Bases de la programmation Java

Syntaxe Java de base : les variables, les types de données, les opérateurs et les tableaux

Variables et syntaxe de base

https://fre.myservername.com/java-basics-java-syntax

https://www.jmdoudoux.fr/java/dej/chap-syntaxe.htm

https://gayerie.dev/epsi-b3-java/langage_java/types_primitifs.html

https://gayerie.dev/epsi-b3-java/langage_java/la_classe_string.html

Opérateurs

https://koor.fr/Java/Tutorial/JavaOperators.wp

https://gayerie.dev/epsi-b3-java/langage_java/operateurs.html

Structures de contrôles

https://gayerie.dev/epsi-b3-java/langage_java/structures_de_controle.html#les-structures-de-controle

Tableaux

https://koor.fr/Java/Tutorial/java_tableaux.wp

https://gayerie.dev/epsi-b3-java/langage_java/tableau.html

Les concepts de la programmation orientée objet : classes, objets, encapsulation, héritage et polymorphisme

Classes

https://gayerie.dev/epsi-b3-java/langage_java/premiere_classe.html

https://koor.fr/Java/Tutorial/java_poo_encapsulation.wp

Attributs et méthodes

https://gayerie.dev/epsi-b3-java/langage_java/attributs_et_methodes.html

https://koor.fr/Java/Tutorial/java_poo_encapsulation.wp

Héritage & Implémentation

https://gayerie.dev/epsi-b3-java/langage_java/heritage_composition.html

https://koor.fr/Java/Tutorial/java_poo_heritage.wp

https://gayerie.dev/epsi-b3-java/langage_java/interface.html

https://koor.fr/Java/Tutorial/java_poo_type_abstrait.wp#interface

https://koor.fr/Java/Tutorial/java_poo_interface.wp

Polymorphisme

https://gayerie.dev/epsi-b3-java/langage_java/polymorphisme.html

Collections

https://gayerie.dev/epsi-b3-java/langage_java/les_collections.html

https://gayerie.dev/epsi-b3-java/langage_java/streams.html

Enumérations

https://koor.fr/Java/Tutorial/java_enum.wp

https://gayerie.dev/epsi-b3-java/langage_java/enumeration.html

TP bases de la programmation Java

Dans ce TP, vous allez développer un système de gestion pour une bibliothèque. Nous allons créer plusieurs classes et une interface pour modéliser le fonctionnement de la bibliothèque.

Partie 1 : Création de la classe `Publisher`

1. Dans un nouveau fichier `Publisher.java`, créez une classe `Publisher` qui représente un éditeur de livres. Cette classe doit avoir les attributs privés suivants :
   - `name` : le nom de l'éditeur (String)
   - `address` : l'adresse de l'éditeur (String)
2. Ajoutez un constructeur à la classe `Publisher` qui prend en entrée un nom et une adresse, et initialise les attributs correspondants.
3. Ajoutez des méthodes getters et setters pour les attributs de la classe `Publisher`.

Partie 2 : Création de la classe `Book`

Un livre doit avoir les attributs privés suivants :

- `title` : Le nom du livre
- `author` : L'auteur du livre
- `numberOfPages` : Le nombre de pages

1. Créez votre classe `Book`, y inclure un nouvel attribut privé `publisher` de type `Publisher`.
2. Modifiez le constructeur de `Book` pour prendre en compte le nouvel attribut `publisher`.
3. Ajoutez une méthode getter et une méthode setter pour l'attribut `publisher`.

Partie 3 : Création de l'interface `Loanable`

1. Créez une nouvelle interface `Loanable`. Cette interface doit définir les méthodes suivantes :
   - `loanTo(String borrower)` : Cette méthode ne retourne rien et modifie l'état de l'objet pour indiquer qu'il est emprunté par l'emprunteur passé en paramètre.
   - `returnBook()` : Cette méthode ne retourne rien et modifie l'état de l'objet pour indiquer qu'il n'est plus emprunté.

Partie 4 : Modification de la classe `Book` pour implémenter `Loanable`

1. Modifiez la classe `Book` pour qu'elle implémente l'interface `Loanable`.
2. Ajoutez un nouvel attribut privé `borrower` de type `String` à la classe `Book`. Cette variable doit être `null` quand le livre n'est pas emprunté.
3. Implémentez les méthodes `loanTo` et `returnBook` pour modifier l'attribut `borrower`.

Partie 5 : Création de la classe `Library`

1. Dans un nouveau fichier `Library.java`, créez une classe `Library` qui représente une bibliothèque. Cette classe doit avoir un seul attribut privé `books`, qui est une liste de `Book` .
2. Ajoutez un constructeur sans arguments à la classe `Library` qui initialise l'attribut `books` comme une nouvelle `ArrayList`.
3. Ajoutez une méthode `addBook` à la classe `Library` qui prend un `Book` en argument et l'ajoute à la liste `books`.
4. Ajoutez une méthode `searchBook` à la classe `Library` qui prend un titre de livre en argument et renvoie le `Book` correspondant s'il est dans la liste, et `null` sinon.
5. Ajoutez une méthode `loanBook` à la classe `Library` qui prend un titre de livre et un emprunteur en argument, et emprunte le livre correspondant s'il est dans la liste et n'est pas déjà emprunté. Cette méthode doit lancer une exception si le livre n'est pas disponible.

Partie 6 : Création de la classe `LibraryApp`

1. Enfin, dans un nouveau fichier `LibraryApp.java`, créez une classe `LibraryApp` avec une méthode `main`. Dans cette méthode, créez plusieurs objets `Book`, ajoutez-les à un objet `Library`, et testez la recherche et l'emprunt de livres.

Arborescence de fichiers

Voici une possible arborescence de fichiers pour ce TP :

```
src/
├── main/
│   ├── java/
│   │   ├── LibraryApp.java
```

```
│   │   ├── Book.java
│   │   ├── Publisher.java
│   │   ├── Loanable.java
│   │   ├── Library.java
```

Code complet

Voici une implémentation possible de ce TP, en supposant que chaque fichier se trouve dans le package `com.example.library` :

Publisher.java

```
package com.example.library;

public class Publisher {
    private String name;
    private String address;

    public Publisher(String name, String address) {
        this.name = name;
        this.address = address;
    }

    // Getters and setters
    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return this.address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
```

```
}
```

Loanable.java

```
package com.example.library;

public interface Loanable {
    void loanTo(String borrower);
    void returnBook();
}
```

Book.java

```
package com.example.library;

public class Book implements Loanable {
    private String title;
    private String author;
    private int numberOfPages;
    private Publisher publisher;
    private String borrower;

    public Book(String title, String author, int numberOfPages, Publisher publisher) {
        this.title = title;
        this.author = author;
        this.numberOfPages = numberOfPages;
        this.publisher = publisher;
        this.borrower = null;
    }

    // Getters and setters
    public String getTitle() {
        return this.title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getAuthor() {
```

```
        return this.author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public int getNumberOfPages() {
        return this.numberOfPages;
    }

    public void setNumberOfPages(int numberOfPages) {
        this.numberOfPages = numberOfPages;
    }

    public Publisher getPublisher() {
        return this.publisher;
    }

    public void setPublisher(Publisher publisher) {
        this.publisher = publisher;
    }

    public String getBorrower() {
        return this.borrower;
    }

    @Override
    public void loanTo(String borrower) {
        if (this.borrower != null) {
            throw new IllegalStateException("Book is already borrowed.");
        }
        this.borrower = borrower;
    }

    @Override
    public void returnBook() {
        this.borrower = null;
    }
}
```

Library.java

```
package com.example.library;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.Optional;

public class Library {
    private List<Book> books;

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

    public void addBook(Book book) {
        books.add(book);
    }

    public Book searchBook(String title) {
        Optional<Book> foundBook = books.stream()
                .filter(book -> Objects.equals(book.getTitle(), title))
                .findFirst();
        return foundBook.orElse(null);
    }

    public void loanBook(String title, String borrower) {
        Book book = searchBook(title);
        if (book == null) {
            throw new IllegalArgumentException("Book not found.");
        }
        book.loanTo(borrower);
    }
}
```

LibraryApp.java

```
package com.example.library;
```

```
public class LibraryApp {
    public static void main(String[] args) {
        // Create Publisher
        Publisher penguin = new Publisher("Penguin", "New York");

        // Create Books
        Book book1 = new Book("1984", "George Orwell", 328, penguin);
        Book book2 = new Book("To Kill a Mockingbird", "Harper Lee", 281, penguin);

        // Create Library
        Library library = new Library();

        // Add books to library
        library.addBook(book1);
        library.addBook(book2);

        // Test searching and loaning books
        System.out.println("Searching for '1984': " + library.searchBook("1984"));
        library.loanBook("1984", "John Doe");
        System.out.println("'1984' borrowed by: " + book1.getBorrower());
    }
}
```