Notes sur les tutoriaux d'Oracle

Une petite collection de notes concernant le tutoriel Java d'Oracle. On suivra l'ordre des deux premiers trails du tutorial :

- getting started
- learning the Java langage

Trail getting started

https://docs.oracle.com/javase/tutorial/getStarted/index.html

Une fois le tutoriel lu, vous pourrez l'exécuter en suivant la partie utiliser intellij.

Trail learning the Java Langage

https://docs.oracle.com/javase/tutorial/java/index.html

Concepts

https://docs.oracle.com/javase/tutorial/java/concepts/index.html

Pendant Java de la partie paradigme objet et modelisation uml du cours.

Language basics

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html

Variable

En java tout est typé. Une variable ne peut contenir que des objets d'un seul type. On les déclare ainsi :

```
type nom;
```

Le code ci-dessus a déclaré une variable qui s'appelle `nom` et qui peut contenir des objets de type `type`. Exemple :

```
int x;
String nom;
```

Type primitif

Un énorme piège de Java est que bien qu'il soit un langage objet, le poids de l'histoire fait que les nombres (int float et double) ne sont **pas** des objets. Ce sont des types primitifs.

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

- **int** : entier sur 32 bits
- **float** : réel flottant sur 32 bit
- **double** : réel sur 64 bits
- **char** : nombre correspondant à un caractère (16-bit unicode)
- **boolean** : true et false

En gros les nombres et les booléens. Ce ne sont **pas** des objets (leur type ne commence pas par des majuscules).

- Ils ne possèdent **pas** de méthodes.
- L'égalité se fait avec l'opérateur ==.

tableaux

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Les tableaux en Java :

- sont fixes (on ne peut pas en augmenter la taille)
- contiennent uniquement un type d'objet
- sont construits avec `new typeObjet[nombreObjet];` ou `new typeObjet[] {objet1, …, objetn};`

Ne confondez pas :

- la variable : `int[] tableauEntier;`
- et l'objet : `tableauEntier = new int[10];` (construction avec new)

`tableauEntier` est une variable pouvant contenir un tableau, pas le tableau !

la classe java.util.Arrays contient plein de méthodes utiles pour les tableaux :

- `Arrays.toString()` pour convertir un tableau en chaîne de caractères.
- `Arrays.sort()` pour trier

```
package com.mco;


import java.util.Arrays;

public class Main {
```

```
    public static void main(String[] args) {

        int[] tableauEntier;

        tableauEntier = new int[] {1, 3, 2, 6, 4, 5};

        System.out.println(tableauEntier);
        String tableauConvertiEnString = Arrays.toString(tableauEntier);
        System.out.println(tableauConvertiEnString);

        Arrays.sort(tableauEntier);

        System.out.println(Arrays.toString(tableauEntier));
    }
}
```

La création de l'objet tableau ne crée par les objets. On suppose que j'ai une classe `Card` qui représente une carte Son constructeur est `new Card(int value, String color)`. Le code suivant fait dans l'ordre :

1. crée une variable pouvant contenir un tableau de 2 cartes,
2. crée le tableau,
3. pour chaque case du table crée un objet `Card` que l'on met dedans.

La création des objet à mettre dans un tableau est **indispensable**.

```
Card [] deuxCartes;
deuxCartes = new Card[2];

deuxCartes[0] = new Card(1, "SPADE");
deuxCartes[1] = new Card(7, "HEART");
```

Si l'on omet la création des 2 cartes, les deux objets sont `null` qui est l'objet vide en Java.

Operator

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

Récapitulatif

Je préfère que vous utilisiez `+= 1` et `-= 1` à la place de `++` et `--` qui sont certes classe et font hacker mais peuvent vous trahir. Par exemple qu'affiche le code suivant :

```
int i = 1;
```

```
System.out.println(i++);
```

Expressions, Statements, and Blocks

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html

En python les blocs sont déterminés par des indentations, en Java par des accolades. On indentera également pour des questions de lisibilité.

Je sais bien que des blocs d'une seule ligne n'ont pas besoin d'accolades mais **mettez-en tout de même !**

Un code est fait pour évoluer et lorsque l'on rajoute une instruction on oublie souvent de rajouter l'accolade. Par exemple le code suivant ne fait **pas** ce que l'on pense :

```
int i;
for (i=0 ; i < 10 ; i += 1)
    System.out.println("J'affiche i : ");
    System.out.println(i);
```

Control Flow Statements

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/flow.html

Boucles For : https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html

Pour les boucles `for` :

- préférez dans la mesure du possible la forme en *pour chaque* (`for (int item: numbers) {…}`) qui ressemble au *pour chaque* de python. Elle est plus claire.
- avec les stream de java 8 (bien au-delà de ce cours introductif), il est possible d'avoir une forme de range : intstream en java.

Lesson: Classes and Objects

https://docs.oracle.com/javase/tutorial/java/javaOO/index.html

Comprenez bien cette partie. Tout l'intérêt de Java est la dedans !

Classes

https://docs.oracle.com/javase/tutorial/java/javaOO/classes.html

Pour Java tout est une classe, même le point de départ de notre programme.

Le fichier `Main.java` contient une méthode statique `main` qui est le point de départ de notre programme. Par défaut, lorsque l'on coche *Command Line Application*, Intellij va créer un fichier **Main.java** Contenant une méthode statique `main` qui sera le point de départ de notre application :

```
package com.exemple;

public class Main {

    public static void main(String[] args) {
       //le code
    }
}
```

Declaring Classes

https://docs.oracle.com/javase/tutorial/java/javaOO/classdecl.html

Un programme Java est constitué de classes. Chaque classe est décrite dans un fichier ayant le nom de la classe et chaque fichier ne contient qu'une seule classe. Fichier `MaClasse.java` typique :

```
package com.domaine.application;

import les.méthodes.Utilisée.ici;
import mais.définies.ailleurs;

class MaClasse {

  // des attributs, constructeurs et méthodes.
}
```

Une classe particulière (donc un fichier) contient une méthode statique nommée "main" qui constituera le point de départ du programme.

Declaring Member Variables

https://docs.oracle.com/javase/tutorial/java/javaOO/variables.html

La plupart du temps les attributs sont réservés au package ou privés (accessibles directement uniquement par la classe) et on y accède *via* des méthodes dites *getter* (`getAttribut` pourconnaitre l'attribut) et *setter* (`setAttribut` pour changer l'attribut).

Fichier `MaClasse.java` suite :

```
// package et imports

class MaClasse {
  private type1 attribut;

  public type1 getAttribut();
  public void  setAttribut(type1 attribut);

  // constructeurs et méthodes.

}
```

Defining Methods

https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html

Une méthode est définie par une **signature** : typeRetour nomMethode(typeParametre1 parametre1, ...)

Pour Java 2 méthodes sont différentes si :

- leurs noms sont différents
- leurs noms sont identiques mais le nombre de paramètres ou le type des paramètres sont différents.

On peut donc avoir plusieurs méthodes avec le même nom du moment que leurs paramètres sont différents. C'est ce que l'on appelle la **surcharge**.

Pour Java, deux méthodes de même nom ont même type de retour. Cela évite les ambiguïtés avec des méthodes n'étant différentes que par leur type de sortie et non distinguables par le compilateur.

Providing Constructors for Your Classes

https://docs.oracle.com/javase/tutorial/java/javaOO/constructors.html

On a coutume de présenter (grâce à la surcharge) plusieurs constructeurs. Celui ayant le plus de paramètres étant appelé par tous les autres (avec le mot clé this) :

```
// package et imports

class MaClasse {
  private type1 attribut;

  public type1 getAttribut();
  public void  setAttribut(type1 attribut);
```

```
  public MaClasse (type1 attribut) {
      this.attribut = attribut;
  }

  public MaClasse() {
      this(objetDeType1); // on suppose que l'on a un objet de type1
  }

  // méthodes.

}
```

Passing Information to a Method or a Constructor

https://docs.oracle.com/javase/tutorial/java/javaOO/arguments.html

Object

La partie https://docs.oracle.com/javase/tutorial/java/javaOO/objectcreation.html est cruciale pour comprendre comment fonctionne un programme Java.

Creating Objects

On n'utilise que des **objets** : c'est eux la réalité. Les classes ne servent qu'à créer des objets.

https://docs.oracle.com/javase/tutorial/java/javaOO/objectcreation.html

On utilisera **toujours** le mot clé new pour créer des objets. Ne confondez pas variables et objets :

```
Point originOne; //déclare une variable pouvant nommer un objet
origineOne = new Point(); // on crée un Point que l'on nomme par origineOne.

Point aPoint = origineOne; // aPoint est un autre nom pour l'objet également
nommé origineOne.
```

Using Objects

https://docs.oracle.com/javase/tutorial/java/javaOO/usingobject.html

Attention à l'égalité entre objets. L'opérateur == compare si ce sont les mêmes objets (`is` en python), pas si leur contenu est identique ainsi

```
new String("coucou") == "coucou"
```

est faux. Ce sont deux objets différents. Pour vérifier que leur contenu est identique on

utilise la méthode `toString` :

```
(new String("coucou")).equals("coucou");
```

est ainsi vrai.

En python c'est le contraire :

- is pour savoir si deux objets sont les mêmes,
- == pour savoir si le contenu de deux objets est identique

Ne vous laissez pas abuser : `"coucou" == "coucou"` va répondre `true`, mais c'est parce que le compilateur Java est malin. Il associe aux deux éléments le même objet. Les String étant non mutables (on ne peut les modifier) cela ne porte pas à conséquence et accélère le code.

More on Classes

Using the this Keyword

https://docs.oracle.com/javase/tutorial/java/javaOO/thiskey.html

S'adresse à l'objet qui utilise la méthode. Le code ci-après se sert de `this` trois fois :

- **dans le premier constructeur**. L'objet courant est ici celui qui est construit. On affecte à son attribut x (`this.x`) l'entier x passé en paramètre.
- **dans le constructeur sans paramètre**. On appelle le constructeur avec un paramètre.
- **dans la méthode**. Elle rend un tableau contenant l'objet.

```
class Exemple {
    int x;

    Exemple(int x) {
        this.x = x;
    }

    Exemple() {
        this(42);
    }

    Exemple[] encapsule() {
        Exemple[] tableau = new Exemple[1];
        tableau[0] = this;
        return tableau;
    }
```

```
}
```

Controlling Access to Members of a Class

https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

Permet de contrôler l'usage de méthodes et attributs de classes par le *monde extérieur*, c'est-à-dire les autres classes du programme.

En Java, on peut contrôler cette visibilité de 4 façons (qui ont leur pendant en UML) :

- `public` : tout le monde peut voir et utiliser la méthode/attribut.
- `protected` : la classe et ses descendants peuvent voir et utiliser la méthode/attribut.
- `private` : uniquement la classe peut voir et utiliser la méthode/attribut.
- aucun modificateur : les classes du package peuvent voir et utiliser la méthode/attribut.

Si l'on ne veut pas s'embêter, une bonne règle est :

- de laisser les attributs sans modificateur (ou à la rigueur en `private`, mais cela risque de poser des problèmes pour les tests)
- de placer les méthodes/constructeurs en `public`.

Understanding Class Members

https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html

Les méthodes et attributs de classes sont caractérisés par le mot clé `static`. On s'en sert essentiellement de trois façons :

- **pour la classe principale**. Dans la méthode main bien sûr mais aussi pour les différentes parties du programme principal.
- **pour les classes sans objet ou les constantes**. La classe java.lang.Math est un bon exemple avec ses méthodes qui sont toutes statiques et les définitions de constantes comme `PI` et `E`.
- **pour construire des objets**. En utilisant le pattern Factory pour créer des objets.

Nested Classes

https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

Vous pouvez sauter cette partie en première lecture. Cela permet des techniques de programmation très utiles mais ce n'est pas pour des débutants.

Enum Type

https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html

Idéal pour traiter avec des constantes comme les jours de la semaine, les couleurs d'un jeu de carte, etc.

Annotations

https://docs.oracle.com/javase/tutorial/java/annotations/index.html

L'IDE les met souvent tout seul comme `@Override` lorsque l'on récrit une méthode d'un ancêtre. Sachez que ça existe, mais nous ne l'utiliserons pas plus que ça.

Un tutorial sympa (et en français) : https://fr.wikibooks.org/wiki/Programmation_Java/Annotations

Lesson: Interfaces and Inheritance

https://docs.oracle.com/javase/tutorial/java/IandI/index.html

Grouper les objets par :

- ce qu'ils **sont** : héritage
- ce qu'ils **font** : interfaces

Interfaces

https://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html

Permet de définir des objets par ce qu'ils **FONT**.

Les variables peuvent être définies comme pouvant contenir des objets implémentant une interface particulière. Les objets doivent tout de même être créés de façon normale (un new suivi d'un de ses constructeur) mais ils peuvent être rangés et véhiculés par des variables respectant des fonctionnalités particulières.

Héritage

https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

Le mot clé super permet d'accéder aux constructeurs et méthodes des ancêtres.

Object as a Superclass

https://docs.oracle.com/javase/tutorial/java/IandI/objectclass.html

Les trois méthodes ici qu'il faudra a priori toujours redéfinir sont :

- `String toString()` qui convertit un objet en chaîne de caractères
- `boolean equals(Object o)` qui vérifie que le contenu de 2 objets coïncide

- `int hashcode()` qui convertit un objet en nombre

Number and Strings

https://docs.oracle.com/javase/tutorial/java/data/index.html

Number

https://docs.oracle.com/javase/tutorial/java/data/numbers.html

Les classes et objets associés aux nombres.

int n'est pas une classe alors qu'**Integer** en est une.

String

https://docs.oracle.com/javase/tutorial/java/data/strings.html

Les chaînes de caractères se comportent comme en python.

Generics

https://docs.oracle.com/javase/tutorial/java/generics/index.html

Peut être extrêmement pénible et compliqué. Nous serons forcés de l'utiliser (et cela peut être utile), mais il faut toujours un peu penser et tester si cela fonctionne ou pas.

Les generics ne fonctionnent qu'avec des classes. Ainsi pour faire une `ArrayList` d'entiers il faut utiliser : `ArrayList<Integer>` et non pas `ArrayList<int>`

Un tutorial sur les types génériques en java qui montre un peu tous les cas d'utilisations.

Packages

https://docs.oracle.com/javase/tutorial/java/package/index.html