Objet, référence, égalité, nullabilité, mutabilité

Exercice 1 - Livre

On cherche à écrire une classe `Book` représentant un livre avec un titre et le nom de l'auteur.

1. Déclarer une classe `Book` contenant les champs privés `title` et `author`.
2. Puis essayer le code suivant dans une méthode `main` de la classe `Book`.

```
var book = new Book();
System.out.println(book.title + ' ' + book.author);
```

   Expliquer.
3. Créer une classe `Main` (dans un fichier `Main.java`) et déplacer le `main` de la classe `Book` dans la classe `Main`.
   Quel est le problème ? Comment peut-on le corriger ?
4. Rappeler les 4 visibilités possibles en Java.
   Pourquoi doit-on toujours déclarer les champs privés sous peine de mort certaine ?
5. Qu'est ce qu'un accesseur ?
   Sachant qu'il n'y a aucune raison de changer les champs d'un livre une fois celui-ci créé (au moins pour l'instant), quels sont les accesseurs que l'on doit mettre ici ?
6. Comment indiquer à un futur développeur que la valeur du champ `title` (ou `author`) ne doit pas être modifiée ?
   Pourquoi est-ce important ?
7. Ajouter un constructeur initialisant (un constructeur initialise, il ne construit pas vraiment) le livre avec un titre (`ptitle`) et un auteur (`pauthor`).
   Pourquoi le code du `main` ne fonctionne plus ? Changer celui-ci.
8. Modifier le constructeur précédent pour que les deux paramètres s'appellent `title` et `author`. Quel est le problème ? Quelle est la solution ?
9. Écrire un autre constructeur qui prend juste un titre et pas d'auteur.
   On initialisera le champ `author` avec "<no author>" dans ce cas.
10. Comment le compilateur fait-il pour savoir quel constructeur appeler ?
11. Comment faire maintenant pour que le second constructeur appelle le premier ?

Exercice 2 - Liberté, Égalité, Fraternité

1.
```
var b1 = new Book("Da Java Code", "Duke Brown");
var b2 = b1;
var b3 = new Book("Da Java Code", "Duke Brown");

System.out.println(b1 == b2);
System.out.println(b1 == b3);
```

   Qu'affiche le code ci-dessus ?
   Pourquoi ?
2. Écrire dans la classe `Book` une méthode (à vous de trouver le nom et la signature exacte de la méthode) qui renvoie `true` si deux livres ont les mêmes nom et description.
   Attention à la comparaison de chaînes de caractères.
3. La classe `java.util.ArrayList` correspond à un tableau qui s'agrandit dynamiquement.
   A quoi sert la méthode `indexOf` de `ArrayList` (RTFM) ?
4. Exécutez le code suivant :

```
public static void main(String[] args){
  var b1 = new Book("Da Java Code", "Duke Brown");
```

```
    var b2 = b1;
    var b3 = new Book("Da Java Code", "Duke Brown");

    var list = new ArrayList();
    list.add(b1);
    System.out.println(list.indexOf(b2));
    System.out.println(list.indexOf(b3));
  }
```

Quel est le problème avec les résultats affichés sur la console.
Note : ici, le compilateur génère un *warning* au niveau du `add`. Nous verrons dans les prochains TD comment l'éviter.

5. Quelle méthode de `Book` est appelée par `ArrayList.indexOf` (RTFM again) ?
6. Modifier la classe `Book` pour que `indexOf()` fait sur l'`ArrayList` teste si les deux livres ont les mêmes caractéristiques.
7. Utiliser l'annotation `@Override` (`java.lang.Override`) sur la méthode ajoutée à `Book`.
8. A quoi sert l'annotation `@Override` ?
9. Qu'affiche le code ci-dessous ?

```
    var aBook = new Book(null, null);
    var anotherBook = new Book(null, null);
    var list = new ArrayList();
    list.add(aBook);
    System.out.println(list.indexOf(anotherBook));
```

Où se situe le problème ?
Rappeler pourquoi un code doit arrêter de fonctionner si celui-ci est mal utilisé par un développeur.
Que doit-on faire pour corriger le problème ?

10. Rappeler quelle est la règle de bonne pratique concernant l'utilisation de `null`.
11. A quoi sert la méthode `java.util.Objects.requireNonNull` (RTFM) ?
    Comment l'utiliser ici pour empêcher de construire un livre avec des champs `null` ?

Exercice 3 - Comment afficher un livre ?

On aimerait pouvoir afficher les caractéristiques d'un livre, par le code Java suivant :

```
var book = new Book("Da Java Code", "Dan Duke");
System.out.println(book);
```

Java sait faire cela, à condition de mettre dans la classe `Book` une méthode `public String toString()` (la définition de cette méthode est dans la classe `java.lang.Object` (RTFM !)) retournant une chaîne de caractères, qu'on construit typiquement à partir des attributs de l'objet.
Rappel: en Java on peut faire un '+' entre une `String` et n'importe quoi, le résultat est la concaténation entre la `String` et le n'importe quoi vu comme une chaîne de caractère.

1. Écrire cette méthode, pour obtenir par exemple l'affichage suivant :

```
Da Java Code by Dan Duke
```

2. Peut-on utiliser l'annotation @Override, ici ?
3. Plus difficile, on souhaite maintenant afficher uniquement le titre du livre si le livre est construit avec le constructeur avec un seul argument. Attention, l'affichage de `new Book("", "<no author>")` devra bien afficher "<no author>".
   Si vous entrevoyez une solution qui utilise `null`, pensez plus fort :)

Exercice 4 - Tri à caillou [à la maison]

1. Écrire une méthode `swap` qui échange les valeurs de deux cases d'un tableau. `void swap(int[] array,int index1,int index2)`

2. Écrire une méthode `indexOfMin` qui renvoie l'indice de la valeur minimale d'un tableau.
3. Modifier la méthode `indexOfMin` en ajoutant deux indices indiquant que l'on cherche l'indice du minimum non pas sur tout le tableau mais sur la partie de tableau entre ces deux indices (le premier inclus, le deuxième exclu).
4. Écrire la méthode `sort` qui prend un tableau d'entier en paramètre et qui trie celui-ci en utilisant pour cela les méthodes `indexOfMin` et `swap`.
