Daouda KONE

COMPTE RENDU DU TP4

Exercice 1

Question 1

Question 2

kddaouda@Daouda:~/Esiee/Java/tp4$ java --version openjdk 23.0.2 2025-01-21 OpenJDK Runtime Environment (build 23.0.2+7-Ubuntu-1ubuntu124.04) OpenJDK 64-Bit Server VM (build 23.0.2+7-Ubuntu-1ubuntu124.04, mixed mode, sharing) kddaouda@Daouda:~/Esiee/Java/tp4$

Question 3

public class Main{ public static void main(String [] args){ System.out.println("Hello Eclipse"); } }

Question 4

Exercice2

Question 1

```java public class Library { private final ArrayList books;

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

} ```

## Question 2

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

Question 3

```
java public Book findByTitle (String title) { Objects.requireNonNull(title); for (var book: books) { if(book.title() == title) return book; } return null ; }
```

Question 4

Le compilateuru utilise la mÃ©thode itÃ©rative pour faire la collection (**// Method java/util/ArrayList.iterator:()Ljava/util/Iterator;**). Et Ã une meÃ©thode **hasNext()** (// InterfaceMethod java/util/Iterator.hasNext:()Z).

Question 5

On renvoie **null** pour permet au code de tourner. Si on fait une exception notre code s'arrÃªtera et il se peut que nopus voulions continuer notre recherche.

Question 6

```java @Override public String toString() { var str = new StringBuilder(); str.append("Library: \n"); for(var book : books) { str.append(book).append("\n"); } return str.toString();

```
}
```

```

# Exercice 3

## Question 1
```

La complexitÃ© de findByTitle est : O(n)

Question 2

ImplÃ©mentation de l' interface Map basÃ©e sur une table de hachage . Cette implÃ©mentation fournit toutes les opÃ©rations de mappage facultatives et autorise les valeurs nulles et la clÃ© nulle .

Pour amÃ©liorer le code de **findByTitle** en utilisasant la classe **java.util.HashMap** on peut la recherche par clÃ© dans les livres. Dans ce cas on aura une complexitÃ© de O(1).

Question 3

import java.util.HashMap; import java.util.Objects;

public class Library { // Remplacement de l'ArrayList par une HashMap pour optimiser la recherche par titre private final HashMap books;

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


public void addbook(Book book) {
    Objects.requireNonNull(book);

    if (books.containsKey(book.title())) {
        throw new IllegalArgumentException("A book with this title already exists : " + book.title());
    }

    books.put(book.title(), book);
}



public Book findByTitle(String title) {
    Objects.requireNonNull(title);
    return books.get(title);  // Recherche instantanÃ©e en O(1) en moyenne
}

// Ancienne implÃ©mentation basÃ©e sur une ArrayList (commentÃ©e pour archivage)
/*
private final ArrayList<Book> books;

public Library() {
    this.books = new ArrayList<Book>();
}

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

public Book findByTitle(String title) {
    Objects.requireNonNull(title);
    for (var book : books) {
        if (book.title().equals(title))
            return book;
    }
    return null;
}
*/

public static void main(String[] args) {
    Library library = new Library();
    Book book = new Book("Da Vinci Code", "Dan Brown");
    library.addbook(book);
    System.out.println(library.findByTitle("Da Vinci Code"));
}

}
```

Question 4

Une classe est plus adaptÃ©e car Library est un objet mutable, contenant une structure Ã©volutive et de la logique mÃ©tier. Un record conviendrait uniquement si Library Ã©tait immuable et ne faisait que stocker des donnÃ©es.

En rÃ©sumÃ©, Library est un ensemble dynamique de livres, donc une classe est le meilleur choix !

Question 5

On va utiliser la mÃ©thode **map.value()** qui retourne l'ensemble des livres stockÃ©s **values()** retourne une collection contenant toutes les valeurs de la HashMap. Dans notre cas, elle renverra une collection de Book.

@Override public String toString() { var str = new StringBuilder("Library:\n"); for (Book book : books.values()) { // books.values() retourne la collection des livres str.append(book).append("\n"); // Utilisation du toString() de Book } return str.toString(); }

public static void main(String[] args) { Library library = new Library(); library.addbook(new Book("Da Vinci Code", "Dan Brown")); library.addbook(new Book("1984", "George Orwell")); library.addbook(new Book("Harry Potter", "J.K. Rowling"));

```
    System.out.println(library);
}
```

Remarque: on voit que l'orde d'intersertion des livres ne sont pas respectÃ© pendant l'affichage

kddaouda@Daouda:~/Esiee/Java/tp4$ java Library Library: 1984 by George Orwell Da Vinci Code by Dan Brown Harry Potter by J.K. Rowling

Question 6

On fait **import java.util.LinkedHashMap;** dans le notre fichier Library.java et on change tous les **HashMap** par **LinkedHashMap** dans le fichier.

Resulat:

kddaouda@Daouda:~/Esiee/Java/tp4$ java Library Library: Da Vinci Code by Dan Brown 1984 by George Orwell Harry Potter by J.K. Rowling

Question 7

public void removeAllBooksFromAuthor(String author) { Objects.requireNonNull(author);

```
    for (Book book : books.values()) {
        if (book.author().equals(author)) {
            books.remove(book.title()); // Suppression directe dans for-each
        }
    }
}
```

ProblÃ¨me avec LinkedHashMap.values()

books.values() retourne une vue dynamique de la LinkedHashMap (c'est-Ã -dire une collection liÃ©e Ã la HashMap elle-mÃªme).

Modifier books (via books.remove(title)) pendant qu'on itÃ¨re sur sa vue (dans le for-each) provoque une **ConcurrentModificationException**:

Exception in thread "main" java.util.ConcurrentModificationException

Explication technique :

books.values() dÃ©pend directement de books.

Quand on parcourt books.values() en for-each, on utilise un itÃ©rateur interne.

Lorsque books.remove(title) est appelÃ©, cela modifie la structure de books, ce qui invalide l'itÃ©rateur.

Java dÃ©tecte cette modification et lance l'exception pour Ã©viter des incohÃ©rences.

Question 8

J'ai fait ce code :

public void removeAllBooksFromAuthor(String author) { Objects.requireNonNull(author);

```
    // Utilisation d'un Iterator pour Ã©viter ConcurrentModificationException
    Iterator<Book> iterator = books.values().iterator();
    while (iterator.hasNext()) {
        Book book = iterator.next();
        if (book.author().equals(author)) {
            iterator.remove();  // Supprime l'entrÃ©e actuelle de la LinkedHashMap
        }
    }
}
```

et j'ai importÃ© : **import java.util.Iterator;**

Question 9

public void removeAllBooksFromAuthor(String author) { Objects.requireNonNull(author); books.values().removeIf(book -> book.author().equals(author)); }

Exercice 4

import java.util.Objects; import java.util.concurrent.ThreadLocalRandom;

public record Price(int copperTotal) { public static final int SILVER*PER*GOLD = 13; public static final int COPPER*PER*SILVER = 25;

```
public Price {
    if (copperTotal < 0) {
        throw new IllegalArgumentException("Invalid price.");
    }
}

// Constructeur avec or, argent et cuivre
public Price(int gold, int silver, int copper) {
    this(gold * SILVER_PER_GOLD * COPPER_PER_SILVER + silver * COPPER_PER_SILVER + copper);
}

// Constructeur avec or et argent uniquement (cuivre par dÃ©faut Ã  0)
public Price(int gold, int silver) {
    this(gold, silver, 0);
}

// RÃ©cupÃ¨re le nombre de piÃ¨ces d'or
private int gold() {
    return copperTotal / (SILVER_PER_GOLD * COPPER_PER_SILVER);
}

// RÃ©cupÃ¨re le nombre de piÃ¨ces d'argent
private int silver() {
    return (copperTotal % (SILVER_PER_GOLD * COPPER_PER_SILVER)) / COPPER_PER_SILVER;
}

// RÃ©cupÃ¨re le nombre de piÃ¨ces de cuivre restantes
private int copper() {
    return copperTotal % COPPER_PER_SILVER;
}

@Override
public String toString() {
    return gold() + "g, " + silver() + "s and " + copper() + "c";
}

public static Price randomBelow(Price cost) {
    Objects.requireNonNull(cost);
    var rng = ThreadLocalRandom.current();
    return new Price(rng.nextInt(cost.copperTotal));
}

public boolean isLowerThan(Price other) {
    Objects.requireNonNull(other);
    return this.copperTotal < other.copperTotal;
}

public Price substract(Price other) {
    Objects.requireNonNull(other);
    if (this.isLowerThan(other)) {
        throw new IllegalArgumentException("Not enough gold pieces.");
    }
    return new Price(this.copperTotal - other.copperTotal);
}

}
```

Daouda KONE