

Spring Data

What is Spring Data?

- ◆ **Spring Data** is an umbrella project, not a single component. It's a collection of modules, each focusing on a different aspect of data access or a specific type of data store.
- ◆ **Features**
 - Powerful repository and custom object-mapping abstractions
 - Dynamic query derivation from repository method names
 - Implementation domain base classes providing basic properties
 - Support for transparent auditing (created, last changed)
 - Possibility to integrate custom repository code
 - Easy Spring integration via JavaConfig and custom XML namespaces
 - Advanced integration with Spring MVC controllers
 - Experimental support for cross-store persistence

Main modules

- [Spring Data Commons](#) - Core Spring concepts underpinning every Spring Data module.
- [Spring Data JDBC](#) - Spring Data repository support for JDBC.
- [Spring Data R2DBC](#) - Spring Data repository support for R2DBC.
- [Spring Data JPA](#) - Spring Data repository support for JPA.
- [Spring Data KeyValue](#) - Map based repositories and SPIs to easily build a Spring Data module for key-value stores.
- [Spring Data LDAP](#) - Spring Data repository support for [Spring LDAP](#).
- [Spring Data MongoDB](#) - Spring based, object-document support and repositories for MongoDB.
- [Spring Data Redis](#) - Easy configuration and access to Redis from Spring applications.
- [Spring Data REST](#) - Exports Spring Data repositories as hypermedia-driven RESTful resources.
- [Spring Data for Apache Cassandra](#) - Easy configuration and access to Apache Cassandra or large scale, highly available, data oriented Spring applications.
- [Spring Data for Apache Geode](#) - Easy configuration and access to Apache Geode for highly consistent, low latency, data oriented Spring applications.

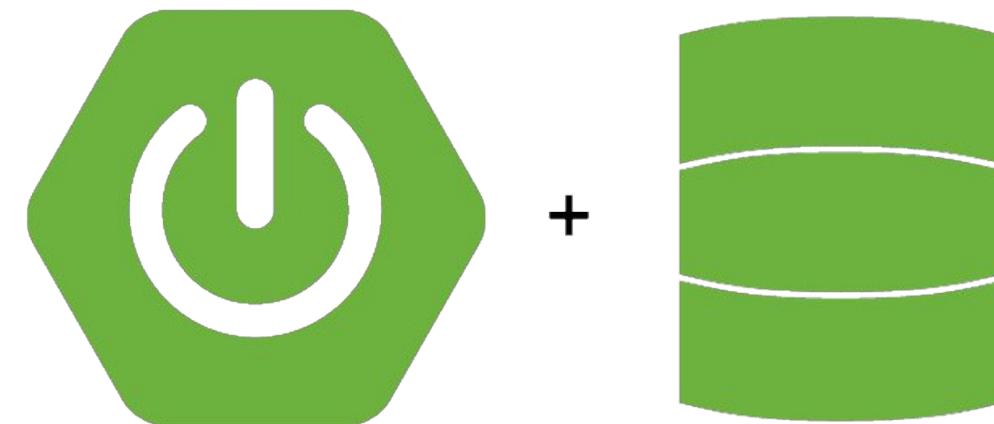
Spring Data JPA

Objectives

- ❖ What is Spring Data?
- ❖ What is Spring Data JPA?
- ❖ Core Concepts
- ❖ Repository Interface
- ❖ Query Methods
- ❖ JPQL and Native Queries
- ❖ Spring Data JPA with Spring Boot
- ❖ Benefits and Best Practices

What is Spring Data JPA ?

- Spring Data JPA is a part of the Spring Framework that simplifies the development of data access layers for Java applications, specifically those using JPA (Java Persistence API). It provides a higher-level abstraction over JPA, allowing developers to interact with databases using simpler and more expressive methods.



Key Features of Spring Data JPA

- ❖ **Repository Abstraction:** It introduces the Repository interface, which defines common data access methods like save, find, delete, etc. Spring Data JPA automatically generates implementations for these methods based on method names and domain class conventions.
- ❖ **Spring Data JPA Repositories:** You create custom repositories by extending the Repository interface and defining custom query methods. Spring Data JPA uses a naming convention to automatically generate query implementations.
- ❖ **Query Annotation:** You can use the @Query annotation to write custom JPQL or native SQL queries.
- ❖ **Specification API:** You can create dynamic query criteria using the Specification API, allowing for complex filtering and sorting.
- ❖ **Integration with Spring Framework:** It seamlessly integrates with other Spring components like Spring MVC, Spring Security, and Spring Boot.

Benefits of using Spring Data JPA

- ❖ **Reduced Boilerplate Code:** It significantly reduces the amount of code you need to write for data access operations.
- ❖ **Improved Productivity:** By simplifying data access, it increases developer productivity.
- ❖ **Enhanced Maintainability:** The code becomes more maintainable due to its simplicity and adherence to conventions.
- ❖ **Flexibility:** It supports various database systems through JPA and provides options for custom query methods and specifications.
- ❖ **Strong Community and Support:** It benefits from the large Spring community and has extensive documentation and support resources.

Core Concepts of Spring Data JPA

Spring Data JPA Annotations

- ❖ **@Entity:** Marks a class as a JPA entity, indicating that it represents a database table.
- ❖ **@Table:** Specifies the name of the database table to which the entity is mapped.
- ❖ **@Id:** Marks a field as the primary key of the entity.
- ❖ **@GeneratedValue:** Specifies the strategy for generating primary key values (e.g., GenerationType.IDENTITY, GenerationType.AUTO, GenerationType.SEQUENCE, GenerationType.TABLE).
- ❖ **@Column:** Configures column-specific properties like name, length, nullable, and unique constraints.

Relationship Mapping Annotations

- ❖ **@OneToOne:** Defines a one-to-one relationship between two entities.
- ❖ **@OneToMany:** Defines a one-to-many relationship between two entities.
- ❖ **@ManyToOne:** Defines a many-to-one relationship between two entities.
- ❖ **@ManyToMany:** Defines a many-to-many relationship between two entities.
- ❖ **@JoinColumn:** Specifies the foreign key column in the database table

Query Annotations

- ❖ **@Query:** Defines a custom JPQL or native SQL query.
- ❖ **@Modifying:** Indicates that a query modifies data.
- ❖ **@Param:** Binds a parameter to a query.
- ❖ **Other Annotations**
- ❖ **@Temporal:** Specifies the temporal type of a field.
(e.g., *TemporalType.DATE*, *TemporalType.TIME*, *TemporalType.TIMESTAMP*)
- ❖ **@Enumerated:** Specifies the enumeration type for a field.
- ❖ **@Lob:** Maps large object types (e.g., CLOB, BLOB).
- ❖ **@Transient:** Excludes a field from persistence.
- ❖ **@Version:** Specifies a version field for optimistic locking.

JPA Query Methods

JPA Query

Keyword	Sample	JPQL snippet
Distinct	findDistinctByLastnameAndFirstname	select distinct ... where x.lastname = ?1 and x.firstname = ?2
And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Is , Equals	findByFirstname, findByFirstnameIs, findByFirstnameEquals	... where x.firstname = ?1 (or ... where x.firstname IS NULL if the argument is null)
Between	findByStartDateBetween	... where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	... where x.age <= ?1
Greater Than	findByAgeGreaterThan	... where x.age > ?1
Greater Than Equal	findByAgeGreaterThanEqual	... where x.age >= ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull , Null	findByAge(Is)Null	... where x.age is null
IsNotNull , NotNull	findByAge(Is)NotNull	... where x.age is not null

<https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html>

JPA Query

Like	<code>findByFirstnameLike</code>	<code>... where x.firstname like ?1</code>
NotLike	<code>findByFirstnameNotLike</code>	<code>... where x.firstname not like ?1</code>
StartingWith	<code>findByFirstnameStartingWith</code>	<code>... where x.firstname like ?1 (parameter bound with appended %)</code>
EndingWith	<code>findByFirstnameEndingWith</code>	<code>... where x.firstname like ?1 (parameter bound with prepended %)</code>
Containing	<code>findByFirstnameContaining</code>	<code>... where x.firstname like ?1 (parameter bound wrapped in %)</code>
OrderBy	<code>findByAgeOrderByLastnameDesc</code>	<code>... where x.age = ?1 order by x.lastname desc</code>
Not	<code>findByLastnameNot</code>	<code>... where x.lastname <> ?1</code>
In	<code>findByAgeIn(Collection<Age> ages)</code>	<code>... where x.age in ?1</code>
NotIn	<code>findByAgeNotIn(Collection<Age> ages)</code>	<code>... where x.age not in ?1</code>
True	<code>findByActiveTrue()</code>	<code>... where x.active = true</code>
False	<code>findByActiveFalse()</code>	<code>... where x.active = false</code>
IgnoreCase	<code>findByFirstnameIgnoreCase</code>	<code>... where UPPER(x.firstname) = UPPER(?1)</code>

<https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html>

Repository query keywords

Keyword	Description
<code>find...By</code> , <code>read...By</code> , <code>get...By</code> , <code>query...By</code> , <code>search...By</code> , <code>stream...By</code>	General query method returning typically the repository type, a <code>Collection</code> or <code>Streamable</code> subtype or a result wrapper such as <code>Page</code> , <code>GeoResults</code> or any other store-specific result wrapper. Can be used as <code>findBy...</code> , <code>findMyDomainTypeBy...</code> or in combination with additional keywords.
<code>exists...By</code>	Exists projection, returning typically a <code>boolean</code> result.
<code>count...By</code>	Count projection returning a numeric result.
<code>delete...By</code> , <code>remove...By</code>	Delete query method returning either no result (<code>void</code>) or the delete count.
<code>...First<number>...</code> , <code>...Top<number>...</code>	Limit the query results to the first <code><number></code> of results. This keyword can occur in any place of the subject between <code>find</code> (and the other keywords) and <code>by</code> .
<code>...Distinct...</code>	Use a distinct query to return only unique results. Consult the store-specific documentation whether that feature is supported. This keyword can occur in any place of the subject between <code>find</code> (and the other keywords) and <code>by</code> .

<https://docs.spring.io/spring-data/jpa/reference/repositories/query-keywords-reference.html>

Reserved methods

```
deleteAllById(Iterable<ID> identifiers)
```

```
deleteById(ID identifier)
```

```
existsById(ID identifier)
```

```
findAllById(Iterable<ID> identifiers)
```

```
findById(ID identifier)
```

<https://docs.spring.io/spring-data/jpa/reference/repositories/query-keywords-reference.html>

Supported query method predicate keywords and modifiers

Logical keyword	Keyword expressions
AND	And
OR	Or
AFTER	After , IsAfter
BEFORE	Before , IsBefore
CONTAINING	Containing , IsContaining , Contains
BETWEEN	Between , IsBetween
ENDING_WITH	EndingWith , IsEndingWith , EndsWith
EXISTS	Exists
FALSE	False , IsFalse
GREATER_THAN	GreaterThan , IsGreaterThan
GREATER_THAN_EQUALS	GreaterThanEqual , IsGreaterThanEqual
IN	In , IsIn

<https://docs.spring.io/spring-data/jpa/reference/repositories/query-keywords-reference.html>

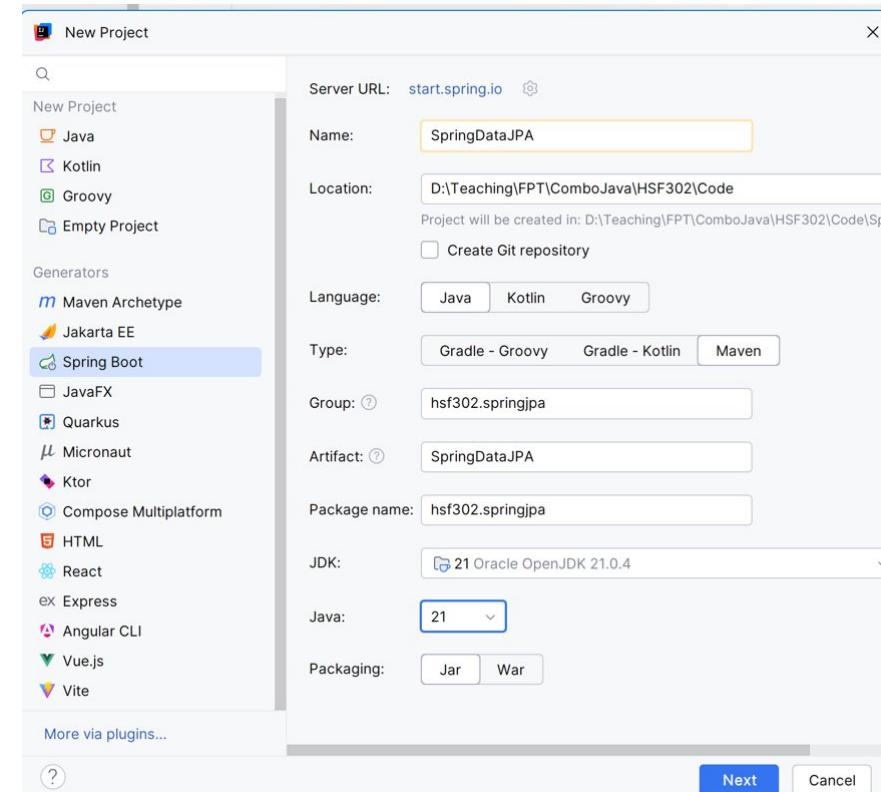
Supported query method predicate keywords and modifiers

IS	Is, Equals, (or no keyword)
IS_EMPTY	IsEmpty, Empty
IS_NOT_EMPTY	IsNotEmpty, NotEmpty
IS_NOT_NULL	NotNull, IsNotNull
IS_NULL	Null, IsNull
LESS_THAN	LessThan, IsLessThan
LESS_THAN_EQUAL	LessThanEqual, IsLessThanEqual
LIKE	Like, IsLike
NEAR	Near, IsNear
NOT	Not, IsNot
NOT_IN	NotIn, IsNotIn
NOT_LIKE	NotLike, IsNotLike
REGEX	Regex, MatchesRegex, Matches
STARTING_WITH	StartingWith, IsStartingWith, StartsWith
TRUE	True, IsTrue
WITHIN	Within, IsWithin

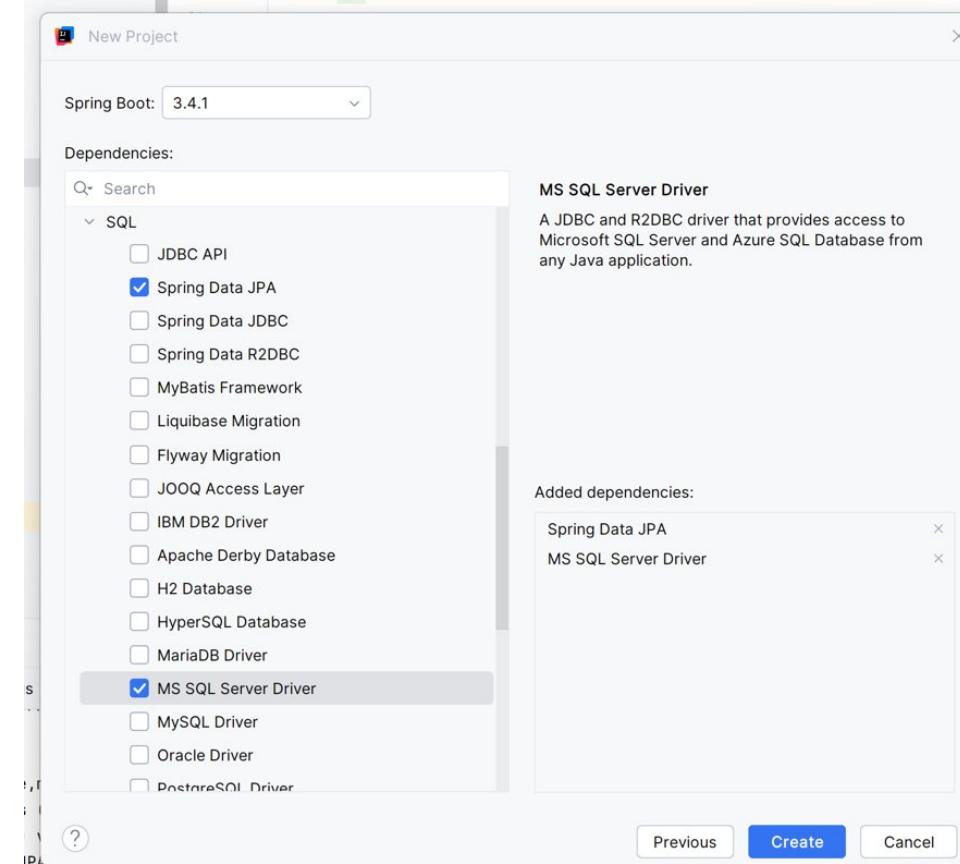
<https://docs.spring.io/spring-data/jpa/reference/repositories/query-keywords-reference.html>

Spring Data JPA Programming Demo (ManyToMany)

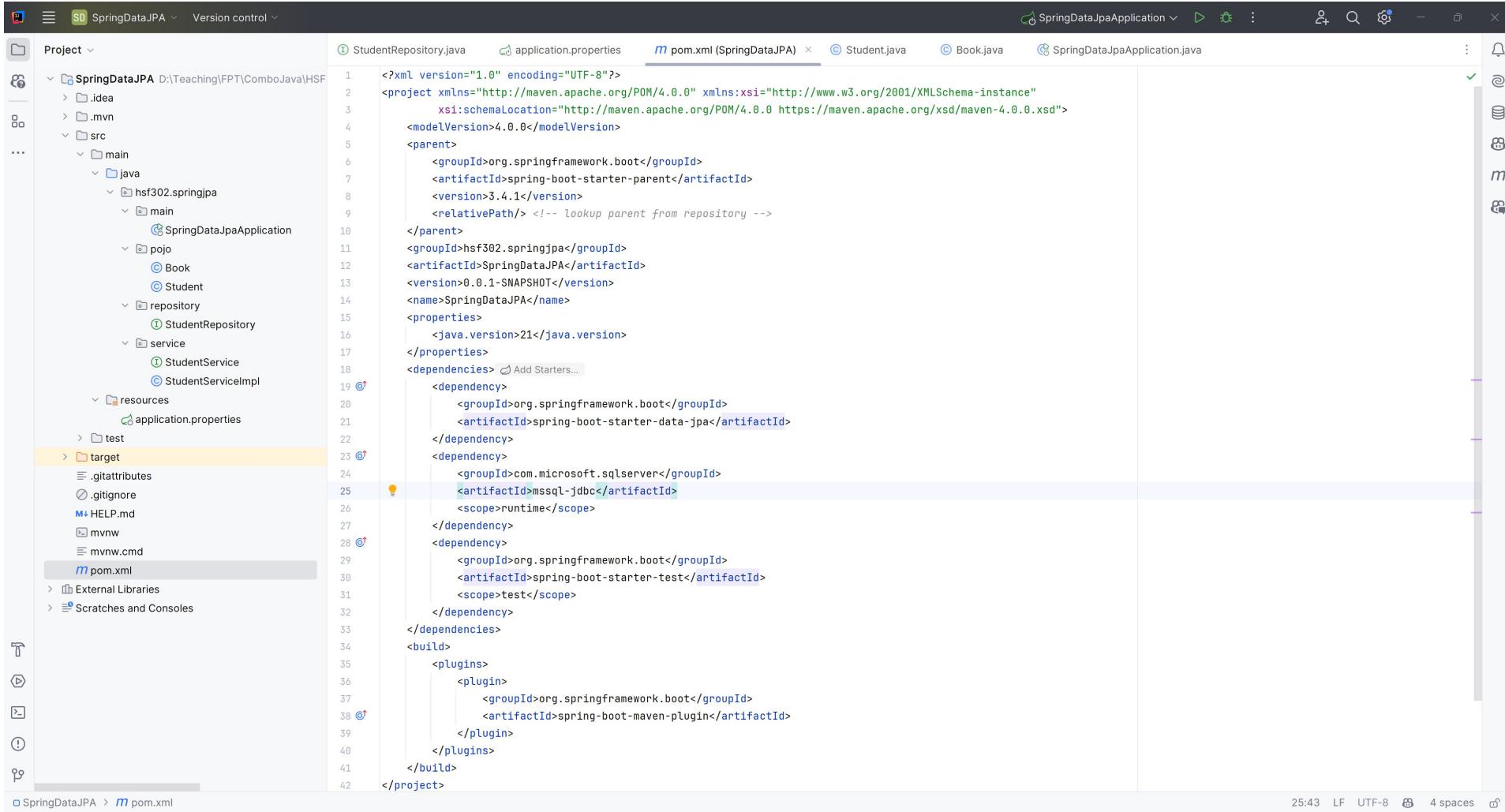
Open IntelliJ, File | New | Spring Boot -> Maven Project



Add Dependencies



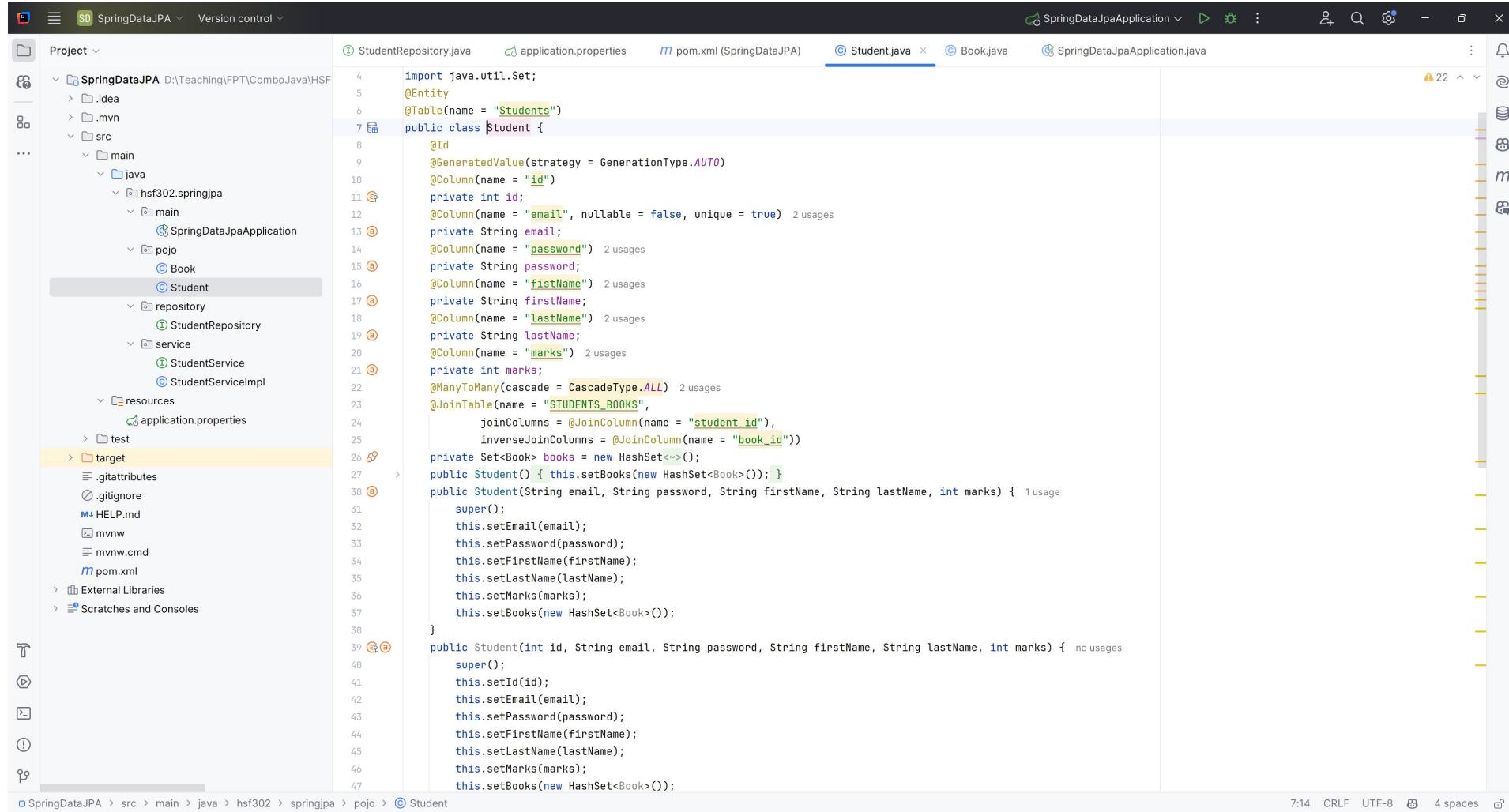
Create the Structure Project



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "SpringDataJPA". It includes the following packages:
 - src/main/java:** Contains `hsf302.springjpa`, which has `main`, `pojo`, `repository`, and `service` packages. `StudentRepository` and `StudentService` are defined here.
 - src/resources:** Contains `application.properties`.
 - target:** Contains build artifacts.
- pom.xml (SpringDataJPA):** The current file being edited, showing the Maven configuration for the project. It specifies the parent group ID as `org.springframework.boot` and artifact ID as `spring-boot-starter-parent`. The version is set to `3.4.1`. The project name is `SpringDataJPA`. The `<dependencies>` section includes dependencies for `spring-boot-starter-data-jpa` (group ID `com.microsoft.sqlserver`, artifact ID `mssql-jdbc`, scope `runtime`) and `spring-boot-starter-test` (group ID `org.springframework.boot`, artifact ID `spring-boot-starter-test`, scope `test`). The `<build>` section includes a plugin for `spring-boot-maven-plugin`.
- Other Files:** Shows `application.properties`, `Student.java`, `Book.java`, and `SpringDataJpaApplication.java` in the editor tabs.

Create the Student.java in pojo

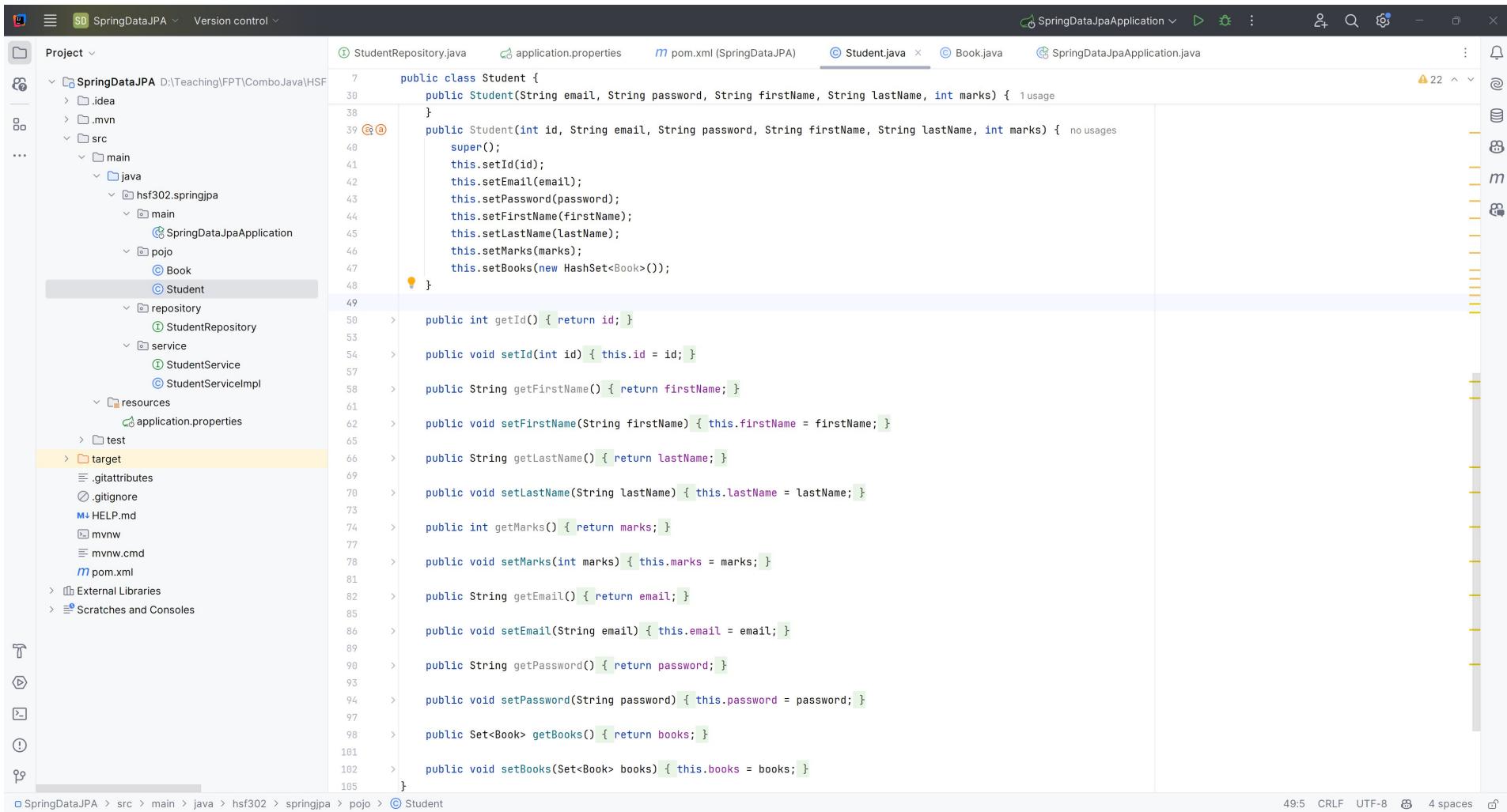


The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The left sidebar shows the project structure under "SpringDataJPA". The "Student" class is selected in the "src/main/java/pojo" directory.
- Editor Tab:** The "Student.java" tab is active in the top navigation bar.
- Code Content:** The code for the Student.java class is displayed:

```
import java.util.Set;
@Entity
@Table(name = "Students")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private int id;
    @Column(name = "email", nullable = false, unique = true) 2 usages
    private String email;
    @Column(name = "password") 2 usages
    private String password;
    @Column(name = "firstName") 2 usages
    private String firstName;
    @Column(name = "lastName") 2 usages
    private String lastName;
    @Column(name = "marks") 2 usages
    private int marks;
    @ManyToMany(cascade = CascadeType.ALL) 2 usages
    @JoinTable(name = "STUDENTS_BOOKS",
               joinColumns = @JoinColumn(name = "student_id"),
               inverseJoinColumns = @JoinColumn(name = "book_id"))
    private Set<Book> books = new HashSet<>();
    public Student() { this.setBooks(new HashSet<Book>()); }
    public Student(String email, String password, String firstName, String lastName, int marks) { 1 usage
        super();
        this.setEmail(email);
        this.setPassword(password);
        this.setFirstName(firstName);
        this.setLastName(lastName);
        this.setMarks(marks);
        this.setBooks(new HashSet<Book>());
    }
    public Student(int id, String email, String password, String firstName, String lastName, int marks) { no usages
        super();
        this.setId(id);
        this.setEmail(email);
        this.setPassword(password);
        this.setFirstName(firstName);
        this.setLastName(lastName);
        this.setMarks(marks);
        this.setBooks(new HashSet<Book>());
    }
}
```
- Toolbars and Status Bar:** The top bar includes tabs for "SpringDataJPA", "application.properties", "pom.xml (SpringDataJPA)", "Book.java", and "SpringDataJpaApplication.java". The status bar at the bottom shows "7:14 CRLF UTF-8 4 spaces".

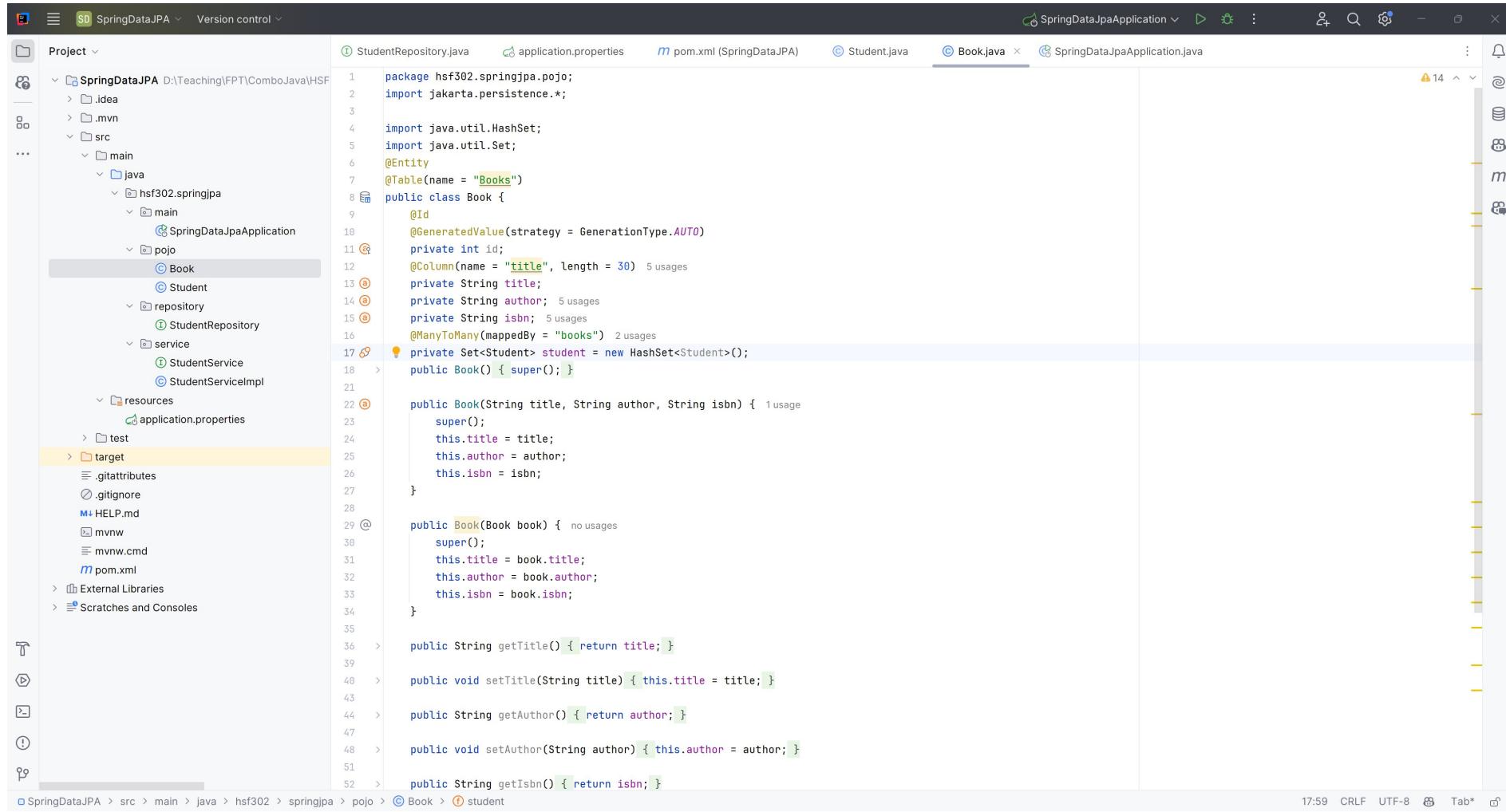
Create the Student.java in pojo



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The left sidebar shows the project structure under "SpringDataJPA". The "Student" class is selected in the "src/main/java/hsf302.springjpa/pojo" package.
- Edit Window:** The main window displays the code for the `Student.java` class. The code defines a constructor taking email, password, first name, last name, and marks, and a second constructor taking id, email, password, first name, last name, and marks. It also includes getters and setters for id, first name, last name, marks, email, and password, along with a method to get books and set books.
- Status Bar:** The bottom status bar shows the file path as "SpringDataJPA > src > main > java > hsf302 > springjpa > pojo > Student", and the bottom right corner shows "49:5 CRLF UTF-8 8 4 spaces".

Create the Book.java in pojo



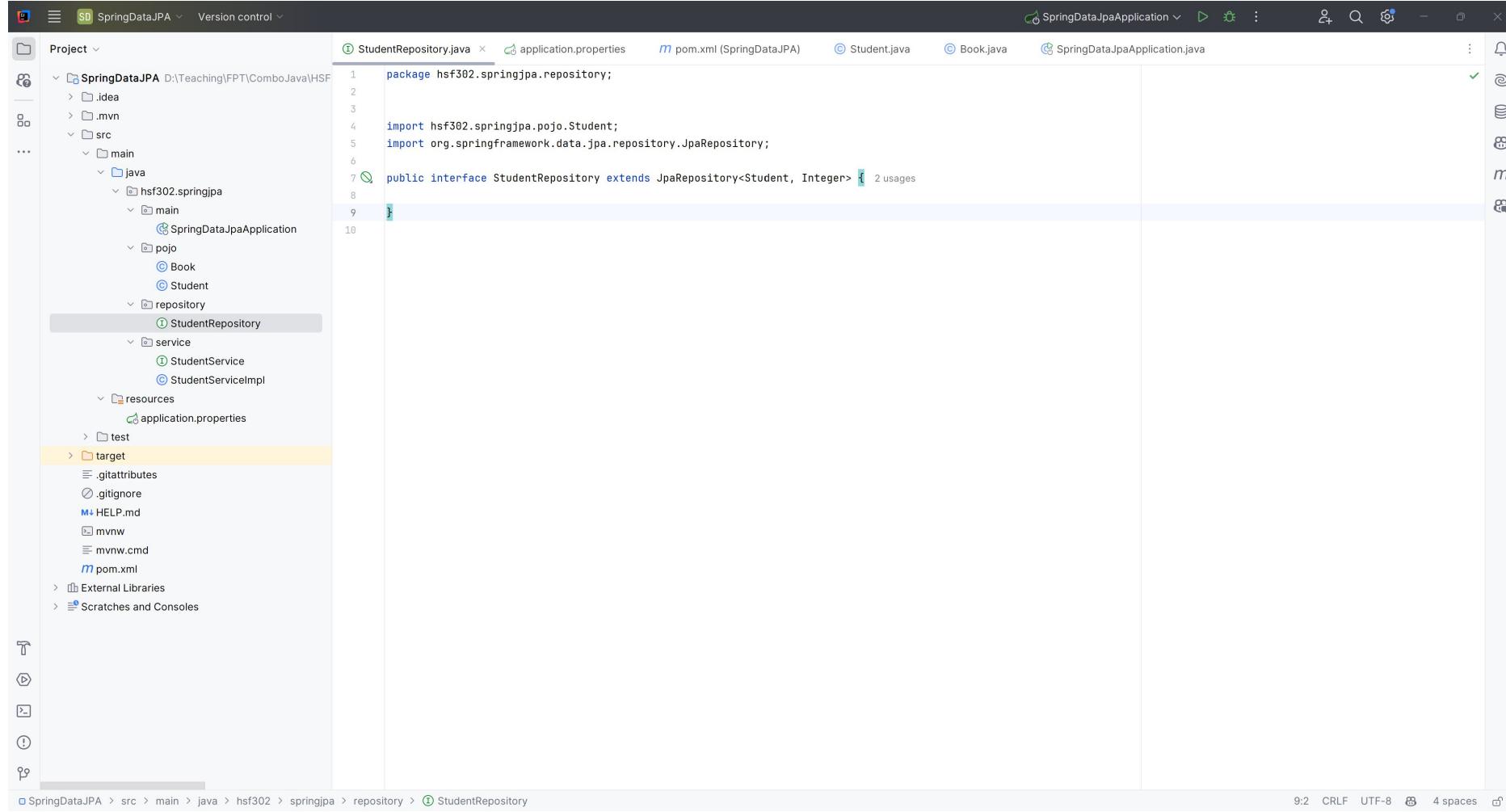
The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "SpringDataJPA". The "target" folder is highlighted.
- Editor Tab Bar:** Displays tabs for StudentRepository.java, application.properties, pom.xml, Student.java, Book.java (which is the active tab), and SpringDataJpaApplication.java.
- Code Editor:** The content of the Book.java file is displayed. The code defines a Book entity with fields for id, title, author, and isbn, and a many-to-many relationship with Student.

```
1 package hsf302.springjpa.pojo;
2 import jakarta.persistence.*;
3
4 import java.util.HashSet;
5 import java.util.Set;
6
7 @Entity
8 @Table(name = "Books")
9 public class Book {
10     @Id
11     @GeneratedValue(strategy = GenerationType.AUTO)
12     private int id;
13     @Column(name = "title", length = 30) 5 usages
14     private String title;
15     private String author; 5 usages
16     private String isbn; 5 usages
17     @ManyToMany(mappedBy = "books") 2 usages
18     private Set<Student> student = new HashSet<Student>();
19     public Book() { super(); }
20
21     public Book(String title, String author, String isbn) { 1 usage
22         super();
23         this.title = title;
24         this.author = author;
25         this.isbn = isbn;
26     }
27
28     public Book(Book book) { no usages
29         super();
30         this.title = book.title;
31         this.author = book.author;
32         this.isbn = book.isbn;
33     }
34
35     public String getTitle() { return title; }
36     public void setTitle(String title) { this.title = title; }
37
38     public String getAuthor() { return author; }
39     public void setAuthor(String author) { this.author = author; }
40
41     public String getIsbn() { return isbn; }
42 }
```

- Bottom Status Bar:** Shows the current time (17:59), file encoding (CRLF), character set (UTF-8), and other system information.

Create the StudentRepository.java in repository package

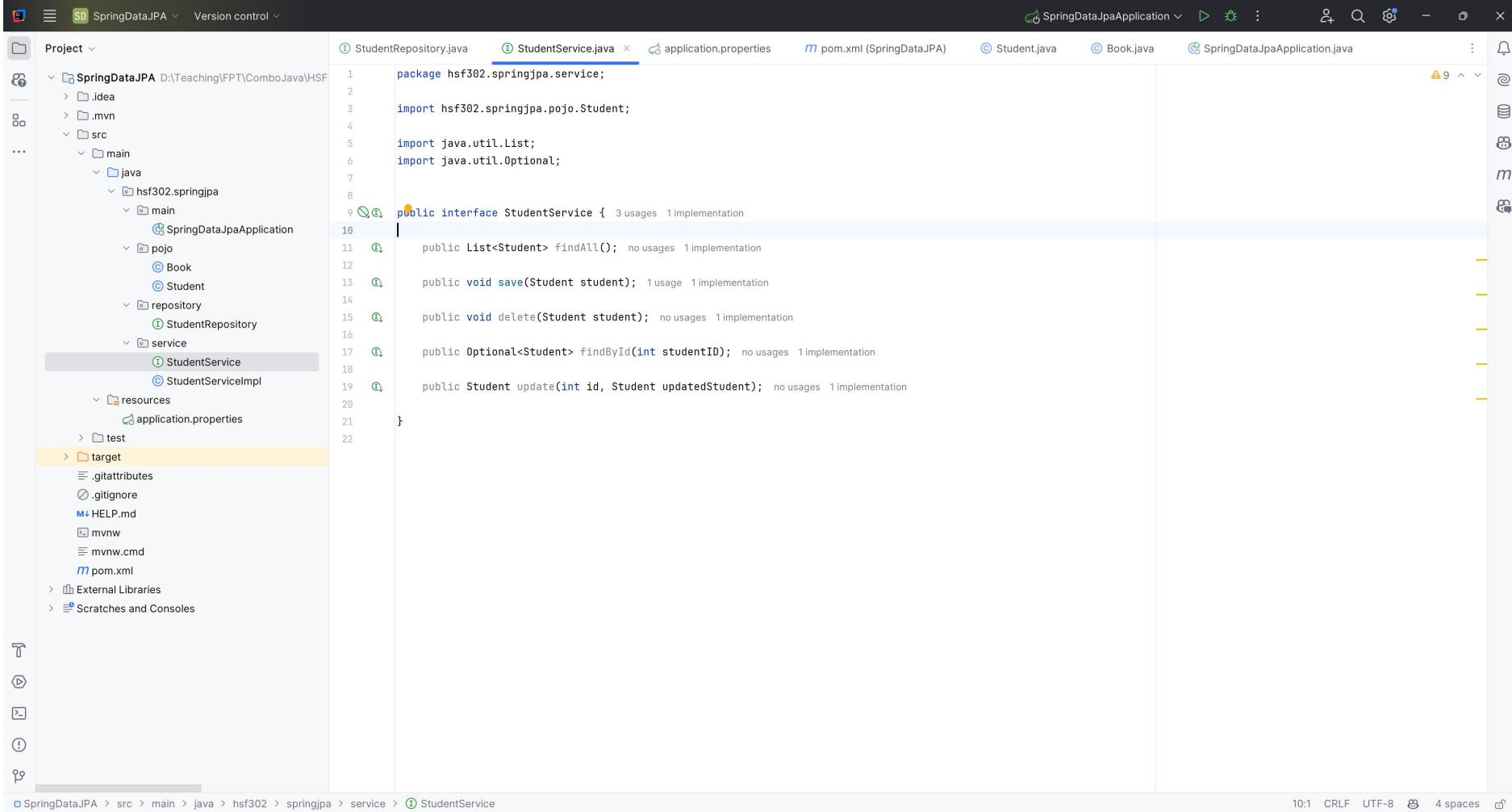


The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "SpringDataJPA". The "src" directory contains "main" and "test" packages. "main" has "java", "pojo", and "repository" sub-packages. "repository" contains "StudentRepository".
- Code Editor:** The file "StudentRepository.java" is open. The code is as follows:

```
1 package hsf302.springjpa.repository;
2
3 import hsf302.springjpa.pojo.Student;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface StudentRepository extends JpaRepository<Student, Integer> { 2 usages}
```

Create the StudentService.java in services package



The screenshot shows the IntelliJ IDEA interface with the following details:

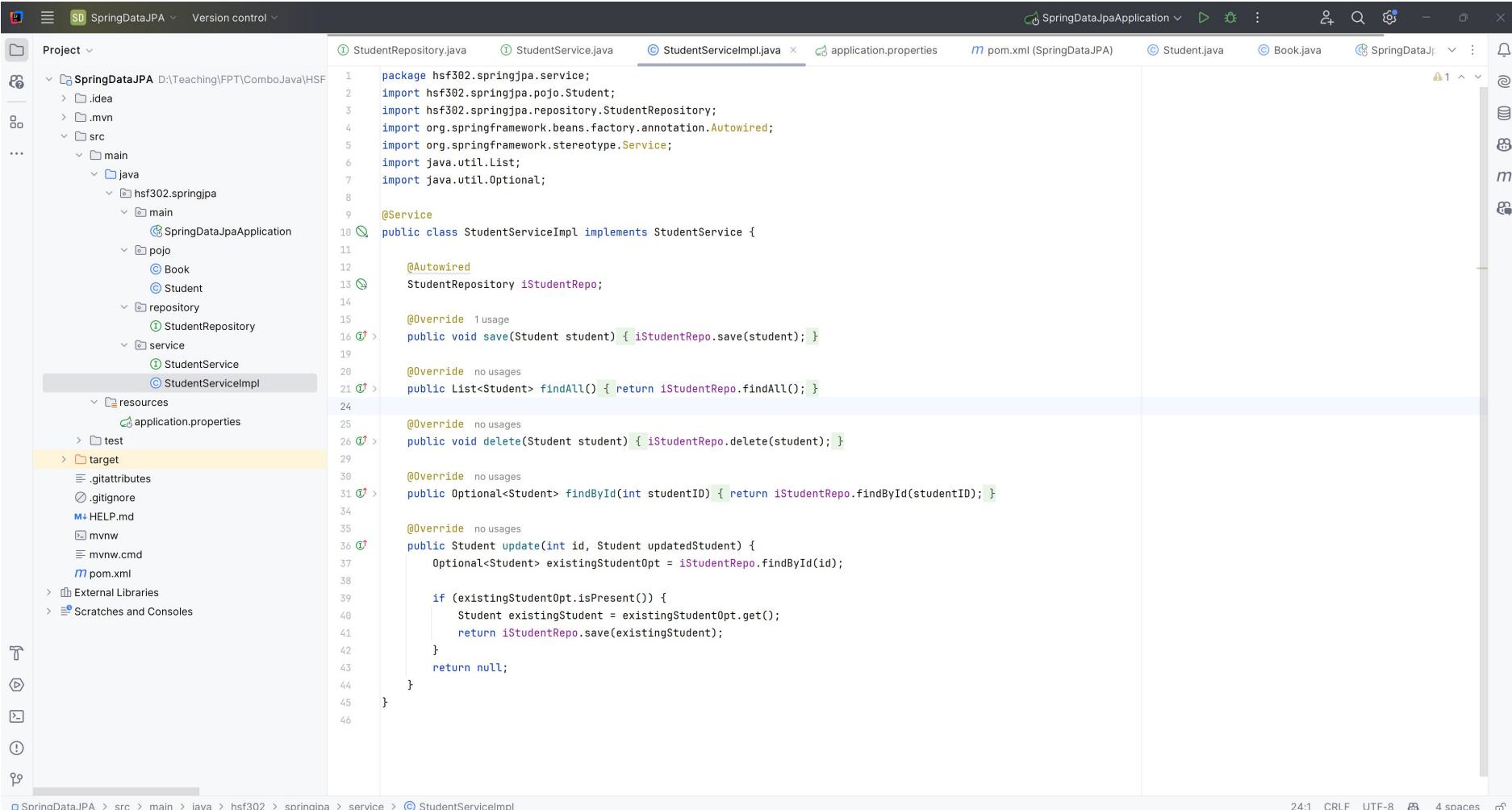
- Project Structure:** The project is named "SpringDataJPA". The "src" directory contains "main" and "test" packages. Under "main", there are "hsf302.springjpa", "pojo", "repository", and "service" packages. The "service" package is currently selected.
- Code Editor:** The file "StudentService.java" is open. The code defines a public interface "StudentService" with the following methods:

```
package hsf302.springjpa.service;

import hsf302.springjpa.pojo.Student;
import java.util.List;
import java.util.Optional;

public interface StudentService {
    public List<Student> findAll();
    public void save(Student student);
    public void delete(Student student);
    public Optional<Student> findById(int studentID);
    public Student update(int id, Student updatedStudent);
}
```
- Toolbars and Status Bar:** The top bar shows tabs for "StudentRepository.java", "application.properties", "pom.xml (SpringDataJPA)", "Student.java", "Book.java", and "SpringDataJpaApplication.java". The status bar at the bottom right shows "10:1 CRLF UTF-8 4 spaces".

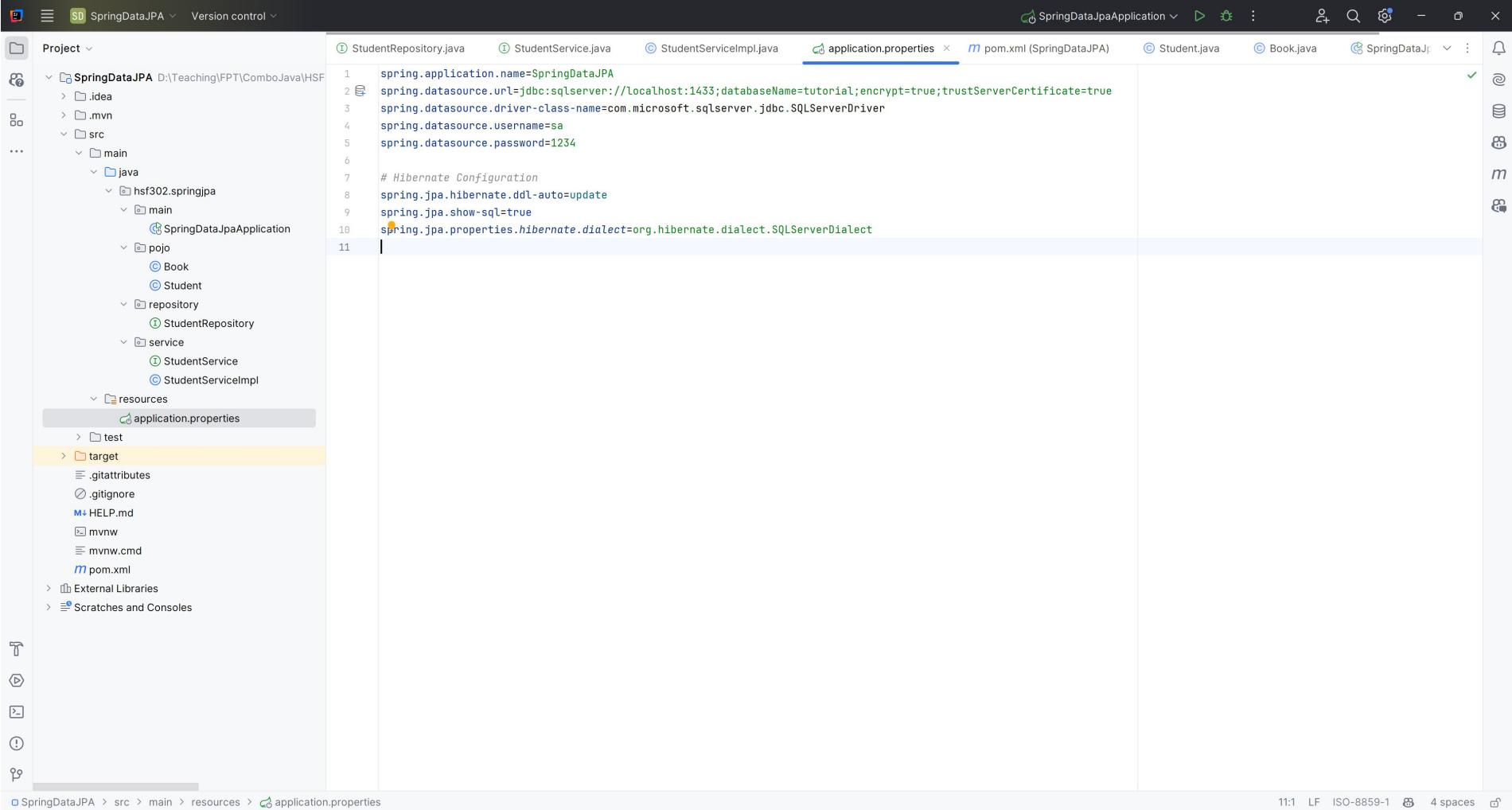
Create the StudentServiceImpl.java in services package



The screenshot shows a Java development environment with the following details:

- Project Structure:** The project is named "SpringDataJPA" located at "D:\Teaching\FPT\ComboJava\HSF". It contains a "src" directory with "main" and "test" packages. "main" has "hsf302.springjpa", "pojo", "repository", and "service" packages. "service" contains "StudentService" and "StudentServiceImpl". "resources" folder includes "application.properties".
- Current File:** The file "StudentServiceImpl.java" is open in the editor.
- Code Content:** The code implements the "StudentService" interface. It imports "hsf302.springjpa.Student", "hsf302.springjpa.repository.StudentRepository", "org.springframework.beans.factory.annotation.Autowired", "org.springframework.stereotype.Service", "java.util.List", and "java.util.Optional". The class uses autowiring to inject "StudentRepository" and overrides methods like "save", "findAll", "delete", and "update".
- IDE Interface:** The IDE has tabs for "StudentRepository.java", "StudentService.java", "StudentServiceImpl.java", "application.properties", "pom.xml", "Student.java", "Book.java", and "SpringDataJPA.java". The status bar at the bottom shows "SpringDataJPA > src > main > java > hsf302 > springjpa > service > StudentServiceImpl".

Create the application.properties in resources



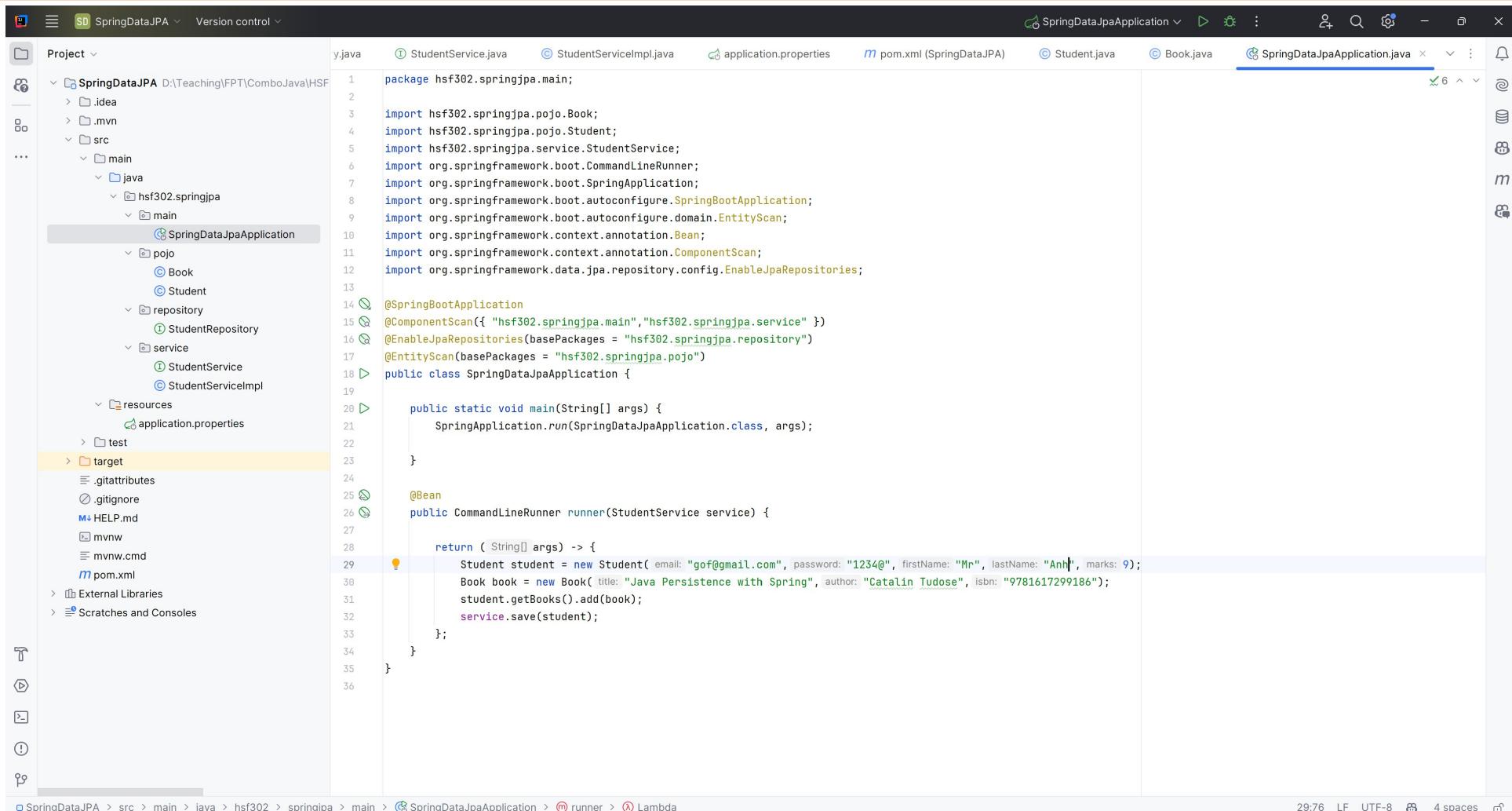
The screenshot shows the IntelliJ IDEA interface with the project 'SpringDataJPA' open. The left sidebar displays the project structure, including the 'resources' folder which contains the 'application.properties' file. The main editor window shows the contents of the 'application.properties' file:

```
spring.application.name=SpringDataJPA
spring.datasource.url=jdbc:sqlserver://localhost:1433;databaseName=tutorial;encrypt=true;trustServerCertificate=true
spring.datasource.driver-class-name=com.microsoft.sqlserver.jdbc.SQLServerDriver
spring.datasource.username=sa
spring.datasource.password=1234

# Hibernate Configuration
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.SQLServerDialect
```

The status bar at the bottom indicates the file is 111 lines long, uses LF line endings, is ISO-8859-1 encoded, has 4 spaces per indentation, and was last modified at 11:11.

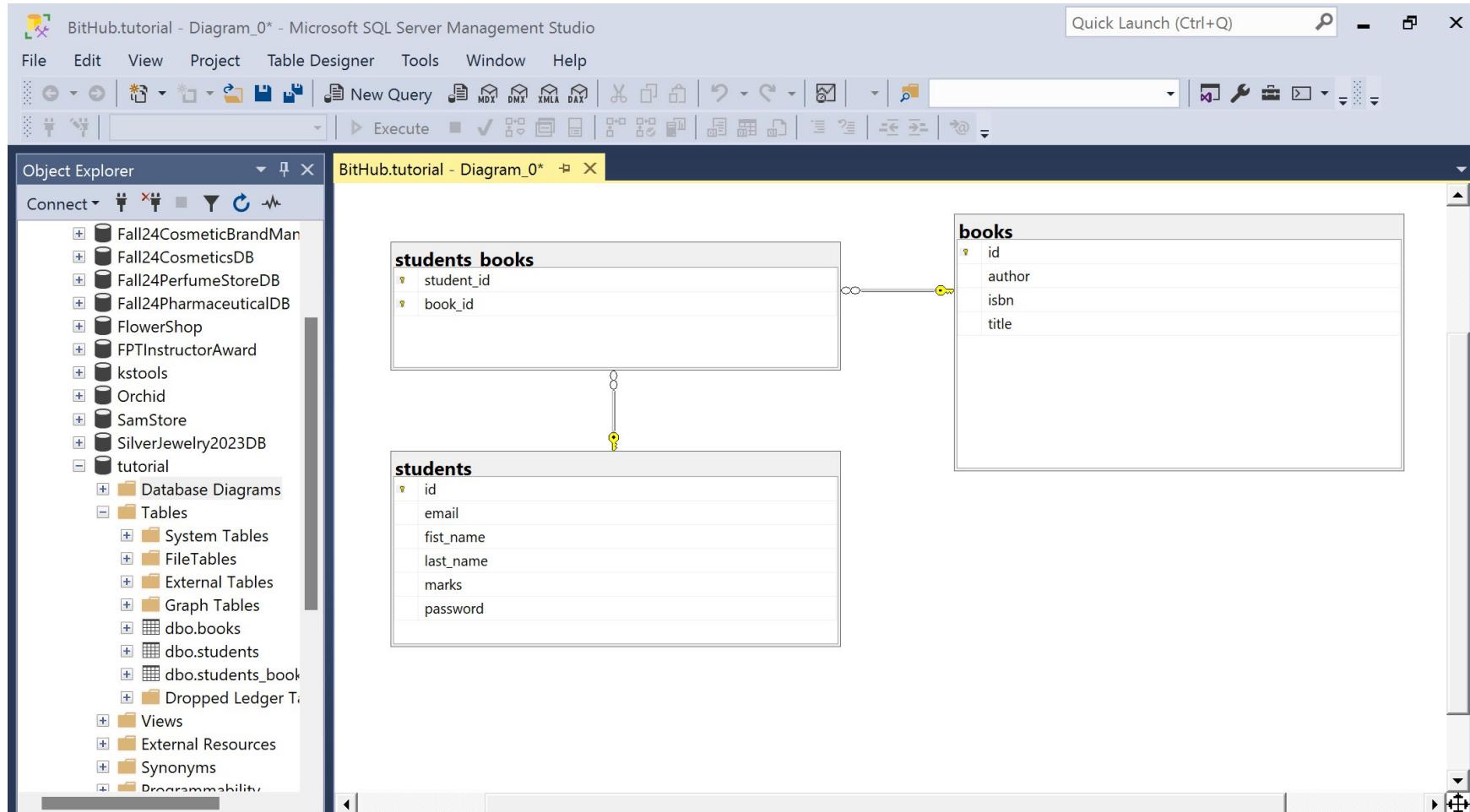
Edit main function



The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "SpringDataJPA" located at "D:\Teaching\FPT\ComboJava\HSF". It contains a "src" folder with "main" and "test" subfolders. "main" contains "hsf302.springjpa" which has "main", "pojo", "repository", "service", and "resources" packages. "resources" contains "application.properties". "test" contains ".gitattributes", ".gitignore", "HELP.md", "mvnw", "mvnw.cmd", and "pom.xml". "target" contains ".gitattributes", ".gitignore", and "Scrapes and Consoles".
- Code Editor:** The file "SpringDataJpaApplication.java" is open. The code defines a main function that runs a Spring Boot application. It includes annotations like @SpringBootApplication, @ComponentScan, and @EnableJpaRepositories. It also defines a CommandLineRunner named "runner" that creates a student and a book, adds the book to the student's books, and saves the student.
- Status Bar:** The status bar at the bottom shows the file path as "SpringDataJPA > src > main > java > hsf302 > springjpa > main > SpringDataJpaApplication > runner > Lambda", and the status "29:76 LF UTF-8 4 spaces".

Run Program



Result student table

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists several databases, with 'tutorial' selected. The 'Tables' node under 'tutorial' is expanded, showing 'dbo.students'. The central pane displays a query window with the following SQL code:

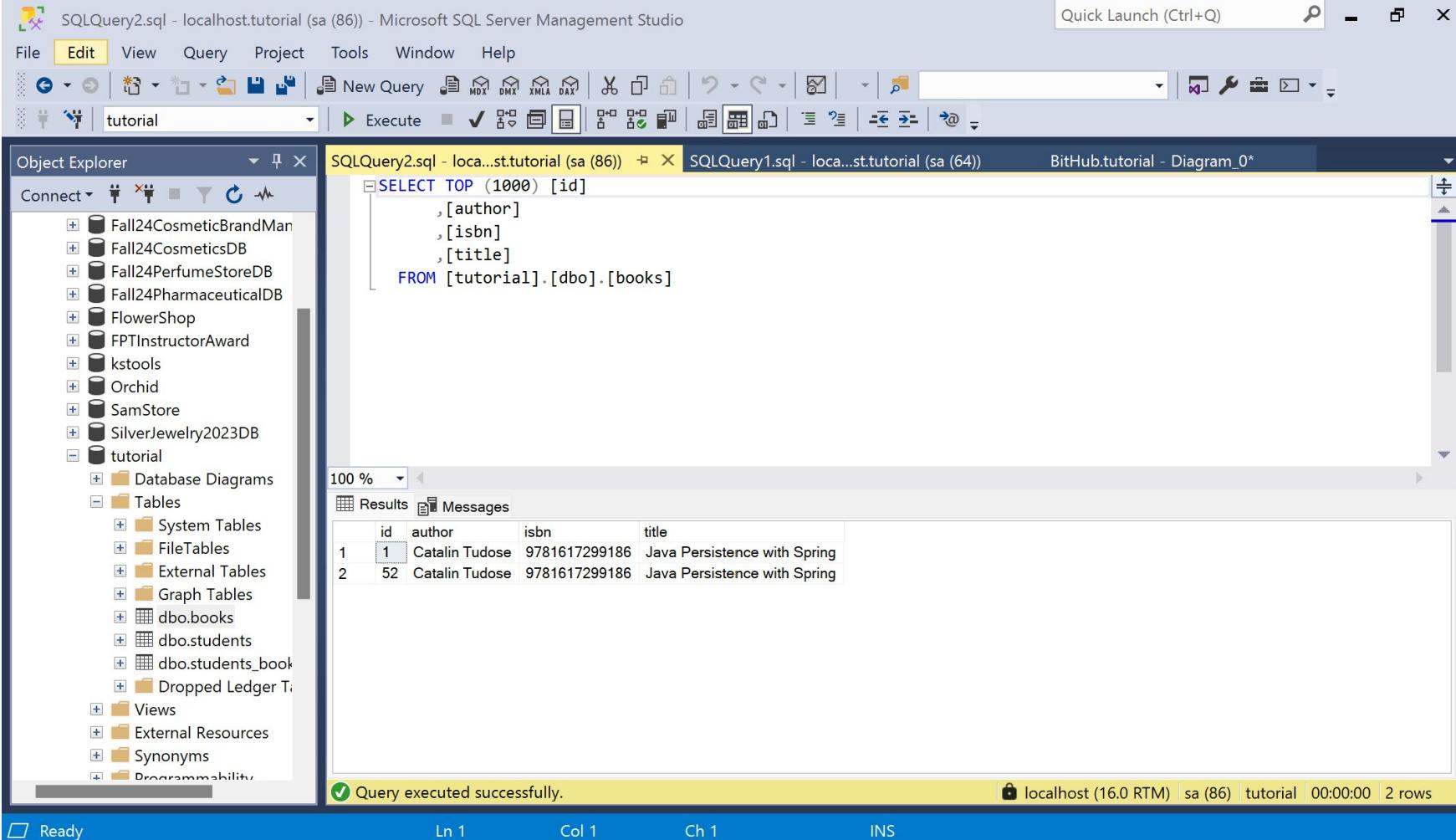
```
SELECT TOP (1000) [id]
    ,[email]
    ,[first_name]
    ,[last_name]
    ,[marks]
    ,[password]
FROM [tutorial].[dbo].[students]
```

The results pane below shows a table with two rows of data:

	id	email	first_name	last_name	marks	password
1	402	lamnn@gmail.com	Lam	Nguyen	9	1234@
2	502	gof@gmail.com	Mr	Anh	9	1234@

At the bottom, a message bar indicates "Query executed successfully." and shows connection details: localhost (16.0 RTM) | sa (64) | tutorial | 00:00:00 | 2 rows.

Result book table



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists several databases, with 'tutorial' selected. The central pane displays a query window with the following SQL code:

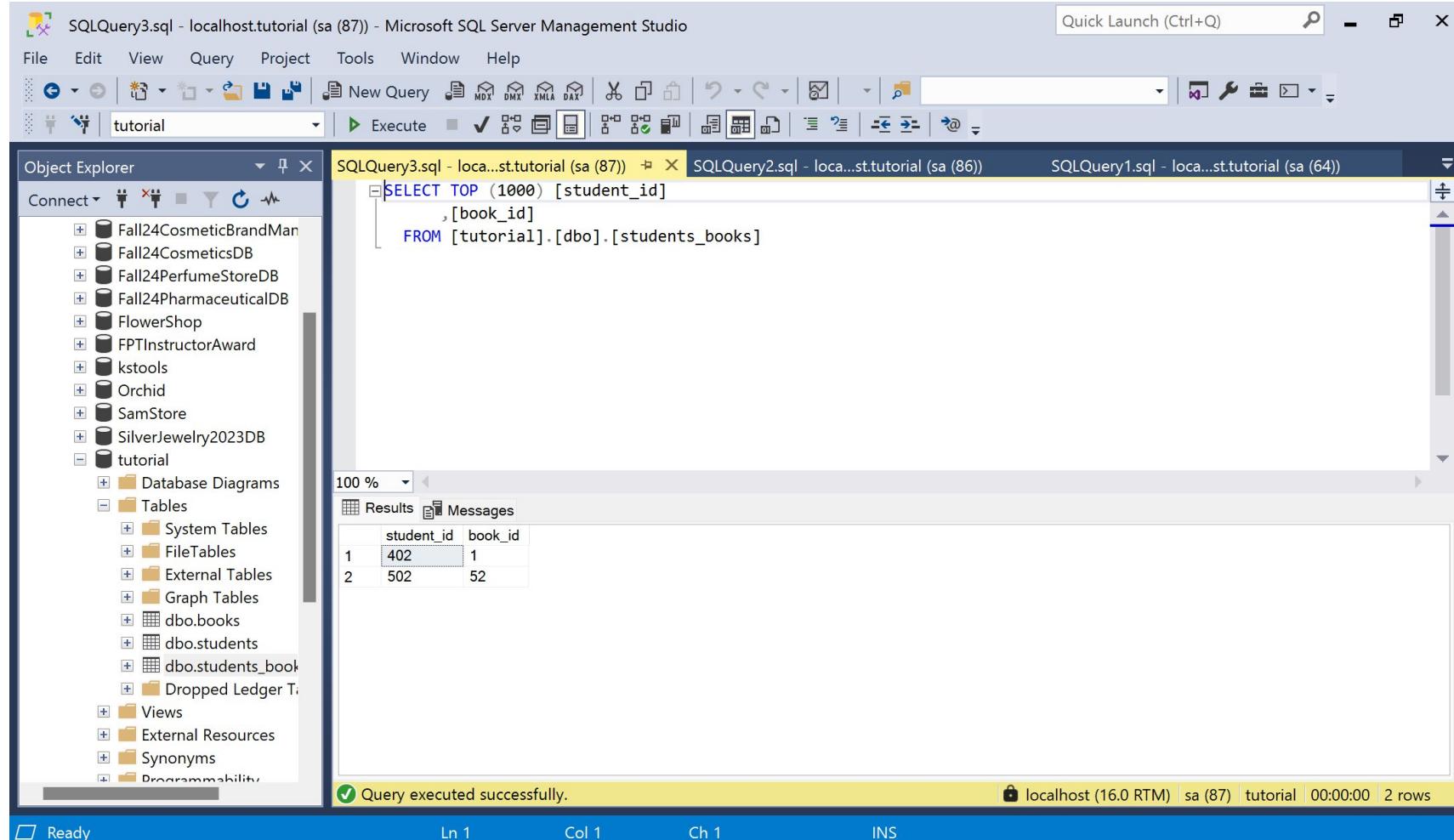
```
SELECT TOP (1000) [id],  
       [author],  
       [isbn],  
       [title]  
  FROM [tutorial].[dbo].[books]
```

The results pane below shows the output of the query:

	id	author	isbn	title
1	1	Catalin Tudose	9781617299186	Java Persistence with Spring
2	52	Catalin Tudose	9781617299186	Java Persistence with Spring

A message at the bottom of the results pane states "Query executed successfully."

Result student-book table



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists several databases, with 'tutorial' selected. The 'Tables' node under 'tutorial' is expanded, showing 'dbo.books', 'dbo.students', 'dbo.students_book', and 'Dropped Ledger T...'. The 'Results' tab in the center displays the output of the following SQL query:

```
SELECT TOP (1000) [student_id]
      ,[book_id]
  FROM [tutorial].[dbo].[students_books]
```

The results are as follows:

	student_id	book_id
1	402	1
2	502	52

A status bar at the bottom indicates: 'Query executed successfully.' and 'localhost (16.0 RTM) | sa (87) | tutorial | 00:00:00 | 2 rows'.

Summary

Concepts were introduced:

- ◆ Spring Data JPA
- ◆ Advantages of using Spring Data JPA
- ◆ Key features of Spring Data JPA
 - Dependency Injection and Inversion of Control
 - N-Layer