Java & Spring Boot

- Mise en place de l'environnement de développement
  - Installer Java
  - Installer IntelliJ (Standalone)
  - Vérifier que les outils de compilation et d'exécution sont correctement configurés
- Bases de la programmation Java
  - Syntaxe Java de base : les variables, les types de données, les opérateurs et les tableaux
  - Les concepts de la programmation orientée objet : classes, objets, encapsulation, héritage et polymorphisme
  - Collections
  - Enumérations
  - TP bases de la programmation Java
- Les exceptions
- Gestion des Tests
  - Pourquoi tester
  - Les bonnes pratiques
- Les designs patterns
- Spring Boot
  - Découverte
  - Gestion de la persistance avec JPA et Hibernate
  - Clean architecture
  - TP Spring Boot
- Compilation et distribution
  - Spring Boot

- Maven

Mise en place de l'environnement de développement

Installer Java

Une des manières la plus simple d'installer java sur une distribution **Debian**, serait d'installer une version `openjdk`, car cette dernière est déjà présente dans les dépôts de base.

Mettre à jour la liste des dépendances

```
sudo apt-get update && sudo apt-get upgrade
```

Installer OpenJDK 11 development kit (qui contient à la fois les outils pour compiler, que pour lancer une application java)

```
sudo apt-get install openjdk-11-jdk
```

Tester la version de java installé

```
java -version
```

Sources

https://www.linode.com/docs/guides/how-to-install-openjdk-on-debian-10/

Installer IntelliJ (Standalone)

IntelliJ est surement l'un des meilleurs IDE du marché pour le développement Java.

On peut installer la version **Ultimate**, si l'on possède la licence, ou bien la version **Community**, plus limité, mais gratuite.

Télécharger IntelliJ

https://www.jetbrains.com/idea/download/#section=linux

Installer IntelliJ Ultimate

```
sudo tar -xzf ideaIU-*.tar.gz -C /opt
```

Installer IntelliJ Community

```
sudo tar -xzf ideaIC-*.tar.gz -C /opt
```

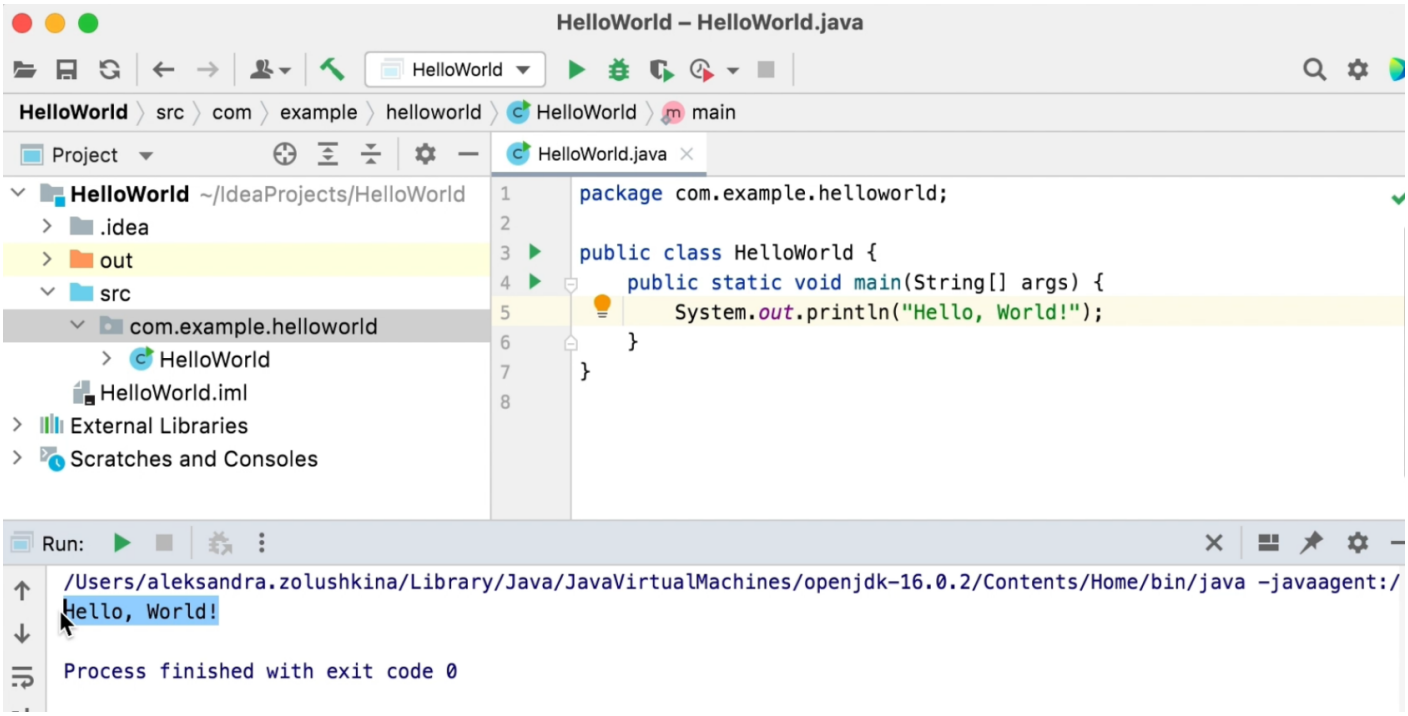
Ensuite, exécuter le fichier `idea.sh` se trouvant dans le chemin `/opt/idea-*/bin/idea.sh`

Sources

https://www.jetbrains.com/help/idea/installation-guide.html#standalone

Vérifier que les outils de compilation et d'exécution sont correctement configurés

Pour ce faire, il suffira de créer un projet Java tout simple dans IntelliJ, pour vérifier que vous obtenez bien le petit message **Hello** World dans la console !



Sources

https://www.jetbrains.com/help/idea/creating-and-running-your-first-java-application.html#get-started

Premier projet

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

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

Bases de la programmation Java

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());
    }
}
```

Les exceptions

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

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

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

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

Gestion des Tests

Pourquoi tester

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

Les bonnes pratiques

https://openclassrooms.com/fr/courses/6100311-testez-votre-code-java-pour-realiser-des-applications-de-qualite/exercises/3699

https://openclassrooms.com/fr/courses/6100311-testez-votre-code-java-pour-realiser-des-applications-de-qualite/6465561-structurez-vos-tests-unitaires-avec-les-annotations-junit

https://zestedesavoir.com/tutoriels/274/les-tests-unitaires-en-java/

https://www.baeldung.com/java-unit-testing-best-practices

Les designs patterns

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

Meilleur site pour les design patterns : https://refactoring.guru/design-patterns

Spring Boot

Spring Boot

Découverte

https://devstory.net/11267/tutoriel-spring-boot-pour-debutant

https://bnguimgo.developpez.com/tutoriels/spring/services-rest-avec-springboot-et-spring-resttemplate/?page=premiere-partie-le-serveur

https://www.youtube.com/playlist?list=PLtyD11a_24egpX6V-BXtVnBmrbR60I1P6

Spring Boot

Gestion de la persistance avec JPA et Hibernate

https://gayerie.dev/docs/spring/spring/spring_data_jpa.html

https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa

https://www.baeldung.com/spring-boot-hibernate

https://www.javadevjournal.com/spring-boot/spring-boot-with-hibernate/

Spring Boot

Clean architecture

https://www.baeldung.com/spring-boot-clean-architecture

https://medium.com/swlh/clean-architecture-java-spring-fea51e26e00

https://reflectoring.io/java-components-clean-boundaries/

TP Spring Boot

Dans ce TP, vous allez développer un système de gestion pour une bibliothèque en utilisant Spring Boot et Spring Data JPA. Nous allons créer plusieurs classes, des interfaces pour les dépôts de données et des contrôleurs pour exposer nos services via une API REST.

Partie 1: Initialisation du projet Spring Boot

1. Allez sur Spring Initializr pour générer un nouveau projet Spring Boot. Choisissez "Maven Project", "Java", et la dernière version de Spring Boot.
2. Dans la section "Project Metadata", donnez un nom de groupe et d'artefact approprié, par exemple "com.example" et "library".
3. Dans la section "Dependencies", ajoutez les dépendances suivantes: Spring Web, Spring Data JPA, et Spring Boot DevTools.
4. Cliquez sur "Generate" pour télécharger un fichier zip du projet. Décompressez le fichier et ouvrez le projet dans votre IDE favori.

Partie 2: Création de la classe `Publisher`

1. Dans le package `model`, créez une classe `Publisher` qui représente un éditeur de livres. Cette classe doit avoir les attributs suivants :
   - `id` : l'ID de l'éditeur (Long)
   - `name` : le nom de l'éditeur (String)
   - `address` : l'adresse de l'éditeur (String)
2. Annoter la classe avec `@Entity` pour indiquer qu'il s'agit d'une entité JPA.
3. Annoter l'attribut `id` avec `@Id` et `@GeneratedValue` pour indiquer qu'il s'agit de la clé primaire et qu'elle est générée automatiquement.

Partie 3: Création de l'interface `PublisherRepository`

1. Dans le package `repository`, créez une interface `PublisherRepository` qui étend `JpaRepository<Publisher, Long>`. Cela nous donne gratuitement plusieurs méthodes pour interagir avec la base de données comme `findAll()`, `findById()`, `save()`, `delete()`, etc.

Partie 4: Création de la classe `Book`

1. Dans le package `model`, créez une classe `Book` qui représente un livre. Cette classe doit avoir les attributs suivants :
   - `id` : l'ID du livre (Long)
   - `title` : le titre du livre (String)
   - `author` : l'auteur du livre (String)
   - `publisher` : l'éditeur du livre (`Publisher`)
2. Annoter la classe avec `@Entity` pour indiquer qu'il s'agit d'une entité JPA.
3. Annoter l'attribut `id` avec `@Id` et `@GeneratedValue` pour indiquer qu'il s'agit de la clé primaire et qu'elle est générée automatiquement.
4. Annoter l'attribut `publisher` avec `@ManyToOne` pour indiquer la relation entre `Book` et `Publisher`.

Partie 5: Création de l'interface `BookRepository`

1. Dans le package `repository`, créez une interface `BookRepository` qui étend `JpaRepository<Book, Long>`.

Partie 6: Création des contrôleurs

1. Dans le package `controller`, créez une classe `PublisherController` avec une méthode `getAllPublishers()` qui retourne tous les éditeurs, et une méthode `addPublisher()` qui ajoute un nouvel éditeur.
2. Dans le même package, créez une classe `BookController` avec une méthode `getAllBooks()` qui retourne tous les livres, et une méthode `addBook()` qui ajoute un nouveau livre.

Partie 7: Test de l'application

1. Exécutez l'application et utilisez un outil comme Postman pour tester votre API REST.

Arborescence de fichiers

Voici une possible arboressence de fichiers pour ce TP :

```
src/
├── main/
│   ├── java/
│   │   ├── com.example.library
│   │   │   ├── LibraryApplication.java
│   │   │   ├── model/
│   │   │   │   ├── Book.java
```

```
│   │   │   │   ├── Publisher.java
│   │   │   ├── repository/
│   │   │   │   ├── BookRepository.java
│   │   │   │   ├── PublisherRepository.java
│   │   │   ├── controller/
│   │   │   │   ├── BookController.java
│   │   │   │   ├── PublisherController.java
│   ├── resources/
│   │   ├── application.properties
```

Code

Model

Book

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

import javax.persistence.*;

@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String author;

    @ManyToOne
    private Publisher publisher;

    public Book() {
    }

    public Book(String title, String author, Publisher publisher) {
        this.title = title;
        this.author = author;
        this.publisher = publisher;
    }
```

```
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

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

    public String getAuthor() {
        return author;
    }

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

    public Publisher getPublisher() {
        return publisher;
    }

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

Publisher

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

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;

@Entity
public class Publisher {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String address;

    public Publisher() {
    }

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

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

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

    public String getAddress() {
        return address;
    }

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

```
    }
}
```

Repository

BookRepository

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

import com.example.library.model.Book;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {
}
```

PublisherRepository

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

import com.example.library.model.Publisher;
import org.springframework.data.jpa.repository.JpaRepository;

public interface PublisherRepository extends JpaRepository<Publisher, Long> {
}
```

Controller

BookController

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

import com.example.library.model.Book;
import com.example.library.repository.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
public class BookController {
    @Autowired
```

```
    private BookRepository bookRepository;

    @GetMapping("/books")
    public List<Book> getAllBooks() {
        return bookRepository.findAll();
    }

    @PostMapping("/books")
    public Book addBook(@RequestBody Book book) {
        return bookRepository.save(book);
    }
}
```

PublisherController

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

import com.example.library.model.Publisher;
import com.example.library.repository.PublisherRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
public class PublisherController {
    @Autowired
    private PublisherRepository publisherRepository;

    @GetMapping("/publishers")
    public List<Publisher> getAllPublishers() {
        return publisherRepository.findAll();
    }

    @PostMapping("/publishers")
    public Publisher addPublisher(@RequestBody Publisher publisher) {
        return publisherRepository.save(publisher);
    }
}
```

Compilation et distribution

Compilation et distribution

Spring Boot

https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/

https://www.baeldung.com/spring-boot-run-maven-vs-executable-jar

https://www.yawintutor.com/how-to-create-executable-jar-file-in-spring-boot-using-maven/

Compilation et distribution

Maven

https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

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

https://www.jetbrains.com/help/idea/maven-support.html

https://www.baeldung.com/executable-jar-with-maven