TP3 JAVA

Exercice 1

1) Déclarer un record Book avec les composants title et author.

```
public record Book(String title, String author) {}
```

2) Puis essayer le code suivant dans une méthode main du record Book.

-le code crée un nouvel objet livre, puis affiche le titre et l'auteur du livre, avec un espace entre les deux

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 ?

-On ne peut plus appeler les champs de nos objet, il faut passer par les setters. book.title -> book.title()

4) On peut remarquer que le code permet de créer des livres ayant un titre ou un auteur null. Comment faire pour éviter ce problème sachant qu'il existe une méthode static requireNonNull dans la classe java.util.Objects.

-cette ligne dans le constructeur permet de lever une erreur si le titre est null

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

5) En fait, on peut simplifier le code que vous avez écrit à la question précédente en utilisant un constructeur compact (compact constructor). Commenter le code précédent et utiliser un constructor compact à la place.

```
    public Book{
        Objects.requireNonNull(title, "title is null");
        Objects.requireNonNull(author, "author is null");
    }
```

6) Écrire un autre constructeur qui prend juste un titre et pas d'auteur et ajouter un code de test dans le main. On initialisera le champ author avec "" dans ce cas

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

7) Comment le compilateur fait-il pour savoir quel constructeur appeler ?

-Le compilateur choisi quel constructeur appeler en fonction du nombre d'agruments.

8) On souhaite maintenant pouvoir changer le titre d'un livre déjà existant en utilisant une méthode nommée withTitle qui prend en paramètre le nouveau titre.

Pourquoi le code suivant ne marche pas ? Écrire le code correspondant et ajouter un code de test dans le main.

-Le code suivant ne marche pas car les objets sont imuables. Il faut créer un nouvel objet.

```
    public Book withTitle(String title) {
        return new Book(title, this.author);
    }
```

Exercice 2

1) Qu'affiche le code ci-dessous ?

-Le code affiche true false parce que == compare l'adresse des variables, b2 est copié sur l'adresse de b1. b3 est un nouvel objet

2) Comment faire pour tester si deux objets ont le même contenu ?

-avec la methode .equals() Écrire le code qui affiche si b1 et b2, puis b1 et b3 ont le même contenu.

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

3) Écrire une méthode isFromTheSameAuthor() qui renvoie vrai si deux livres sont du même auteur.

```
    public boolean isFromTheSameAuthor(Book book1, Book book2){
        return book1.author.equals(book2.author);
    }
```

4) Comment faire pour que le code suivant affiche "Da Java Code by Duke Brown".

-Il faut modifier la methode toString() dans le record Book, lorsque l'on voudra afficher un objet book il sera passé en string, et grâce à la modification de la methode toString, "by" sera ajouté.

5) Utiliser l'annotation @Override (java.lang.Override) sur la méthode ajoutée à Book.

```
    @Override
    public String toString(){
         return this.title + " by " + this.author;
    }
```

6) A quoi sert l'annotation @Override ?

-L'oppération @Override sert à réécrire une méthode existante.

Exercice 3

1) Quel est le problème ?

-le problème est que "equals()" n'est pas défini dans les classes et compare les adresses des variables.

2) Comment corriger le problème si on s'entête à utiliser une classe ?

```
    @Override
    public boolean equals(Object object) {
        Book2 book2 = (Book2) object;
        return title == book2.title && author == book2.author;
      }
```

Exercice 4

1) Écrire une méthode swap qui échange les valeurs de deux cases d'un tableau :

-la methode swap est défini dans la classe "Ex4"

```
    public static void swap(int[] array, int index1, int index2){
        int a = array[index1];
        int b = array[index2];
        array[index1]=b;
        array[index2]=a;
    }
```

```
        int[] t = {1,2,3};
        Ex4.swap(t, 1, 2);
        System.out.println(t[0]);
        System.out.println(t[1]);
        System.out.println(t[2]);
```

2) Écrire une méthode indexOfMin qui renvoie l'indice de la valeur minimale d'un tableau.

```
    public static int indexOfMin(int[] array){
        int min=array[0];
        for(int i: array){
            if(i<min){
                min=i;
            }
        }
        return min;
    }
```

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).

```
    public static int indexOfMin2(int[] array, int index1, int index2){
        int min=array[index1];
        int i;
        for(i=index1; i<index2; i++){
            if(array[i]<min){
```

```
                min=i;
            }
        }
        return min;
    }
```

4) Écrire la méthode sort qui prend un tableau d'entiers en paramètre et qui trie celui-ci en utilisant pour cela les méthodes indexOfMin et swap.

```
    public static void sort(int[] array, int index1, int index2){
        if(index2<index1){
            return;
        }
        swap(array, index1, indexOfMin2(array, index1, index2));
        sort(array, index1+1, index2);
    }

    public static void sortW(int[] array){
        int index1=0;
        int index2= array.length;
        sort(array, index1, index2);
    }
```