TP4

Exercice 1

1) Que fait sysout + Ctrl + Space dans un main ? -"sysout" + Ctrl + Space dans un main permet de générer automatiquement "System.out.println() 2) Que fait toStr + Ctrl + Space dans une classe ? -"toStr" + Ctrl + Space dans une classe permet de générer automatiquement la méthode "toString()" 3) Définir un champs foo de type int, que fait get + Ctrl + Space, et set + Ctrl + Space .Dans le menu Source, comment générer un constructeur initialisant le champ foo ? -"get" + Ctrl + Space permet de générer automatiquement un getter pour le champ "foo", tandis que "set" + Ctrl + Space permet de générer automatiquement un setter 4) Dans le menu Source, comment générer un constructeur initialisant le champ foo ? -Pour générer un constructeur initialisant le champ "foo", il faut cliquer sur "Source" dans la barre de menu, puis de sélectionner "Generate Constructor using Fields" 5) Sélectionner le nom de la classe puis Alt + Shift + R, qu'obtient-on ? Même question avec le champ foo . -En appuyant sur Alt + Shift + R, on peut renommer la classe. en sélectionnant le champ "foo" puis en appuyant sur Alt + Shift + R, on peut renommer le champ. 6) Écrire a = 2 + 3 + 4, puis sélectionner 2 + 3 puis Alt + Shift + L . -Extract local variable 7) Écrire new Integer(2), en gardant le curseur après ')', appuyer sur Ctrl + 1, que se passe-t-il ? -"2" est écrit 8) Déclarer une variable s de type String et cliquer sur String en maintenant la touche Ctrl . Que se passe-t-il ? -On retourne sur la classe String 9) Dans la méthode toString(), que fait un Ctrl + Clic sur super.toString() ? -on accède directement à la méthode toString() de la superclasse de notre objet. 10) Sélectionner le champs foo, puis Ctrl + Shift + G. Que se passe-t-il ? -affiche toutes les occurrences du champ foo dans le code 11) À quoi sert Ctrl + Shift + O ? -Ctrl + Shift + O permet d'organiser les imports dans le code, en ajoutant ou en supprimant les importations nécessaires pour les classes utilisées. 12) À quoi sert Ctrl + Shift + C ? -Ctrl + Shift + C permet de commenter ou décommenter une ligne

Exercice 2

1) Écrire une classe Library avec un champs books de type ArrayList ainsi qu'un constructeur sans paramètre initialisant le champ books.

```
public class Library {
    private final ArrayList<Book> books;

    public Library(){
        books = new ArrayList<>();
    }
}
```

2) Ajouter une méthode add qui permet d'ajouter des books (non null) à la liste de livres.

```
    public void add(Book book){
        Objects.requireNonNull(book);
        books.add(book);
    }
```

3) Écrire une méthode findByTitle qui permet de trouver un livre en fonction de son titre dans la bibliothèque. La méthode doit renvoyer null dans le cas où aucun livre n'a le bon titre.

```
    public Book findByTitle(String title){
        for(Book book : books){
            if(book.title().equals(title)){
                return book;
            }
        }
        return null;
    }
```

4) Comment le compilateur compile-t-il une boucle foreach sur une collection ? -le compilateur compile une boucle foreach en utilisant java/util/Iterator

5) Expliquer pourquoi la méthode findByTitle doit renvoier null plutôt que de lever une exception. -Car c'est une methode plus propre pour gérer les erreurs.

6) Écrire une méthode toString permettant d'afficher les livres de la bibliothèque dans l'ordre d'insertion, un livre par ligne.

```
    @Override
    public String toString() {
        var builder = new StringBuilder();
        var separator="";
        for(Book i : books){
            builder.append(i);
            builder.append(separator);
            separator="\n";
        }
        return builder.toString();
    }
```

Exercice 3

1) Quelle est la complexité de la méthode findByTitle de la classe Library ? -la complexité de la méthode findByTitle est de O(n)

2) Quelle est la structure de données algorithmique dont java.util.HashMap est une implantation ? -java.util.HashMap est une implémentation des tables de hachage Sachant que l'on veut améliorer la performance de findByTitle comment peut on utiliser la classe java.util.HashMap pour cela ? -On peut remplacer l'arrayList de notre librairie par une hashMap, chaque livre sera associé à une clée dans la hashmap. Quelle sera alors la complexité de findByTitle ? -La complexité sera de O(1)

3) Commenter entièrement le code de la classe Library (pour ne pas perdre votre travail) et recopier les signatures des méthodes commentées. Pour l'instant, laisser la méthode toString de côté Modifier les champs afin d'utiliser une java.util.HashMap et implanter les méthodes le constructeur et les méthodes add et findByTitle.

```
    private Map<String, Book> books = new HashMap<>();
```

```
    public Library(){
        books = new HashMap<>();
    }

    public void add(String key, Book book){
        books.put(key, book);
    }

    public Book findByTitle(String title){
        return books.get(title);
    }
```

4) Expliquer pourquoi, ici, on a préféré utiliser une classe pour représenter Libary plutôt qu'un record. -Car notre map library va être modifié au fils du temps, dans un record, les éléments sont immuables.

5) Pour l'implantation de la méthode toString, quelle méthode de java.util.HashMap doit-on utiliser pour obtenir l'ensemble des valeurs stockées ? Si vous ne savez pas, lisez la javadoc ! -On va utiliser map.values() pour obtenir l'ensemble des valeurs stockées.

```
    public String toString(){
        var builder = new StringBuilder();
        var separator="";
        for(Book i : books.values()){
            builder.append(i);
            builder.append(separator);
            separator="\n";
        }
        return builder.toString();
    }
```

6) En fait, la méthode toString ne fait pas exactement ce qui est demandé, car elle ne permet pas d'afficher les éléments dans l'ordre d'insertion. Sachant qu'il existe une classe LinkedHashMap, comment peut-on résoudre ce problème ? -on va remplacer notre map par une LinkedHashMap, la LinkedHashMap conserve l'ordre d'insertion des élémetens.

```
    private Map<String, Book> books = new LinkedHashMap<>();

    public Library(){
        books = new LinkedHashMap<>();
    }
```

7) Pourquoi votre implantation lève-t-elle une exception dans l'exemple suivant ? -mon implémentation lève une exception car je supprime un élément de la map en même temps que je la parcours

```
    public void removeAllBooksFromAuthor(String nomAuthor){
        for(Book book : books.values()){
            if(book.author().equals(nomAuthor)){
                books.remove(book.title());
            }
```

```
        }
    }
```

8) En fait, il existe une méthode remove sur l'interface Iterator qui n'a pas ce problème, car le parcours et la suppression se font sur le même itérateur. Implanter correctement la méthode removeAllBooksFromAuthor.

```
    public void removeAllBooksFromAuthor2(String nomAuthor) {
        Iterator<Book> iterator = books.values().iterator();
        while (iterator.hasNext()) {
            Book book = iterator.next();
            if (book.author().equals(nomAuthor)) {
                iterator.remove();
            }
        }
    }
```

9) De façon optionnelle, si vous êtes balèze, il existe une méthode removeIf sur Collection qui permet d'écrire la méthode removeAllBooksFromAuthor en une ligne ! -On va parcourir les values de notre map books, et appliquer removeIf sur les livres dont l'auteur est égal au paramètre de notre fonction.

```
    public void removeAllBooksFromAuthor3(String nomAuthor) {
        books.values().removeIf(book -> book.author().equals(nomAuthor));
    }
```