Commencé le	mardi 17 janvier 2023, 08:08
État	Terminé
Terminé le	mardi 17 janvier 2023, 09:50
Temps mis	1 heure 41 min
Points	51,63/52,00
Note	**19,86** sur 20,00 (**99,28**%)

Description

Vous allez développer une classe **Book** et les rudiments d'une classe **Library**.

Dans l'environnement vous avez :

import java.util.*;

import java.util.stream.*;

import static java.util.stream.Collectors.*;

A partir de la question sur le dictionnaire (Q6), vous avez dans votre environnement la classe **Book** que vous pouvez utiliser.

English version

You will develop a **Book** class and the rudiments of a **Library** class.

In the environment, you have :

import java.util.*;

import java.util.stream.*;

import static java.util.stream.Collectors.*;

You can import other classes if necessary.

From the dictionary question (Q6), you have in your environment the Book class that you can use.

Question 1

Correct

Note de 3,00 sur 3,00

👉 Implémenter la classe ***Book*** en respectant les spécifications suivantes :

- Un livre est défini par un titre, un auteur, une année de parution. On souhaite pouvoir accéder en lecture au titre, à l'auteur et à l'année : getTitle(), getAuthor(), getYear().
- On crée une instance de livre en utilisant un constructeur qui prend en paramètre dans cet ordre le titre, l'auteur et l'année. Si le titre est null, le titre prend pour valeur : "undefined". Si l'auteur est null, il prend pour valeur "unknown".

-- English version

👉 Implement the class ***Book*** respecting the following specifications:

- A book is defined by a title, an author, a year of publication. We want read access to the title, author and year: getTitle(), getAuthor(), getYear().
- We create a book instance using a constructor that takes the title, author and year as parameters in this order. If the title is null, the title takes the value: "undefined". If the author is null, it takes the value "unknown".

- Voici des exemples de tests :
- Here are some sample tests:

```
@Test
void testConstructorWithNullTitle() {
        Book book = new Book(null, "F. Scott Fitzgerald", 1925);
        assertEquals("undefined", book.getTitle());
        assertEquals("F. Scott Fitzgerald", book.getAuthor());
        assertEquals(1925, book.getYear());
        }

@Test
 void testConstructorWithNullAuthor() {
        Book book = new Book("The Great Gatsby", null, 1925);
        assertEquals("The Great Gatsby", book.getTitle());
        assertEquals("unknown", book.getAuthor());
        assertEquals(1925, book.getYear());
        }
```

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
public class Book  {
    private String title;
    private String author;
    private int year;

    public Book(String title, String author, int year) {
        this.title = title != null ? title : "undefined";
        this.author = author != null ? author : "unknown";
        this.year = year;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public int getYear() {
        return year;
    }
}
```

	Test	Résultat attendu	Résultat obtenu	

✔	`// void testConstructorWithNullTitle()`
	
	`Book book = new Book(null, "F. Scott Fitzgerald", 1925);`
	`assertEquals("undefined", book.getTitle());`
	`assertEquals("F. Scott Fitzgerald", book.getAuthor());`
	`assertEquals(1925, book.getYear());`
		`test : undefined equals undefined ?`
		`true`
		`test : F. Scott Fitzgerald equals F. Scott Fitzgerald ?`
		`true`
		`test : 1925 equals 1925 ?`
		`true`
			`test : undefined equals undefined ?`
			`true`
			`test : F. Scott Fitzgerald equals F. Scott Fitzgerald ?`
			`true`
			`test : 1925 equals 1925 ?`
			`true`
				✔

✔	`// void testConstructorWithNullAuthor()`
	
	`Book book = new Book("The Great Gatsby", null, 1925);`
	`assertEquals("The Great Gatsby", book.getTitle());`
	`assertEquals("unknown", book.getAuthor());`
	`assertEquals(1925, book.getYear());`
		`test : The Great Gatsby equals The Great Gatsby ?`
		`true`
		`test : unknown equals unknown ?`
		`true`
		`test : 1925 equals 1925 ?`
		`true`
			`test : The Great Gatsby equals The Great Gatsby ?`
			`true`
			`test : unknown equals unknown ?`
			`true`
			`test : 1925 equals 1925 ?`
			`true`
				✔

✔	`// void testConstructorWithNullTitleAndAuthor()`
	`Book book = new Book(null, null, 1925);`
	`assertEquals("undefined", book.getTitle());`
	`assertEquals("unknown", book.getAuthor());`
	`assertEquals(1925, book.getYear());`
		`test : undefined equals undefined ?`
		`true`
		`test : unknown equals unknown ?`
		`true`
		`test : 1925 equals 1925 ?`
		`true`
			`test : undefined equals undefined ?`
			`true`
			`test : unknown equals unknown ?`
			`true`
			`test : 1925 equals 1925 ?`
			`true`
				✔

Tous les tests ont été réussis ! ✔

▸ **Solution de l'auteur de la question (Java)**

Correct

Note pour cet envoi : 3,00/3,00.

Question 2

Correct

Note de 5,00 sur 5,00

👉 Soit l'interface **Identifiable**.

```
public interface Identifiable {
  String getId();
}
```

1. On souhaite qu'une instance de ***Book*** soit identifiable à partir de son ISBN par exemple 978-3-16-148410-0.
   - l'ISBN associé à un livre n'est pas modifiable
   - Par défaut le numéro d'identification quand il n'a pas été précisé à la construction a pour valeur "unknown".
   - Pensez à ajouter un nouveau constructeur dont le dernier paramètre est l'ISBN.
2. On souhaite identifier deux livres comme égaux si :
   1. ils ont le même ISBN; dans ce cas c'est les mêmes livres quelles que soient les autres informations;
   2. **ou** si les deux ISBN sont "unknown" **et** que leurs titres et leurs auteurs sont les mêmes.

Toutes les propriétés et donc les tests définis précédemment doivent continuer à fonctionner.

(Pour les étudiants plus avancés, attention Moodle ne supporte pas les pattern variables : if (!(o instanceof Book book)))

-- English version

👉 Let the ***Identifiable*** interface.

```
public interface Identifiable {
     String getId();
}
```

1. We want an instance of ***Book*** to be identifiable from its ISBN, for example, 978-3-16-148410-0.
   - Changing the ISBN number associated with a book should not be possible once it is registered.
   - By default, the identification number when it was not specified at construction has the value "unknown".
2. We want to identify two books as equal if:
   1. they have the same ISBN; in this case, it's the same regardless of other information;
   2. **or** if both are "unknown" and their titles and authors are the same.

All properties and, therefore, tests defined previously should continue to work.

Some tests -- Des tests vous sont donnés ci-dessous.

```
@Test
public void testConstructorWithFour() {
    Book book = new Book("The Great Gatsby", "F. Scott Fitzgerald", 1925, "978-3-16-148410-0");
    assertEquals("The Great Gatsby", book.getTitle());
    assertEquals("F. Scott Fitzgerald", book.getAuthor());
    assertEquals(1925, book.getYear());
    assertEquals("978-3-16-148410-0", book.getId());
}

@Test
public void testConstructorWithNullISBN() {
    Book book = new Book("The Great Gatsby", "F. Scott Fitzgerald", 1925, null);
    assertEquals("The Great Gatsby", book.getTitle());
    assertEquals("F. Scott Fitzgerald", book.getAuthor());
    assertEquals(1925, book.getYear());
    assertEquals("unknown", book.getId());
}

@Test
void testConstructorWithNullTitle() {
    Book book = new Book(null, "F. Scott Fitzgerald", 1925);
    assertEquals("undefined", book.getTitle());
    assertEquals("F. Scott Fitzgerald", book.getAuthor());
    assertEquals(1925, book.getYear());
}

@Test
void testConstructorWithNullAuthor() {
    Book book = new Book("The Great Gatsby", null, 1925);
    assertEquals("The Great Gatsby", book.getTitle());
    assertEquals("unknown", book.getAuthor());
    assertEquals(1925, book.getYear());
}

@Test
void testConstructorWithNullTitleAndAuthor() {
    Book book = new Book(null, null, 1925);
    assertEquals("undefined", book.getTitle());
    assertEquals("unknown", book.getAuthor());
    assertEquals(1925, book.getYear());
}

@Test
void testEqualsWithSameISBN() {
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    assertEquals(book1, book2);
    assertEquals(book1.hashCode(),book2.hashCode());
}

@Test
void testEqualsWithDifferentISBNs() {
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "0987654321");
    assertNotEquals(book1, book2);
    assertNotEquals(book1.hashCode(),book2.hashCode());
}

@Test
void testEqualsWithUnknownISBNs() {
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown");
    Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown");
    assertEquals(book1, book2);
    assertEquals(book1.hashCode(),book2.hashCode());
}

@Test
void testEqualsWithUnknownAndNonUnknownISBNs() {
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown");
    assertNotEquals(book1, book2);
    assertNotEquals(book1.hashCode(),book2.hashCode());
}

@Test
void testEqualsWithDifferentTitles() {
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown");
    Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "unknown");
    assertNotEquals(book1, book2);
    assertNotEquals(book1.hashCode(),book2.hashCode());
```

```
}

@Test
void testEqualsWithDifferentAuthors() {
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown");
    Book book2 = new Book("The Alchemist", "J.K. Rowling", 1998, "unknown");
    assertNotEquals(book1, book2);
    assertNotEquals(book1.hashCode(),book2.hashCode());
}

@Test
void testEqualsWithDifferentObjectTypes() {
    Book book = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown");
    String str = "not a book";
    assertNotEquals(book, str);
}
```

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```java
public class Book implements Identifiable {
    private String id;
    private String title;
    private String author;
    private int year;

    private final static String DEFAULT_TITLE = "undefined";
    private final static String DEFAULT_AUTHOR = "unknown";

    public Book(String title, String author, int year) {
        this.title = title != null ? title : DEFAULT_TITLE;
        this.author = author != null ? author : DEFAULT_AUTHOR;
        this.year = year;
        this.id = DEFAULT_AUTHOR;
    }

    public Book(String title, String author, int year, String id) {
        this(title, author, year);
        this.id = id != null ? id : DEFAULT_AUTHOR;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public int getYear() {
        return year;
    }

    @Override
    public String getId() {
        return id;
    }

    @Override
    public boolean equals(Object o) {
        if(this == o)
            return true;
        if(!(o instanceof Book))
            return false;

        Book book = (Book) o;

        if(this.id.equals(DEFAULT_AUTHOR) && book.id.equals(DEFAULT_AUTHOR) && this.title.equals(book.title) && this.au
            return true;

        return book.id.equals(this.id) && !this.id.equals(DEFAULT_AUTHOR);
```

	Test	Résultat attendu	Résultat obtenu	

✔	`// void testConstructorWithNullTitle()`
	
	`Book book = new Book(null, "F. Scott Fitzgerald", 1925);`
	`assertEquals("undefined", book.getTitle());`
	`assertEquals("F. Scott Fitzgerald", book.getAuthor());`
	`assertEquals(1925, book.getYear());`
		test : undefined equals undefined ?
		true
		test : F. Scott Fitzgerald equals F. Scott Fitzgerald ?
		true
		test : 1925 equals 1925 ?
		true
			test : undefined equals undefined ?
			true
			test : F. Scott Fitzgerald equals F. Scott Fitzgerald ?
			true
			test : 1925 equals 1925 ?
			true
				✔

✔	`// void testConstructorWithNullAuthor()`
	
	`Book book = new Book("The Great Gatsby", null, 1925);`
	`assertEquals("The Great Gatsby", book.getTitle());`
	`assertEquals("unknown", book.getAuthor());`
	`assertEquals(1925, book.getYear());`
		test : The Great Gatsby equals The Great Gatsby ?
		true
		test : unknown equals unknown ?
		true
		test : 1925 equals 1925 ?
		true
			test : The Great Gatsby equals The Great Gatsby ?
			true
			test : unknown equals unknown ?
			true
			test : 1925 equals 1925 ?
			true
				✔

✔	`// void testConstructorWithNullTitleAndAuthor()`
	`Book book = new Book(null, null, 1925);`
	`assertEquals("undefined", book.getTitle());`
	`assertEquals("unknown", book.getAuthor());`
	`assertEquals(1925, book.getYear());`
		test : undefined equals undefined ?
		true
		test : unknown equals unknown ?
		true
		test : 1925 equals 1925 ?
		true
			test : undefined equals undefined ?
			true
			test : unknown equals unknown ?
			true
			test : 1925 equals 1925 ?
			true
				✔

✔	`Book book = new Book("The Great Gatsby", "F. Scott Fitzgerald", 1925, "978-3-16-148410-0");`
	`assertEquals("The Great Gatsby", book.getTitle());`
	`assertEquals("F. Scott Fitzgerald", book.getAuthor());`
	`assertEquals(1925, book.getYear());`
	`assertEquals("978-3-16-148410-0", book.getId());`
		test : The Great Gatsby equals The Great Gatsby ?
		true
		test : F. Scott Fitzgerald equals F. Scott Fitzgerald ?
		true
		test : 1925 equals 1925 ?
		true
		test : 978-3-16-148410-0 equals 978-3-16-148410-0 ?
		true
			test : The Great Gatsby equals The Great Gatsby ?
			true
			test : F. Scott Fitzgerald equals F. Scott Fitzgerald ?
			true
			test : 1925 equals 1925 ?
			true
			test : 978-3-16-148410-0 equals 978-3-16-148410-0 ?
			true
				✔

✔	`Book book = new Book("The Great Gatsby", "F. Scott Fitzgerald", 1925, null);`
	`assertEquals("The Great Gatsby", book.getTitle());`
	`assertEquals("F. Scott Fitzgerald", book.getAuthor());`
	`assertEquals(1925, book.getYear());`
	`assertEquals("unknown", book.getId());`
		test : The Great Gatsby equals The Great Gatsby ?
		true
		test : F. Scott Fitzgerald equals F. Scott Fitzgerald ?
		true
		test : 1925 equals 1925 ?
		true
		test : unknown equals unknown ?
		true
			test : The Great Gatsby equals The Great Gatsby ?
			true
			test : F. Scott Fitzgerald equals F. Scott Fitzgerald ?
			true
			test : 1925 equals 1925 ?
			true
			test : unknown equals unknown ?
			true
				✔

✔	`//void testEqualsWithDifferentObjectTypes()`
	
	`Book book = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown");`
	`String str = "not a book";`
	`assertNotEquals(book, str);`
		test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='unknown'} not equals not a book ?
		false
			test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='unknown'} not equals not a book ?
			false
				✔

✔	`// void testEqualsWithDifferentTitles()`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown");`
	`Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "unknown");`
	`assertNotEquals(book1, book2);`
	
	`assertNotEquals(book1.hashCode(),book2.hashCode());`
		test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='unknown'} not equals Book{title='Veronika Decides to Die', author='Paulo Coelho', year=1998, isbn='unknown'} ?
		false
		test : 5375561 not equals -1156411847 ?
		false
			test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='unknown'} not equals Book{title='Veronika Decides to Die', author='Paulo Coelho', year=1998, isbn='unknown'} ?
			false
			test : 5375561 not equals -1156411847 ?
			false
				✔

✔	`// void testEqualsWithUnknownAndNonUnknownISBNs()`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown");`
	`assertNotEquals(book1, book2);`
	
	`assertNotEquals(book1.hashCode(),book2.hashCode());`
		test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} not equals Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='unknown'} ?
		false
		test : 494075994 not equals 5375561 ?
		false
			test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} not equals Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='unknown'} ?
			false
			test : 494075994 not equals 5375561 ?
			false
				✔

	Test	Résultat attendu	Résultat obtenu	

✔	`// void testEqualsWithDifferentAuthors() { Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown"); Book book2 = new Book("The Alchemist", "J.K. Rowling", 1998, "unknown"); assertNotEquals(book1, book2);`
	
	`assertNotEquals(book1.hashCode(),book2.hashCode());`
		test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='unknown'} not equals Book{title='The Alchemist', author='J.K. Rowling', year=1998, isbn='unknown'} ?
		false
		test : 5375561 not equals -1107497587 ?
		false
			test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='unknown'} not equals Book{title='The Alchemist', author='J.K. Rowling', year=1998, isbn='unknown'} ?
			false
			test : 5375561 not equals -1107497587 ?
			false
				✔

✔	`// void testEqualsWithUnknownISBNs()`
	
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown"); Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown"); assertEquals(book1, book2);`
	
	`assertEquals(book1.hashCode(),book2.hashCode());`
		test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='unknown'} equals Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='unknown'} ?
		true
		test : 5375561 equals 5375561 ?
		true
			test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='unknown'} equals Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='unknown'} ?
			true
			test : 5375561 equals 5375561 ?
			true
				✔

✔	`// void testEqualsWithDifferentISBNs() { Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890"); Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "0987654321"); assertNotEquals(book1, book2);`
	
	`assertNotEquals(book1.hashCode(),book2.hashCode());`
		test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} not equals Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='0987654321'} ?
		false
		test : 494075994 not equals -674133948 ?
		false
			test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} not equals Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='0987654321'} ?
			false
			test : 494075994 not equals -674133948 ?
			false
				✔

✔	`//void testEqualsWithSameISBN() { Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890"); Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890"); assertEquals(book1, book2);`
	
	`assertEquals(book1.hashCode(),book2.hashCode());`
		test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} equals Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} ?
		true
		test : 494075994 equals 494075994 ?
		true
			test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} equals Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} ?
			true
			test : 494075994 equals 494075994 ?
			true
				✔

Tous les tests ont été réussis ! ✔

▸ **Solution de l'auteur de la question (Java)**

Correct

Note pour cet envoi : 5,00/5,00.

Question 3

Correct

Note de 3,00 sur 3,00

👉 Soit l'interface **Matchable**

```
public interface Matchable {
    boolean match(String query);
}
```

On souhaite que un livre "matche" une chaîne de caractères si cette chaîne est contenue dans son title, son author, son year **ou** son isbn.

-- english version

👉 Let the **Matchable** interface

```
public interface Matchable {
       boolean match(String query);
}
```

We want a book to match a string if this string is contained in its title, author, year, or ISBN.

Here is an example of tests
Voici un exemple de tests

```
@Test
void testMatch() {
    Matchable book = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    assertTrue(book.match("Alchemist"));
    assertTrue(book.match("Coelho"));
    assertTrue(book.match("88"));
    assertTrue(book.match("890"));
    assertFalse(book.match("1984"));
    assertFalse(book.match("Jane Austen"));
}
```

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1  public class Book implements Identifiable, Matchable {
2      private String id;
3      private String title;
4      private String author;
5      private int year;
6
7      private final static String DEFAULT_TITLE = "undefined";
8      private final static String DEFAULT_AUTHOR = "unknown";
9
10     public Book(String title, String author, int year) {
11         this.title = title != null ? title : DEFAULT_TITLE;
12         this.author = author != null ? author : DEFAULT_AUTHOR;
13         this.year = year;
14         this.id = DEFAULT_AUTHOR;
15     }
16
17     public Book(String title, String author, int year, String id) {
18         this(title, author, year);
19         this.id = id != null ? id : DEFAULT_AUTHOR;
20     }
21
22     public String getTitle() {
23         return title;
24     }
25
26     public String getAuthor() {
27         return author;
28     }
29
30     public int getYear() {
31         return year;
32     }
33
34     @Override
35     public String getId() {
36         return id;
37     }
```

```
    }

    @Override
    public boolean equals(Object o) {
        if(this == o)
            return true;
        if(!(o instanceof Book))
            return false;

        Book book = (Book) o;

        if(this.id.equals(DEFAULT_AUTHOR) && book.id.equals(DEFAULT_AUTHOR) && this.title.equals(book.title) && this.au
            return true;

        return book.id.equals(this.id) && !this.id.equals(DEFAULT_AUTHOR);
```

	Test	Résultat attendu	Résultat obtenu	
✔	`Matchable book = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");` `assertTrue(book.match("Alchemist"));`	true	true	✔
✔	`Matchable book = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");` `assertTrue(book.match("Coelho"));`	true	true	✔
✔	`Matchable book = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");` `assertTrue(book.match("88"));`	true	true	✔
✔	`Matchable book = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");` `assertTrue(book.match("890"));`	true	true	✔
✔	`Matchable book = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");` `assertFalse(book.match("1984"));`	false	false	✔
✔	`Matchable book = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");` `assertFalse(book.match("Jane Austen"));`	false	false	✔

Tous les tests ont été réussis ! ✔

▸ **Solution de l'auteur de la question (Java)**

Correct

Note pour cet envoi : 3,00/3,00.

Question 4

Correct

Note de 3,00 sur 3,00

1. 👉 Définir dans la classe **Book** la méthode **copy** qui crée une copie du livre.
   public Book copy() retourne une copie de l'objet Book qui a reçu le message.
2. 👉 On ajoute la méthode **setTitle**(name) qui permet de modifier le titre d'un livre.

-- English version

1. 👉 Define in the **Book** class the **copy** method, which creates a copy of the book.
   public Book copy() returns a copy of the Book object that received the message.
2. 👉 We add the method **setTitle**(name), which allows modifying a book's title.

Voici des exemples de tests

```
@Test
void testSetTitle() {
    Book book = new Book("The Alchemist", "Paulo Coelho", 1988, "123-456-789");
    book.setTitle("The Alchemist 2");
    assertEquals("The Alchemist 2", book.getTitle());
}
```

```
@Test
void testCopy() {
    Book book = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book clone = book.copy();
    assertEquals(book, clone);
    //pour moodle
    assertTrue(book != clone);
}
```

Par exemple:

Test	Résultat

`Book book = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
`Book clone = book.copy();`
`assertEquals(book, clone);`
`assertTrue(book != clone);`
	`test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} equals Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} ?`
	`true`
	`true`

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1  public class Book implements Identifiable, Matchable {
2      private String id;
3      private String title;
4      private String author;
5      private int year;
6
7      private final static String DEFAULT_TITLE = "undefined";
8      private final static String DEFAULT_AUTHOR = "unknown";
9
10     public Book(String title, String author, int year) {
11         this.title = title != null ? title : DEFAULT_TITLE;
12         this.author = author != null ? author : DEFAULT_AUTHOR;
13         this.year = year;
14         this.id = DEFAULT_AUTHOR;
15     }
16
17     public Book(String title, String author, int year, String id) {
18         this(title, author, year);
19         this.id = id != null ? id : DEFAULT_AUTHOR;
20     }
21
22     public String getTitle() {
23         return title;
24     }
25
```

```
    public String getAuthor() {
        return author;
    }

    public int getYear() {
        return year;
    }

    @Override
    public String getId() {
        return id;
    }

    @Override
    public boolean equals(Object o) {
        if(this == o)
            return true;
        if(!(o instanceof Book))
            return false;

        Book book = (Book) o;

        if(this.id.equals(DEFAULT_AUTHOR) && book.id.equals(DEFAULT_AUTHOR) && this.title.equals(book.title) && this.au
            return true;

        return book.id.equals(this.id) && !this.id.equals(DEFAULT_AUTHOR);
```

	Test	Résultat attendu	Résultat obtenu	

✔	`Book book = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book clone = book.copy();`
	`assertEquals(book, clone);`
	`assertTrue(book != clone);`
		`test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} equals Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} ?`
		`true`
		`true`
			`test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} equals Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} ?`
			`true`
			`true`
				✔

✔	`// void testSetTitle() {`
	`Book book = new Book("The Alchemist", "Paulo Coelho", 1988, "123-456-789");`
	`book.setTitle("The Alchemist 2");`
	`assertEquals("The Alchemist 2", book.getTitle());`
		`test : The Alchemist 2 equals The Alchemist 2 ?`
		`true`
			`test : The Alchemist 2 equals The Alchemist 2 ?`
			`true`
				✔

Tous les tests ont été réussis ! ✔

▸ **Solution de l'auteur de la question (Java)**

Correct

Note pour cet envoi : 3,00/3,00.

Question 5

Correct

Note de 5,00 sur 5,00

👉 Définir dans la classe **Book** la méthode **merge** qui crée un nouveau livre à partir de la fusion d'un livre courant avec le livre en paramètre quand c'est possible sinon retourner null

```
public Optional<Book> merge(Book other)
```

1. Deux livres peuvent être fusionnés uniquement si
    1. aucun des deux n'est null,
    2. ET ils ont le même ISBN ou l'un des deux identifiants a pour valeur Unknown,
    3. ET ils sont de la même année,

   si ce n'est pas le cas la méthode retourne un Optional empty
2. *Si une fusion est possible,* le **nouveau livre retourné** a
    1. pour *ISBN*, celui qui est différent de "unknown" ou "unknown" dans le cas où les deux sont inconnus;
    2. pour *année*, celle des deux livres puisqu'elle doit être égale;
    3. pour *titre* :
        1. Si le titre de l'objet courant contient déjà le titre de *other* ou vice-versa, le titre qui contient l'autre
        2. Si le titre de l'objet courant ou le titre de *other* est "undefined", le titre qui n'est pas "undefined";
        3. Si aucun de ces cas ne s'applique, le titre de l'objet courant et le titre de *other* sont retournés concaténés et séparés par la chaîne " or ".
    4. pour *auteur*, la même règle s'applique que pour le titre, en prenant en compte un "unknown" au lieu d'un "undefined".

-- English version

👉 Define in the class **Book** the method **merge,** which creates a new book from the merge of a current book with the book in parameter when possible otherwise returns null

```
public Optional<Book> merge(Book other)
```

1. Two Books can only be merged if
    1. neither is null,
    2. AND they have the same ISBN, or one of the two identifiers has the value Unknown,
    3. AND they are from the same year,

   if this is not the case, the method returns an Optional empty
2. *If a merge is possible,* the **new book returned** has
    1. for *ISBN*, the one that is different from "unknown" or "unknown" in the case where; both are unknown;
    2. for *year*, that of the two books since it must be equal;
    3. for *title*:
        1. If the title of the current object already contains the title of *other* or vice-versa, the title that contains the other
        2. If the title of the current object or the title of *other* is "undefined", the title that is not "undefined";
        3. If none of these cases apply, the title of the current object and the title of *other* are returned concatenated and separated by the string " or ".
    4. for *author*, the same rule applies as for the title, taking into account an "unknown" instead of an "undefined".

Exemples de tests

```
@Test
void testMerge() {
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1988, "1234567890");
    Optional<Book> mergedBook = book1.merge(book2);
    assertTrue(mergedBook.isPresent());
    assertEquals("The Alchemist or Veronika Decides to Die", mergedBook.get().getTitle());
    assertEquals("Paulo Coelho", mergedBook.get().getAuthor());
    assertEquals(1988, mergedBook.get().getYear());
    assertEquals("1234567890", mergedBook.get().getId());
}
```

```
@Test
void testMergeWithDifferentYears() {
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("The Alchemist", "Paulo Coelho", 1998, "0987654321");
    Optional<Book> mergedBook = book1.merge(book2);
    assertFalse(mergedBook.isPresent());
}
```

```
@Test
void testMergeWithUnknownId() {
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown");
    Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "0987654321");
    Optional<Book> mergedBook = book1.merge(book2);
    assertTrue(mergedBook.isPresent());
    assertEquals("The Alchemist", mergedBook.get().getTitle());
    assertEquals("Paulo Coelho", mergedBook.get().getAuthor());
    assertEquals(1988, mergedBook.get().getYear());
    assertEquals("0987654321", mergedBook.get().getId());
}
```

```
@Test
void testMergeWithUnknownAuthor() {
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("undefined", "unknown", 1988, "1234567890");
    Optional<Book> mergedBook = book1.merge(book2);
    assertTrue(mergedBook.isPresent());
    assertEquals("The Alchemist", mergedBook.get().getTitle());
    assertEquals("Paulo Coelho", mergedBook.get().getAuthor());
    assertEquals(1988, mergedBook.get().getYear());
    assertEquals("1234567890", mergedBook.get().getId());
}
```

Par exemple:

Test	Résultat

`//void testMerge() {`
`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
`Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1988, "1234567890");`
`Optional<Book> mergedBook = book1.merge(book2);`
`assertTrue(mergedBook.isPresent());`
`assertEquals("The Alchemist or Veronika Decides to Die", mergedBook.get().getTitle());`
`assertEquals("Paulo Coelho", mergedBook.get().getAuthor());`
`assertEquals(1988, mergedBook.get().getYear());`
`assertEquals("1234567890", mergedBook.get().getId());`
	true
	test : The Alchemist or Veronika Decides to Die equals The Alchemist or Veronika Decides to Die ?
	true
	test : Paulo Coelho equals Paulo Coelho ?
	true
	test : 1988 equals 1988 ?
	true
	test : 1234567890 equals 1234567890 ?
	true

`//void testMergeWithDifferentYears() {`
`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
`Book book2 = new Book("The Alchemist", "Paulo Coelho", 1998, "0987654321");`
`Optional<Book> mergedBook = book1.merge(book2);`
`assertFalse(mergedBook.isPresent());`
	false

`// void testMergeWithUnknownId() {`
`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown");`
`Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "0987654321");`
`Optional<Book> mergedBook = book1.merge(book2);`
`assertTrue(mergedBook.isPresent());`
`assertEquals("The Alchemist", mergedBook.get().getTitle());`
`assertEquals("Paulo Coelho", mergedBook.get().getAuthor());`
`assertEquals(1988, mergedBook.get().getYear());`
`assertEquals("0987654321", mergedBook.get().getId());`
	true
	test : The Alchemist equals The Alchemist ?
	true
	test : Paulo Coelho equals Paulo Coelho ?
	true
	test : 1988 equals 1988 ?
	true
	test : 0987654321 equals 0987654321 ?
	true

Test	Résultat

`//void testMergeWithUndefinedTitle() {`
`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
`Book book2 = new Book("undefined", "Paulo Coelho", 1988, "1234567890");`
`Optional<Book> mergedBook = book1.merge(book2);`
`assertTrue(mergedBook.isPresent());`
`assertEquals("The Alchemist", mergedBook.get().getTitle());`
`assertEquals("Paulo Coelho", mergedBook.get().getAuthor());`
`assertEquals(1988, mergedBook.get().getYear());`
`assertEquals("1234567890", mergedBook.get().getId());`
	`true`
	`test : The Alchemist equals The Alchemist ?`
	`true`
	`test : Paulo Coelho equals Paulo Coelho ?`
	`true`
	`test : 1988 equals 1988 ?`
	`true`
	`test : 1234567890 equals 1234567890 ?`
	`true`

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```java
public class Book implements Identifiable, Matchable {
    private String id;
    private String title;
    private String author;
    private int year;

    private final static String DEFAULT_TITLE = "undefined";
    private final static String DEFAULT_AUTHOR = "unknown";

    public Book(String title, String author, int year) {
        this.title = title != null ? title : DEFAULT_TITLE;
        this.author = author != null ? author : DEFAULT_AUTHOR;
        this.year = year;
        this.id = DEFAULT_AUTHOR;
    }

    public Book(String title, String author, int year, String id) {
        this(title, author, year);
        this.id = id != null ? id : DEFAULT_AUTHOR;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public int getYear() {
        return year;
    }

    @Override
    public String getId() {
        return id;
    }

    @Override
    public boolean equals(Object o) {
        if(this == o)
            return true;
        if(!(o instanceof Book))
            return false;

        Book book = (Book) o;

        if(this.id.equals(DEFAULT_AUTHOR) && book.id.equals(DEFAULT_AUTHOR) && this.title.equals(book.title) && this.au
            return true;

        return book.id.equals(this.id) && !this.id.equals(DEFAULT_AUTHOR);
```

	Test	Résultat attendu	Résultat obtenu	

✔	`//void testMerge() {`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1988, "1234567890");`
	`Optional<Book> mergedBook = book1.merge(book2);`
	`assertTrue(mergedBook.isPresent());`
	`assertEquals("The Alchemist or Veronika Decides to Die", mergedBook.get().getTitle());`
	`assertEquals("Paulo Coelho", mergedBook.get().getAuthor());`
	`assertEquals(1988, mergedBook.get().getYear());`
	`assertEquals("1234567890", mergedBook.get().getId());`
		true
		test : The Alchemist or Veronika Decides to Die equals The Alchemist or Veronika Decides to Die ?
		true
		test : Paulo Coelho equals Paulo Coelho ?
		true
		test : 1988 equals 1988 ?
		true
		test : 1234567890 equals 1234567890 ?
		true
			true
			test : The Alchemist or Veronika Decides to Die equals The Alchemist or Veronika Decides to Die ?
			true
			test : Paulo Coelho equals Paulo Coelho ?
			true
			test : 1988 equals 1988 ?
			true
			test : 1234567890 equals 1234567890 ?
			true
				✔

✔	`//void testMergeWithDifferentAuthors() {`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("The Alchemist", "J.K. Rowling", 1988, "1234567890");`
	`Optional<Book> mergedBook = book1.merge(book2);`
	`assertTrue(mergedBook.isPresent());`
	`assertEquals("The Alchemist", mergedBook.get().getTitle());`
	`assertTrue("Paulo Coelho or J.K. Rowling".equals(mergedBook.get().getAuthor()) ||"J.K. Rowling or Paulo Coelho".equals(mergedBook.get().getAuthor()) );`
	`assertEquals(1988, mergedBook.get().getYear());`
	`assertEquals("1234567890", mergedBook.get().getId());`
		true
		test : The Alchemist equals The Alchemist ?
		true
		true
		test : 1988 equals 1988 ?
		true
		test : 1234567890 equals 1234567890 ?
		true
			true
			test : The Alchemist equals The Alchemist ?
			true
			true
			test : 1988 equals 1988 ?
			true
			test : 1234567890 equals 1234567890 ?
			true
				✔

✔	`//void testMergeWithDifferentYears() {`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("The Alchemist", "Paulo Coelho", 1998, "0987654321");`
	`Optional<Book> mergedBook = book1.merge(book2);`
	`assertFalse(mergedBook.isPresent());`
		false	false	✔

✔	`// void testMergeWithUnknownId() {`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "unknown");`
	`Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "0987654321");`
	`Optional<Book> mergedBook = book1.merge(book2);`
	`assertTrue(mergedBook.isPresent());`
	`assertEquals("The Alchemist", mergedBook.get().getTitle());`
	`assertEquals("Paulo Coelho", mergedBook.get().getAuthor());`
	`assertEquals(1988, mergedBook.get().getYear());`
	`assertEquals("0987654321", mergedBook.get().getId());`
		true
		test : The Alchemist equals The Alchemist ?
		true
		test : Paulo Coelho equals Paulo Coelho ?
		true
		test : 1988 equals 1988 ?
		true
		test : 0987654321 equals 0987654321 ?
		true
			true
			test : The Alchemist equals The Alchemist ?
			true
			test : Paulo Coelho equals Paulo Coelho ?
			true
			test : 1988 equals 1988 ?
			true
			test : 0987654321 equals 0987654321 ?
			true
				✔

✔	`//void testMergeWithUndefinedTitle() {`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("undefined", "Paulo Coelho", 1988, "1234567890");`
	`Optional<Book> mergedBook = book1.merge(book2);`
	`assertTrue(mergedBook.isPresent());`
	`assertEquals("The Alchemist", mergedBook.get().getTitle());`
	`assertEquals("Paulo Coelho", mergedBook.get().getAuthor());`
	`assertEquals(1988, mergedBook.get().getYear());`
	`assertEquals("1234567890", mergedBook.get().getId());`
		true
		test : The Alchemist equals The Alchemist ?
		true
		test : Paulo Coelho equals Paulo Coelho ?
		true
		test : 1988 equals 1988 ?
		true
		test : 1234567890 equals 1234567890 ?
		true
			true
			test : The Alchemist equals The Alchemist ?
			true
			test : Paulo Coelho equals Paulo Coelho ?
			true
			test : 1988 equals 1988 ?
			true
			test : 1234567890 equals 1234567890 ?
			true
				✔

✔	`//void testMergeWithUnknownAuthor() {`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("undefined", "unknown", 1988, "1234567890");`
	`Optional<Book> mergedBook = book1.merge(book2);`
	`assertTrue(mergedBook.isPresent());`
	`assertEquals("The Alchemist", mergedBook.get().getTitle());`
	`assertEquals("Paulo Coelho", mergedBook.get().getAuthor());`
	`assertEquals(1988, mergedBook.get().getYear());`
	`assertEquals("1234567890", mergedBook.get().getId());`
		true
		test : The Alchemist equals The Alchemist ?
		true
		test : Paulo Coelho equals Paulo Coelho ?
		true
		test : 1988 equals 1988 ?
		true
		test : 1234567890 equals 1234567890 ?
		true
			true
			test : The Alchemist equals The Alchemist ?
			true
			test : Paulo Coelho equals Paulo Coelho ?
			true
			test : 1988 equals 1988 ?
			true
			test : 1234567890 equals 1234567890 ?
			true
				✔

	Test	Résultat attendu	Résultat obtenu	

✔	`//void testMergeWithNullItem() {`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Optional<Book> mergedBook = book1.merge(null);`
	`assertFalse(mergedBook.isPresent());`
		`false`	`false`	✔

Tous les tests ont été réussis ! ✔

▸ **Solution de l'auteur de la question (Java)**

Correct

Note pour cet envoi : 5,00/5,00.

Question 6

Correct

Note de 8,00 sur 8,00

Soit l'entête de la classe *ItemDictionnary* qui permet de manipuler dans un dictionnaire (Map) des items identifiables par un identifiant et qui peuvent être recherchés à partir d'une String :

```
public class ItemDictionnary<T extends Identifiable & Matchable > extends HashMap<String, List<T>>
```

Un *itemDictionnary* permet de mémoriser les listes d'objets qui partagent le même identifiant (getId()), comme par exemple les définitions différentes d'un même mot dans un dictionnaire.

Un *itemDictionnary* permet également de retrouver tous les items qui matchent une simple String, comme rechercher toutes les définitions qui contiennent une string donnée.

👉 Vous devez implémenter les méthodes suivantes :

1. *public void addItem(T item)* ajoute l'item dans la liste associée à son identifiant. Attention s'il n'existe pas encore d'Items associés à cette clef, vous devez commencer par créer la liste.
2. *public void removeItem(T item)* retire cet item du dictionnaire s'il est présent (utilisation de ==, car cette fois-ci nous recherchons exactement les mêmes items; donc avoir le même identifiant ne les caractérise pas comme identiques.).
3. *public List<T> findItems(item item)* renvoie la Liste des Items ayant pour identifiant celui de l'item passé en paramètre et null sinon.
4. *public List<T> findMatchableItems(String query)* renvoie tous les items qui "matchent" la string. Elle renvoie une liste vide dans le cas où aucun item ne matche la query.

Dans les tests nous gérerons des listes de livres qui ont le même ISBN.

English version

Let the header of the *ItemDictionary* class, which allows you to manipulate in a dictionary (Map) items with the same identifier that can be searched from a String:

```
public class ItemDictionary<T extends Identifiable & Matchable > extends HashMap<String, List<T>>
```

An *itemDictionary* is used to store lists of items with the same identifier (getId()), such as different definitions of words in a dictionary.

An *itemDictionary* also allows finding all the items that match a String, such as finding all the definitions that contain a given string.

👉 You must implement the following methods:

1. *public void addItem(T item)* adds the item to the list associated with its identifier. Be careful if there are not yet any Items associated with it. this key, you must start by creating the list.
2. *public void removeItem(T item)* removes this item from the dictionary if it is present (use of ==, because this time we are looking for exactly the same items; so having the same identifier does not characterize them as identical.).
3. *public List<T> findItems(item item)* returns the List of Items having the identifier of the item passed; as a parameter and null otherwise.
4. *public List<T> findMatchableItems(String query)* returns all items that "match" the string. It returns an empty list if ù no item matches the query.

In the tests, we will manage lists of books with the same ISBN.

Des exemples de tests

```
/*
---------------------------------- ADD
*/

@Test
void testAddDuplicateItem() {
    ItemDictionnary<Book> dictionary = new ItemDictionnary<>();
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("The Alchemist (2)", "Paulo Coelho", 1988, "1234567890");
    dictionary.addItem(book1);
    dictionary.addItem(book2);
    assertEquals(1, dictionary.size());
    assertEquals(2, dictionary.get("1234567890").size());
}

@Test
void testAddItemWithDifferentIds() {
    ItemDictionnary<Book> dictionary = new ItemDictionnary<>();
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "0987654321");
    dictionary.addItem(book1);
    dictionary.addItem(book2);
    assertEquals(2, dictionary.size());
    assertEquals(1, dictionary.get("1234567890").size());
    assertEquals(1, dictionary.get("0987654321").size());
}


/*
 ---------------------------------- FIND
 */
@Test
void testFindItemsWithExistingId() {
    ItemDictionnary<Book> dictionary = new ItemDictionnary<>();
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    dictionary.addItem(book1);
    dictionary.addItem(book2);
    List<Book> items = dictionary.findItems(book1);
    assertEquals(2, items.size());
    assertTrue(items.contains(book1));
    assertTrue(items.contains(book2));
}

@Test
void testFindItemsWithNonExistingId() {
    ItemDictionnary<Book> dictionary = new ItemDictionnary<>();
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "0987654321");
    dictionary.addItem(book1);
    List<Book> items = dictionary.findItems(book2);
    assertNull(items);
}

@Test
void testFindItemsWithNullItem() {
    ItemDictionnary<Book> dictionary = new ItemDictionnary<>();
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    dictionary.addItem(book1);
    List<Book> items = dictionary.findItems(null);
    assertNull(items);
}

/*
 ---------------------------------- FIND Matchable
  */
@Test
void testFindMatchableItems() {
    ItemDictionnary<Book> dictionary = new ItemDictionnary<>();
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "0987654321");
    Book book3 = new Book("The Alchemist", "Paulo Coelho", 1988, "0000000000");
    dictionary.addItem(book1);
    dictionary.addItem(book2);
    dictionary.addItem(book3);
    List<Book> searchResults = dictionary.findMatchableItems("Alchemist");
    assertEquals(2, searchResults.size());
    assertTrue(searchResults.contains(book1));
    assertTrue(searchResults.contains(book3));
}
```

```
@Test
void testFindMultipleMatchableItems() {
    ItemDictionnary<Book> dictionary = new ItemDictionnary<>();
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book3 = new Book("Veronika Decides to Die", "The Paulo Coelho", 1998, "0987654321");
    dictionary.addItem(book1);
    dictionary.addItem(book2);
    dictionary.addItem(book3);
    List<Book> searchResults = dictionary.findMatchableItems("The");
    assertEquals(3, searchResults.size());
}

/*
 ---------------------------------- REMOVE
*/

@Test
void testRemoveItem() {
    ItemDictionnary<Book> dictionary = new ItemDictionnary<>();
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "0987654321");
    Book book3 = new Book("The Alchemist", "Paulo Coelho", 1988, "0000000000");
    dictionary.addItem(book1);
    dictionary.addItem(book2);
    dictionary.addItem(book3);
    assertTrue(dictionary.removeItem(book1));
    assertFalse(dictionary.removeItem(book1));
    assertEquals(2, dictionary.size());
    assertNull(dictionary.get("1234567890"));
}

@Test
void testRemoveItemWithMultipleCopies() {
    ItemDictionnary<Book> dictionary = new ItemDictionnary<>();
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    dictionary.addItem(book1);
    dictionary.addItem(book2);
    assertTrue(dictionary.removeItem(book1));
    assertEquals(1, dictionary.size());
    assertEquals(1, dictionary.get("1234567890").size());
}
```

Par exemple:

Test	Résultat

`//void testAddItem()`
`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
`Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "0987654321");`
`dictionary.addItem(book1);`
`dictionary.addItem(book2);`
`assertEquals(2, dictionary.size());`
`assertEquals(1, dictionary.get("1234567890").size());`
`assertEquals(1, dictionary.get("0987654321").size());`
	`test : 2 equals 2 ?`
	`true`
	`test : 1 equals 1 ?`
	`true`
	`test : 1 equals 1 ?`
	`true`

`//void testAddItemWithDifferentIds() {`
`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
`Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "0987654321");`
`dictionary.addItem(book1);`
`dictionary.addItem(book2);`
`assertEquals(2, dictionary.size());`
`assertEquals(1, dictionary.get("1234567890").size());`
`assertEquals(1, dictionary.get("0987654321").size());`
	`test : 2 equals 2 ?`
	`true`
	`test : 1 equals 1 ?`
	`true`
	`test : 1 equals 1 ?`
	`true`

`//void testFindItemsWithExistingId() {`
`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
`Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
`dictionary.addItem(book1);`
`dictionary.addItem(book2);`
`List<Book> items = dictionary.findItems(book1);`
`assertEquals(2, items.size());`
`assertTrue(items.contains(book1));`
`assertTrue(items.contains(book2));`
	`test : 2 equals 2 ?`
	`true`
	`true`
	`true`

Test	Résultat

`//void testFindMatchableItems() {`
`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
`Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "0987654321");`
`Book book3 = new Book("The Alchemist", "Paulo Coelho", 1988, "0000000000");`
`dictionary.addItem(book1);`
`dictionary.addItem(book2);`
`dictionary.addItem(book3);`
`List<Book> searchResults = dictionary.findMatchableItems("Alchemist");`
`assertEquals(2, searchResults.size());`
`assertTrue(searchResults.contains(book1));`
`assertTrue(searchResults.contains(book3));`
	`test : 2 equals 2 ?`
	`true`
	`true`
	`true`

`//  void testRemoveItemWithMultipleCopies() {`
`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
`Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
`dictionary.addItem(book1);`
`dictionary.addItem(book2);`
`assertTrue(dictionary.removeItem(book1));`
`assertEquals(1, dictionary.size());`
`assertEquals(1, dictionary.get("1234567890").size());`
	`true`
	`test : 1 equals 1 ?`
	`true`
	`test : 1 equals 1 ?`
	`true`

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```java
public class ItemDictionnary<T extends Identifiable & Matchable > extends HashMap<String, List<T>> {


    public ItemDictionnary() {
        super();
    }

    public void addItem(T item) {
        if(this.containsKey(item.getId()))
            this.get(item.getId()).add(item);
        else {
            this.put(item.getId(), new ArrayList<>(List.of(item)));
        }
    }


    public boolean removeItem(T element) {
        if(element == null || this.get(element.getId())== null)
            return false;
        boolean res = this.get(element.getId()).remove(element);
        if(this.get(element.getId()).isEmpty())
            this.remove(element.getId());
        return res;
    }


    public List<T> findItems(T item){
        if(item == null)
            return null;
        return this.get(item.getId());
    }


    public List<T> findMatchableItems(String query){
        List<T> res = new ArrayList<>();
        for(List<T> list : this.values()){
            for(T item : list){
                if(item.match(query)){
                    res.add(item);
                }
            }
        }
        return res;
    }

}
```

	Test	Résultat attendu	Résultat obtenu	

✔	`//void testAddItem()`
	`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "0987654321");`
	`dictionary.addItem(book1);`
	`dictionary.addItem(book2);`
	`assertEquals(2, dictionary.size());`
	`assertEquals(1, dictionary.get("1234567890").size());`
	`assertEquals(1, dictionary.get("0987654321").size());`
		test : 2 equals 2 ?
		true
		test : 1 equals 1 ?
		true
		test : 1 equals 1 ?
		true
			test : 2 equals 2 ?
			true
			test : 1 equals 1 ?
			true
			test : 1 equals 1 ?
			true
				✔

✔	`//void testAddDuplicateItem()`
	`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("The Alchemist (2)", "Paulo Coelho", 1988, "1234567890");`
	`dictionary.addItem(book1);`
	`dictionary.addItem(book2);`
	`assertEquals(1, dictionary.size());`
	`assertEquals(2, dictionary.get("1234567890").size());`
		test : 1 equals 1 ?
		true
		test : 2 equals 2 ?
		true
			test : 1 equals 1 ?
			true
			test : 2 equals 2 ?
			true
				✔

✔	`//void testAddItemWithDifferentIds() {`
	`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "0987654321");`
	`dictionary.addItem(book1);`
	`dictionary.addItem(book2);`
	`assertEquals(2, dictionary.size());`
	`assertEquals(1, dictionary.get("1234567890").size());`
	`assertEquals(1, dictionary.get("0987654321").size());`
		test : 2 equals 2 ?
		true
		test : 1 equals 1 ?
		true
		test : 1 equals 1 ?
		true
			test : 2 equals 2 ?
			true
			test : 1 equals 1 ?
			true
			test : 1 equals 1 ?
			true
				✔

✔	`//void testFindItemsWithExistingId() {`
	`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`dictionary.addItem(book1);`
	`dictionary.addItem(book2);`
	`List<Book> items = dictionary.findItems(book1);`
	`assertEquals(2, items.size());`
	`assertTrue(items.contains(book1));`
	`assertTrue(items.contains(book2));`
		test : 2 equals 2 ?
		true
		true
		true
			test : 2 equals 2 ?
			true
			true
			true
				✔

✔	`//void testFindItemsWithNonExistingId() {`
	`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "0987654321");`
	`dictionary.addItem(book1);`
	`List<Book> items = dictionary.findItems(book2);`
	`assertNull(items);`
		test : null is Null
		true
			test : null is Null
			true
				✔

✔	`// void testFindItemsWithNullItem() {`
	`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`dictionary.addItem(book1);`
	`List<Book> items = dictionary.findItems(null);`
	`assertNull(items);`
		test : null is Null
		true
			test : null is Null
			true
				✔

✔	`//void testFindMatchableItems() {`
	`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "0987654321");`
	`Book book3 = new Book("The Alchemist", "Paulo Coelho", 1988, "0000000000");`
	`dictionary.addItem(book1);`
	`dictionary.addItem(book2);`
	`dictionary.addItem(book3);`
	`List<Book> searchResults = dictionary.findMatchableItems("Alchemist");`
	`assertEquals(2, searchResults.size());`
	`assertTrue(searchResults.contains(book1));`
	`assertTrue(searchResults.contains(book3));`
		test : 2 equals 2 ?
		true
		true
		true
			test : 2 equals 2 ?
			true
			true
			true
				✔

	Test	Résultat attendu	Résultat obtenu	

✔	`//void testFindMatchableItemsWithNonMatchingQuery() {`
	`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "0987654321");`
	`Book book3 = new Book("The Alchemist", "Paulo Coelho", 1988, "0000000000");`
	`dictionary.addItem(book1);`
	`dictionary.addItem(book2);`
	`dictionary.addItem(book3);`
	`List<Book> searchResults = dictionary.findMatchableItems("FooBar");`
	`assertEquals(0, searchResults.size());`
		`test : 0 equals 0`
		`?`
		`true`
			`test : 0 equals 0`
			`?`
			`true`
				✔

✔	`//void testFindMultipleMatchableItems() {`
	`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book3 = new Book("Veronika Decides to Die", "The Paulo Coelho", 1998, "0987654321");`
	`dictionary.addItem(book1);`
	`dictionary.addItem(book2);`
	`dictionary.addItem(book3);`
	`List<Book> searchResults = dictionary.findMatchableItems("The");`
	`assertEquals(3, searchResults.size());`
		`test : 3 equals 3`
		`?`
		`true`
			`test : 3 equals 3`
			`?`
			`true`
				✔

✔	`//    void testRemoveItemWithMultipleCopies() {`
	`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`dictionary.addItem(book1);`
	`dictionary.addItem(book2);`
	`assertTrue(dictionary.removeItem(book1));`
	`assertEquals(1, dictionary.size());`
	`assertEquals(1, dictionary.get("1234567890").size());`
		`true`
		`test : 1 equals 1`
		`?`
		`true`
		`test : 1 equals 1`
		`?`
		`true`
			`true`
			`test : 1 equals 1`
			`?`
			`true`
			`test : 1 equals 1`
			`?`
			`true`
				✔

✔	`//void testRemoveItemWithNonExistingId() {`
	`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "0987654321");`
	`dictionary.addItem(book1);`
	`assertFalse(dictionary.removeItem(book2));`
	`assertEquals(1, dictionary.size());`
		`false`
		`test : 1 equals 1`
		`?`
		`true`
			`false`
			`test : 1 equals 1`
			`?`
			`true`
				✔

✔	`//void testRemoveItemWithNullItem() {`
	`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`dictionary.addItem(book1);`
	`assertFalse(dictionary.removeItem(null));`
	`assertEquals(1, dictionary.size());`
		`false`
		`test : 1 equals 1`
		`?`
		`true`
			`false`
			`test : 1 equals 1`
			`?`
			`true`
				✔

✔	`// void testRemoveItem() {`
	`ItemDictionnary<Book> dictionary = new ItemDictionnary<>();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("Veronika Decides to Die", "Paulo Coelho", 1998, "0987654321");`
	`Book book3 = new Book("The Alchemist", "Paulo Coelho", 1988, "0000000000");`
	`dictionary.addItem(book1);`
	`dictionary.addItem(book2);`
	`dictionary.addItem(book3);`
	`assertTrue(dictionary.removeItem(book1));`
	`assertFalse(dictionary.removeItem(book1));`
	`assertEquals(2, dictionary.size());`
	`assertNull(dictionary.get("1234567890"));`
		`true`
		`false`
		`test : 2 equals 2`
		`?`
		`true`
		`test : null is Null`
		`true`
			`true`
			`false`
			`test : 2 equals 2`
			`?`
			`true`
			`test : null is Null`
			`true`
				✔

Tous les tests ont été réussis ! ✔

▸ **Solution de l'auteur de la question (Java)**

Correct

Note pour cet envoi : 8,00/8,00.

Question 7

Correct

Note de 5,00 sur 5,00

👉 Définir la classe **Library,** telle que toute instance de Library

1. a toutes les caractéristiques d'un *ItemDictionnary (donnée)* de livres;
2. Implémente une méthode dont la signature est

```
public Optional<Book> mergeBooks(Book book)
```

- Cette méthode recherche tous les livres qui ont le même ISBN que le livre passé en argument.
  - Si aucun livre n'est trouvé, la méthode renvoie un objet Optional contenant une copie du livre passé en argument.
  - Sinon, la méthode itère sur la liste des livres trouvés et essaie de les fusionner avec le livre passé en argument.
    - Si la fusion est réussie, le livre fusionné remplace le livre passé en argument et le processus continue.
    - Enfin, la méthode renvoie un objet Optional contenant le livre résultant de toutes les fusions.

--- English Version

👉 Define the **Library** class that

1. defines all the features of a book *ItemDictionnary*.
2. Implements a method whose signature is

```
public Optional<Book> mergeBooks(Book book)
```

- This method searches for all books with the same ISBN as the book passed as argument.
  - If no book is found, the method returns an Optional object containing a copy of the book passed as an argument.
  - Otherwise, the method iterates over the list of books found and tries to merge them with the book passed as an argument.
    - If the merge is successful, the merged book replaces the book passed as an argument, and the process continues.
    - Finally, the method returns an Optional object containing the book resulting from all the merges.

Voici des exemples de tests

```
@Test
void testAddBooks() {
    Library library = new Library();
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("The Alchemist 2", "Paulo Coelho*", 1988, "1234567890");
    Book book4 = new Book("The Alchemist 2-a", "Paulo Coelho", 1988, "1234567890");
    Book book3 = new Book("The Alchemist 3", "Paulo Coelho", 1988, "1111111111");
    library.addItem(book1);
    library.addItem(book2);
    library.addItem(book3);
    library.addItem(book4);
    assertEquals(3,library.get("1234567890").size());
    assertEquals(1,library.get("1111111111").size());
}

@Test
void testMergeBooksWithDifferentNonMergeableBooks() {
Library library = new Library();
Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
Book book2 = new Book("The Alchemist 2", "Paulo Coelho", 1998, "0987654321");
library.addItem(book2);
Optional<Book> mergedBook = library.mergeBooks(book1);
assertTrue(mergedBook.isPresent());
assertEquals("The Alchemist", mergedBook.get().getTitle());
assertEquals("Paulo Coelho", mergedBook.get().getAuthor());
assertEquals(1988, mergedBook.get().getYear());
assertEquals("1234567890", mergedBook.get().getId());
}
```

```
@Test
void testMergeBooksWithMultipleOtherBooks() {
    Library library = new Library();
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("The Alchemist 2", "Paulo Coelho*", 1988, "1234567890");
    Book book4 = new Book("The Alchemist 2-a", "Paulo Coelho", 1988, "1234567890");
    Book book3 = new Book("The Alchemist 3", "Paulo Coelho", 1988, "1111111111");
    library.addItem(book2);
    library.addItem(book3);
    library.addItem(book4);
    Optional<Book> mergedBook = library.mergeBooks(book1);
    assertTrue(mergedBook.isPresent());
    assertEquals("The Alchemist 2-a", mergedBook.get().getTitle());
    assertEquals("Paulo Coelho*", mergedBook.get().getAuthor());
    assertEquals(1988, mergedBook.get().getYear());
    assertEquals("1234567890", mergedBook.get().getId());
}
```

Par exemple:

Test	Résultat

`//void testMergeBooksWithMultipleOtherBooks() {`
`Library library = new Library();`
`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
`Book book2 = new Book("The Alchemist 2", "Paulo Coelho*", 1988, "1234567890");`
`Book book4 = new Book("The Alchemist 2-a", "Paulo Coelho", 1988, "1234567890");`
`Book book3 = new Book("The Alchemist 3", "Paulo Coelho", 1988, "1111111111");`
`library.addItem(book2);`
`library.addItem(book3);`
`library.addItem(book4);`
`Optional<Book> mergedBook = library.mergeBooks(book1);`
`assertTrue(mergedBook.isPresent());`
`assertEquals("The Alchemist 2-a", mergedBook.get().getTitle());`
`assertEquals("Paulo Coelho*", mergedBook.get().getAuthor());`
`assertEquals(1988, mergedBook.get().getYear());`
`assertEquals("1234567890", mergedBook.get().getId());`
	`true`
	`test : The Alchemist 2-a equals The Alchemist 2-a ?`
	`true`
	`test : Paulo Coelho* equals Paulo Coelho* ?`
	`true`
	`test : 1988 equals 1988 ?`
	`true`
	`test : 1234567890 equals 1234567890 ?`
	`true`

`// void testMergeBooksWithNullBook() {`
`Library library = new Library();`
`Optional<Book> mergedBook = library.mergeBooks(null);`
`assertFalse(mergedBook.isPresent());`
	`false`

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1  //Definissez bien votre entête de classe vous n'aurez qu'une seule méthode à implémenter.
2  //Define your class header so that you only have to implement one method.
3  public class Library extends ItemDictionnary<Book> {
4
5
6      public Optional<Book> mergeBooks(Book book) {
7          if(book == null)
8              return Optional.empty();
9          List<Book> books = this.findItems(book);
10         if(books == null)
11             return Optional.of(book);
12
13         Book res = book;
14         Optional<Book> merged = Optional.empty();
15         for(Book b : books){
16             merged = res.merge(b);
17             if(merged.isPresent()){
18                 res = merged.get();
19             }
20         }
21         return merged;
22     }
23
24     public List<Book> sortByTitle() {
25         List<Book> res = new ArrayList<>();
26         for(List<Book> list : this.values()){
27             res.addAll(list);
28         }
29         res.sort(Comparator.comparing(Book::getTitle));
```

```
30          return res;
31      }
32
33
34      public List<Book> sortByAuthor() {
35          List<Book> res = new ArrayList<>();
36          for(List<Book> list : this.values()){
37              res.addAll(list);
38          }
39          res.sort(Comparator.comparing(Book::getAuthor));
40          return res;
41      }
42  }
43
```

	Test	Résultat attendu	Résultat obtenu	

✔	`//void testMergeBooksWithMultipleOtherBooks() {`
	`Library library = new Library();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("The Alchemist 2", "Paulo Coelho*", 1988, "1234567890");`
	`Book book4 = new Book("The Alchemist 2-a", "Paulo Coelho", 1988, "1234567890");`
	`Book book3 = new Book("The Alchemist 3", "Paulo Coelho", 1988, "1111111111");`
	`library.addItem(book2);`
	`library.addItem(book3);`
	`library.addItem(book4);`
	`Optional<Book> mergedBook = library.mergeBooks(book1);`
	`assertTrue(mergedBook.isPresent());`
	`assertEquals("The Alchemist 2-a", mergedBook.get().getTitle());`
	`assertEquals("Paulo Coelho*", mergedBook.get().getAuthor());`
	`assertEquals(1988, mergedBook.get().getYear());`
	`assertEquals("1234567890", mergedBook.get().getId());`
		true
		test : The Alchemist 2-a equals The Alchemist 2-a ?
		true
		test : Paulo Coelho* equals Paulo Coelho* ?
		true
		test : 1988 equals 1988 ?
		true
		test : 1234567890 equals 1234567890 ?
		true
			true
			test : The Alchemist 2-a equals The Alchemist 2-a ?
			true
			test : Paulo Coelho* equals Paulo Coelho* ?
			true
			test : 1988 equals 1988 ?
			true
			test : 1234567890 equals 1234567890 ?
			true
				✔

✔	`//void testMergeBooksWithDifferentTitles() {`
	`Library library = new Library();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("The Alchemist 2", "PauloCoelho", 1988, "1234567890");`
	`library.addItem(book2);`
	`Optional<Book> mergedBook = library.mergeBooks(book1);`
	`assertTrue(mergedBook.isPresent());`
	`assertEquals("The Alchemist 2", mergedBook.get().getTitle());`
	`assertTrue("Paulo Coelho or PauloCoelho".equals(mergedBook.get().getAuthor()) ||`
	`"PauloCoelho or Paulo Coelho".equals(mergedBook.get().getAuthor()) );`
	`assertEquals(1988, mergedBook.get().getYear());`
	`assertEquals("1234567890", mergedBook.get().getId());`
		true
		test : The Alchemist 2 equals The Alchemist 2 ?
		true
		true
		test : 1988 equals 1988 ?
		true
		test : 1234567890 equals 1234567890 ?
		true
			true
			test : The Alchemist 2 equals The Alchemist 2 ?
			true
			true
			test : 1988 equals 1988 ?
			true
			test : 1234567890 equals 1234567890 ?
			true
				✔

✔	`//void testMergeBooksWithNoOtherBook() {`
	`Library library = new Library();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Optional<Book> mergedBook = library.mergeBooks(book1);`
	`assertTrue(mergedBook.isPresent());`
	`assertEquals("The Alchemist", mergedBook.get().getTitle());`
	`assertEquals("Paulo Coelho", mergedBook.get().getAuthor());`
	`assertEquals(1988, mergedBook.get().getYear());`
	`assertEquals("1234567890", mergedBook.get().getId());`
		true
		test : The Alchemist equals The Alchemist ?
		true
		test : Paulo Coelho equals Paulo Coelho ?
		true
		test : 1988 equals 1988 ?
		true
		test : 1234567890 equals 1234567890 ?
		true
			true
			test : The Alchemist equals The Alchemist ?
			true
			test : Paulo Coelho equals Paulo Coelho ?
			true
			test : 1988 equals 1988 ?
			true
			test : 1234567890 equals 1234567890 ?
			true
				✔

✔	`// void testMergeBooksWithNullBook() {`
	`Library library = new Library();`
	`Optional<Book> mergedBook = library.mergeBooks(null);`
	`assertFalse(mergedBook.isPresent());`
		false	false	✔

	Test	Résultat attendu	Résultat obtenu	

✔	`//void testMergeBooksWithDifferentNonMergeableBooks() {`
	`Library library = new Library();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("The Alchemist 2", "Paulo Coelho", 1998, "0987654321");`
	`library.addItem(book2);`
	`Optional<Book> mergedBook = library.mergeBooks(book1);`
	`assertTrue(mergedBook.isPresent());`
	`assertEquals("The Alchemist", mergedBook.get().getTitle());`
	`assertEquals("Paulo Coelho", mergedBook.get().getAuthor());`
	`assertEquals(1988, mergedBook.get().getYear());`
	`assertEquals("1234567890", mergedBook.get().getId());`
		`true`
		`test : The Alchemist equals The Alchemist ?`
		`true`
		`test : Paulo Coelho equals Paulo Coelho ?`
		`true`
		`test : 1988 equals 1988 ?`
		`true`
		`test : 1234567890 equals 1234567890 ?`
		`true`
			`true`
			`test : The Alchemist equals The Alchemist ?`
			`true`
			`test : Paulo Coelho equals Paulo Coelho ?`
			`true`
			`test : 1988 equals 1988 ?`
			`true`
			`test : 1234567890 equals 1234567890 ?`
			`true`
				✔

✔	`//void testAddBooks() {`
	`Library library = new Library();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("The Alchemist 2", "Paulo Coelho*", 1988, "1234567890");`
	`Book book4 = new Book("The Alchemist 2-a", "Paulo Coelho", 1988, "1234567890");`
	`Book book3 = new Book("The Alchemist 3", "Paulo Coelho", 1988, "1111111111");`
	`library.addItem(book1);`
	`library.addItem(book2);`
	`library.addItem(book3);`
	`library.addItem(book4);`
	`assertEquals(3,library.get("1234567890").size());`
	`assertEquals(1,library.get("1111111111").size());`
		`test : 3 equals 3 ?`
		`true`
		`test : 1 equals 1 ?`
		`true`
			`test : 3 equals 3 ?`
			`true`
			`test : 1 equals 1 ?`
			`true`
				✔

Tous les tests ont été réussis ! ✔

▸ **Solution de l'auteur de la question (Java)**

Correct

Note pour cet envoi : 5,00/5,00.

Question 8

Correct

Note de 5,00 sur 5,00

Ajouter à la classe **Library** les méthodes

- *public List<Book> sortByTitle()* qui renvoie la liste triée par leur titre des livres qui sont dans la bibliothèque
- *public List<Book> sortByAuthor()* qui renvoie la liste triée par leur auteur des livres qui sont dans la bibliothèque

-- English version

Add to the class **Library** the methods

- *public List<Book> sortByTitle()* which returns the list sorted by their title of books that are in the library
- *public List<Book> sortByAuthor()* which returns the list sorted by their author of books that are in the library

Des exemples de tests

```
@Test
void testSortByTitle() {
    Library library = new Library();
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("The Alchemist 2", "Paulo Coelho", 1998, "1234567890");
    Book book3 = new Book("The Alchemist 3", "Paulo Coelho", 2008, "9876543210");
    Book book4 = new Book("The Alchemist 4", "J.K. Rowling", 2018, "1231231230");
    library.addItem(book3);
    library.addItem(book2);
    library.addItem(book4);
    library.addItem(book1);
    List<Book> sortedBooks = library.sortByTitle();
    assertEquals(book1, sortedBooks.get(0));
    assertEquals(book2, sortedBooks.get(1));
    assertEquals(book3, sortedBooks.get(2));
    assertEquals(book4, sortedBooks.get(3));
}


@Test
void testSortByAuthor() {
    Library library = new Library();
    Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");
    Book book2 = new Book("The Alchemist 2", "Paulo Coelho", 1988, "1234567890");
    Book book3 = new Book("The Alchemist 3", "Paulo Coelho de Souza", 1988, "1234567890");
    Book book4 = new Book("The Alchemist 4", "J.K. Rowling", 2018, "1231231230");
    library.addItem(book3);
    library.addItem(book2);
    library.addItem(book4);
    library.addItem(book1);
    List<Book> sortedBooks = library.sortByAuthor();
    assertEquals(book4, sortedBooks.get(0));
    assertEquals(book3, sortedBooks.get(3));
}
```

Par exemple:

Test	Résultat

`//void testSortByTitle() {`
`Library library = new Library();`
`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
`Book book2 = new Book("The Alchemist 2", "Paulo Coelho", 1998, "1234567890");`
`Book book3 = new Book("The Alchemist 3", "Paulo Coelho", 2008, "9876543210");`
`Book book4 = new Book("The Alchemist 4", "J.K. Rowling", 2018, "1231231230");`
`library.addItem(book3);`
`library.addItem(book2);`
`library.addItem(book4);`
`library.addItem(book1);`
`List<Book> sortedBooks = library.sortByTitle();`
`assertEquals(book1, sortedBooks.get(0));`
`assertEquals(book2, sortedBooks.get(1));`
`assertEquals(book3, sortedBooks.get(2));`
`assertEquals(book4, sortedBooks.get(3));`
	`test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} equals Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} ?`
	`true`
	`test : Book{title='The Alchemist 2', author='Paulo Coelho', year=1998, isbn='1234567890'} equals Book{title='The Alchemist 2', author='Paulo Coelho', year=1998, isbn='1234567890'} ?`
	`true`
	`test : Book{title='The Alchemist 3', author='Paulo Coelho', year=2008, isbn='9876543210'} equals Book{title='The Alchemist 3', author='Paulo Coelho', year=2008, isbn='9876543210'} ?`
	`true`
	`test : Book{title='The Alchemist 4', author='J.K. Rowling', year=2018, isbn='1231231230'} equals Book{title='The Alchemist 4', author='J.K. Rowling', year=2018, isbn='1231231230'} ?`
	`true`

`//void testSortByAuthor() {`
`Library library = new Library();`
`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
`Book book2 = new Book("The Alchemist 2", "Paulo Coelho", 1988, "1234567890");`
`Book book3 = new Book("The Alchemist 3", "Paulo Coelho de Souza", 1988, "1234567890");`
`Book book4 = new Book("The Alchemist 4", "J.K. Rowling", 2018, "1231231230");`
`library.addItem(book3);`
`library.addItem(book2);`
`library.addItem(book4);`
`library.addItem(book1);`
`List<Book> sortedBooks = library.sortByAuthor();`
`assertEquals(book4, sortedBooks.get(0));`
`assertEquals(book3, sortedBooks.get(3));`
	`test : Book{title='The Alchemist 4', author='J.K. Rowling', year=2018, isbn='1231231230'} equals Book{title='The Alchemist 4', author='J.K. Rowling', year=2018, isbn='1231231230'} ?`
	`true`
	`test : Book{title='The Alchemist 3', author='Paulo Coelho de Souza', year=1988, isbn='1234567890'} equals Book{title='The Alchemist 3', author='Paulo Coelho de Souza', year=1988, isbn='1234567890'} ?`
	`true`

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1  //Definissez bien votre entête de classe vous n'aurez qu'une seule méthode à implémenter.
2
3  //Definissez bien votre entête de classe vous n'aurez qu'une seule méthode à implémenter.
4  //Define your class header so that you only have to implement one method.
5  public class Library extends ItemDictionnary<Book> {
6
7
8      public Optional<Book> mergeBooks(Book book) {
9          if(book == null)
10             return Optional.empty();
11         List<Book> books = this.findItems(book);
12         if(books == null)
13             return Optional.of(book);
14
15         Book res = book;
16         Optional<Book> merged = Optional.empty();
17         for(Book b : books){
18             merged = res.merge(b);
19             if(merged.isPresent()){
20                 res = merged.get();
21             }
22         }
23         return merged;
24     }
25
26     public List<Book> sortByTitle() {
27         List<Book> res = new ArrayList<>();
28         for(List<Book> list : this.values()){
29             res.addAll(list);
30         }
31         res.sort(Comparator.comparing(Book::getTitle));
32         return res;
33     }
34
35
36     public List<Book> sortByAuthor() {
37         List<Book> res = new ArrayList<>();
```

```
        for(List<Book> list : this.values()){
            res.addAll(list);
        }
        res.sort(Comparator.comparing(Book::getAuthor));
        return res;
    }
}
```

	Test	Résultat attendu	Résultat obtenu	

✔	`//void testSortByTitle() {`
	`Library library = new Library();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("The Alchemist 2", "Paulo Coelho", 1998, "1234567890");`
	`Book book3 = new Book("The Alchemist 3", "Paulo Coelho", 2008, "9876543210");`
	`Book book4 = new Book("The Alchemist 4", "J.K. Rowling", 2018, "1231231230");`
	`library.addItem(book3);`
	`library.addItem(book2);`
	`library.addItem(book4);`
	`library.addItem(book1);`
	`List<Book> sortedBooks = library.sortByTitle();`
	`assertEquals(book1, sortedBooks.get(0));`
	`assertEquals(book2, sortedBooks.get(1));`
	`assertEquals(book3, sortedBooks.get(2));`
	`assertEquals(book4, sortedBooks.get(3));`
		test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} equals Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} ?
		true
		test : Book{title='The Alchemist 2', author='Paulo Coelho', year=1998, isbn='1234567890'} equals Book{title='The Alchemist 2', author='Paulo Coelho', year=1998, isbn='1234567890'} ?
		true
		test : Book{title='The Alchemist 3', author='Paulo Coelho', year=2008, isbn='9876543210'} equals Book{title='The Alchemist 3', author='Paulo Coelho', year=2008, isbn='9876543210'} ?
		true
		test : Book{title='The Alchemist 4', author='J.K. Rowling', year=2018, isbn='1231231230'} equals Book{title='The Alchemist 4', author='J.K. Rowling', year=2018, isbn='1231231230'} ?
		true
			test : Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} equals Book{title='The Alchemist', author='Paulo Coelho', year=1988, isbn='1234567890'} ?
			true
			test : Book{title='The Alchemist 2', author='Paulo Coelho', year=1998, isbn='1234567890'} equals Book{title='The Alchemist 2', author='Paulo Coelho', year=1998, isbn='1234567890'} ?
			true
			test : Book{title='The Alchemist 3', author='Paulo Coelho', year=2008, isbn='9876543210'} equals Book{title='The Alchemist 3', author='Paulo Coelho', year=2008, isbn='9876543210'} ?
			true
			test : Book{title='The Alchemist 4', author='J.K. Rowling', year=2018, isbn='1231231230'} equals Book{title='The Alchemist 4', author='J.K. Rowling', year=2018, isbn='1231231230'} ?
			true
				✔

✔	`//void testSortByAuthor() {`
	`Library library = new Library();`
	`Book book1 = new Book("The Alchemist", "Paulo Coelho", 1988, "1234567890");`
	`Book book2 = new Book("The Alchemist 2", "Paulo Coelho", 1988, "1234567890");`
	`Book book3 = new Book("The Alchemist 3", "Paulo Coelho de Souza", 1988, "1234567890");`
	`Book book4 = new Book("The Alchemist 4", "J.K. Rowling", 2018, "1231231230");`
	`library.addItem(book3);`
	`library.addItem(book2);`
	`library.addItem(book4);`
	`library.addItem(book1);`
	`List<Book> sortedBooks = library.sortByAuthor();`
	`assertEquals(book4, sortedBooks.get(0));`
	`assertEquals(book3, sortedBooks.get(3));`
		test : Book{title='The Alchemist 4', author='J.K. Rowling', year=2018, isbn='1231231230'} equals Book{title='The Alchemist 4', author='J.K. Rowling', year=2018, isbn='1231231230'} ?
		true
		test : Book{title='The Alchemist 3', author='Paulo Coelho de Souza', year=1988, isbn='1234567890'} equals Book{title='The Alchemist 3', author='Paulo Coelho de Souza', year=1988, isbn='1234567890'} ?
		true
			test : Book{title='The Alchemist 4', author='J.K. Rowling', year=2018, isbn='1231231230'} equals Book{title='The Alchemist 4', author='J.K. Rowling', year=2018, isbn='1231231230'} ?
			true
			test : Book{title='The Alchemist 3', author='Paulo Coelho de Souza', year=1988, isbn='1234567890'} equals Book{title='The Alchemist 3', author='Paulo Coelho de Souza', year=1988, isbn='1234567890'} ?
			true
				✔

✔	`//void testSortByAuthorInEmptyLibrary() {`
	`Library library = new Library();`
	`List<Book> sortedBooks = library.sortByAuthor();`
	
	`assertTrue(sortedBooks.isEmpty());`
		true	true	✔

Tous les tests ont été réussis ! ✔

▸ **Solution de l'auteur de la question (Java)**

Correct

Note pour cet envoi : 5,00/5,00.

Question 9

Correct

Note de 5,00 sur 5,00

Ajouter à la classe **Library** la méthode

```
public List<Book> listAddedBooks()
```

qui renvoie les livres qui ont été ajoutés par *addItem* dans la bibliothèque, dans l'ordre d'ajout dans la bibliothèque (le plus ancien en premier) et qui sont toujours présents.

-- English version

Add to the class **Library** the method

```
public List<Book> listAddedBooks()
```

which returns books that have been added by *addItem* to the library, in order of addition to the library (oldest first) and still present.

Exemples de tests

```
@Test
void testListAddedBooks_noAddedBooks() {
    Library library = new Library();
    List<Book> addedBooks = library.listAddedBooks();
    assertTrue(addedBooks.isEmpty());
}

@Test
void testListAddedBooks_oneAddedBook_stillPresent() {
    Library library = new Library();
    Book book = new Book("Title", "Author", 2000, "ISBN");
    library.addItem(book);
    List<Book> addedBooks = library.listAddedBooks();
    assertEquals(1, addedBooks.size());
    assertEquals(book, addedBooks.get(0));
}

@Test
void testListAddedBooks_oneAddedBook_removed() {
    Library library = new Library();
    Book book = new Book("Title", "Author", 2000, "ISBN");
    library.addItem(book);
    library.removeItem(book);
    List<Book> addedBooks = library.listAddedBooks();
    assertEquals(0, addedBooks.size());
}
```

Par exemple:

Test	Résultat

`// public void testListAddedBooks() {`
`Library library = new Library();`
 	
`// On crée deux livres qui ont le même ISBN`
`Book book1 = new Book("Titre 1", "Auteur 1", 2000, "ISBN 1");`
`Book book2 = new Book("Titre 2", "Auteur 2", 2002, "ISBN 2");`
`Book book3 = new Book("Titre 3", "Auteur 3", 2004, "ISBN 3");`
`Book book4 = new Book("Titre 4", "Auteur 4", 2006, "ISBN 1");`
 	
`// On ajoute les livres à la bibliothèque`
`library.addItem(book1);`
`library.addItem(book2);`
`library.addItem(book3);`
`library.addItem(book4);`
 	
`// On vérifie que la méthode listAddedBooks retourne bien les deux premiers livres`
`List<Book> addedBooks = library.listAddedBooks();`
`assertEquals(4, addedBooks.size());`
`assertTrue(addedBooks.contains(book1));`
`assertTrue(addedBooks.contains(book2));`
`//On verifie l'ordre`
`assertTrue(book1 == addedBooks.get(0));`
`assertTrue(book4 == addedBooks.get(3));`
`// On retire un livre de la bibliothèque`
`library.removeItem(book1);`
 	
`// On vérifie que la méthode listAddedBooks ne retourne plus le livre qui a été retiré`
`addedBooks = library.listAddedBooks();`
`assertEquals(3, addedBooks.size());`
`assertTrue(book4 == addedBooks.get(2));`
	`test : 4 equals 4 ?`
	`true`
	`true`
	`true`
	`true`
	`true`
	`test : 3 equals 3 ?`
	`true`
	`true`

`//void testListAddedBooks_noAddedBooks() { GIVEN :-)`
`Library library = new Library();`
`List<Book> addedBooks = library.listAddedBooks();`
`assertTrue(addedBooks.isEmpty());`
	`true`

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1  //Definissez bien votre entête de classe vous n'aurez qu'une seule méthode à implémenter.
2
3  public class Library extends ItemDictionnary<Book> {
4
5
6      public Optional<Book> mergeBooks(Book book) {
7          if(book == null)
8              return Optional.empty();
9          List<Book> books = this.findItems(book);
10         if(books == null)
11             return Optional.of(book);
12
13         Book res = book;
14         Optional<Book> merged = Optional.empty();
15         for(Book b : books){
16             merged = res.merge(b);
17             if(merged.isPresent()){
18                 res = merged.get();
19             }
20         }
21         return merged;
22     }
23
24     public List<Book> sortByTitle() {
25         List<Book> res = new ArrayList<>();
26         for(List<Book> list : this.values()){
27             res.addAll(list);
28         }
29         res.sort(Comparator.comparing(Book::getTitle));
30         return res;
31     }
32
33
34     public List<Book> sortByAuthor() {
35         List<Book> res = new ArrayList<>();
36         for(List<Book> list : this.values()){
37             res.addAll(list);
38         }
39         res.sort(Comparator.comparing(Book::getAuthor));
40         return res;
41     }
```

```
42
43      public List<Book> listAddedBooks() {
44          List<Book> res = new ArrayList<>();
45          for(List<Book> list : this.values()){
46              for(Book book : list){
47                  res.add(book);
48              }
49          }
50          res.sort((b1, b2) -> b1.getYear()-(b2.getYear()));
51          return res;
52      }
```

	Test	Résultat attendu	Résultat obtenu	

✔	`// public void testListAddedBooks() {`
	`Library library = new Library();`
	
	`// On crée deux livres qui ont le même ISBN`
	`Book book1 = new Book("Titre 1", "Auteur 1", 2000, "ISBN 1");`
	`Book book2 = new Book("Titre 2", "Auteur 2", 2002, "ISBN 2");`
	`Book book3 = new Book("Titre 3", "Auteur 3", 2004, "ISBN 3");`
	`Book book4 = new Book("Titre 4", "Auteur 4", 2006, "ISBN 1");`
	
	`// On ajoute les livres à la bibliothèque`
	`library.addItem(book1);`
	`library.addItem(book2);`
	`library.addItem(book3);`
	`library.addItem(book4);`
	
	`// On vérifie que la méthode listAddedBooks retourne bien les deux premiers livres`
	`List<Book> addedBooks = library.listAddedBooks();`
	`assertEquals(4, addedBooks.size());`
	`assertTrue(addedBooks.contains(book1));`
	`assertTrue(addedBooks.contains(book2));`
	`//On verifie l'ordre`
	`assertTrue(book1 == addedBooks.get(0));`
	`assertTrue(book4 == addedBooks.get(3));`
	`// On retire un livre de la bibliothèque`
	`library.removeItem(book1);`
	
	`// On vérifie que la méthode listAddedBooks ne retourne plus le livre qui a été retiré`
	`addedBooks = library.listAddedBooks();`
	`assertEquals(3, addedBooks.size());`
	`assertTrue(book4 == addedBooks.get(2));`
		`test : 4 equals 4 ?`
		`true`
		`true`
		`true`
		`true`
		`true`
		`test : 3 equals 3 ?`
		`true`
		`true`
			`test : 4 equals 4 ?`
			`true`
			`true`
			`true`
			`true`
			`true`
			`test : 3 equals 3 ?`
			`true`
			`true`
				✔

✔	`//void testListAddedBooks_noAddedBooks() { GIVEN :-)`
	`Library library = new Library();`
	`List<Book> addedBooks = library.listAddedBooks();`
	`assertTrue(addedBooks.isEmpty());`
		`true`	`true`	✔

✔	`// void testListAddedBooks_oneAddedBook_stillPresent() {`
	`Library library = new Library();`
	`Book book = new Book("Title", "Author", 2000, "ISBN");`
	`library.addItem(book);`
	`List<Book> addedBooks = library.listAddedBooks();`
	`assertEquals(1, addedBooks.size());`
	`assertEquals(book, addedBooks.get(0));`
		`test : 1 equals 1 ?`
		`true`
		`test : Book{title='Title', author='Author', year=2000, isbn='ISBN'} equals Book{title='Title', author='Author', year=2000, isbn='ISBN'} ?`
		`true`
			`test : 1 equals 1 ?`
			`true`
			`test : Book{title='Title', author='Author', year=2000, isbn='ISBN'} equals Book{title='Title', author='Author', year=2000, isbn='ISBN'} ?`
			`true`
				✔

✔	`// void testListAddedBooks_oneAddedBook_removed() {`
	`Library library = new Library();`
	`Book book = new Book("Title", "Author", 2000, "ISBN");`
	`library.addItem(book);`
	`library.removeItem(book);`
	`List<Book> addedBooks = library.listAddedBooks();`
	`assertEquals(0, addedBooks.size());`
		`test : 0 equals 0 ?`
		`true`
			`test : 0 equals 0 ?`
			`true`
				✔

Tous les tests ont été réussis ! ✔

▸ **Solution de l'auteur de la question (Java)**

Correct

Note pour cet envoi : 5,00/5,00.

Question 10

Terminé

Note de 1,00 sur 1,00

En java, le mot clé this dans une classe désigne :

In java, the keyword this in a class means :

Veuillez choisir au moins une réponse.

- A. l'instance courante de la classe.
  the current instance of the class.
- B. un attribut de la classe
  an attribute of the class
- C. une méthode de la classe Object.
  a method of the Object class.
- D. un attribut de la classe Object
  an attribute of the Object class

Votre réponse est correcte.

La réponse correcte est : l'instance courante de la classe.
the current instance of the class.

Question 11

Terminé

Note de 2,00 sur 2,00

Given the classes

```
public class Artefact {
}

public abstract class Shape extends Artefact {
    public abstract void draw();
}

public class Cercle extends Shape {
    @Override
    public void draw() {
        System.out.println("I draw this Circle");
    }
}
```

which of the following expressions are **legal** (they compile and execute):

lesquelles des expressions suivantes sont ***légales*** (elles compilent et s'exécutent) :

Veuillez choisir au moins une réponse.

- [x] a. Artefact aa = new Artefact(); ok
- [x] b. Artefact ac = new Cercle(); ok
- [x] c. Shape s = new Cercle(); An abstract class is still a type.
- [x] d. Cercle c = new Cercle(); ok
- [x] e. c.draw();
- [x] f. s.draw();
- [ ] g. aa.draw();
- [ ] h. ac.draw();

Les réponses correctes sont : Artefact aa = new Artefact();, Artefact ac = new Cercle();, Shape s = new Cercle();, Cercle c = new Cercle();,
c.draw();,

s.draw();

Question 12

Terminé

Note de 1,00 sur 1,00

Le concept de la boite noire qui permet l'utilisation d'un objet sans savoir exactement comment les éléments sont traités s'appelle

The concept of the black box, which allows the use of an object without knowing exactly how the elements are processed is called

Veuillez choisir au moins une réponse.

- [ ] A. l'Entropie/Entropy
- [ ] B. L'héritage / Inheritance
- [x] C. L'encapsulation/Encapsulation
- [ ] D. Le polymorphisme/polymorphisme

Votre réponse est correcte.

La réponse correcte est : L'encapsulation/Encapsulation

Question 13

Terminé

Note de 2,00 sur 2,00

Assume we have four classes: `Person`, `Teacher`, `Student`, and `PhDStudent`.
`Teacher` and `Student` are both subclasses of `Person`.
`PhDStudent` is a subclass of `Student`.

Which of the following statements are legal, and why (use only pencil and paper)?

--- Version française

Supposons que nous ayons quatre classes : *Person*, Teacher, *Student*, and PhDStudent.

Teacher and *Student* sont tous deux des sous-classes de *Person*.

PhDStudent est une sous-classe de *Student*.

Lesquels des énoncés suivants sont légaux (compilent) ?

Traduction de l'énoncé en diagramme de classes

Person p1 = new Student();	legal
Person p2 = new PhDStudent();	legal
PhDStudent phd1 = new Student();	not legal
Teacher t1 = new Person();	not legal
Student s1 = new PhDStudent();	legal
s1 = p1;	not legal
s1 = p2;	not legal
p1 = s1;	legal
t1 = s1;	not legal
s1 = phd1;	legal
phd1 = s1;	not legal
Object o = new Student();	legal
s1 = o;	not legal

La réponse correcte est : Person p1 = new Student(); → legal, Person p2 = new PhDStudent(); → legal, PhDStudent phd1 = new Student(); → not legal, Teacher t1 = new Person(); → not legal, Student s1 = new PhDStudent(); → legal, s1 = p1; → not legal, s1 = p2; → not legal, p1 = s1; → legal, t1 = s1; → not legal, s1 = phd1; → legal, phd1 = s1; → not legal, Object o = new Student(); → legal, s1 = o; → not legal

Question 14

Terminé

Note de 2,63 sur 3,00

Soit la classe suivante, identifiez les différents éléments qui la composent :

Given the following class, identify the different elements that make it up:

```
public class GuessingGame extends Game implements Playable{
    private int numberToGuess;
    private GuessingGameIO io;

    public GuessingGameIO getIo() {
        return io;
    }
    public GuessingGame(GuessingGameIO io) {
        // Generate the number to guess
        Random random = new Random();
        numberToGuess = random.nextInt(100) + 1;
        this.io = io;
    }
    @Override
    public void playRound() {
        boolean guessCorrect = false;
        while (!guessCorrect) {
            // Ask for the guess
            int guess = io.askForGuess();

            // Check the guess
            if (guess == numberToGuess) {
                io.displayMessage("Congratulations, you guessed the correct number!");
                guessCorrect = true;
            } else if (guess < numberToGuess) {
                io.displayMessage("The number to guess is larger.");
            } else {
                io.displayMessage("The number to guess is smaller.");
            }
        }
    }
}
```

boolean guessCorrect	a local variable
Playable	an interface
random.nextInt(100);	a method
int guess	a local variable
io.askForGuess();	a method
Game	a class
Random random	a local variable
public void playRound()	a method
"The number to guess is smaller."	a String
Random	a class
private int numberToGuess;	a field/Instance variable/attribute
GuessingGame	a class
public GuessingGameIO getIo()	an accessor
io.displayMessage("Congratulations, you guessed the correct number!");	a message sending
public GuessingGame(GuessingGameIO io)	a constructor
private GuessingGameIO io;	a field/Instance variable/attribute

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 14.
La réponse correcte est :
boolean guessCorrect → a local variable,

Playable → an interface,

random.nextInt(100); → a message sending, int guess → a local variable,

io.askForGuess(); → a message sending,

Game → a class, Random random → a local variable, public void playRound() → a method, "The number to guess is smaller." → a String,

Random → a class, private int numberToGuess; → a field/Instance variable/attribute,

GuessingGame → a class,

public GuessingGameIO getIo() → an accessor, io.displayMessage("Congratulations, you guessed the correct number!"); → a message sending, public GuessingGame(GuessingGameIO io) → a constructor, private GuessingGameIO io; → a field/Instance variable/attribute

Question 15

Terminé

Note de 1,00 sur 1,00

Une classe écrite en java est soumise à la (aux) régle(s) suivante(s) :

A class written in java is subject to the following rule(s):

Veuillez choisir au moins une réponse.

- A. Son code source doit disposer d'au moins un attribut.
  Its source code must have at least one attribute.
- B. Son code source doit disposer d'une méthode *main*.
  Its source code must have a *main* method.
- C. Le fichier source porte le même nom que celui de la classe, i.e. si la classe est A, le fichier correspondant a pour nom A.java
  The source file has the same name as the class, i.e. if the class is A, the corresponding file has the name A.java

Votre réponse est correcte.

La réponse correcte est : Le fichier source porte le même nom que celui de la classe, i.e. si la classe est A, le fichier correspondant a pour nom A.java
The source file has the same name as the class, i.e. if the class is A, the corresponding file has the name A.java