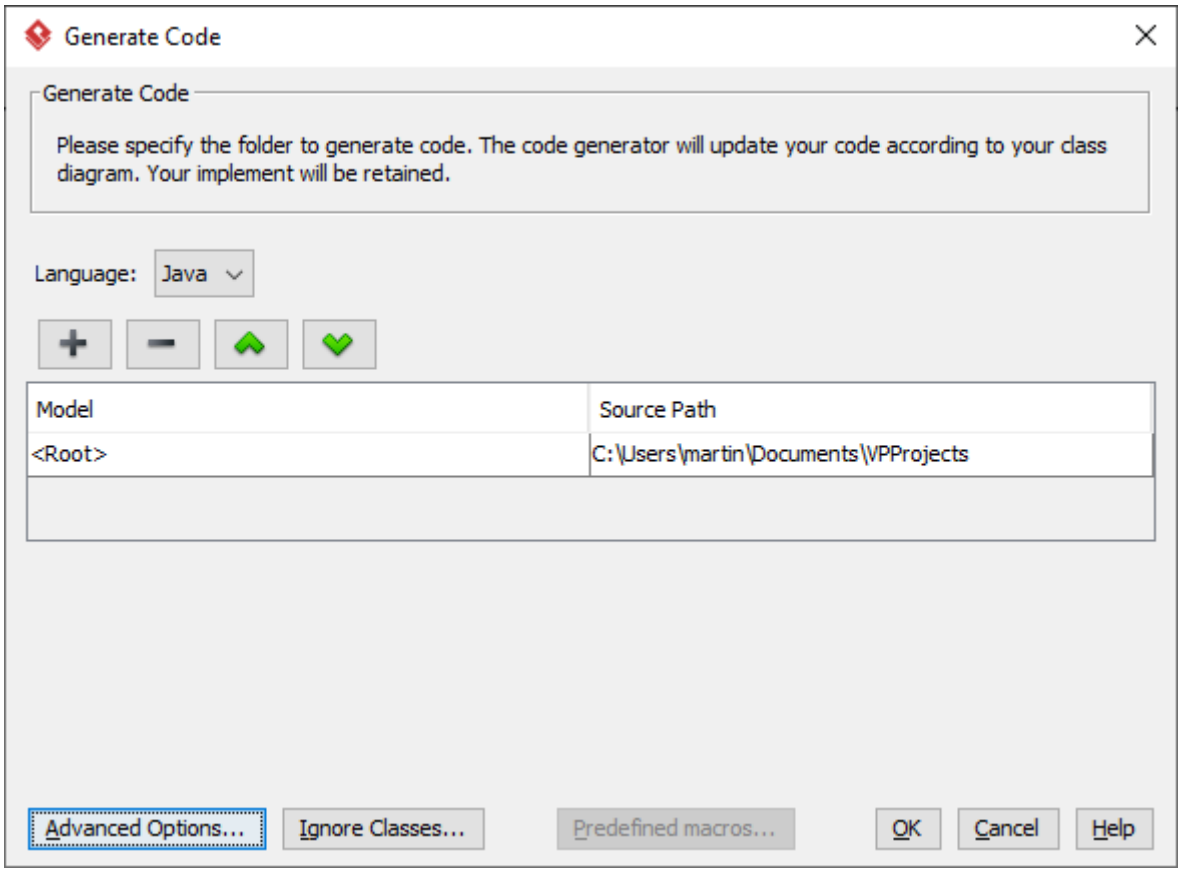
Les valeurs pour Multiplicity indiquent qu'un Magasin *a plusieurs* tables.

Pensez à indiquer un nom pour chaque rôle : ce sera le nom de l'attribut correspondant dans le code Java (si vous n'indiquez pas de valeur pour le champ rôle, Visual Paradigm ne va pas générer de déclaration d'attribut dans Java : il ne saurait pas comment appeler l'attribut !).

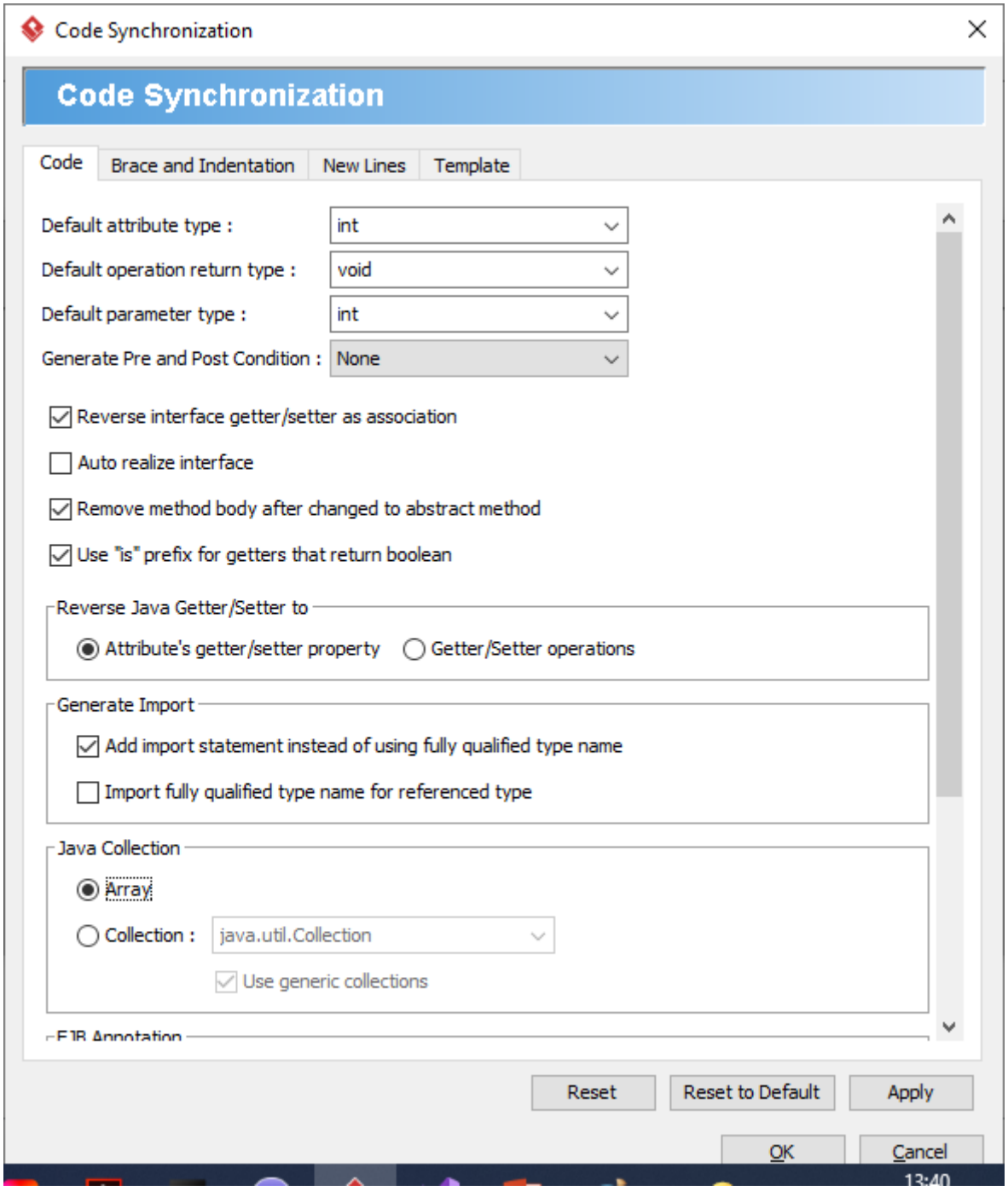
Votre schéma de classes doit maintenant ressembler à cela :

Traduisez ensuite ce diagramme de classe en java en cliquant sur le menu Tools, puis Code, Generate Java Code.

Cliquez sur le bouton “Advanced Options”, en bas à gauche.



Dans la partie “Collections”, cliquez sur “Array” :



Cliquez sur Ok pour générer les sources Java.
Avec Windows Explorer, allez dans le dossier de vos projets Visual Paradigm et ouvrez les fichiers Magasin.java et Table.java.

Examinez le code Java.
Dans Visual Paradigm, complétez votre classe Magasin :

- Ajoutez un attribut nbTables qui contiendra le nombre de tables actuellement vendues par le magasin
- Le constructeur avec les 2 paramètres
- La méthode add pour ajouter une table au tableau
- Regénérez votre code Java et ouvrez les classes Java

EX D : static vs. non static : gérer une pile de données

Une pile est une structure de données qui permet d'empiler des données (par exemple des scores, des codes d'articles, …).
Cette structure de données se manipule de la manière suivante :
On peut ajouter un nouvel élément au sommet de la pile.
On peut demander à voir quel est l'élément au sommet de la pile.
On peut dépiler un élément (ou plusieurs en même temps).
Dans cet exercice nous allons programmer une classe qui permet de gérer une pile de données avec ces opérations. Nous allons le faire de deux manières différentes.

1) Commencez par programmer une classe `PileStatic` avec

- un attribut **static** de type tableau d'entier
- un attribut **static** nbValeurs qui correspond au nombre de valeurs actuellement stockées dans le tableau
- une constante **static** NB_MAX = 100 qui correspond au nombre maximum de valeurs que l'on peut mettre dans la pile
- une méthode **static** void empiler (int i)
- une méthode **static** void afficherSommet() qui affiche la valeur actuellement sur le sommet de la pile
- une méthode **static** void empiler (int i, int j) qui permet d'empiler deux valeurs d'un seul coup
  (on dit que la méthode empiler est SURCHARGEE / OVERLOADED car elle existe en plusieurs versions avec différents types ou nombres de paramètres)
- Une méthode **static** void depiler () qui enlève la valeur actuellement au sommet de la pile
- une méthode **static** main qui permet de tester votre classe

La version de la classe PileStatic que vous venez de programmer n'est pas de l'orienté objet :-(
On ne manipule que des variables globales dans toute la classe.
On n'a pas besoin de créer d'objets.

Est-ce que par exemple vous pouvez créer plusieurs piles
(par exemple une pile de scores, une pile de nombre d'articles, …)
!……

Vous devez éviter d'utiliser ce mot-clé static : ce n'est pas de l'orientée objet.

2) Programmez maintenant une classe `PileObjet` qui utilise le moins possible le mot clé static :
c'est du véritable orienté objet avec des attributs d'objets et des méthodes d'objet !

Est-ce que vous pouvez créer plusieurs piles par exemple une pile de scores, une pile de nombre d'articles, …
? ……

EX E : Passage de paramètres

En Java, le passage de paramètres se fait par …… .

Est-ce que dans un sous-programme, on peut modifier une variable de type de base (par exemple int) passée en paramètre ? ……

Dans un sous-programme auquel on passe une référence o vers un objet en paramètre :

- On peut modifier …… ……
- On ne peut pas modifier ……..

Qu'affiche le programme suivant ?

```
public class A {
    int na ;

    public static void main (String args []) {
        B b = new B();
        A a1 = new A();
        a1.na = 2 ;
        System.out.println ("Avant appel : " + a1.na);
        System.out.println ("Avant appel : " + a1);
        b.m(a1);
        System.out.println ("Après appel : " + a1.na);
        System.out.println ("Après appel : " + a1);
    }
}
class B {
    int nb ;
     void m(A a2) {
        a2.na = 10 ;
        a2 = new A();
        System.out.println ("Durant appel : " + a2);
    }
}
```

Exécutez le.

- ……
  ……
  ……
  ……
  ……
  ……
  ……

Dessinez l'état de la mémoire en mode ruban.

EX G : complément du cours (optionnel)

Cf page 6 du polycopié chapitre 2
"Application
Compléter la classe Personne en ajoutant un attribut numéroSecu
Compléter le main de la classe VoitureProprietaire en créant une 2ème voiture.

Le propriétaire de cette voiture sera le même pour les 2 voitures.
Faire un schéma de la mémoire boite et flèches. "

EX H : site de streaming (optionnel)

Programmez en Java un programme qui gère un site de streaming avec des films, des séries télé avec plusieurs saisons, une liste de favoris, plusieurs comptes pour le même client, un timestamp par vidéo qui mémorise où en est la lecture dans les différentes vidéos, …

L'affichage se fera simplement avec des println pour l'instant
et le choix d'un film avec un Scanner et un menu affiché à l'écran

EX I : pour les étudiants qui ont fait facilement tous les autres exercices : Jeu d'échec (optionnel)

Programmez en Java un programme qui permet de jouer aux échecs.
Il y a plein de documentation sur Internet !

Commencez par représenter avec les classes, attributs et méthodes les concepts de :

- Case de l'échiquier
- Echiquier
- Pièce
- ...

https://harmoniaphilosophica.com/2018/07/23/how-to-code-a-chess-program-in-one-day-c-and-java-examples/

Avec des concepts que nous n'avons pas encore vu (classes abstraites) :
https://www.youtube.com/watch?v=h8fSdSUKttk

Annexe

Quelques tutoriels pour aller plus loin si vous le souhaitez :

- Tutoriel UML Visual Paradigm
  https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/
- Association Vs. Aggregation Vs. Composition
  https://www.guru99.com/association-aggregation-composition-difference.html