

Java DUT 1 Feuille TD4
Université Paris-Est Marne-la-Vallée

Exercice 1.—

On définit une classe `Book` pour représenter des livres. Chaque livre possède un titre (`title`) et un auteur (`author`) qui seront des chaînes de caractères, c'est-à-dire des objets `String`.

- a) Écrire un constructeur pour la classe `Book`. Les champs seront `private`.
- b) Écrire une classe `BookTest` avec une méthode `main` pour tester le constructeur et les méthodes de `Book`. Testez le constructeur.
- c) On considère que les champs d'un `Book` ne seront jamais changés une fois celui-ci créé. Quels sont les accesseurs que l'on doit mettre ? Écrivez les. Que doit-on ajouter aussi ?
- d) Écrire un autre constructeur qui prend juste un `title` et pas d'`author`. On initialisera le champ `author` avec "no author" dans ce cas. Comment le compilateur fait-il pour savoir quel constructeur appeler ?
- e) Comment faire maintenant pour que le second constructeur appelle le premier ? On utilisera l'appel `this()` placé en tête du constructeur de la façon suivante.

```
public Book(String title) {  
    this(title, "no author");  
}
```

- f) On stocke des `Book` dans un tableau qui s'agrandit automatiquement en utilisant la classe `java.util.ArrayList`. Regardez la documentation de la méthode `indexOf` dans cette classe et exécutez le code suivant :

```
public static void main(String[] args){  
    Book b1 = new Book("Les fruits du Congo", "Alexandre Vialatte");  
    Book b2 = new Book("Les fruits du Congo");  
    Book b3 = new Book("Les fruits du Congo", "Alexandre Vialatte");  
    Book b4 = new Book("Lolita", "Vladimir Nabokov");  
    ArrayList<Book> list = new ArrayList<Book>();  
    list.add(b1);  
    list.add(b4);  
    System.out.println(list.indexOf(b2));  
    System.out.println(list.indexOf(b3));  
}
```

- g) Quelle méthode de `Book` est appelée par `indexOf` ? Modifier la classe `Book` pour que `indexOf` fonctionne correctement. Pour cela écrivez dans la classe `Book` une méthode `public boolean equals(Object other)` renvoyant `true` si deux `Book` ont les mêmes champs et `false` sinon. Pourquoi ajoutez-t-on le tag `@Override` ? On complètera le code ci-dessous.

```
@Override  
public boolean equals (Object other){  
    if (other==null) return false;  
    Book b = (Book)other ;  
    ...  
}
```

h) Qu'affiche le code ci-dessous ?

```
public static void main(String[] args){
    Book b = new Book("Lolita", "Nabokov");
    Book c = new Book(null, null);
    ArrayList<Book> list = new ArrayList<Book>();
    list.add(b);
    System.out.println(list.indexOf(c));
}
```

Où se situe le problème ?

i) Corrigez le problème en empêchant la construction d'un Book avec des champs null. On utilisera la méthode `java.util.Objects.requireNonNull` de la façon suivante :

```
public Book(String title, String author) {
    this.title = Objects.requireNonNull(title);
    this.author = Objects.requireNonNull(author);
}
```