

Java Persistence API

Objectives

- ❖ Explain what JPA is and its role in data persistence for Java applications.
- ❖ Highlight the advantages of using JPA over traditional JDBC approaches.
- ❖ Define and illustrate core JPA concepts like entities, annotations, persistence context, entity manager and JPQL.
- ❖ Provide a clear understanding of how these components work together to manage data.
- ❖ Show how JPA simplifies development by reducing boilerplate code and allowing focus on business logic.

Contents

- ❖ Introduction
- ❖ Key Concepts
- ❖ Annotations
- ❖ Relationships
- ❖ Using JPA
- ❖ Demo
- ❖ Advantages and Disadvantages

Object Relational Mapping (ORM)

Benefits of ORM

- ◆ **Increased Productivity:** You write less code because ORM handles the low-level data access tasks.
- ◆ **Improved Maintainability:** Your code is cleaner and easier to understand because it focuses on the business logic, not database details.
- ◆ **Enhanced Performance:** ORM frameworks can optimize data access and caching, potentially improving performance.
- ◆ **Database Independence:** You can switch databases without rewriting your application code (to some extent)

Popular ORM Frameworks for Java

- ❖ **Hibernate**: The most widely used JPA implementation, offering a rich feature set and extensive customization options.
- ❖ **EclipseLink**: Another JPA implementation with a focus on standards compliance and portability.
- ❖ **Spring Data JPA**: An abstraction layer on top of JPA providers, simplifying data access and reducing boilerplate code.
- ❖ **DataNucleus** providing easy persistence to RDBMS datastores. Comes with its own "SQL-like" JPQL query language, so you query your data in a language similar to what your datastore understands.

Java Persistence API (JPA)

What is JPA ?

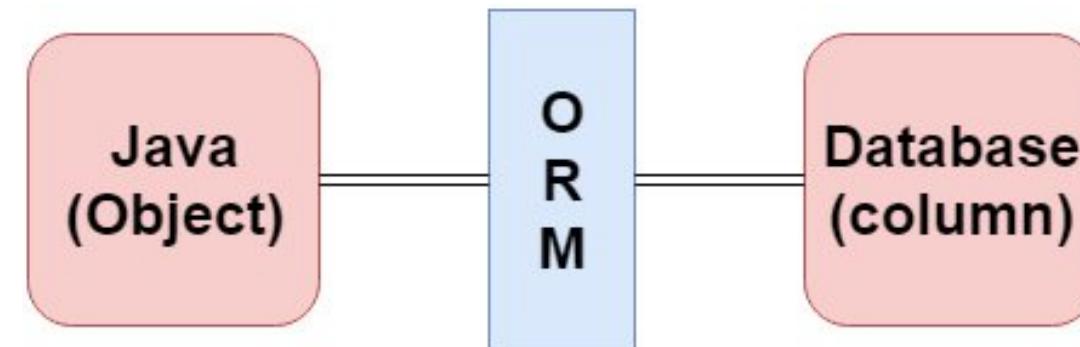
- ❖ JPA stands for Java Persistence API. It is a Java programming interface that allows developers to manage relational data in Java applications using object oriented methodologies.
- ❖ JPA is a specification for managing relational data in Java applications. It provides a set of interfaces, annotations, and an object relational mapping (ORM) framework that simplifies the process of interacting with databases.

Key Feature of JPA

- ❖ **ORM:** JPA allows you to map Java objects (entities) to relational database tables, abstracting away the differences between the object-oriented and relational models.
- ❖ **Annotations:** You can use annotations to define entities, relationships and other persistence related metadata.
- ❖ **EntityManager:** The interface provides methods for persisting, retrieving, updating, and deleting entities.
- ❖ **JPQL** is an object oriented query language that allows you to query entities and their relationships.
- ❖ **Transaction Management:** JPA supports transaction management, ensuring data integrity and consistency.

JPA Object Relational Mapping

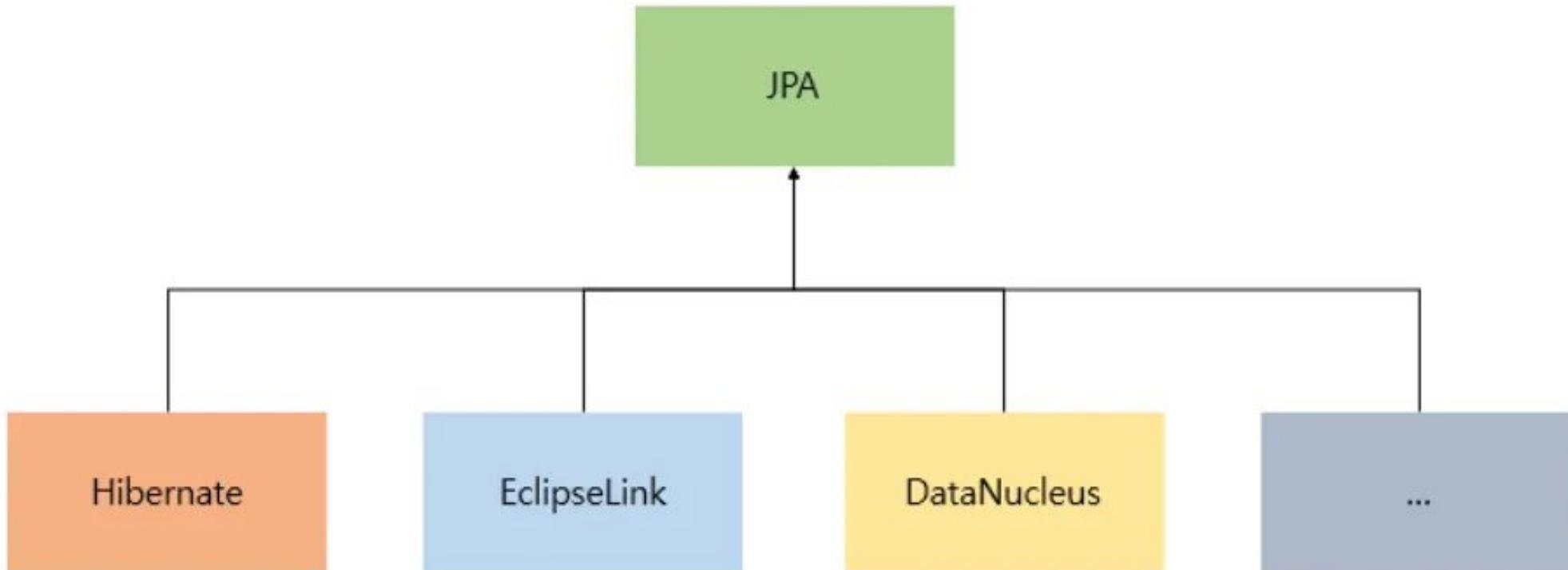
- ❖ ORM makes it easier to work with databases in object-oriented applications, allowing you to focus on the business logic rather than the underlying data storage details.



Benefits of Using JPA

- ❖ **Simplified Data Access:** JPA makes it easier to work with relational data in Java applications by providing a higher-level abstraction.
- ❖ **Code Portability:** JPA is a standard specification, so code written using JPA can be portable across different JPA providers.
- ❖ **Reduced Boilerplate Code:** JPA annotations and the EntityManager interface reduce the amount of code needed for data access.
- ❖ **Improved Data Integrity:** JPA's transaction management and validation features help ensure data integrity.
- ❖ **Enhanced Performance:** Caching and other optimization techniques can improve the performance of JPA applications.

JPA Implementation Options



Entity Annotations in JPA

- ❖ **@Entity:** This annotation marks a Java class as an entity, meaning it represents a table in the database.
- ❖ **@Table:** This annotation allows you to specify the name of the database table that an entity maps to.
- ❖ **@Transient:** This annotation marks a property that should not be persisted to the database.
- ❖ **@NamedQueries and @NamedQuery:** These annotations allow you to define named queries that can be used to retrieve entities.

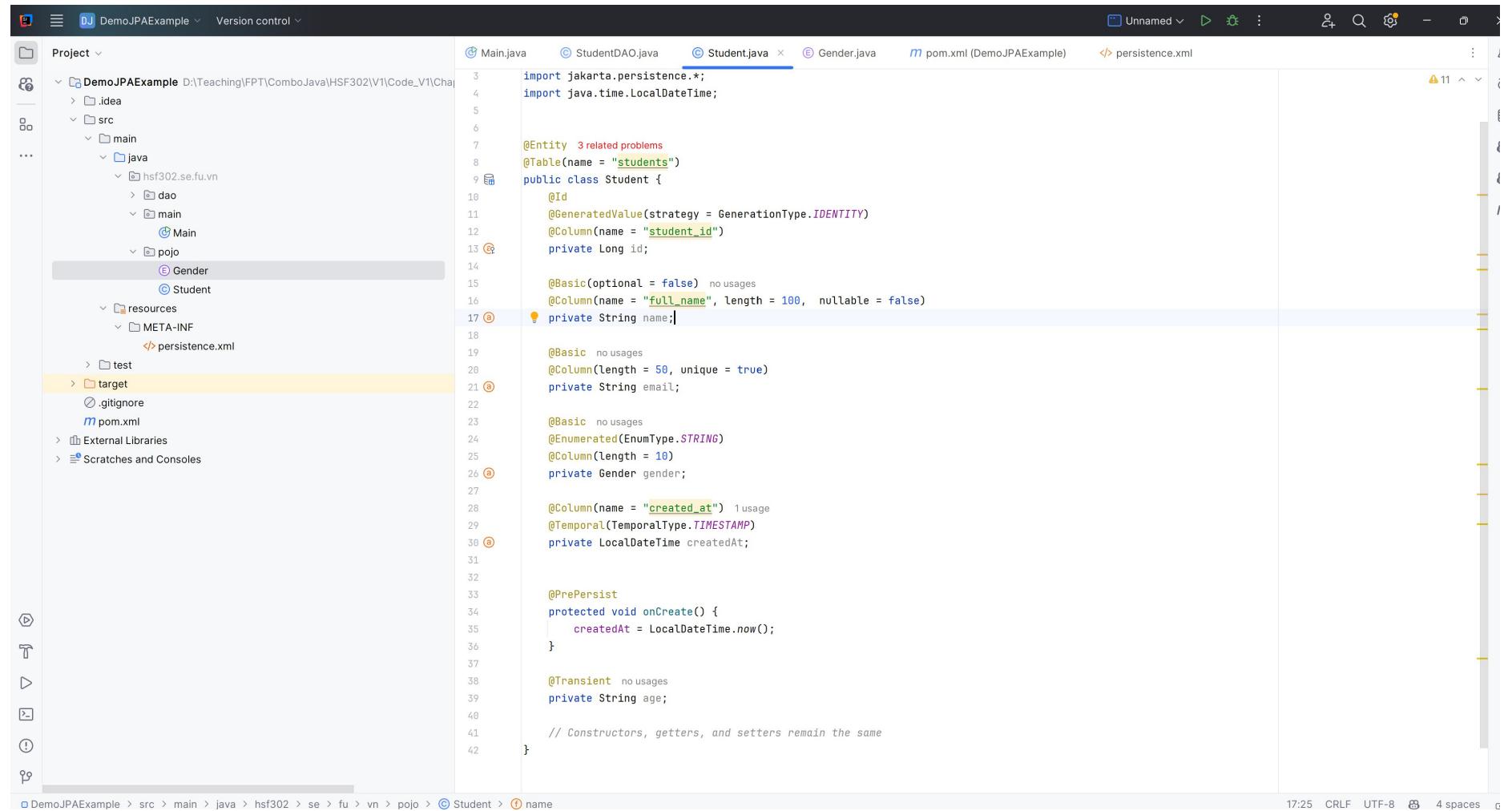
Mapping Annotations in JPA

- ❖ **@Id:** This annotation marks a Java class as an entity, meaning it represents a table in the database.
- ❖ **@Column:** This annotation defines the mapping between a property of an entity and a column in the database table. You can use it to specify the column name, data type, and other attributes.
- ❖ **@Basic:** This annotation specifies that a property is a basic type and should be persisted.
- ❖ **@Enumerated:** This annotation is used to map an Enum type to a database column.

Example

```
package hsf302.se.fu.vn.pojo;

public enum Gender {
    MALE,
    FEMALE,
    OTHER
}
```



The screenshot shows an IDE interface with the following details:

- Project:** DemoJPAExample
- File:** Student.java
- Content:** Java code defining a `Student` entity with attributes `id`, `name`, `email`, `gender`, and `createdAt`. It also includes `@PrePersist` logic for setting `createdAt`.
- Imports:** `jakarta.persistence.*` and `java.time.LocalDateTime`.
- Annotations:** `@Entity`, `@Table(name = "students")`, `@Id`, `@GeneratedValue(strategy = GenerationType.IDENTITY)`, `@Column(name = "student_id")`, `@Basic(optional = false)`, `@Column(name = "full_name", length = 100, nullable = false)`, `@Enumerated(EnumType.STRING)`, `@Column(length = 18)`, `@Column(name = "created_at")`, `@Temporal(TemporalType.TIMESTAMP)`.
- IDE UI:** Shows the project structure on the left, the code editor with syntax highlighting, and various toolbars and status bars at the top and bottom.

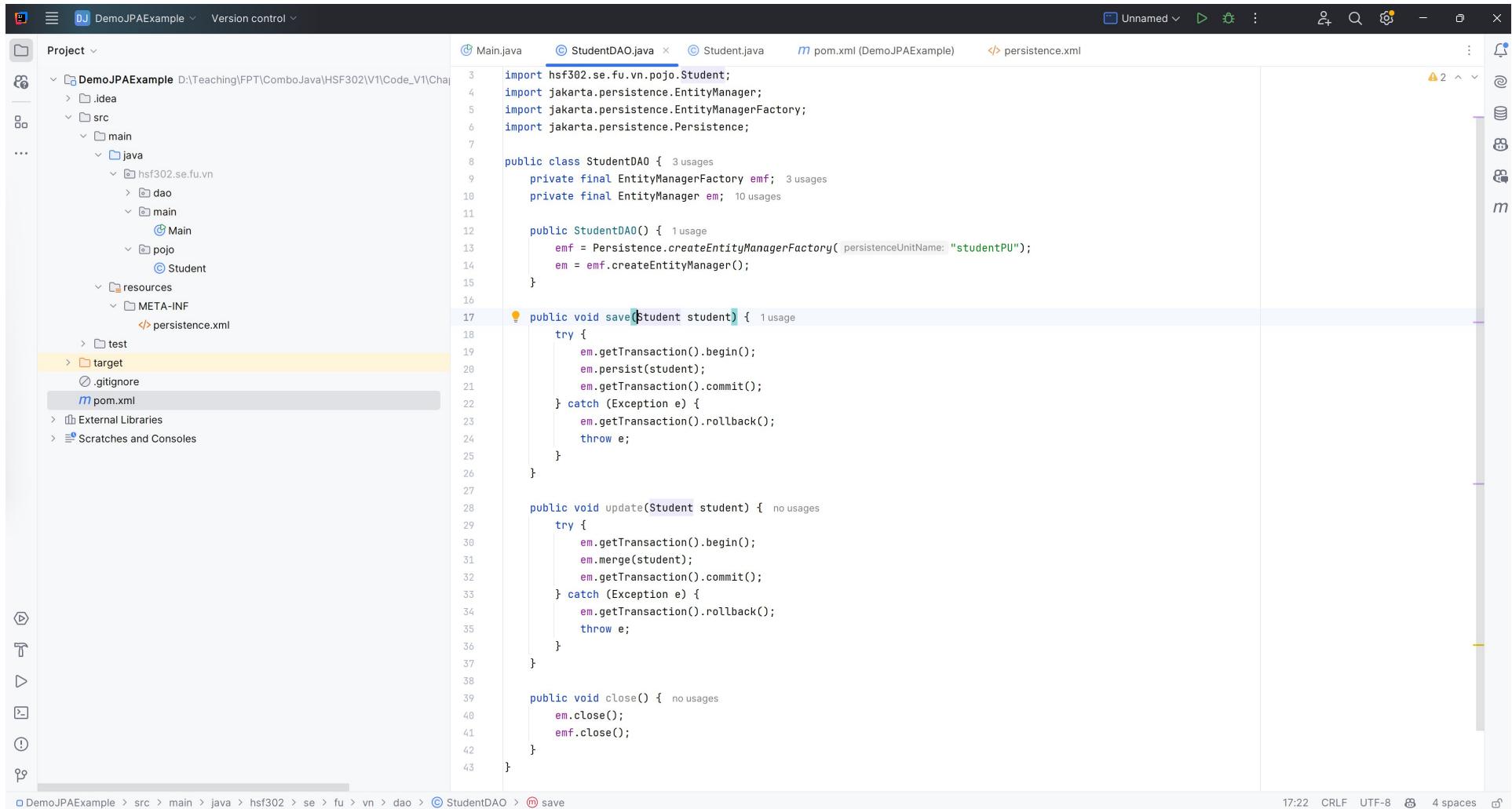
JPA Entity Manager

- ❖ The entity manager implements the API and encapsulates all of them within a single interface.
 - **Manages entity lifecycle:** Persist, find, merge, remove
 - **Controls persistence context:** Cache, flush, detach
 - **Executes queries:** JPQL, Criteria API
 - **Handles transactions:** Begin, commit, rollback
 - **Provides access to entity metadata**

Key Operations of the Entity Manager

- ❖ **Persisting Entities:** `em.persist(entity)` makes a transient entity instance persistent
- ❖ **Finding Entities:** `em.find(entityClass,primaryKey)` retrieves an entity by its primary key.
- ❖ **Merging Entities:** `em.merge(entity)` merges a detached entity instance into the current persistence context.
- ❖ **Removing Entities:** `em.remove(entity)` removes a persistent entity instance.
- ❖ **Querying Data:** `em.createQuery(jpqlString)` creates a JPQL query to retrieve entities based on specified criteria.

Example



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

- Project Structure:** The left sidebar shows the project structure for "DemoJPAExample". It includes a .idea folder, a src directory containing main/java (with packages hsf302.se.fu.vn, dao, main, pojo, and Student), resources, META-INF (with persistence.xml), test, target, .gitignore, pom.xml, External Libraries, and Scratches and Consoles.
- Code Editor:** The main window displays the "StudentDAO.java" file. The code implements a DAO interface with methods for saving and updating student entities using EntityManager.
- Pom.xml:** The pom.xml file is visible in the project structure, indicating the use of Maven for dependency management.
- Status Bar:** The bottom status bar shows the current file path (DemoJPAExample > src > main > java > hsf302 > se > fu > vn > dao > StudentDAO), the save icon, and the current time (17:22).

```
import hsf302.se.fu.vn.pojo.Student;
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;

public class StudentDAO {
    private final EntityManagerFactory emf;
    private final EntityManager em;

    public StudentDAO() {
        emf = Persistence.createEntityManagerFactory("studentPU");
        em = emf.createEntityManager();
    }

    public void save(Student student) {
        try {
            em.getTransaction().begin();
            em.persist(student);
            em.getTransaction().commit();
        } catch (Exception e) {
            em.getTransaction().rollback();
            throw e;
        }
    }

    public void update(Student student) {
        try {
            em.getTransaction().begin();
            em.merge(student);
            em.getTransaction().commit();
        } catch (Exception e) {
            em.getTransaction().rollback();
            throw e;
        }
    }

    public void close() {
        em.close();
        emf.close();
    }
}
```

Example

The screenshot shows an IDE interface with the following details:

- Project:** DemoJPAExample
- File:** StudentDAO.java
- Content:** Java code for a DAO interface. The code includes methods for closing the EntityManager, removing a student by ID, finding a student by ID, finding all students, finding students by age, and finding students by name.

```
public class StudentDAO { 3 usages
    public void close() { 1 usage
        em.close();
        emf.close();
    }

    public void remove(Long id) { no usages
        try {
            em.getTransaction().begin();
            Student student = em.find(Student.class, id);
            if (student != null) {
                em.remove(student);
            }
            em.getTransaction().commit();
        } catch (Exception e) {
            em.getTransaction().rollback();
            throw e;
        }
    }

    public Student find(Long id) { no usages
        return em.find(Student.class, id);
    }

    public List<Student> findAll() { 1 usage
        TypedQuery<Student> query = em.createQuery("SELECT s FROM Student s", Student.class);
        return query.getResultList();
    }

    public List<Student> findByAge(int age) { 1 usage
        TypedQuery<Student> query = em.createQuery(
            "SELECT s FROM Student s WHERE s.age = :age", Student.class);
        query.setParameter("age", age);
        return query.getResultList();
    }

    public List<Student> findByName(String name) { 1 usage
        TypedQuery<Student> query = em.createQuery(
            "SELECT s FROM Student s WHERE s.name LIKE :name", Student.class);
        query.setParameter("name", name, value: "%" + name + "%");
        return query.getResultList();
    }
}
```

- Toolbars and Status Bar:** The status bar at the bottom shows the file path (DemoJPAExample > src > main > java > hsf302 > se > fu > vn > dao > StudentDAO), line numbers (74-81), and other system information (74:6 CRLF UTF-8 4 spaces).

JPA Versions and Key Features

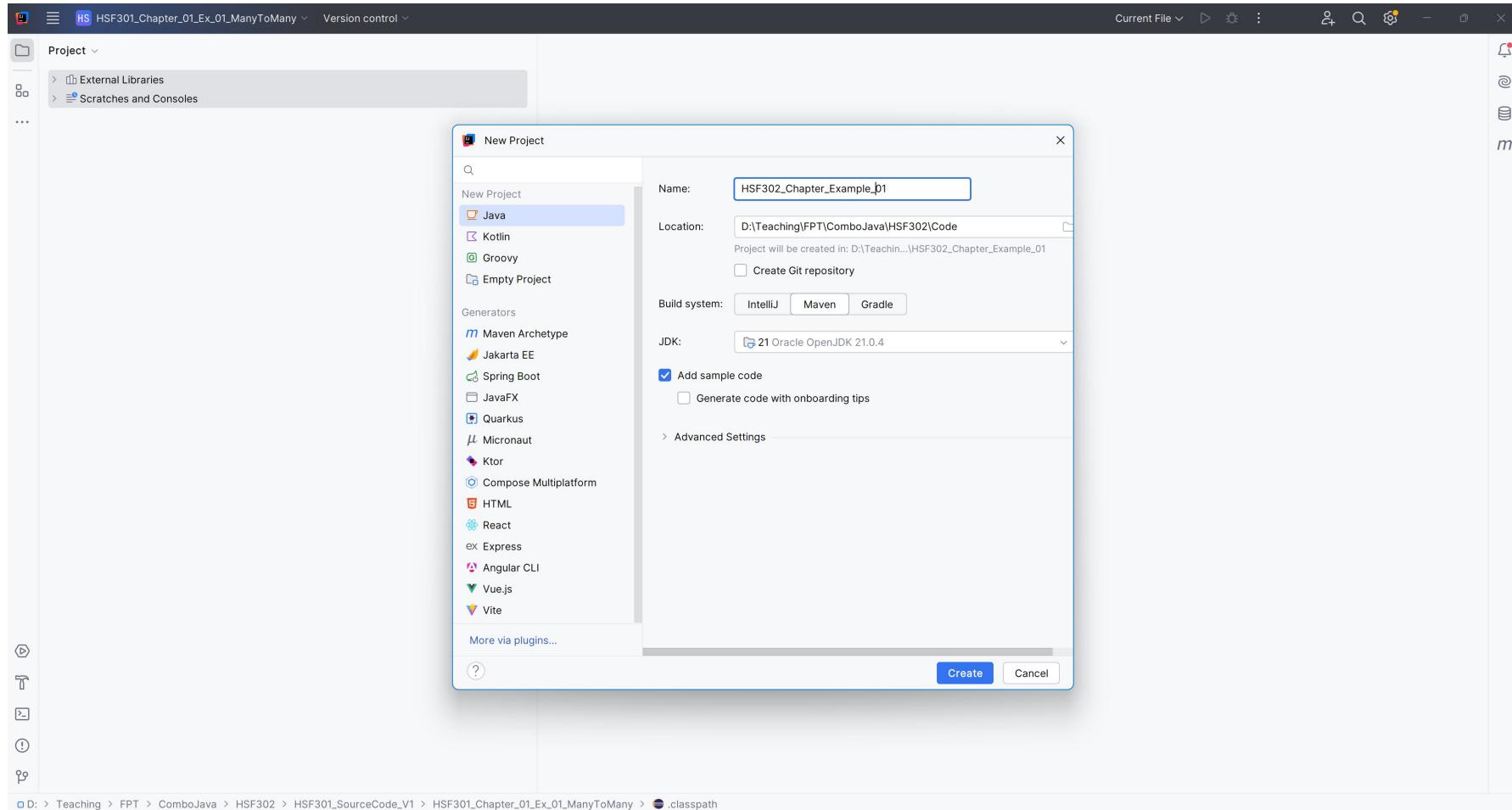
JPA Version	Release Year	Key Features
JPA 1.0	2006	Core ORM functionality: entities, mappings, relationships, inheritance, JPQL, EntityManager
JPA 2.0	2009	Enums, Criteria API, embeddable collections, derived properties, validation, metamodel
JPA 2.1	2013	Entity graphs, converters, stored procedures, Java 8 date/time
JPA 2.2	2019	Streamlined bootstrap, Java 9 modules
JPA 3.0	2022	Renamed to Jakarta Persistence, Java records, Java 17 support

Demo Simple CRUD with JPA

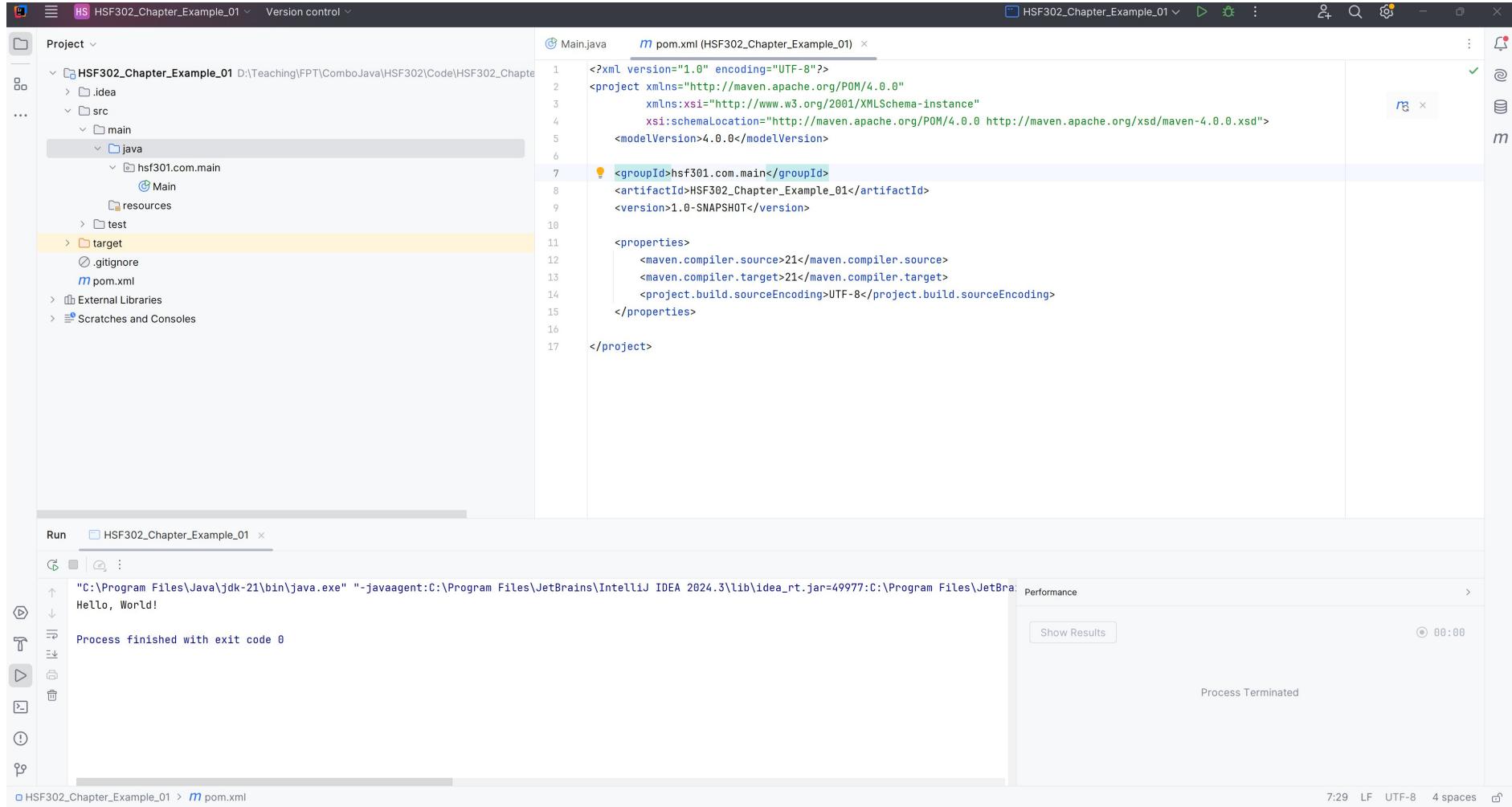
Requirements

- JDK : 21
- JPA : 3.1
- MSSQL : 11.2.3
- Hibernate: 6.5.2

Open IntelliJ, File | New | Maven Project



The New Project



HS FSF302_Chapter_Example_01 Version control

Main.java pom.xml (HSF302_Chapter_Example_01)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>hsf301.com.main</groupId>
<artifactId>HSF302_Chapter_Example_01</artifactId>
<version>1.0-SNAPSHOT</version>
<properties>
    <maven.compiler.source>21</maven.compiler.source>
    <maven.compiler.target>21</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
</project>
```

Project HSF302_Chapter_Example_01 D:\Teaching\FPT\ComboJava\HSF302\Code\HSF302_Chapter_Example_01

- .idea
- src
 - main
 - java
 - hsf301.com.main
 - Main
 - resources
 - test
 - target
 - .gitignore
 - pom.xml
- External Libraries
- Scratches and Consoles

Run HSF302_Chapter_Example_01

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2024.3\lib\idea_rt.jar=49977:C:\Program Files\JetBrains\IntelliJ IDEA 2024.3\bin" -Dfile.encoding=UTF-8 Hello, World!
```

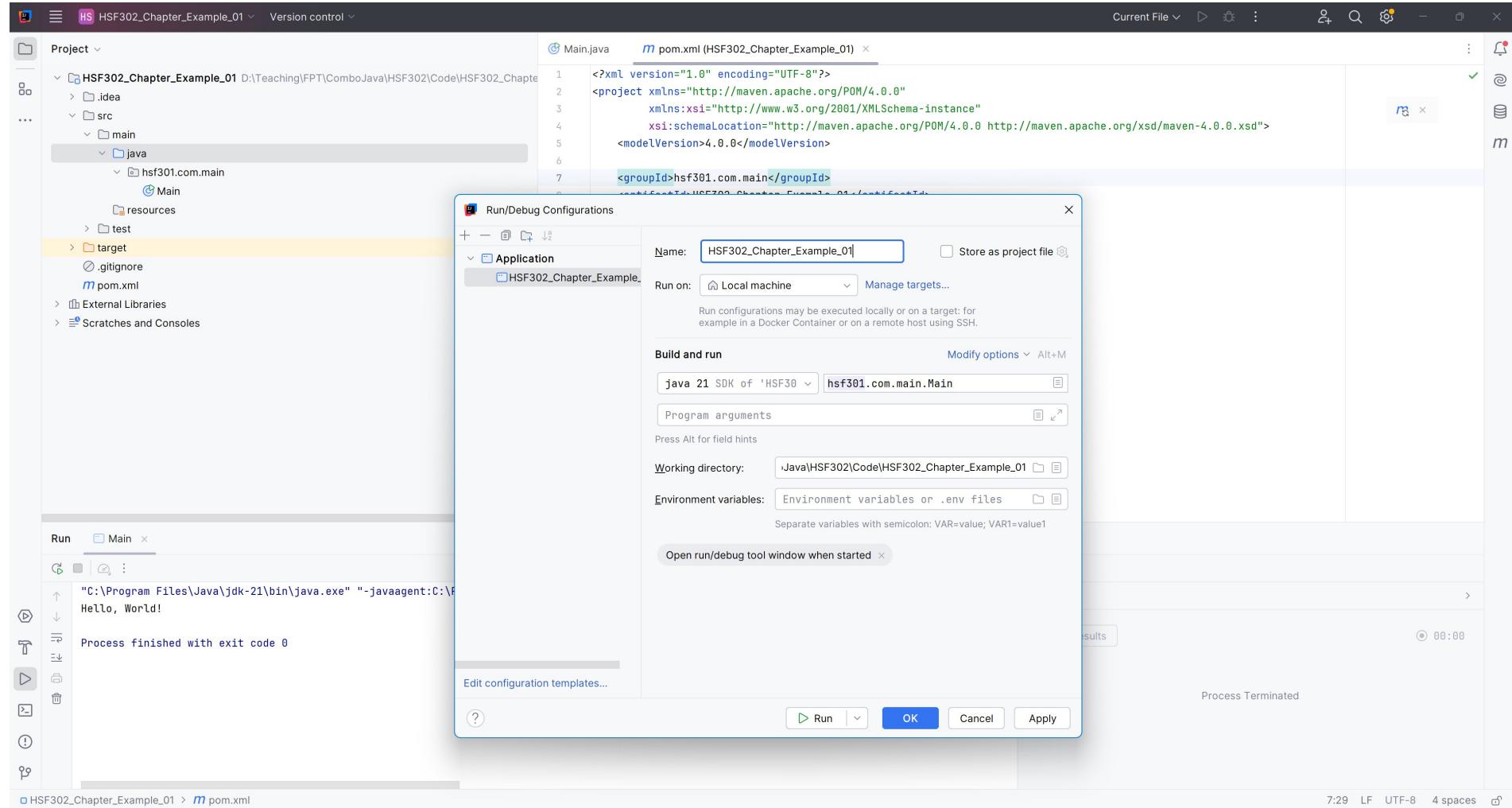
Process finished with exit code 0

Show Results 00:00 Performance

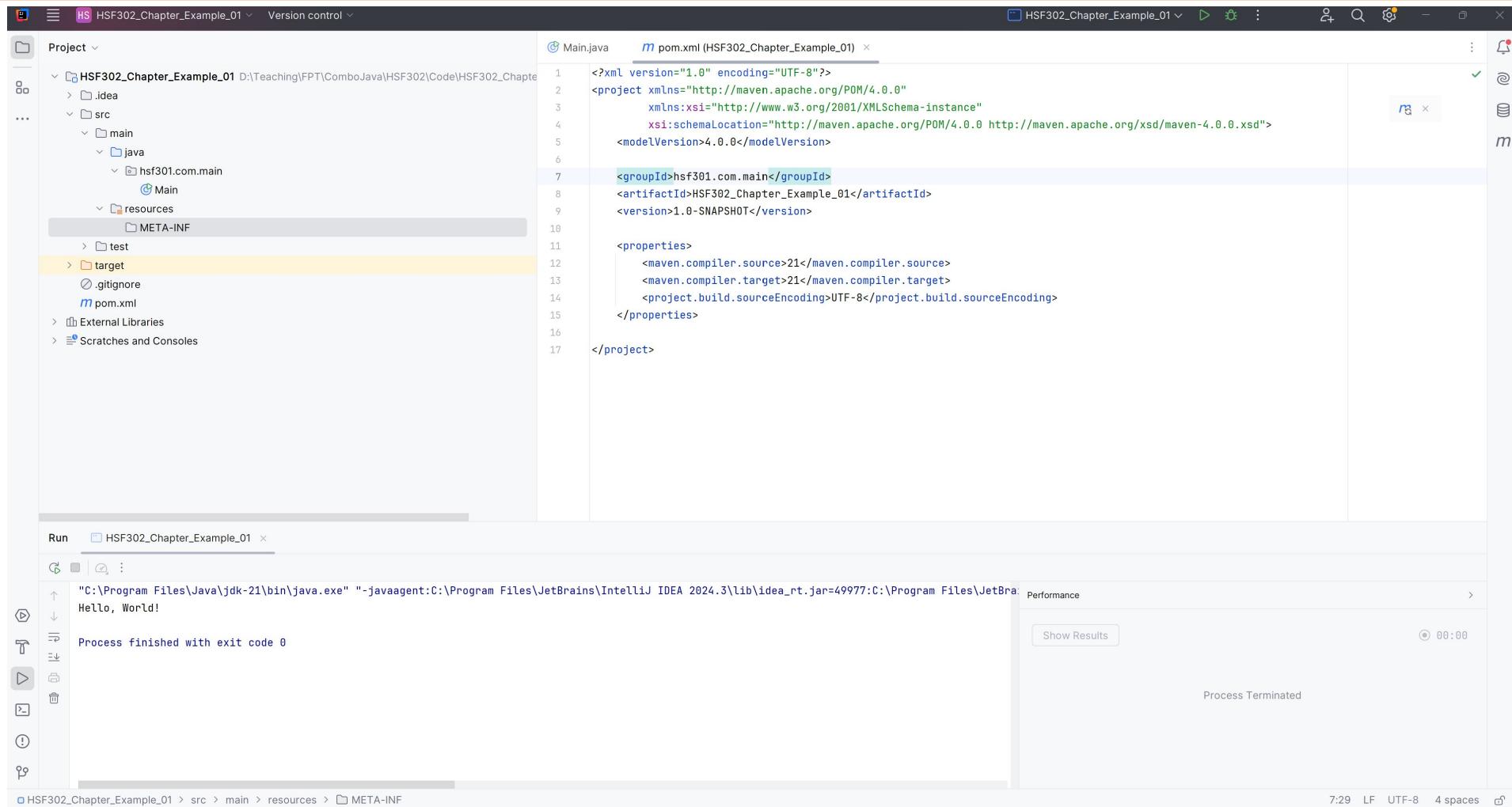
Process Terminated 7:29 LF UTF-8 4 spaces

HSF302_Chapter_Example_01 > pom.xml

Add Run Configurations



Create folder META-INF in src/main/resources

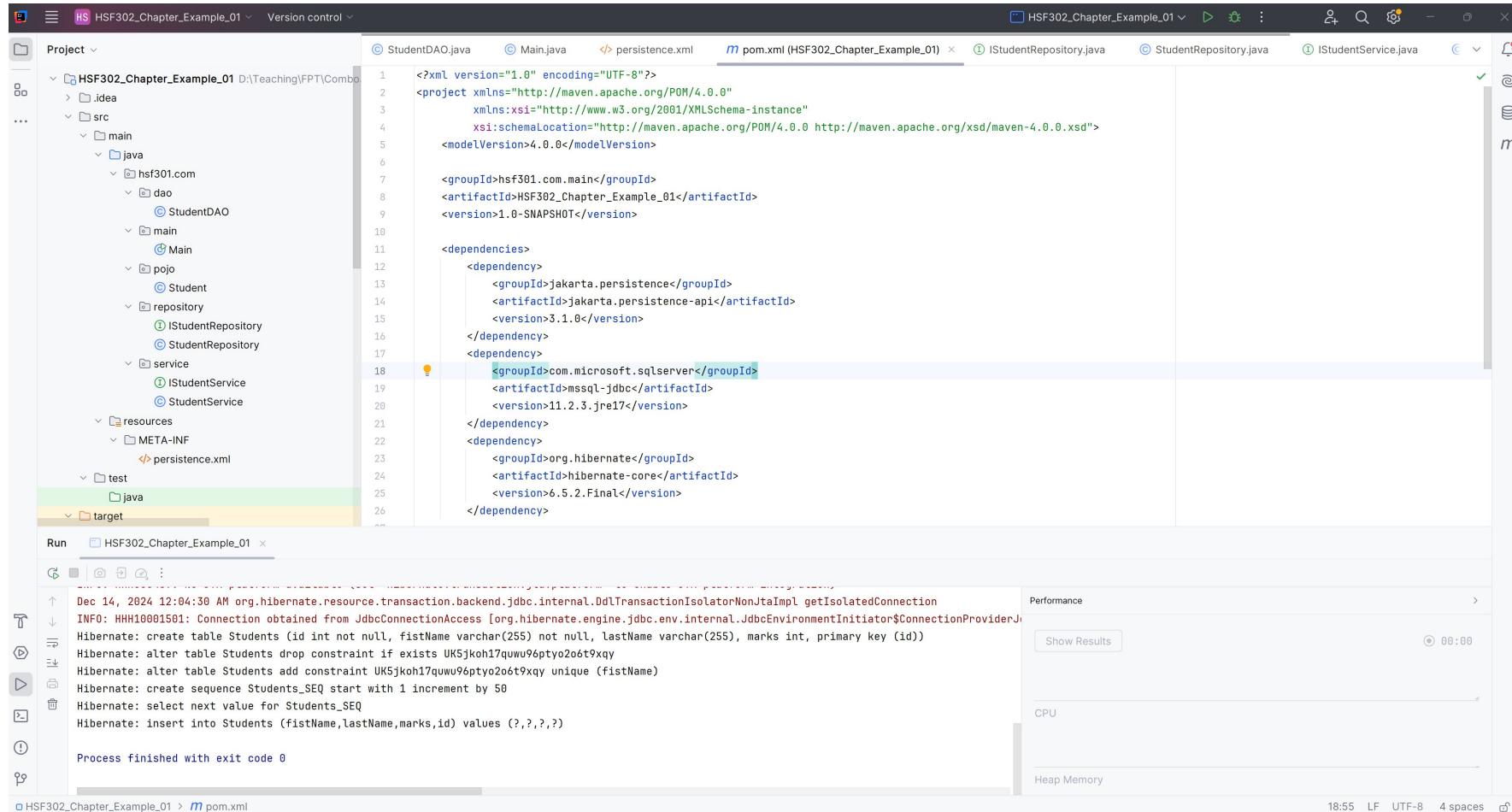


The screenshot shows the IntelliJ IDEA interface with a Maven project named "HSF302_Chapter_Example_01". The project structure on the left includes a .idea folder, a src directory containing main and test subfolders, and a target folder. Inside main/java, there is a Main.java file and a package named hsf301.com.main with a Main class. The resources folder contains a META-INF directory. The pom.xml file is open in the editor, showing the following XML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>hsf301.com.main</groupId>
    <artifactId>HSF302_Chapter_Example_01</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
</project>
```

The Run tab at the bottom shows the command "C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2024.3\lib\idea_rt.jar=49977:C:\Program Files\JetBrains\IntelliJ IDEA 2024.3\bin" was executed, resulting in the output "Hello, World!" and a process exit code of 0. The status bar at the bottom right indicates the time is 7:29, LF, UTF-8, 4 spaces, and the file size is 0.

Edit the pom.xml



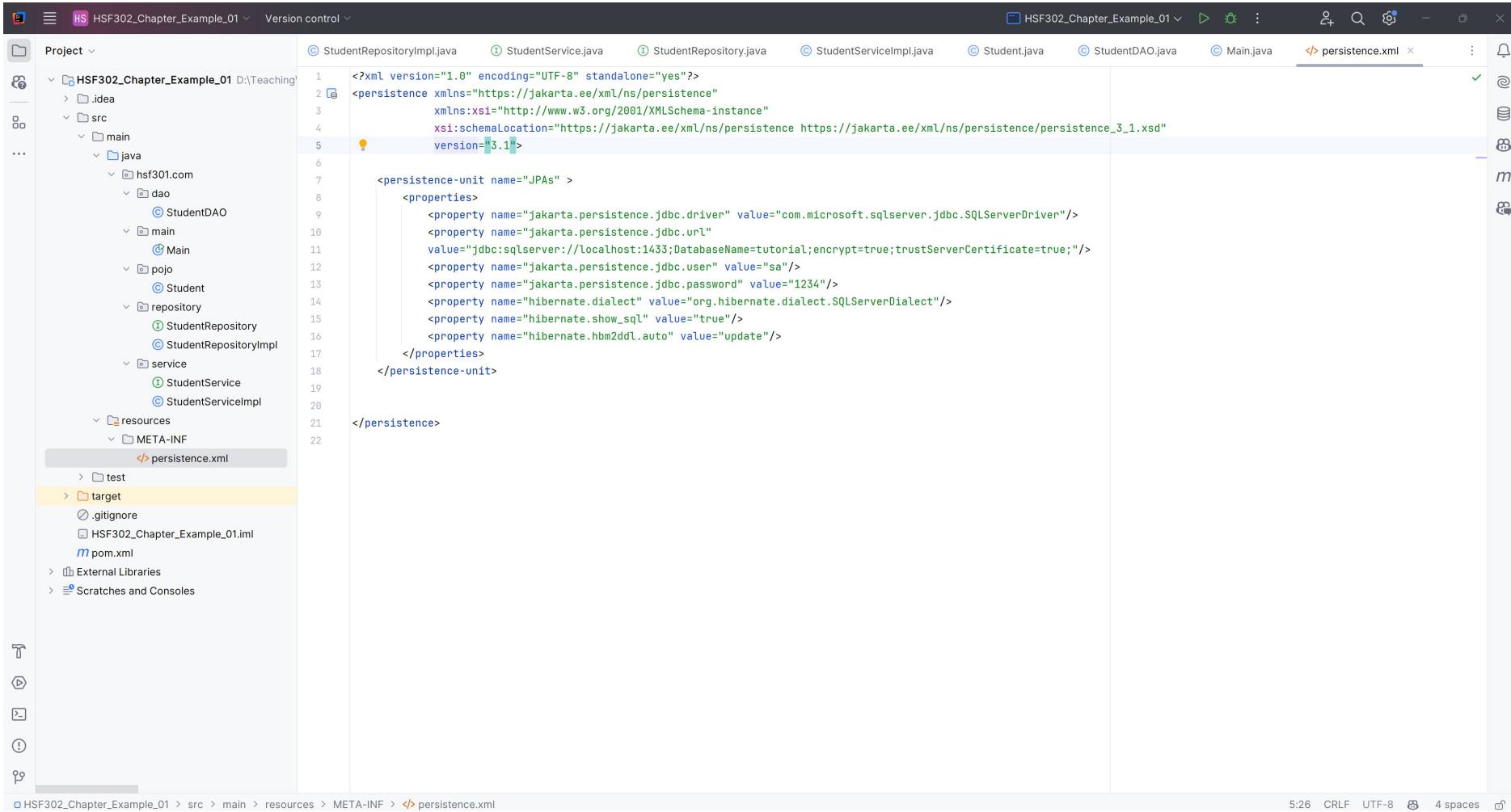
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>hsf301.com.main</groupId>
<artifactId>HSF302_Chapter_Example_01</artifactId>
<version>1.0-SNAPSHOT</version>

<dependencies>
    <dependency>
        <groupId>jakarta.persistence</groupId>
        <artifactId>jakarta.persistence-api</artifactId>
        <version>3.1.0</version>
    </dependency>
    <dependency>
        <groupId>com.microsoft.sqlserver</groupId>
        <artifactId>mssql-jdbc</artifactId>
        <version>11.2.3.jre17</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.5.2.Final</version>
    </dependency>
</dependencies>
```

Dec 14, 2024 12:04:30 AM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator\$ConnectionProviderJdbcConnectionAccess@53333333]
Hibernate: create table Students (id int not null, fistName varchar(255) not null, lastName varchar(255), marks int, primary key (id))
Hibernate: alter table Students drop constraint if exists UK5jkoh17quwu96ptyo2o6t9xqy
Hibernate: alter table Students add constraint UK5jkoh17quwu96ptyo2o6t9xqy unique (fistName)
Hibernate: create sequence Students_SEQ start with 1 increment by 50
Hibernate: select next value for Students_SEQ
Hibernate: insert into Students (fistName,lastName,marks,id) values (?, ?, ?, ?)
Process finished with exit code 0

Create the persistence.xml in META-INF folder



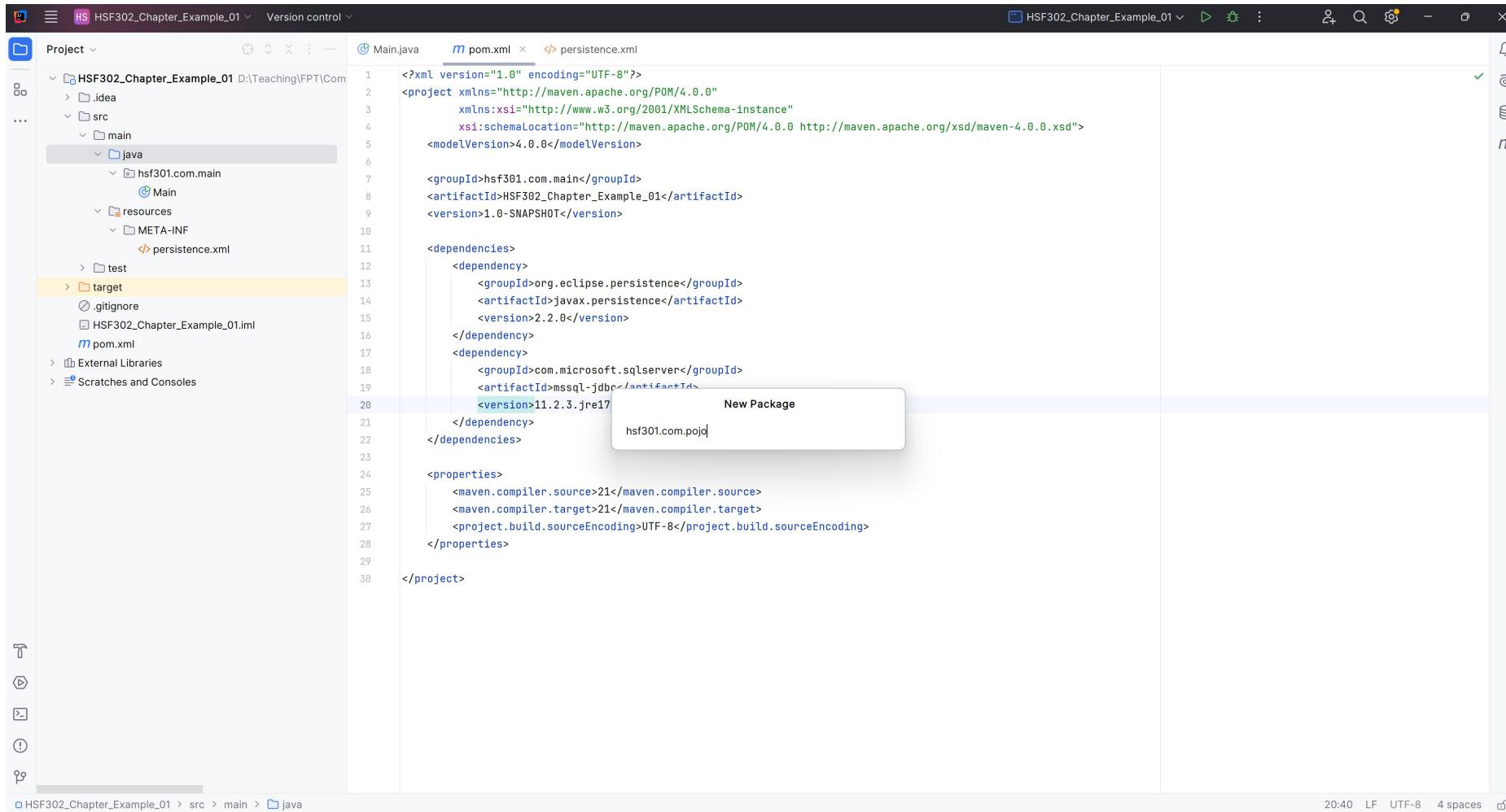
The screenshot shows a Java IDE interface with a project named "HSF302_Chapter_Example_01". The left sidebar displays the project structure, including a "META-INF" folder containing a "persistence.xml" file. The main editor window shows the XML content of "persistence.xml". The XML defines a persistence unit named "JPAs" with properties for a Microsoft SQL Server database, including driver, URL, user, and password. The code is color-coded for syntax highlighting.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence https://jakarta.ee/xml/ns/persistence/persistence_3_1.xsd"
    version="3.1">

    <persistence-unit name="JPAs" >
        <properties>
            <property name="jakarta.persistence.jdbc.driver" value="com.microsoft.sqlserver.jdbc.SQLServerDriver"/>
            <property name="jakarta.persistence.jdbc.url"
                value="jdbc:sqlserver://localhost:1433;databaseName=tutorial;encrypt=true;trustServerCertificate=true;"/>
            <property name="jakarta.persistence.jdbc.user" value="sa"/>
            <property name="jakarta.persistence.jdbc.password" value="1234"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.SQLServerDialect"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
        </properties>
    </persistence-unit>

</persistence>
```

Add hsf301.com.pojo Package in src/main/java



The screenshot shows the IntelliJ IDEA interface with the project 'HSF302_Chapter_Example_01' open. The left sidebar displays the project structure, including the 'src' directory with 'main' and 'java' sub-directories. The 'java' directory contains 'hsf301.com.main' and 'resources'. The 'resources' directory has a 'META-INF' folder containing 'persistence.xml'. The right pane shows the 'pom.xml' file being edited. A code completion tooltip is visible over the XML code, suggesting the addition of a new dependency. The tooltip text is 'New Package' and the suggested package name is 'hsf301.com.pojo'.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

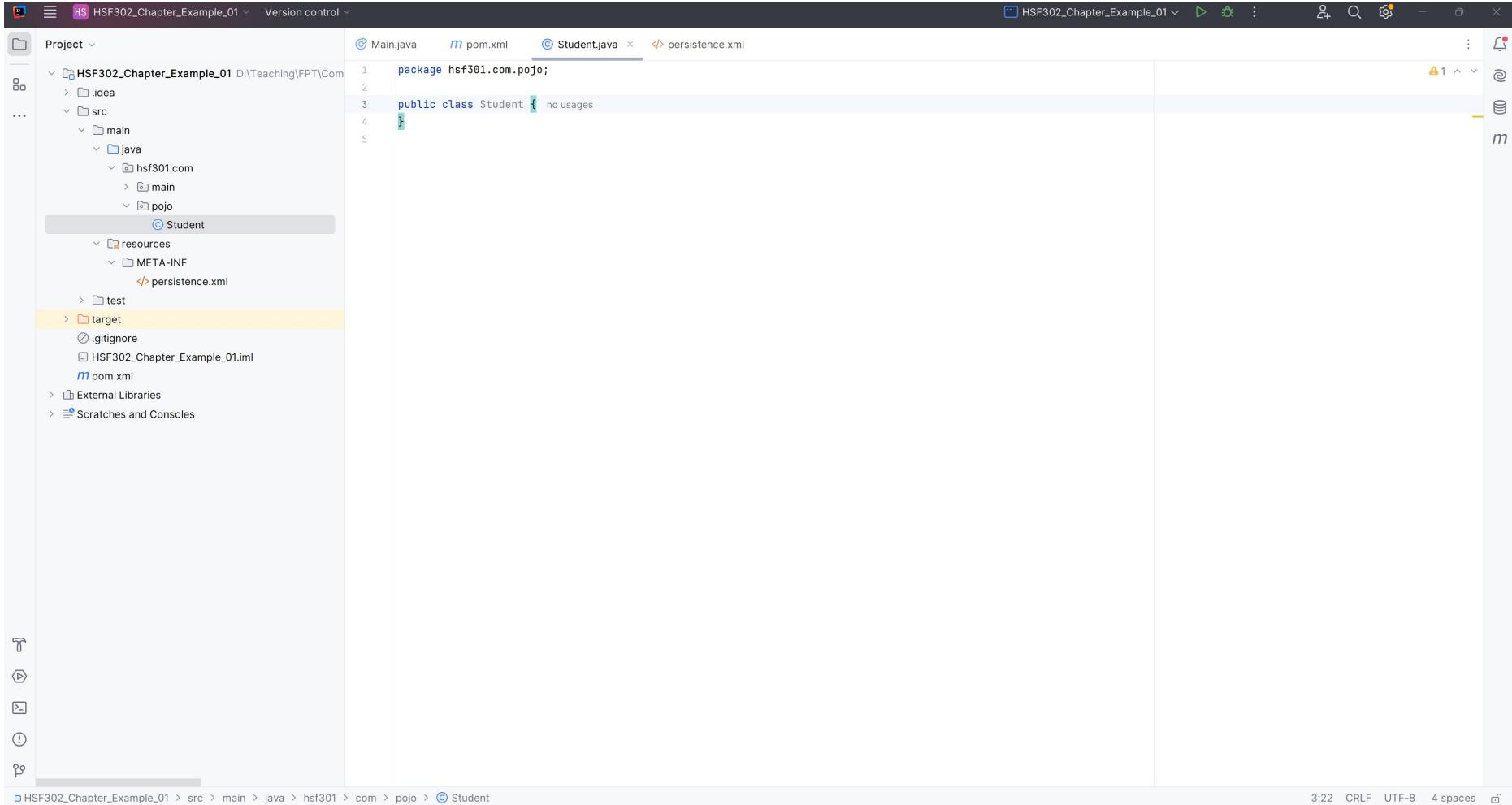
    <groupId>hsf301.com.main</groupId>
    <artifactId>HSF302_Chapter_Example_01</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>javax.persistence</artifactId>
            <version>2.2.0</version>
        </dependency>
        <dependency>
            <groupId>com.microsoft.sqlserver</groupId>
            <artifactId>mssql-jdbc</artifactId>
            <version>11.2.3.jre17</version>
        </dependency>
    </dependencies>

    <properties>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

</project>
```

Add Student.java in hsf301.com.pojo package

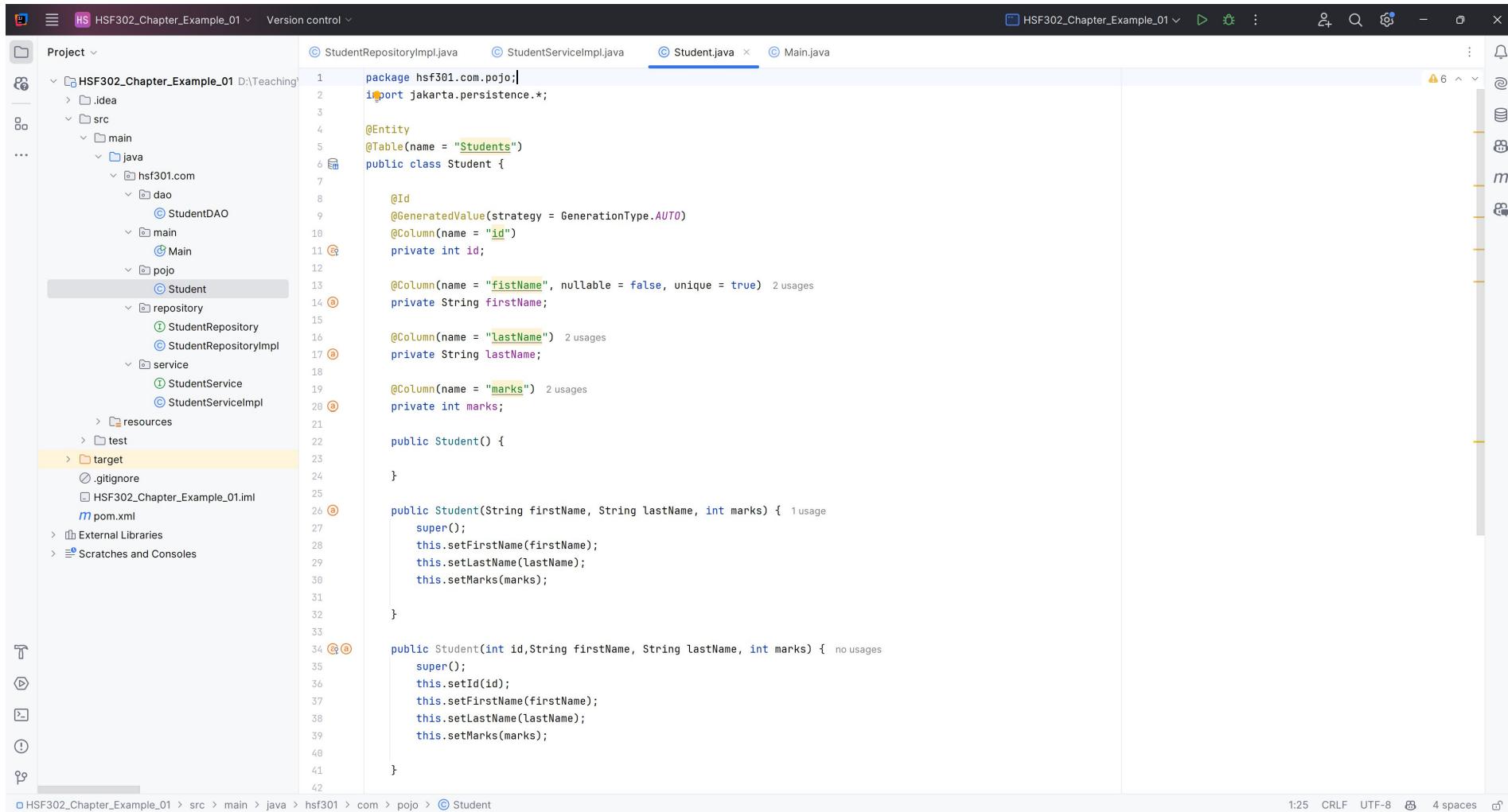


The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The left sidebar shows the project structure under "HSF302_Chapter_Example_01". It includes a "src" directory containing "main" and "test" packages, and a "target" directory.
- Code Editor:** The main window displays the "Student.java" file in the "hsf301.com.pojo" package. The code is as follows:

```
1 package hsf301.com.pojo;
2
3 public class Student { no usages }
```
- Toolbars and Status Bar:** The top bar shows tabs for Main.java, pom.xml, and Student.java. The status bar at the bottom right shows the time as 3:22, encoding as CRLF, character set as UTF-8, and indentation as 4 spaces.

Edit the Student.java



The screenshot shows the IntelliJ IDEA interface with the project 'HSF302_Chapter_Example_01' open. The 'Student.java' file is selected in the editor tab bar. The code editor displays the following Java code:

```
package hsf301.com.pojo;
import jakarta.persistence.*;

@Entity
@Table(name = "Students")
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private int id;

    @Column(name = "firstName", nullable = false, unique = true) 2 usages
    private String firstName;

    @Column(name = "lastName") 2 usages
    private String lastName;

    @Column(name = "marks") 2 usages
    private int marks;

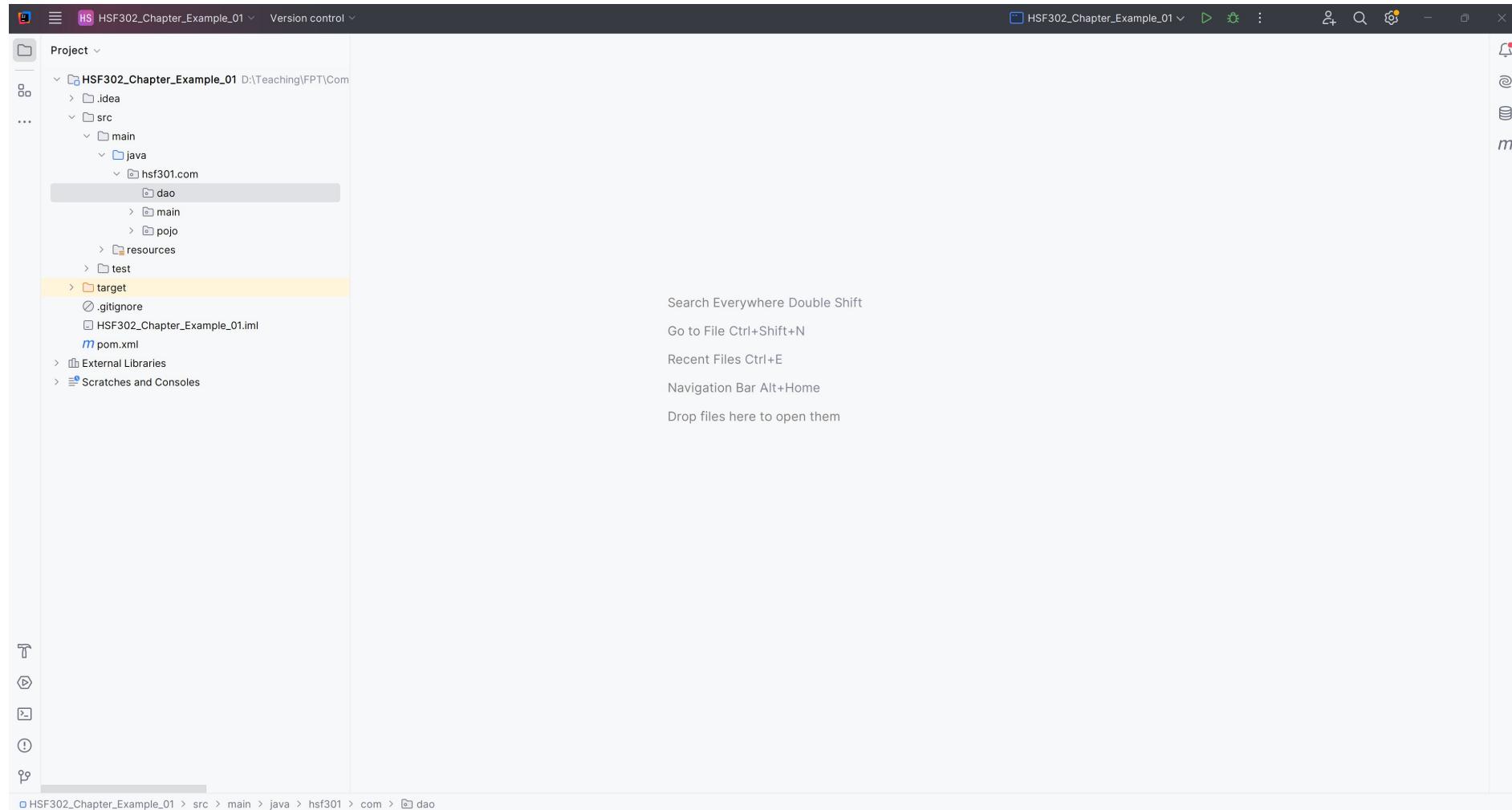
    public Student() {
    }

    public Student(String firstName, String lastName, int marks) { 1 usage
        super();
        this.setFirstName(firstName);
        this.setLastName(lastName);
        this.setMarks(marks);
    }

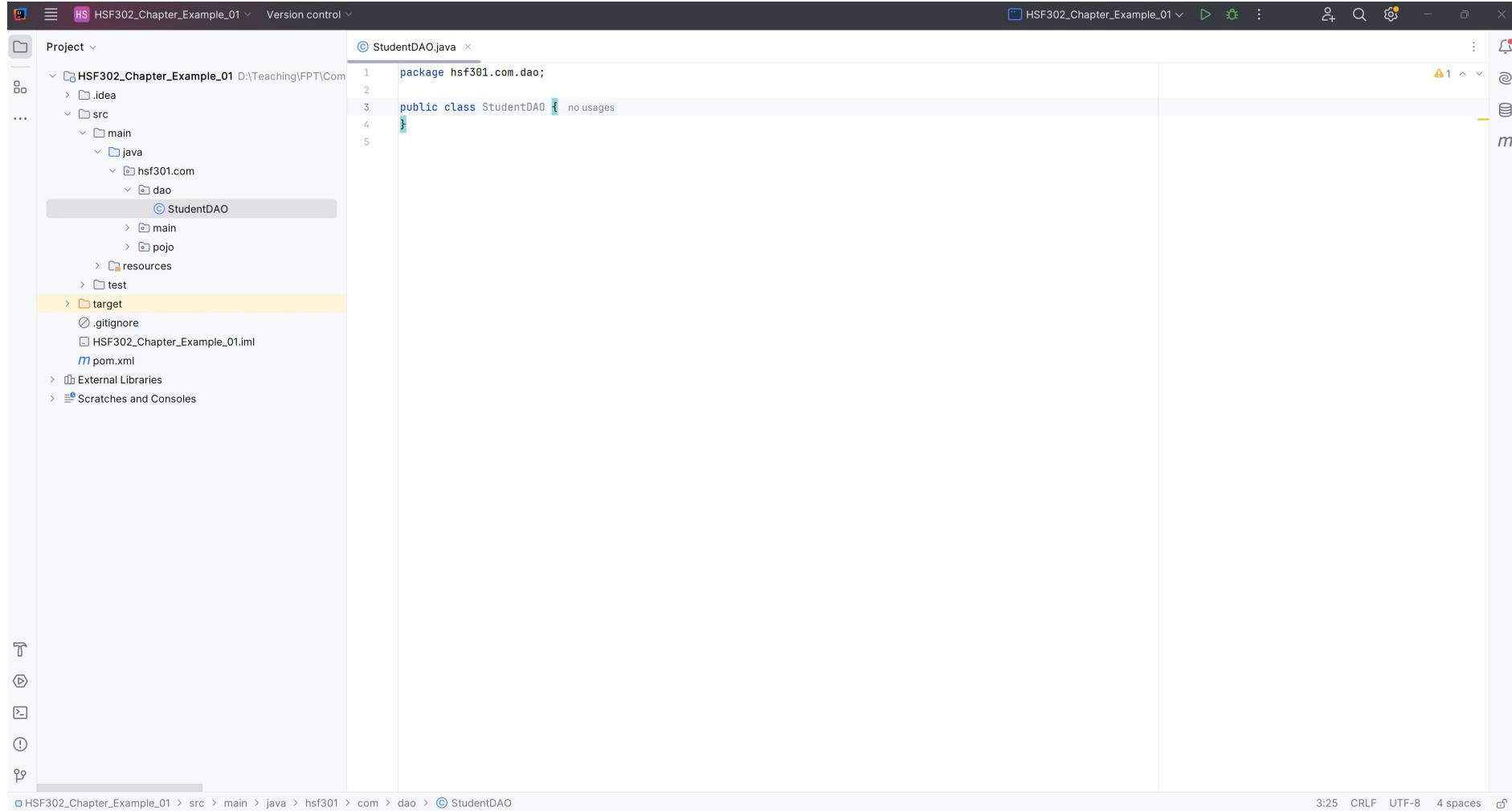
    public Student(int id, String firstName, String lastName, int marks) { no usages
        super();
        this.setId(id);
        this.setFirstName(firstName);
        this.setLastName(lastName);
        this.setMarks(marks);
    }
}
```

The code editor includes syntax highlighting for Java keywords and annotations, and the IntelliJ UI with toolbars and a vertical status bar at the bottom.

Add hsf301.com.dao Package in src/main/java



Create StudentDAO in hsf301.com.dao



The screenshot shows the IntelliJ IDEA interface with a Java project named "HSF302_Chapter_Example_01". The project structure on the left includes a "src" directory containing "main", "java", and "hsf301.com" packages, which further contain "dao" and "pojo" sub-packages. A "StudentDAO" class is selected in the "dao" package. The main editor window displays the following code:

```
1 package hsf301.com.dao;
2
3 public class StudentDAO { no usages
4 }
```

The status bar at the bottom indicates the file is 3:25 CRLF UTF-8 4 spaces.

Edit the StudentDAO in hsf301.com.dao

The screenshot shows an IDE interface with a project structure on the left and a code editor on the right.

Project Tree:

- HSF302_Chapter_Example_01
- src
 - main
 - java
 - hsf301.com
 - dao
 - StudentDAO
 - main
 - Main
 - pojo
 - Student
 - repository
 - StudentRepository
 - StudentRepositoryImpl
 - service
 - StudentService
 - StudentServiceImpl
 - resources
 - test
 - target
 - .gitignore
 - HSF302_Chapter_Example_01.iml
 - pom.xml
 - External Libraries
 - Scratches and Consoles

Code Editor (StudentDAO.java):

```
package hsf301.com.dao;

import hsf301.com.pojo.Student;
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;

import java.util.List;

public class StudentDAO {

    private static EntityManager em; 20 usages
    private static EntityManagerFactory emf; 6 usages

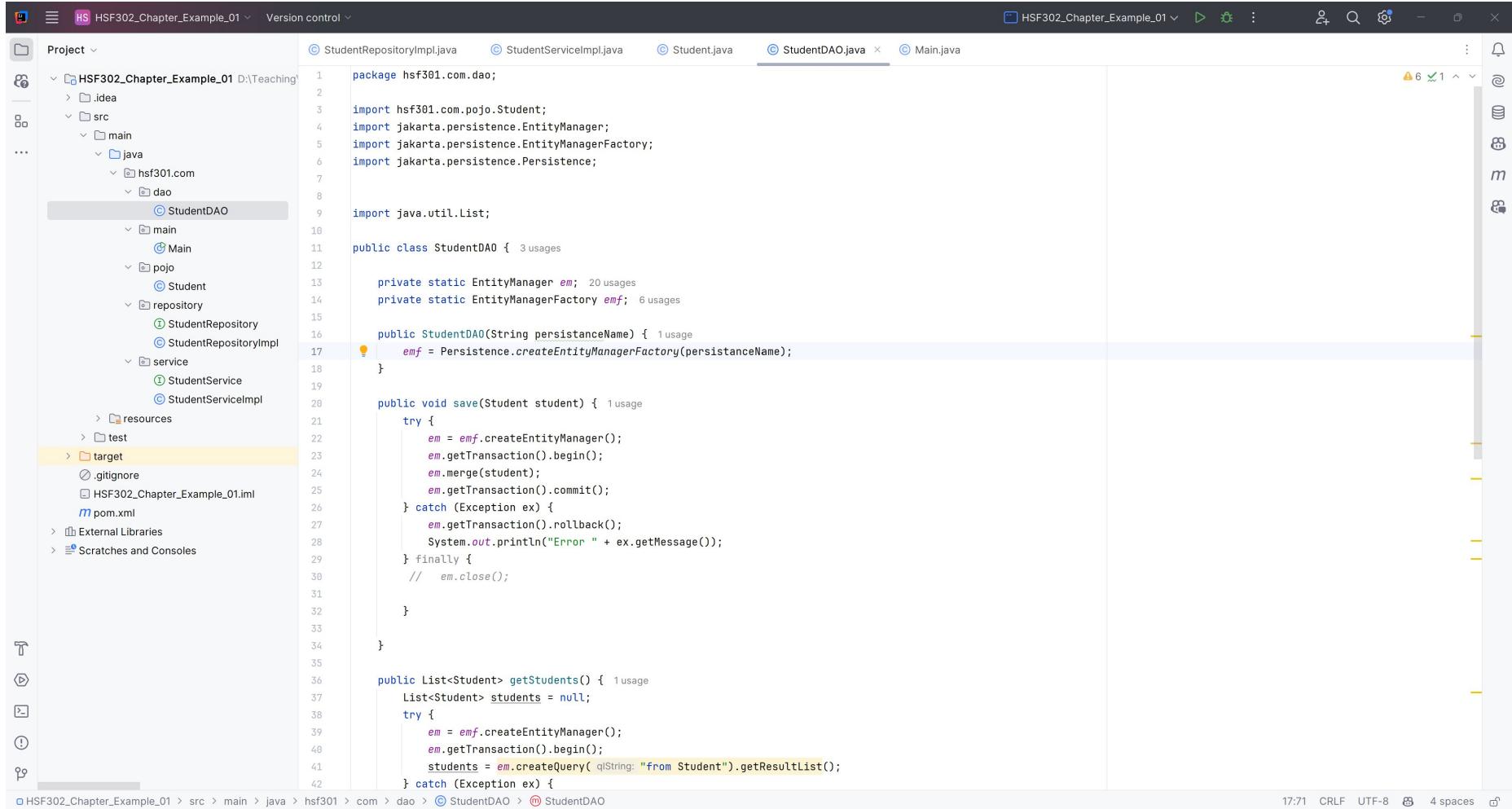
    public StudentDAO(String persistanceName) { 1 usage
        emf = Persistence.createEntityManagerFactory(persistanceName);
    }

    public void save(Student student) { 1 usage
        try {
            em = emf.createEntityManager();
            em.getTransaction().begin();
            em.merge(student);
            em.getTransaction().commit();
        } catch (Exception ex) {
            em.getTransaction().rollback();
            System.out.println("Error " + ex.getMessage());
        } finally {
            // em.close();
        }
    }

    public List<Student> getStudents() { 1 usage
        List<Student> students = null;
        try {
            em = emf.createEntityManager();
            em.getTransaction().begin();
            students = em.createQuery("from Student").getResultList();
        } catch (Exception ex) {
    
```

Status Bar: 17:11 CRLF UTF-8 4 spaces

Save Student Method



The screenshot shows the IntelliJ IDEA interface with the project `HSF302_Chapter_Example_01` open. The `StudentDAO.java` file is selected in the editor tab bar. The code implements a DAO interface for saving students using Entity Manager.

```
package hsf301.com.dao;

import hsf301.com.pojo.Student;
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;

import java.util.List;

public class StudentDAO {

    private static EntityManager em; 20 usages
    private static EntityManagerFactory emf; 6 usages

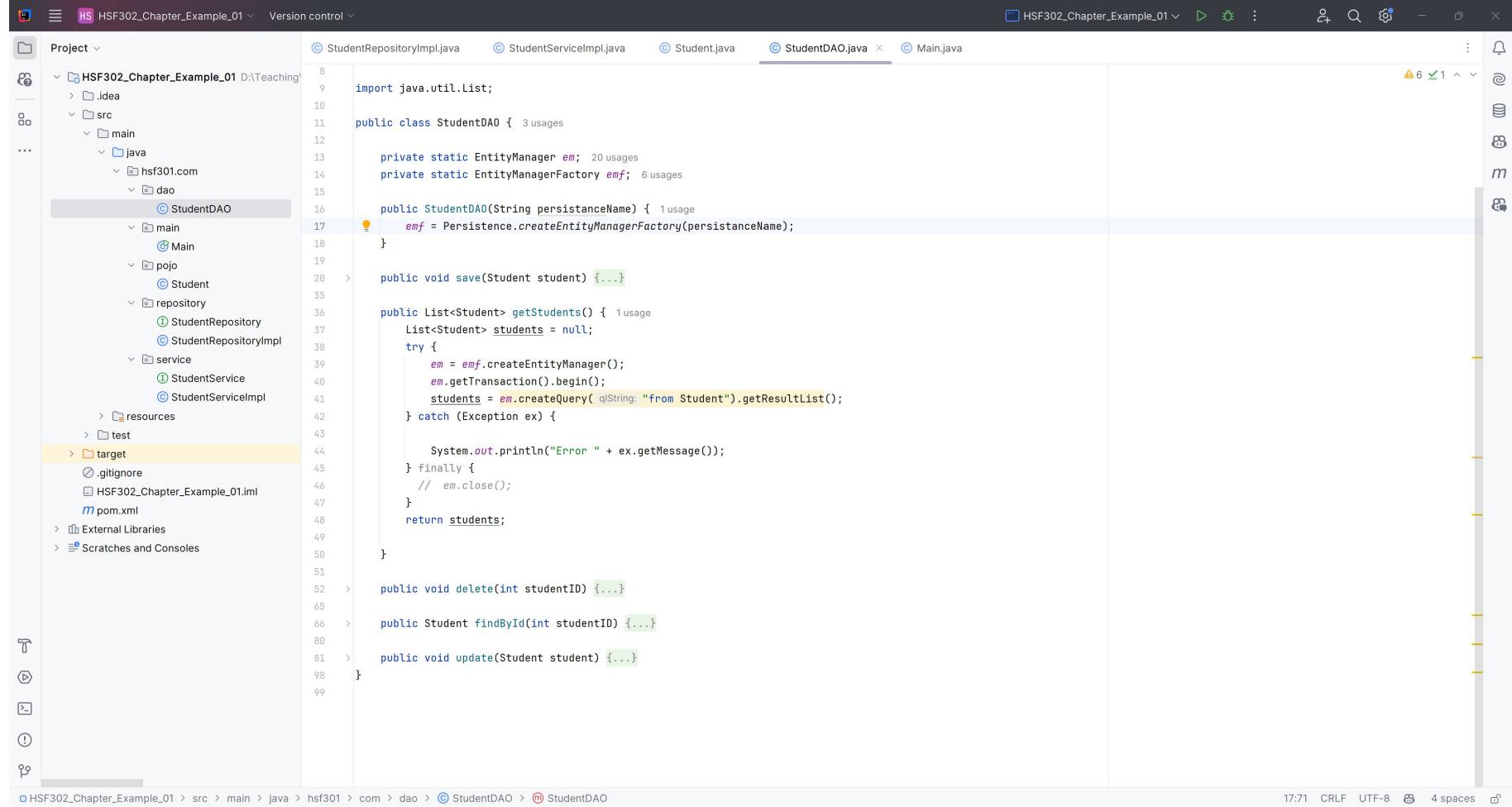
    public StudentDAO(String persistanceName) { 1 usage
        emf = Persistence.createEntityManagerFactory(persistanceName);
    }

    public void save(Student student) { 1 usage
        try {
            em = emf.createEntityManager();
            em.getTransaction().begin();
            em.merge(student);
            em.getTransaction().commit();
        } catch (Exception ex) {
            em.getTransaction().rollback();
            System.out.println("Error " + ex.getMessage());
        } finally {
            // em.close();
        }
    }

    public List<Student> getStudents() { 1 usage
        List<Student> students = null;
        try {
            em = emf.createEntityManager();
            em.getTransaction().begin();
            students = em.createQuery("from Student").getResultList();
        } catch (Exception ex) {
    
```

The code uses EntityManager and EntityManagerFactory from the jakarta.persistence package. It includes a constructor that initializes the EntityManagerFactory. The `save` method starts a transaction, merges the student entity, and commits it. The `getStudents` method queries all student entities from the database.

Get All Students Method



The screenshot shows the IntelliJ IDEA interface with the project `HSF302_Chapter_Example_01` open. The `StudentDAO.java` file is selected in the editor tab bar. The code implements a DAO interface for managing students.

```
import java.util.List;

public class StudentDAO {    3 usages

    private static EntityManager em;    20 usages
    private static EntityManagerFactory emf;    6 usages

    public StudentDAO(String persistanceName) {    1 usage
        emf = Persistence.createEntityManagerFactory(persistanceName);
    }

    public void save(Student student) {...}

    public List<Student> getStudents() {    1 usage
        List<Student> students = null;
        try {
            em = emf.createEntityManager();
            em.getTransaction().begin();
            students = em.createQuery("from Student").getResultList();
        } catch (Exception ex) {

            System.out.println("Error " + ex.getMessage());
        } finally {
            // em.close();
        }
        return students;
    }

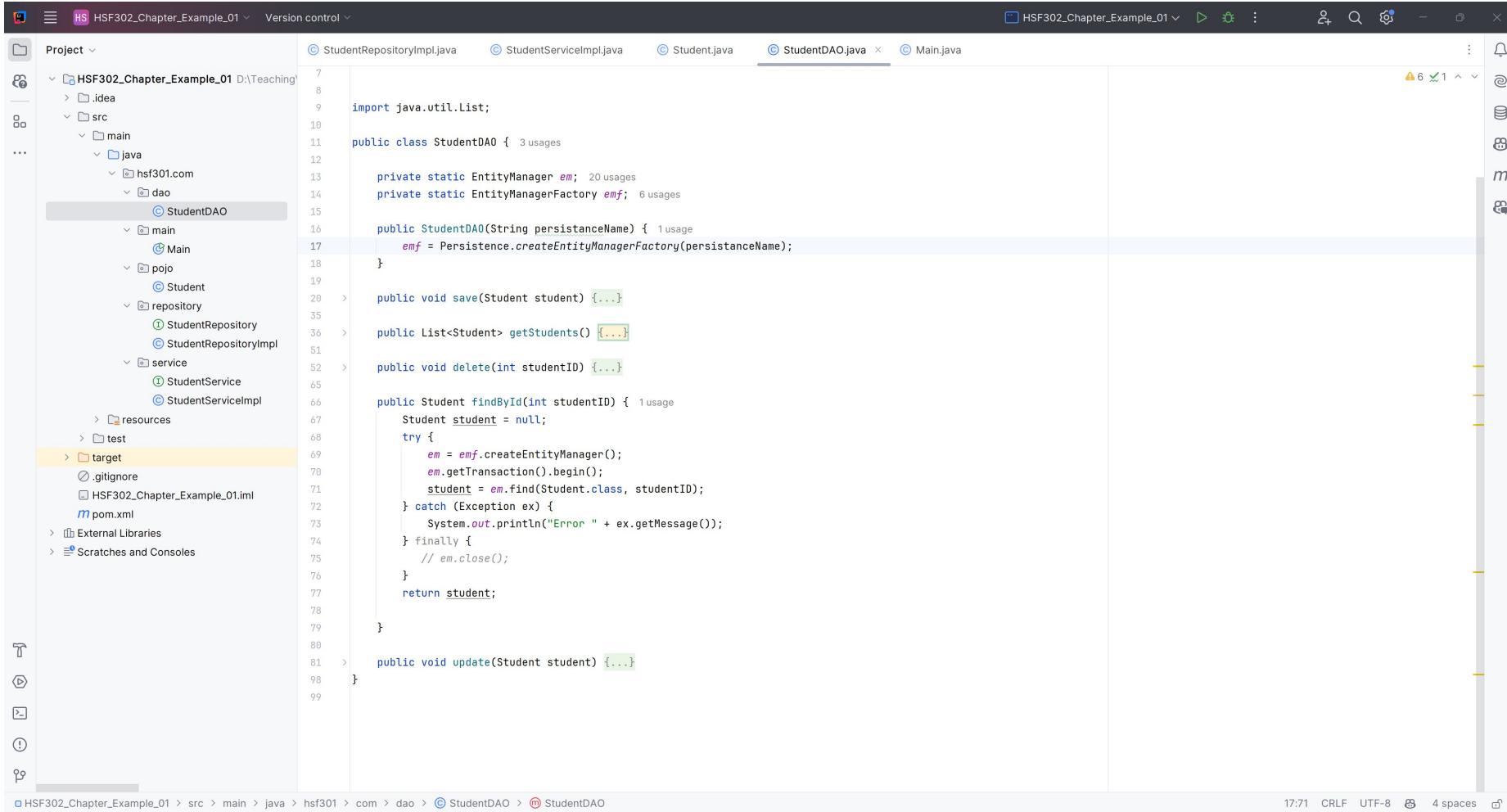
    public void delete(int studentID) {...}

    public Student findById(int studentID) {...}

    public void update(Student student) {...}
}
```

The code uses JPA annotations and EntityManager to interact with a database. The `target` folder is highlighted in yellow in the project tree.

Find a Student Method



The screenshot shows an IDE interface with the following details:

- Project:** HSF302_Chapter_Example_01
- File:** StudentDAO.java
- Code Preview:**

```
import java.util.List;

public class StudentDAO {    3 usages

    private static EntityManager em;    20 usages
    private static EntityManagerFactory emf;    6 usages

    public StudentDAO(String persistanceName) {    1 usage
        emf = Persistence.createEntityManagerFactory(persistanceName);
    }

    public void save(Student student) {...}

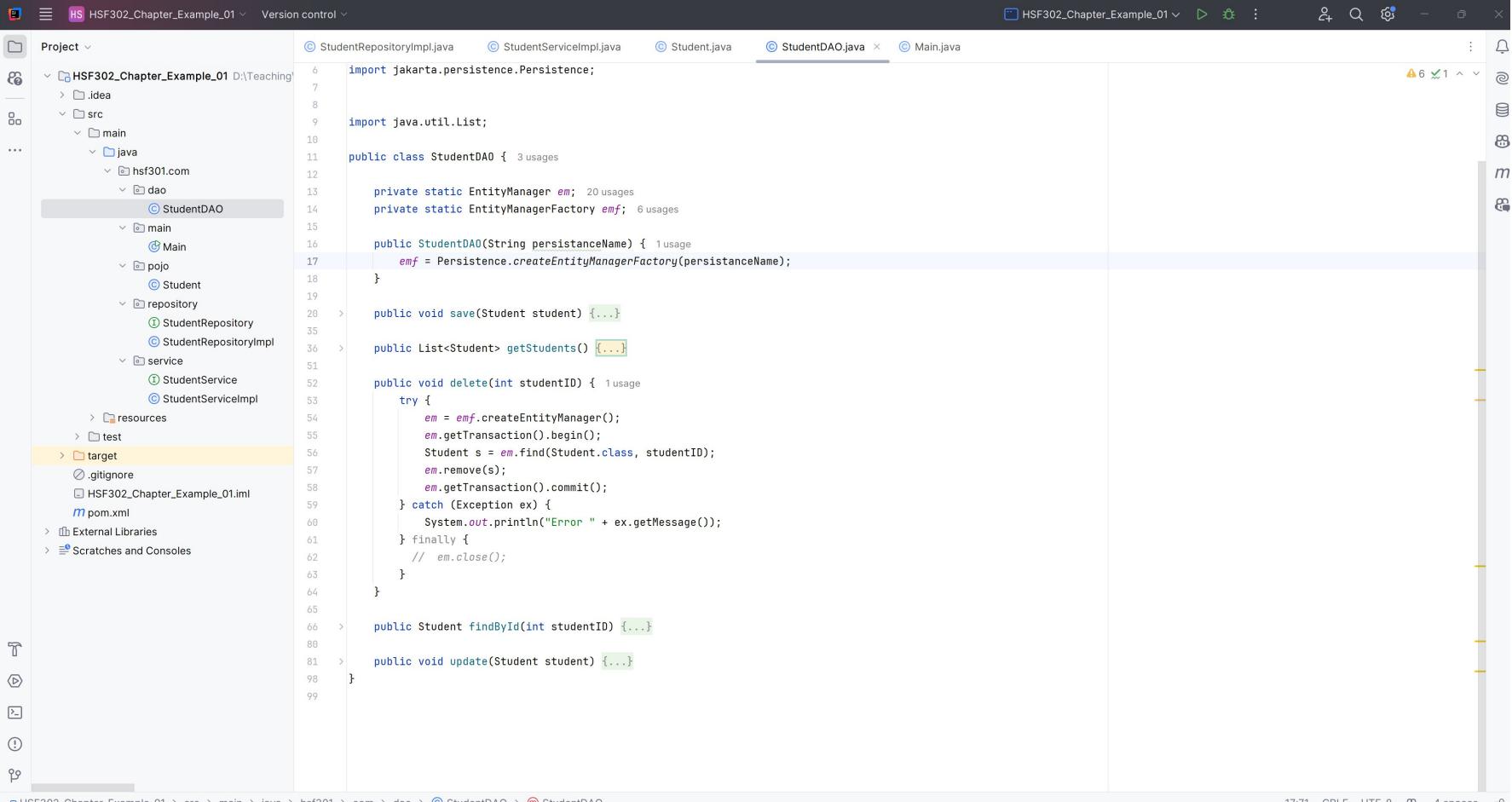
    public List<Student> getStudents() {...}

    public void delete(int studentID) {...}

    public Student findById(int studentID) {    1 usage
        Student student = null;
        try {
            em = emf.createEntityManager();
            em.getTransaction().begin();
            student = em.find(Student.class, studentID);
        } catch (Exception ex) {
            System.out.println("Error " + ex.getMessage());
        } finally {
            // em.close();
        }
        return student;
    }

    public void update(Student student) {...}
}
```
- Toolbars and Status Bar:** Version control, tabs for other files like StudentRepositoryImpl.java, StudentServiceImpl.java, Student.java, Main.java, and Main.java. Status bar at the bottom shows 17:71 CRLF UTF-8 4 spaces.

Delete Student Method



The screenshot shows the IntelliJ IDEA interface with the project 'HSF302_Chapter_Example_01' open. The 'StudentDAO.java' file is selected in the editor tab bar. The code implements a DAO interface for managing student data using EntityManager and EntityManagerFactory.

```
import jakarta.persistence.Persistence;
import java.util.List;

public class StudentDAO { 3 usages

    private static EntityManager em; 20 usages
    private static EntityManagerFactory emf; 6 usages

    public StudentDAO(String persistanceName) { 1 usage
        emf = Persistence.createEntityManagerFactory(persistanceName);
    }

    public void save(Student student) {...}

    public List<Student> getStudents() {...}

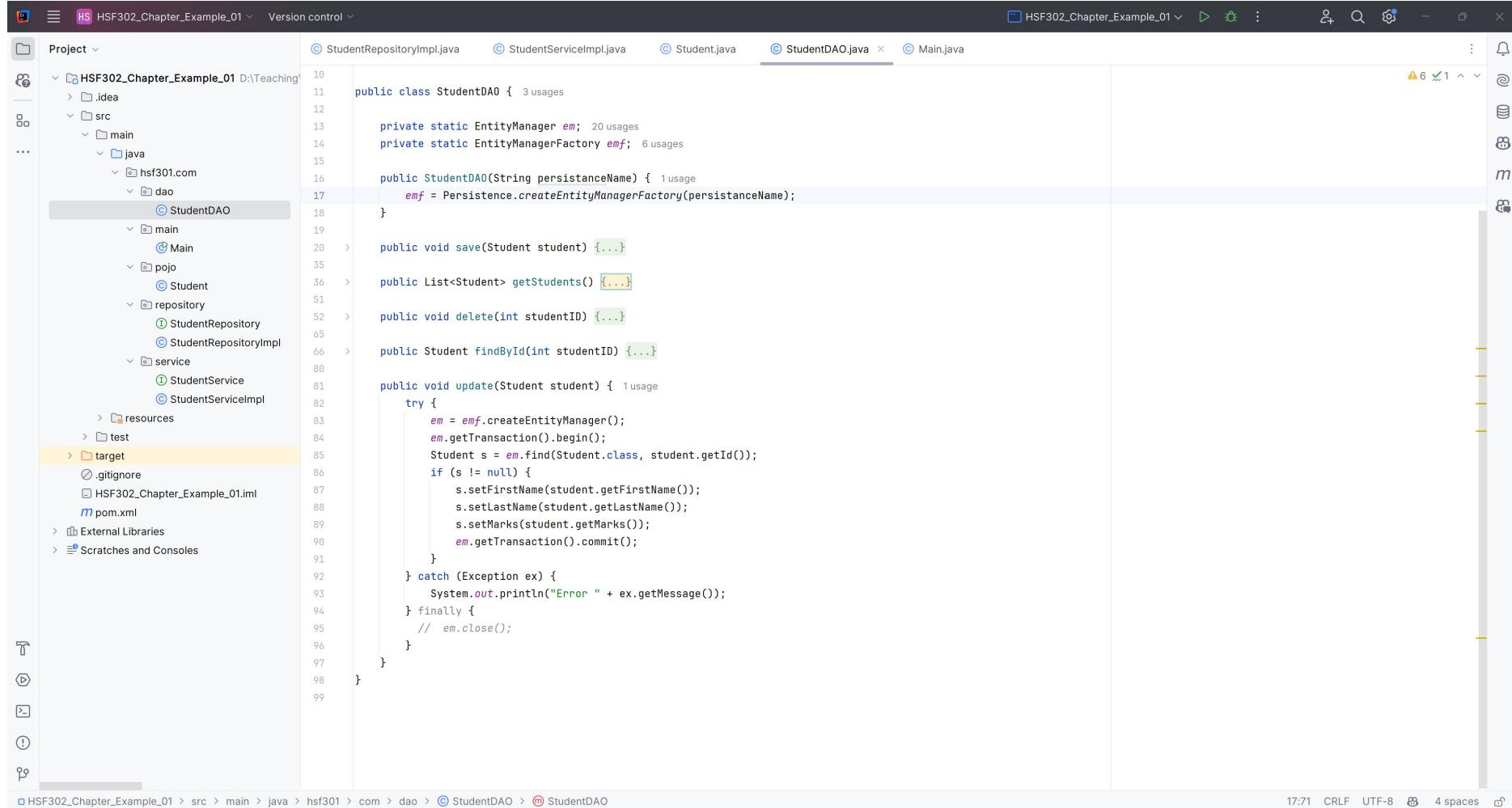
    public void delete(int studentID) { 1 usage
        try {
            em = emf.createEntityManager();
            em.getTransaction().begin();
            Student s = em.find(Student.class, studentID);
            em.remove(s);
            em.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println("Error " + ex.getMessage());
        } finally {
            // em.close();
        }
    }

    public Student findById(int studentID) {...}

    public void update(Student student) {...}
}
```

The code includes imports for jakarta.persistence.Persistence and java.util.List. It defines a constructor that initializes EntityManagerFactory. The save method takes a Student object and persists it. The getStudents method returns a list of Student objects. The delete method removes a student by ID, using a try-finally block to manage the EntityManager. The findById and update methods are also defined but not shown in full.

Update Student Method



The screenshot shows an IDE interface with the following details:

- Project:** HSF302_Chapter_Example_01
- File:** StudentDAO.java
- Code Content:** The code implements a DAO interface for Student entities. It uses EntityManager and EntityManagerFactory from Persistence. The update method performs a transactional update on a student object.

```
public class StudentDAO { 3 usages
    private static EntityManager em; 20 usages
    private static EntityManagerFactory emf; 6 usages

    public StudentDAO(String persistanceName) { 1 usage
        emf = Persistence.createEntityManagerFactory(persistanceName);
    }

    public void save(Student student) {...}

    public List<Student> getStudents() {...}

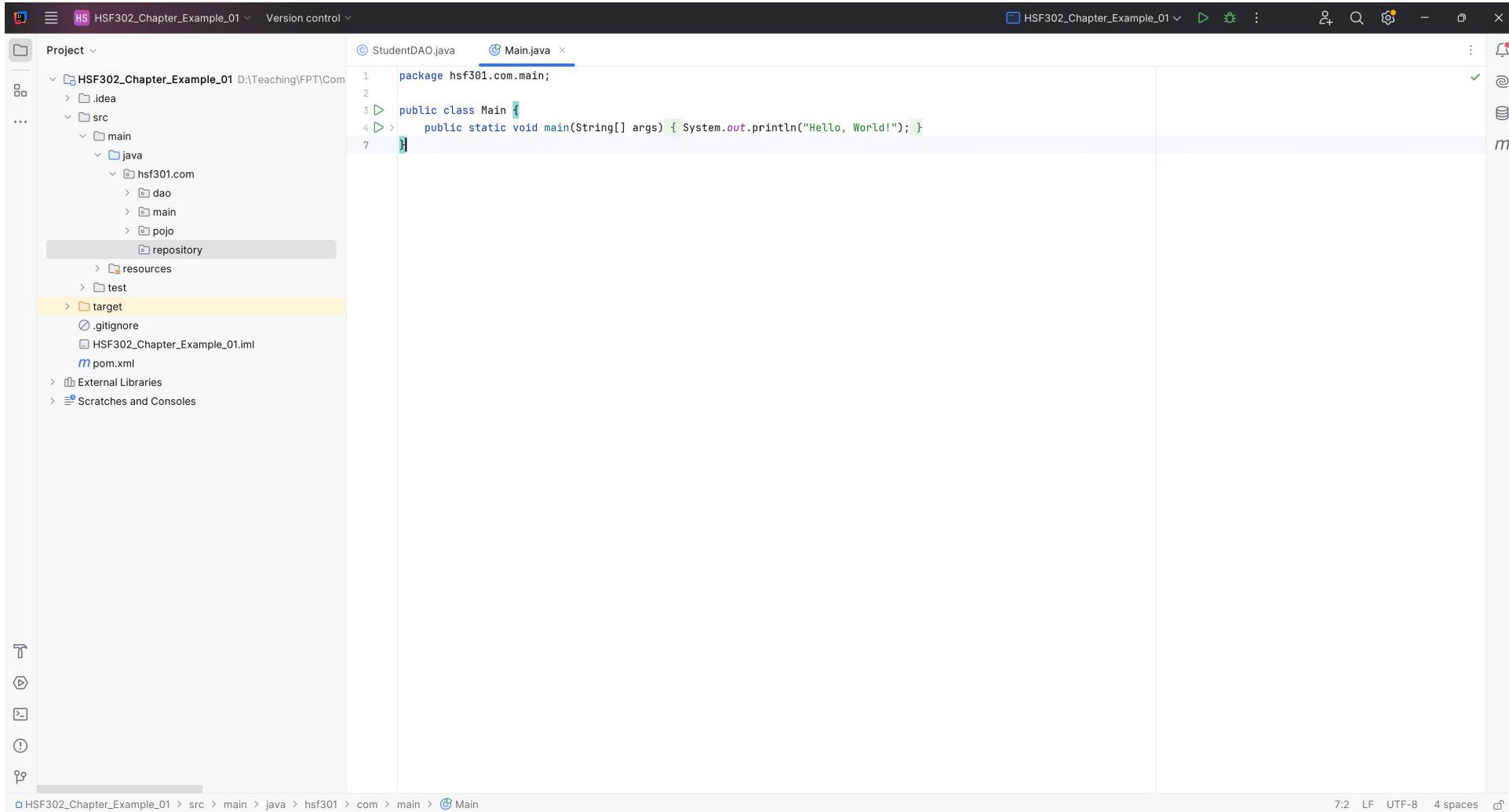
    public void delete(int studentID) {...}

    public Student findById(int studentID) {...}

    public void update(Student student) { 1 usage
        try {
            em = emf.createEntityManager();
            em.getTransaction().begin();
            Student s = em.find(Student.class, student.getId());
            if (s != null) {
                s.setFirstName(student.getFirstName());
                s.setLastName(student.getLastName());
                s.setMarks(student.getMarks());
                em.getTransaction().commit();
            }
        } catch (Exception ex) {
            System.out.println("Error " + ex.getMessage());
        } finally {
            // em.close();
        }
    }
}
```

- IDE Features:** The left sidebar shows the project structure with files like Main.java, Student.java, and StudentServiceImpl.java. The bottom status bar shows file paths, encoding, and other settings.

Create hsf301.com.repository Package in src/main/java

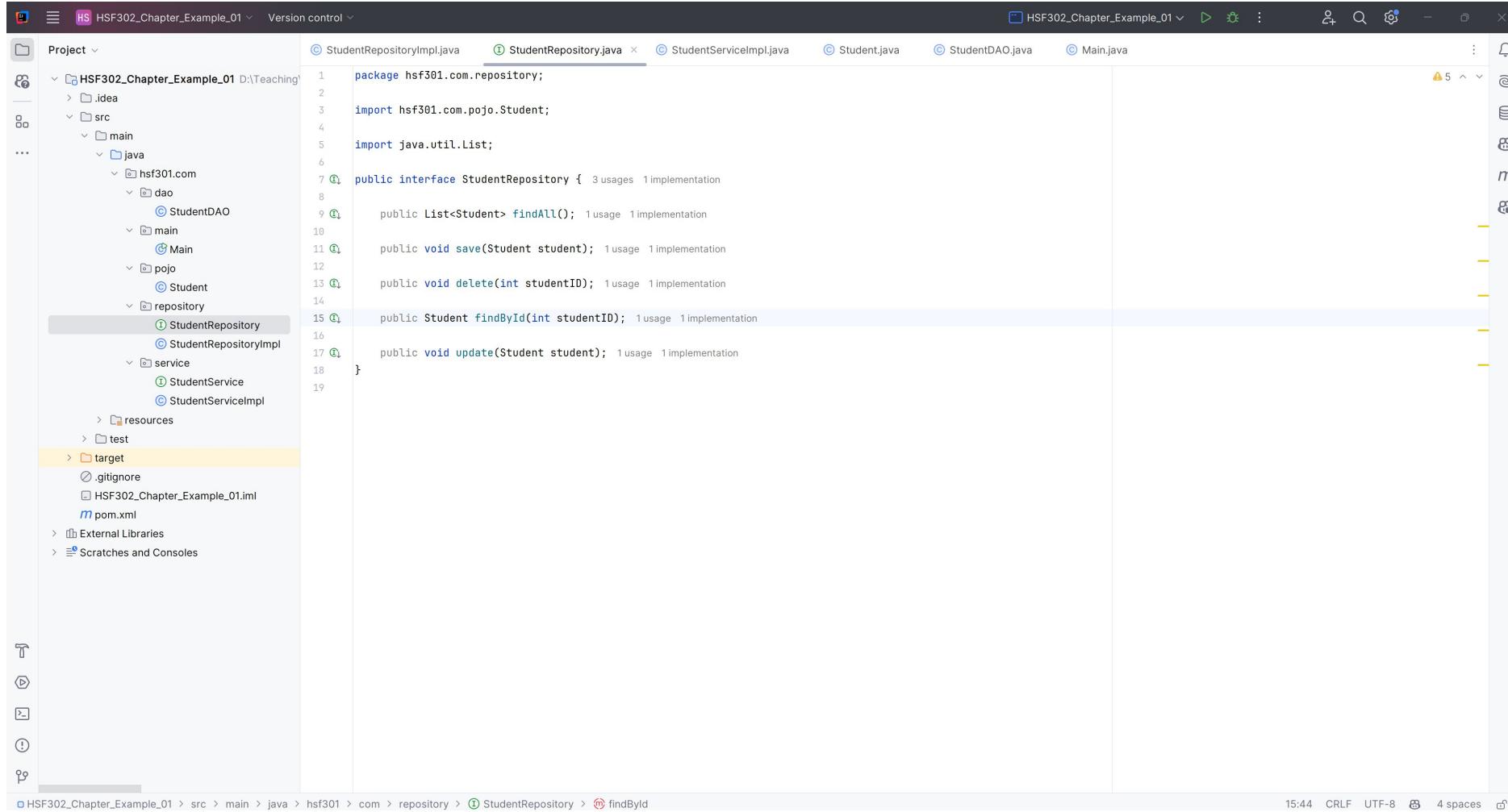


The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** The project structure is displayed under "HSF302_Chapter_Example_01". The "src" directory contains a "main" package, which further contains "java", "hsf301.com", "dao", "main", "pojo", and "repository". The "repository" folder is currently selected.
- Code Editor:** Two files are open: "StudentDAO.java" and "Main.java". "Main.java" is the active file, showing the following code:

```
package hsf301.com.main;
public class Main {
    public static void main(String[] args) { System.out.println("Hello, World!"); }
```
- Status Bar:** The status bar at the bottom shows the file path: "HSF302_Chapter_Example_01 > src > main > java > hsf301 > com > main > Main". It also displays "7:2 LF UTF-8 4 spaces" and a small "C" icon.

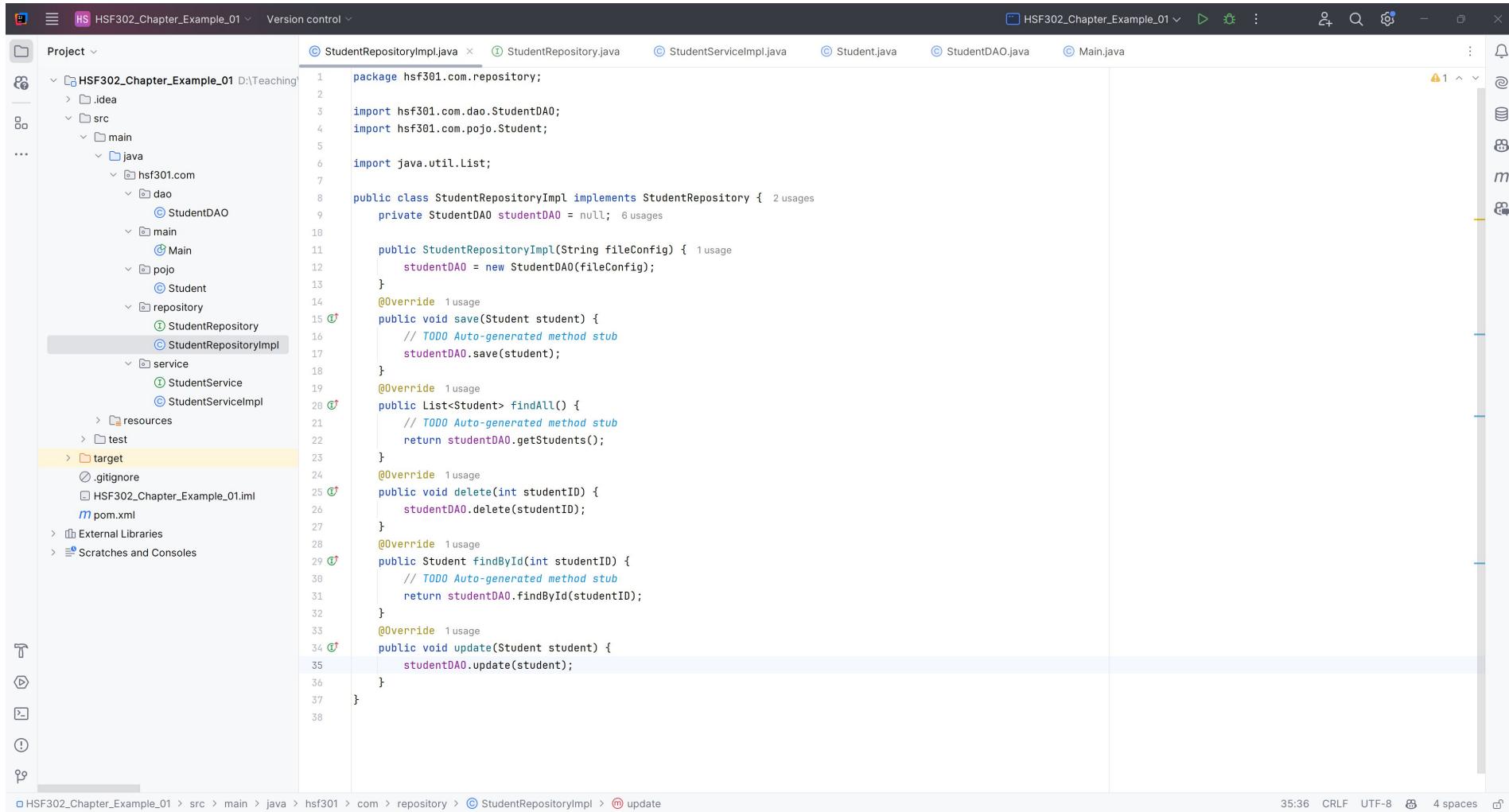
Create StudentRepository



The screenshot shows the IntelliJ IDEA IDE interface. The left sidebar displays the project structure for 'HSF302_Chapter_Example_01'. The 'src' directory contains 'main' and 'test' packages. Under 'main', there are 'java', 'resources', and 'service' packages. The 'java' package contains 'hsf301.com' which has 'dao', 'main', 'pojo', and 'repository' sub-packages. The 'repository' package contains 'StudentRepository' and 'StudentRepositoryImpl' files. The 'service' package contains 'StudentService' and 'StudentServiceImpl' files. The 'test' package contains 'target', '.gitignore', 'HSF302_Chapter_Example_01.iml', and 'pom.xml'. The right panel shows the code editor with 'StudentRepository.java' selected. The code defines an interface named 'StudentRepository' with methods: findAll(), save(Student student), delete(int studentID), findById(int studentID), and update(Student student). The code is annotated with Javadoc comments and usage counts.

```
1 package hsf301.com.repository;
2
3 import hsf301.com.pojo.Student;
4
5 import java.util.List;
6
7 public interface StudentRepository {
8     List<Student> findAll();
9     void save(Student student);
10    void delete(int studentID);
11    Student findById(int studentID);
12    void update(Student student);
13 }
```

Create StudentRepositoryImpl implement StudentRepository



The screenshot shows the IntelliJ IDEA interface with the project 'HSF302_Chapter_Example_01' open. The left sidebar displays the project structure, and the right pane shows the code for `StudentRepositoryImpl.java`. The code implements the `StudentRepository` interface, utilizing `StudentDAO` for database operations.

```
package hsf301.com.repository;

import hsf301.com.dao.StudentDAO;
import hsf301.com.pojo.Student;

import java.util.List;

public class StudentRepositoryImpl implements StudentRepository {
    private StudentDAO studentDAO = null;

    public StudentRepositoryImpl(String fileConfig) {
        studentDAO = new StudentDAO(fileConfig);
    }

    @Override
    public void save(Student student) {
        // TODO Auto-generated method stub
        studentDAO.save(student);
    }

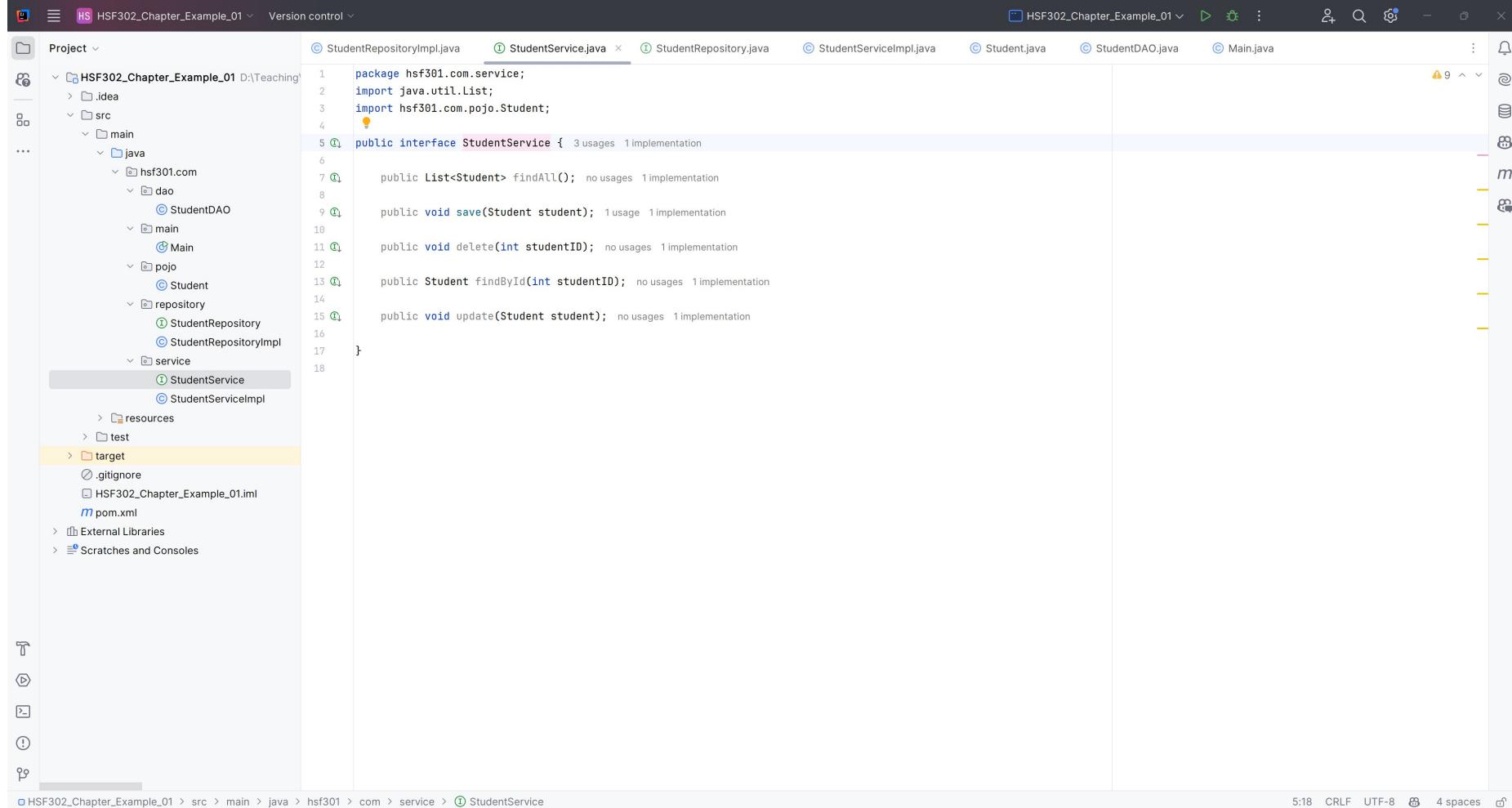
    @Override
    public List<Student> findAll() {
        // TODO Auto-generated method stub
        return studentDAO.getStudents();
    }

    @Override
    public void delete(int studentID) {
        studentDAO.delete(studentID);
    }

    @Override
    public Student findById(int studentID) {
        // TODO Auto-generated method stub
        return studentDAO.findById(studentID);
    }

    @Override
    public void update(Student student) {
        studentDAO.update(student);
    }
}
```

Create StudentService



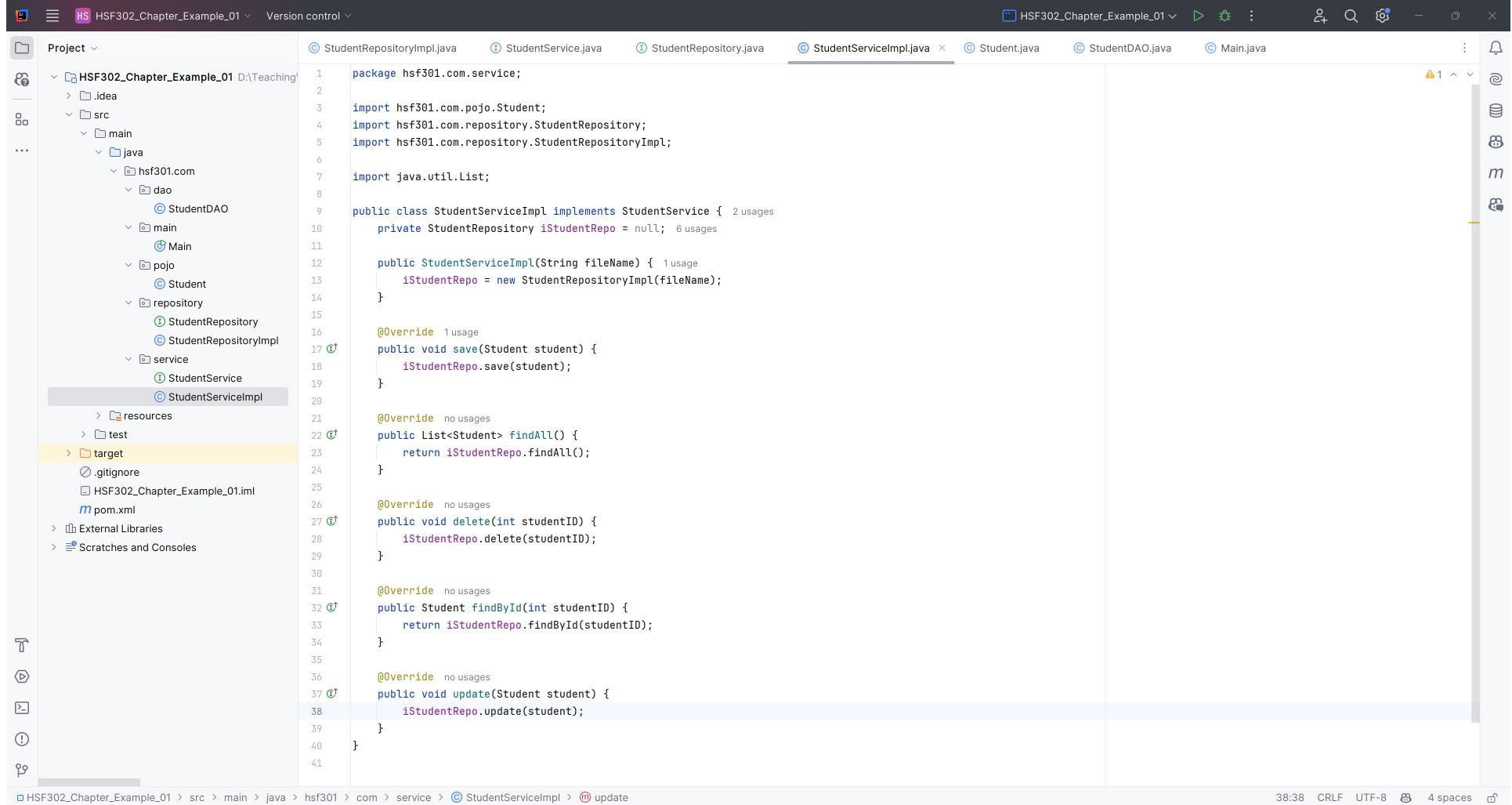
The screenshot shows the IntelliJ IDEA IDE interface with the following details:

- Project Structure:** The project is named "HSF302_Chapter_Example_01". It contains a "src" directory with "main" and "test" packages. The "main" package has "java", "resources", and "service" sub-directories. The "service" directory contains "StudentService.java" and "StudentServiceImpl.java".
- Code Editor:** The "StudentService.java" file is open and displayed. The code defines an interface named "StudentService" with the following methods:

```
package hsf301.com.service;
import java.util.List;
import hsf301.com.pojo.Student;

public interface StudentService {
    public List<Student> findAll();
    public void save(Student student);
    public void delete(int studentID);
    public Student findById(int studentID);
    public void update(Student student);
}
```
- Toolbars and Status Bar:** The top bar shows tabs for other files like "StudentRepositoryImpl.java", "StudentServiceImpl.java", etc. The bottom status bar shows the file path "HSF302_Chapter_Example_01 > src > main > java > hsf301 > com > service > StudentService.java", and the bottom right corner shows the time "5:18", encoding "CRLF", and line endings "4 spaces".

Create StudentServiceImpl implement StudentService



The screenshot shows the IntelliJ IDEA interface with the project structure and code editor for a Java application named HSF302_Chapter_Example_01.

Project Structure:

- HSF302_Chapter_Example_01
- src
 - main
 - java
 - hsf301.com
 - dao
 - StudentDAO
 - main
 - Main
 - pojo
 - Student
 - repository
 - StudentRepository
 - StudentRepositoryImpl
 - service
 - StudentService
 - StudentServiceImpl
 - target
 - .gitignore
 - HSF302_Chapter_Example_01.iml
 - pom.xml
 - External Libraries
 - Scratches and Consoles

Code Editor (StudentServiceImpl.java):

```
package hsf301.com.service;

import hsf301.com.pojo.Student;
import hsf301.com.repository.StudentRepository;
import hsf301.com.repository.StudentRepositoryImpl;

import java.util.List;

public class StudentServiceImpl implements StudentService {
    private StudentRepository iStudentRepo = null;

    public StudentServiceImpl(String fileName) {
        iStudentRepo = new StudentRepositoryImpl(fileName);
    }

    @Override
    public void save(Student student) {
        iStudentRepo.save(student);
    }

    @Override
    public List<Student> findAll() {
        return iStudentRepo.findAll();
    }

    @Override
    public void delete(int studentID) {
        iStudentRepo.delete(studentID);
    }

    @Override
    public Student findById(int studentID) {
        return iStudentRepo.findById(studentID);
    }

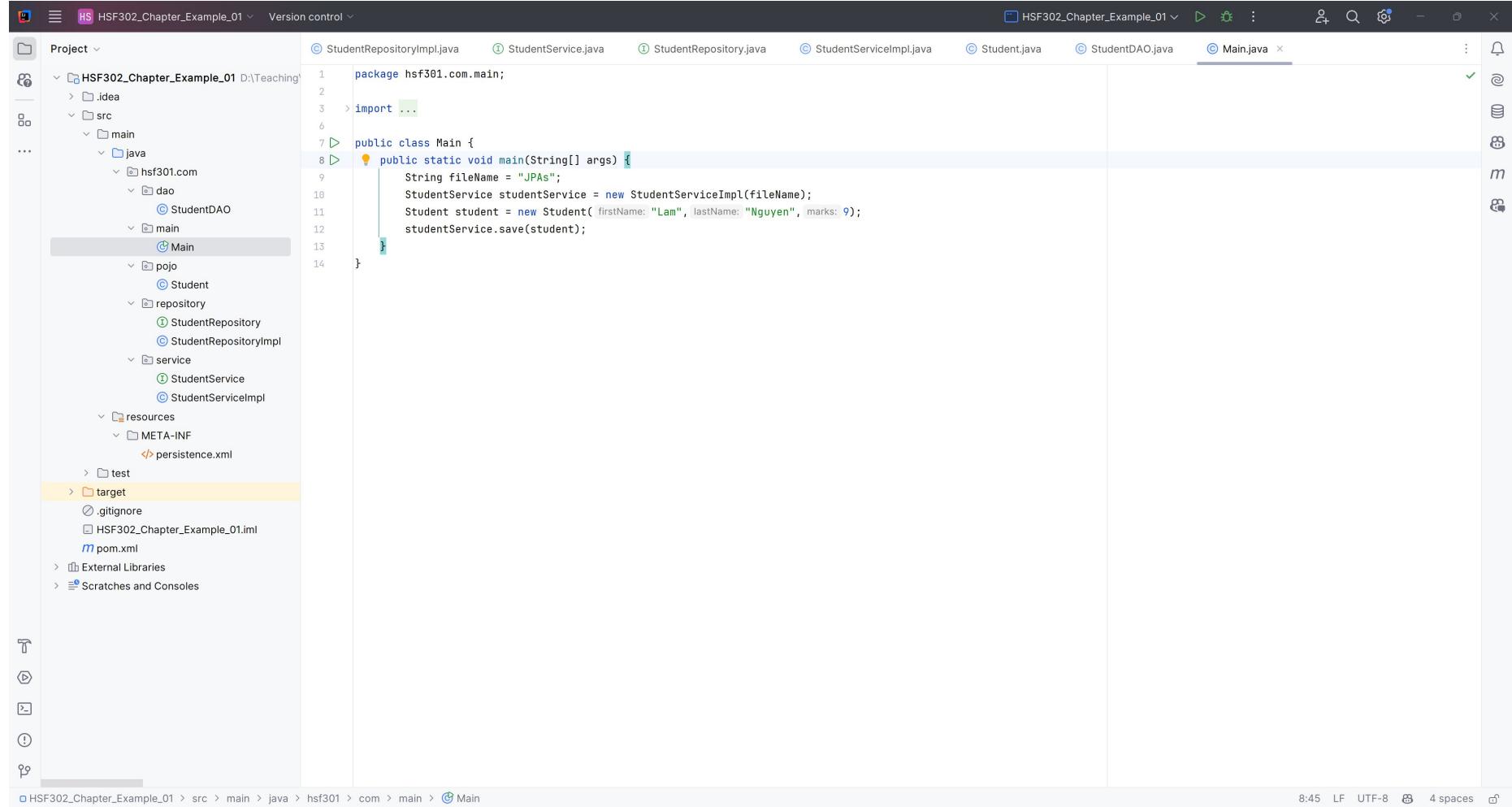
    @Override
    public void update(Student student) {
        iStudentRepo.update(student);
    }
}
```

Status Bar:

HSF302_Chapter_Example_01 > src > main > java > hsf301 > com > service > StudentServiceImpl > update

38:38 CRLF UTF-8 4 spaces

Create Main function

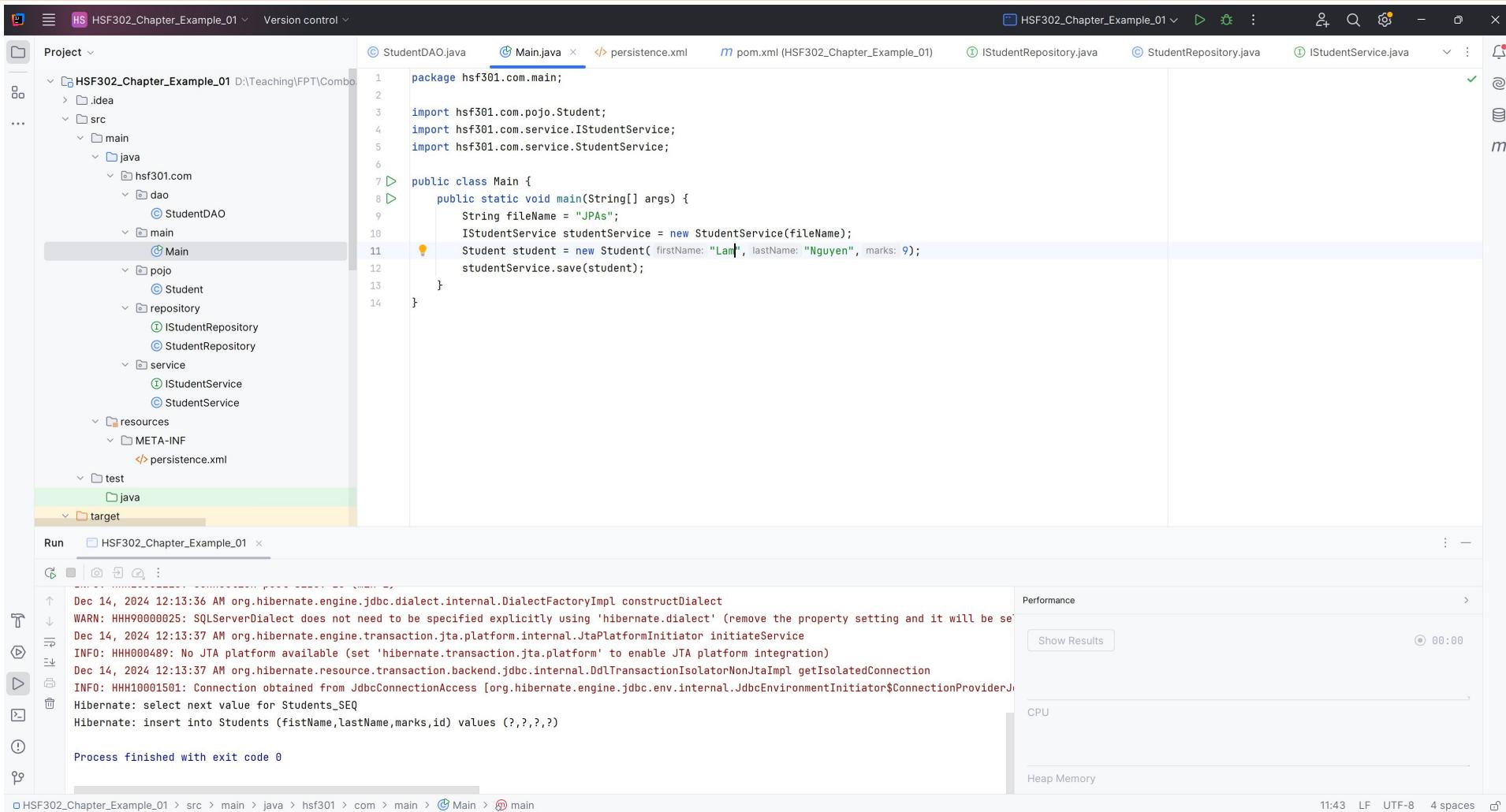


The screenshot shows the IntelliJ IDEA interface with the project `HSF302_Chapter_Example_01` open. The `Main.java` file is selected in the editor, displaying the following code:

```
package hsf301.com.main;
import ...
public class Main {
    public static void main(String[] args) {
        String fileName = "JPAs";
        StudentService studentService = new StudentServiceImpl(fileName);
        Student student = new Student( firstName: "Lam", lastName: "Nguyen", marks: 9 );
        studentService.save(student);
    }
}
```

The `target` folder in the project structure is highlighted in yellow. The status bar at the bottom right shows the time as 8:45, encoding as LF, character set as UTF-8, and code style as 4 spaces.

Run Program



The screenshot shows the IntelliJ IDEA IDE interface during the execution of a Java application. The main window displays the code for the `Main.java` file, which contains the following code:

```
package hsf301.com.main;

import hsf301.com.pojo.Student;
import hsf301.com.service.IStudentService;
import hsf301.com.service.StudentService;

public class Main {
    public static void main(String[] args) {
        String fileName = "JPAs";
        IStudentService studentService = new StudentService(fileName);
        Student student = new Student(firstName: "Lam", lastName: "Nguyen", marks: 9);
        studentService.save(student);
    }
}
```

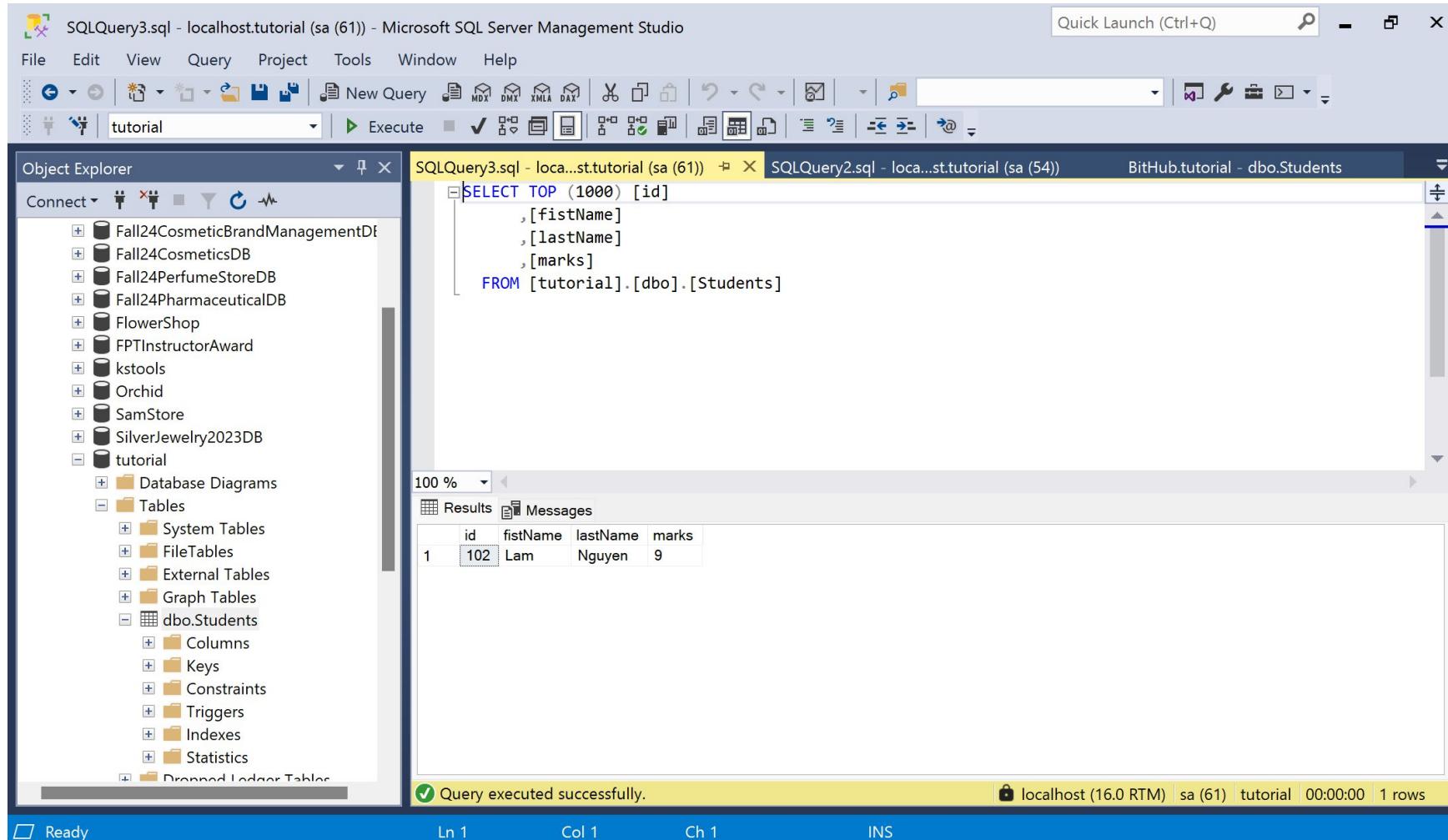
The code uses Java 11 syntax, including the record syntax for the `Student` class. The `HSF302_Chapter_Example_01` project structure is visible on the left, showing packages like `hsf301.com.main`, `hsf301.com.pojo`, `hsf301.com.service`, and `hsf301.com.repository`. The `persistence.xml` file is also present in the resources.

In the bottom-left corner, the `Run` tool window shows the application has finished running successfully with an exit code of 0. The log output includes standard Hibernate initialization messages and a successful database insertion statement:

```
Dec 14, 2024 12:13:36 AM org.hibernate.engine.jdbc.dialect.internal.DialectFactoryImpl constructDialect
WARN: HHH90000025: SQLServerDialect does not need to be specified explicitly using 'hibernate.dialect' (remove the property setting and it will be se...
Dec 14, 2024 12:13:37 AM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
Dec 14, 2024 12:13:37 AM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJ...
Hibernate: select next value for Students_SEQ
Hibernate: insert into Students (firstName,lastName,marks,id) values (?, ?, ?, ?)
Process finished with exit code 0
```

The bottom right corner of the interface shows the current time as 11:43, the line separator as LF, the encoding as UTF-8, and the code style as 4 spaces.

Result



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]
    ,[firstName]
    ,[lastName]
    ,[marks]
FROM [tutorial].[dbo].[Students]
```

The Results pane below shows the output of the query:

	id	firstName	lastName	marks
1	102	Lam	Nguyen	9

A green status bar at the bottom indicates: 'Query executed successfully.' followed by connection information: 'localhost (16.0 RTM) | sa (61) | tutorial | 00:00:00 | 1 rows'.

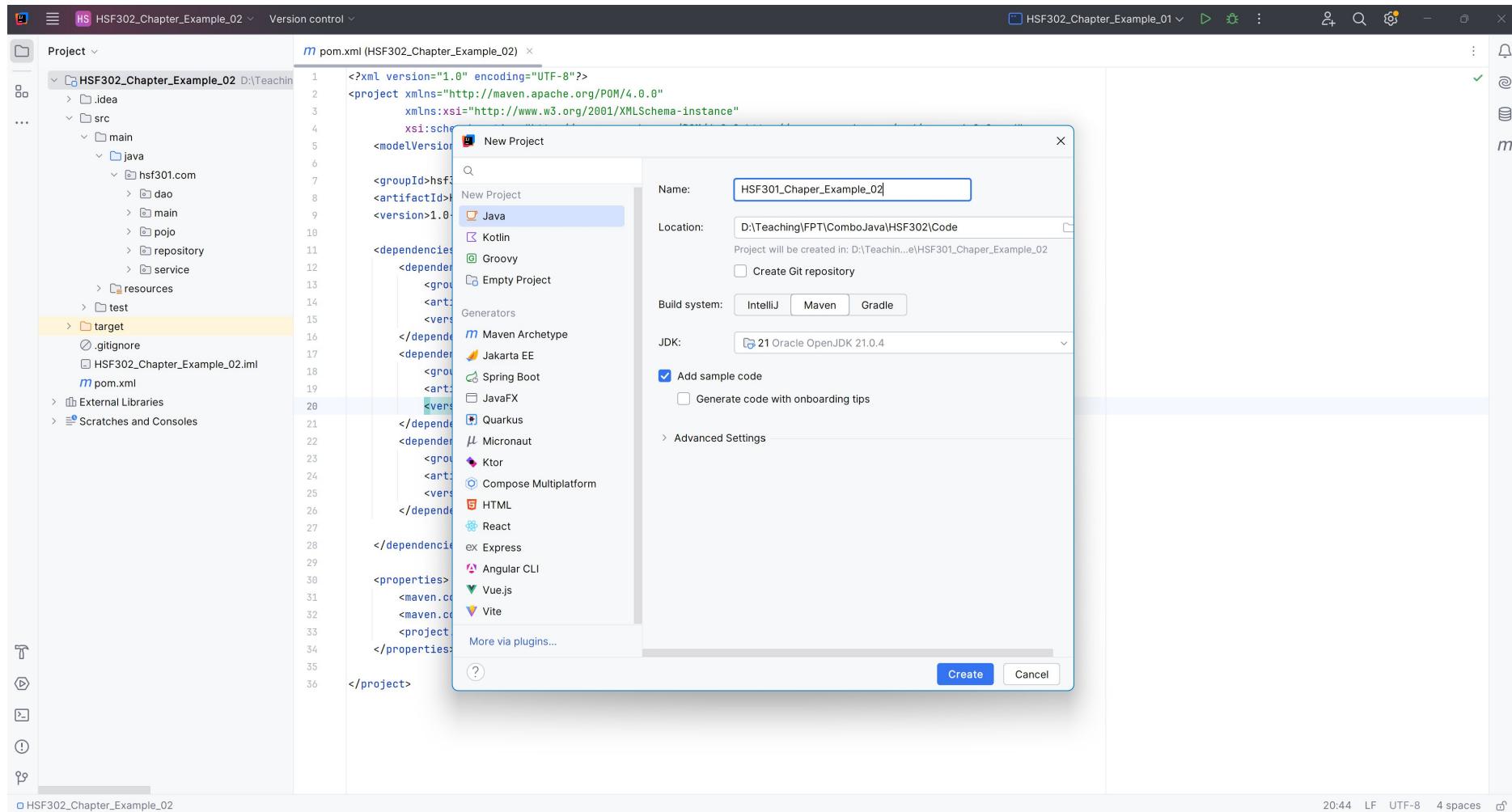
Mapping in JPA

Relationships Annotations in JPA

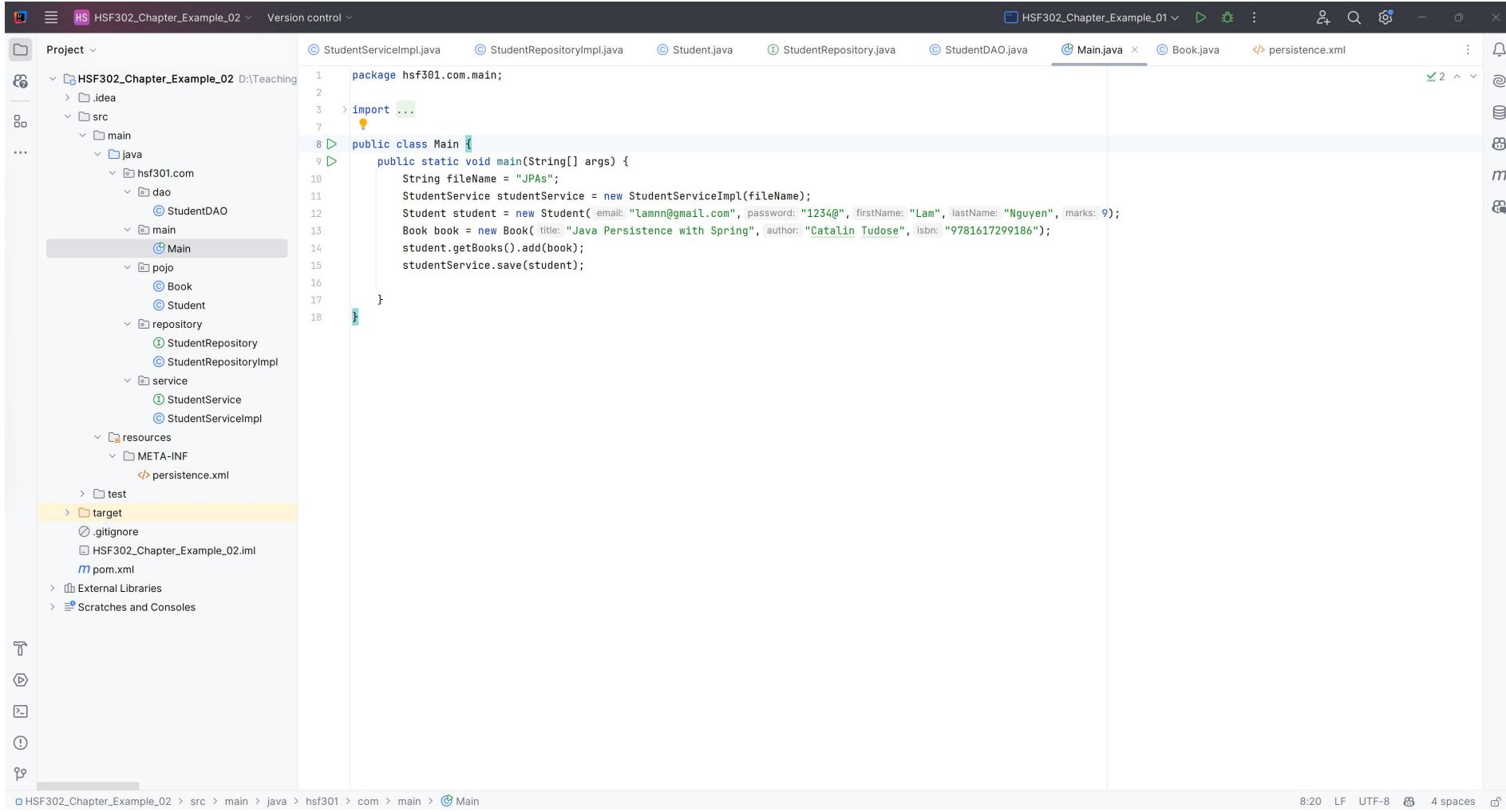
- ❖ **@ManyToOne:** This annotation defines a many-to-one relationship between two entities.
- ❖ **@OneToMany:** This annotation defines a one-to-many relationship between two entities.
- ❖ **@OneToOne:** This annotation defines a one-to-one relationship between two entities.
- ❖ **@ManyToMany:** This annotation defines a many-to-many relationship between two entities.

Demo JPA (One To Many)

Open IntelliJ, File | New | Maven Project



Create a structure of Maven Project



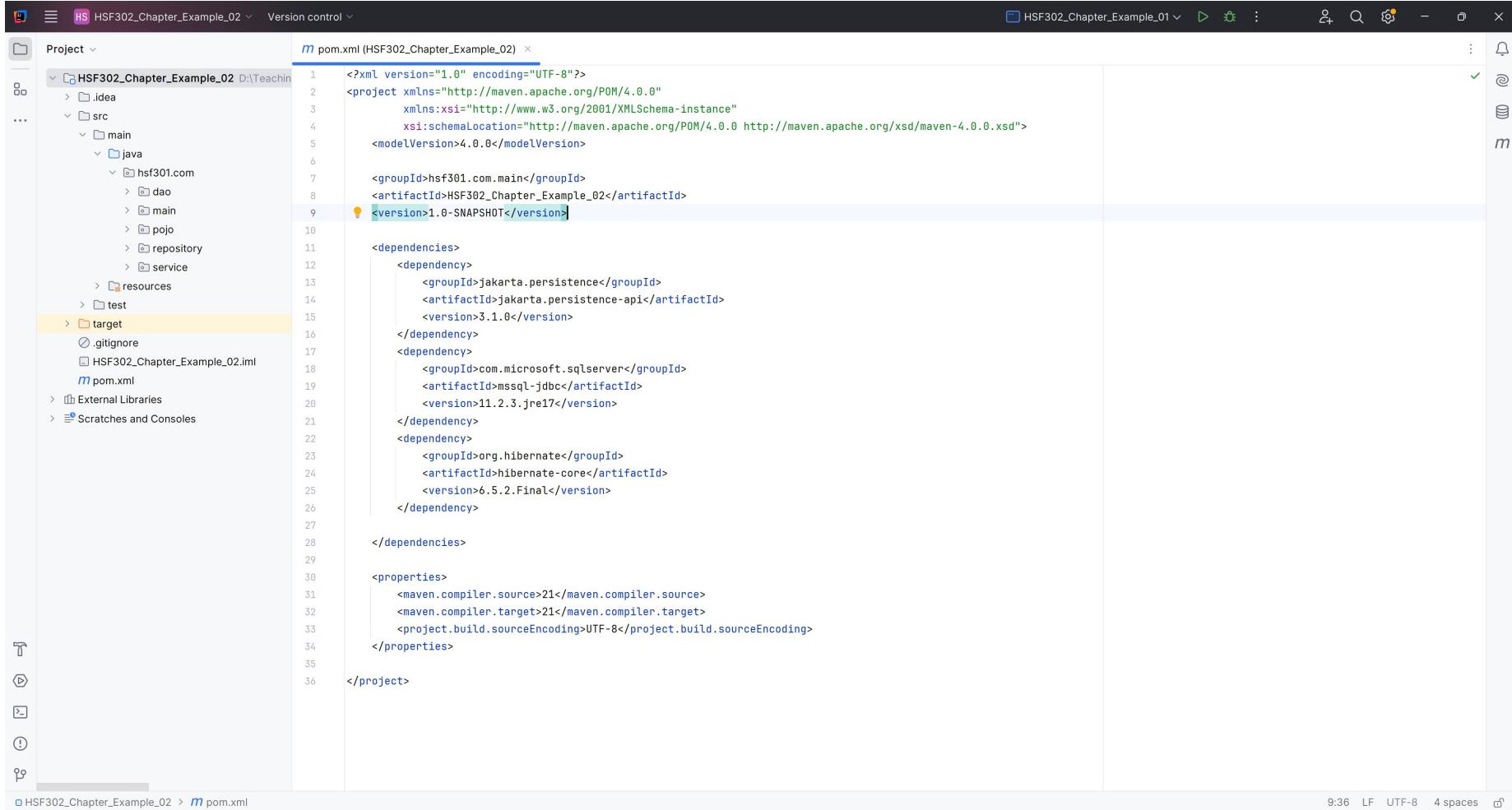
The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** On the left, the project structure is displayed under "HSF302_Chapter_Example_02". It includes a .idea folder, a src directory containing main, java, hsf301.com, dao, StudentDAO, main, Main, pojo, Book, Student, repository, StudentRepository, StudentRepositoryImpl, service, StudentService, StudentServiceImpl, resources, META-INF (with persistence.xml), test, target (highlighted in yellow), .gitignore, HSF302_Chapter_Example_02.iml, and pom.xml. There are also External Libraries and Scratches and Consoles.
- Code Editor:** The main window shows the "Main.java" file with the following code:

```
package hsf301.com.main;
import ...;

public class Main {
    public static void main(String[] args) {
        String fileName = "JPAs";
        StudentService studentService = new StudentServiceImpl(fileName);
        Student student = new Student( email: "Lamnn@gmail.com", password: "1234@", firstName: "Lam", lastName: "Nguyen", marks: 9);
        Book book = new Book( title: "Java Persistence with Spring", author: "Catalin Tudose", isbn: "9781617299186");
        student.getBooks().add(book);
        studentService.save(student);
    }
}
```
- Toolbars and Status Bar:** The top bar shows tabs for "HSF302_Chapter_Example_02" and "HSF302_Chapter_Example_01". The status bar at the bottom shows the file path "HSF302_Chapter_Example_02 > src > main > java > hsf301 > com > main > Main", and the bottom right corner shows "8:20 LF UTF-8 4 spaces".

Edit pom.xml



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

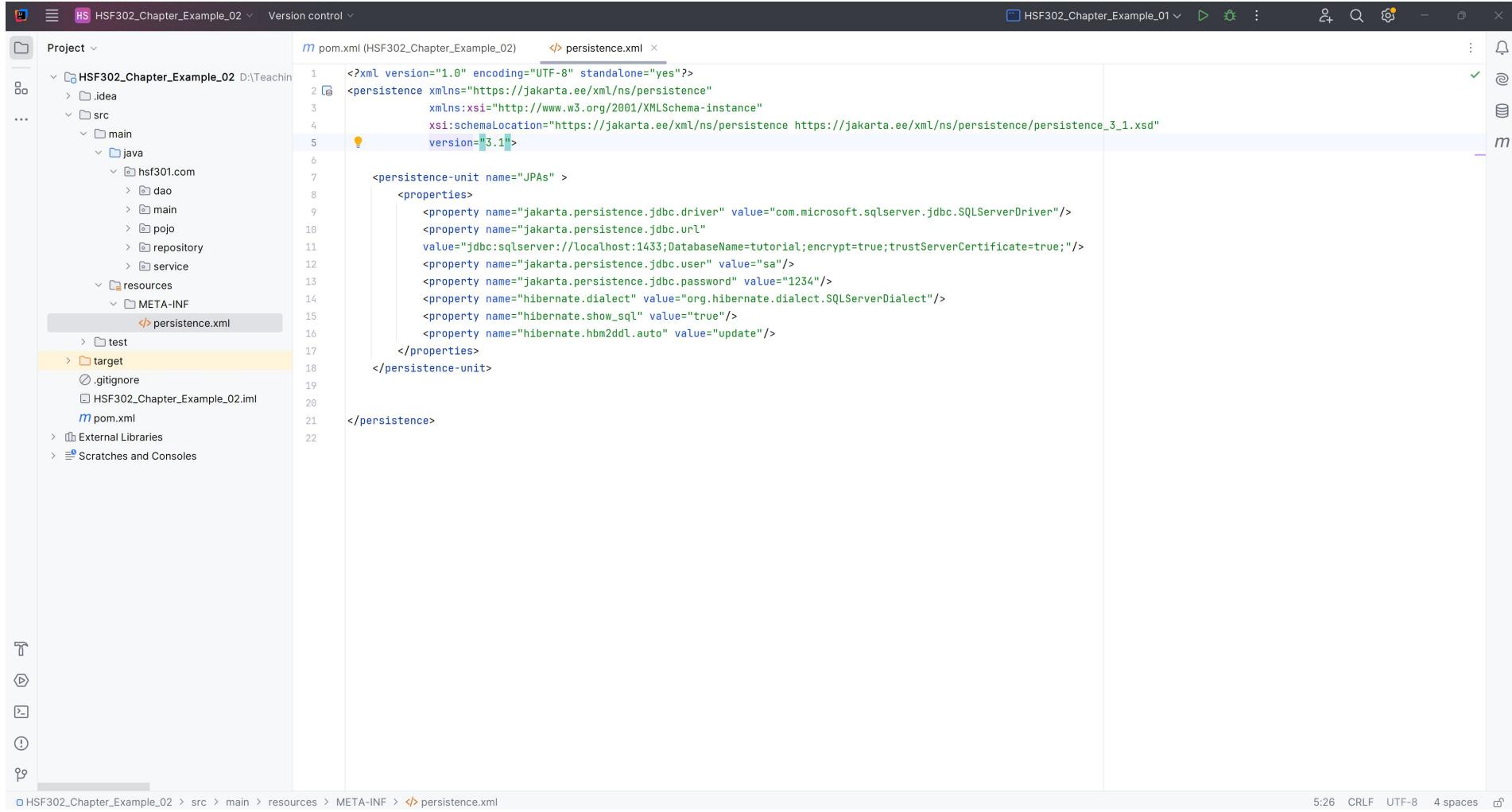
<groupId>hsf301.com.main</groupId>
<artifactId>HSF302_Chapter_Example_02</artifactId>
<version>1.0-SNAPSHOT</version>

<dependencies>
    <dependency>
        <groupId>jakarta.persistence</groupId>
        <artifactId>jakarta.persistence-api</artifactId>
        <version>3.1.0</version>
    </dependency>
    <dependency>
        <groupId>com.microsoft.sqlserver</groupId>
        <artifactId>mssql-jdbc</artifactId>
        <version>11.2.3.jre17</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.5.2.Final</version>
    </dependency>
</dependencies>

<properties>
    <maven.compiler.source>21</maven.compiler.source>
    <maven.compiler.target>21</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

</project>
```

Create persistence.xml in META-INF folder



The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right.

Project Structure:

- Project: HSF302_Chapter_Example_02
- Source Path: D:\Teaching\HSF302\Chapter_Example_02
- File Tree:
 - src
 - main
 - java
 - hsf301.com
 - dao
 - main
 - pojo
 - repository
 - service
 - resources
 - META-INF
 - persistence.xml
 - test
 - target
 - .gitignore
 - HSF302_Chapter_Example_02.iml
 - pom.xml
 - External Libraries
 - Scratches and Consoles

Code Editor (persistence.xml):

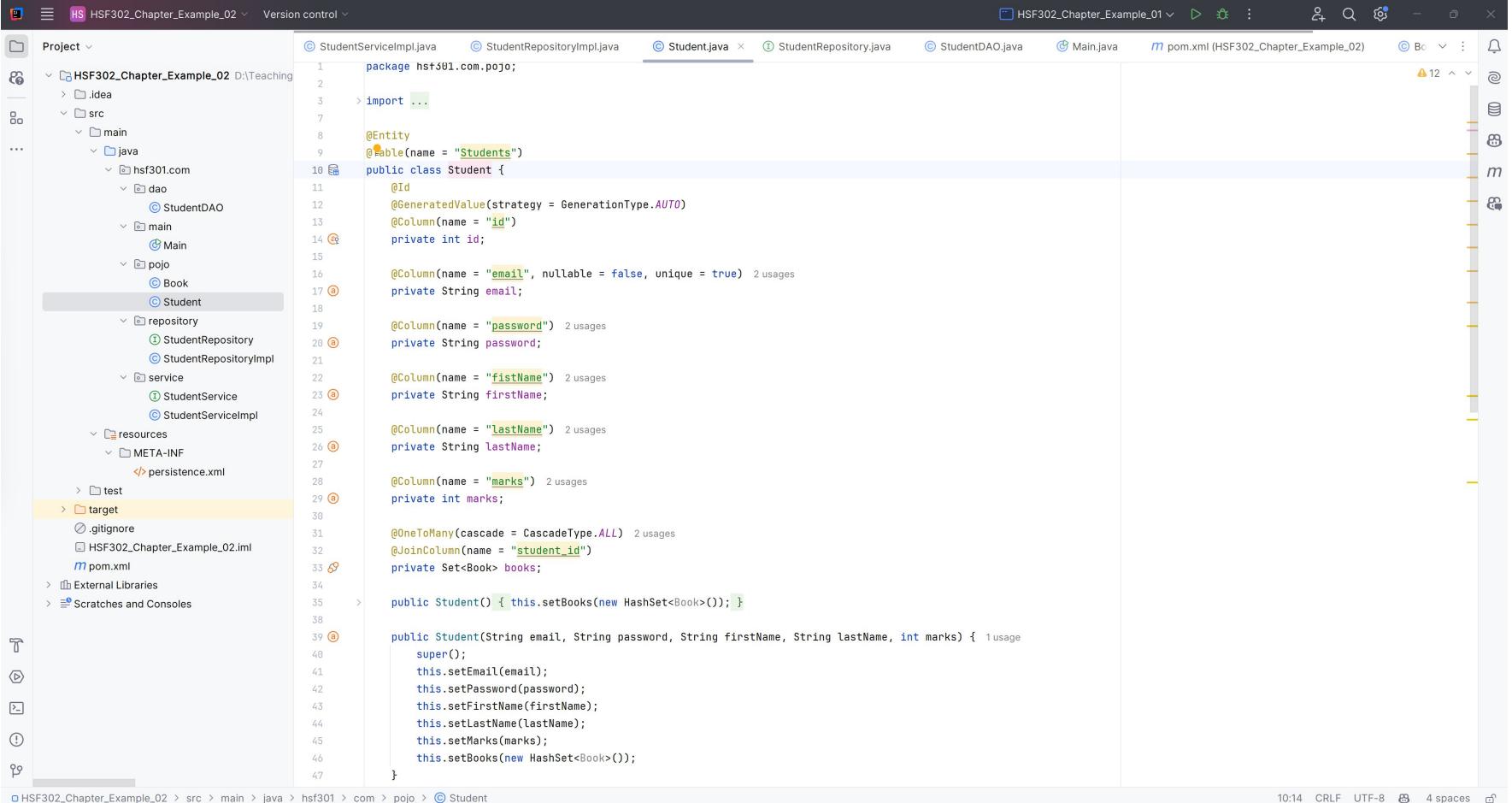
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence https://jakarta.ee/xml/ns/persistence/persistence_3_1.xsd"
    version="3.1">

    <persistence-unit name="JPAs" >
        <properties>
            <property name="jakarta.persistence.jdbc.driver" value="com.microsoft.sqlserver.jdbc.SQLServerDriver"/>
            <property name="jakarta.persistence.jdbc.url"
                value="jdbc:sqlserver://localhost:1433;DatabaseName=tutorial;encrypt=true;trustServerCertificate=true;"/>
            <property name="jakarta.persistence.jdbc.user" value="sa"/>
            <property name="jakarta.persistence.jdbc.password" value="1234"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.SQLServerDialect"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
        </properties>
    </persistence-unit>
</persistence>
```

Status Bar:

5:26 CRLF UTF-8 4 spaces

Create Student.java in Pojo



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

- Project Structure:** The project is named "HSF302_Chapter_Example_02". The "src/main/java" directory contains packages for "hsf301.com" (with "dao", "main", "pojo", "repository", "service", and "resources" sub-packages), "Book", and "Student".
- Code Editor:** The "Student.java" file is open in the editor. The code defines a POJO class "Student" with annotations for Entity, Table, and various columns. It includes fields for id, email, password, firstName, lastName, and marks, along with a set of books.
- Status Bar:** The status bar at the bottom shows the file path "HSF302_Chapter_Example_02 > src > main > java > hsf301 > com > pojo > Student.java", the time "10:14", and encoding "UTF-8".

```
package hsf301.com.pojo;

import ...;

@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)
    private String email;

    @Column(name = "password")
    private String password;

    @Column(name = "firstName")
    private String firstName;

    @Column(name = "lastName")
    private String lastName;

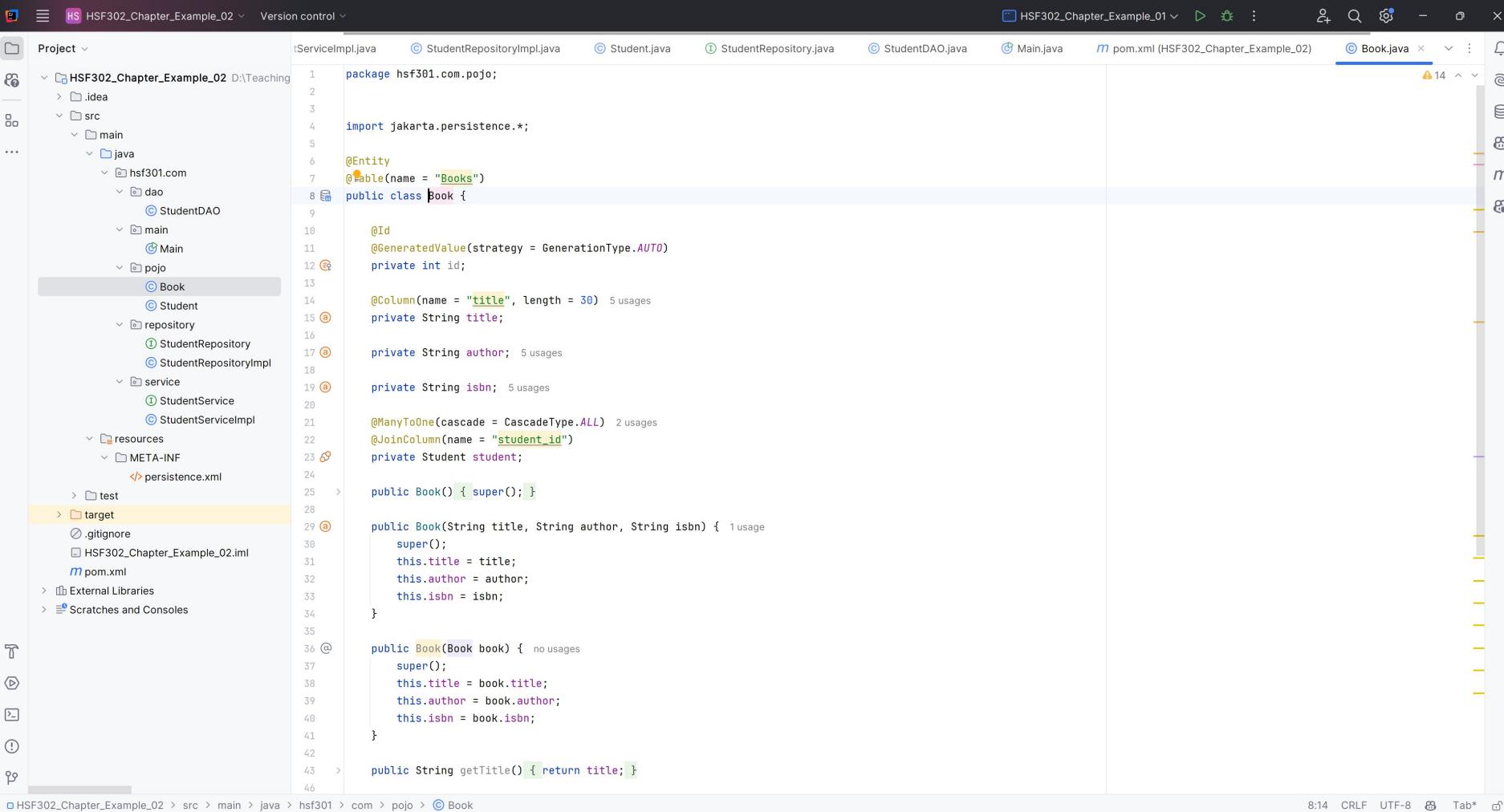
    @Column(name = "marks")
    private int marks;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "student_id")
    private Set<Book> books;

    public Student() { this.setBooks(new HashSet<Book>()); }

    public Student(String email, String password, String firstName, String lastName, int marks) {
        super();
        this.setEmail(email);
        this.setPassword(password);
        this.setFirstName(firstName);
        this.setLastName(lastName);
        this.setMarks(marks);
        this.setBooks(new HashSet<Book>());
    }
}
```

Create Book.java in hsf301.com.pojo



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

- Project Structure:** The project is named "HSF302_Chapter_Example_02". The "src" directory contains "main" and "test" packages. "main" has "java" and "resources" sub-directories. "java" contains "hsf301.com" which has "dao", "main", and "pojo" sub-packages. "pojo" contains "Book" and "Student" classes. "repository" contains "StudentRepository" and "StudentRepositoryImpl" classes. "service" contains "StudentService" and "StudentServiceImpl" classes. "resources" contains "META-INF" and "persistence.xml".
- Code Editor:** The "Book.java" file is open in the code editor. The code defines a class "Book" with annotations for persistence and relationships.
- Toolbars and Status Bar:** The top bar shows tabs for various files like "ServiceImpl.java", "StudentRepositoryImpl.java", etc. The status bar at the bottom right shows the time as 8:14 and encoding as CRLF.

```
package hsf301.com.pojo;

import jakarta.persistence.*;

@Entity
@Table(name = "Books")
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    @Column(name = "title", length = 30)
    private String title;

    private String author;

    private String isbn;

    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "student_id")
    private Student student;

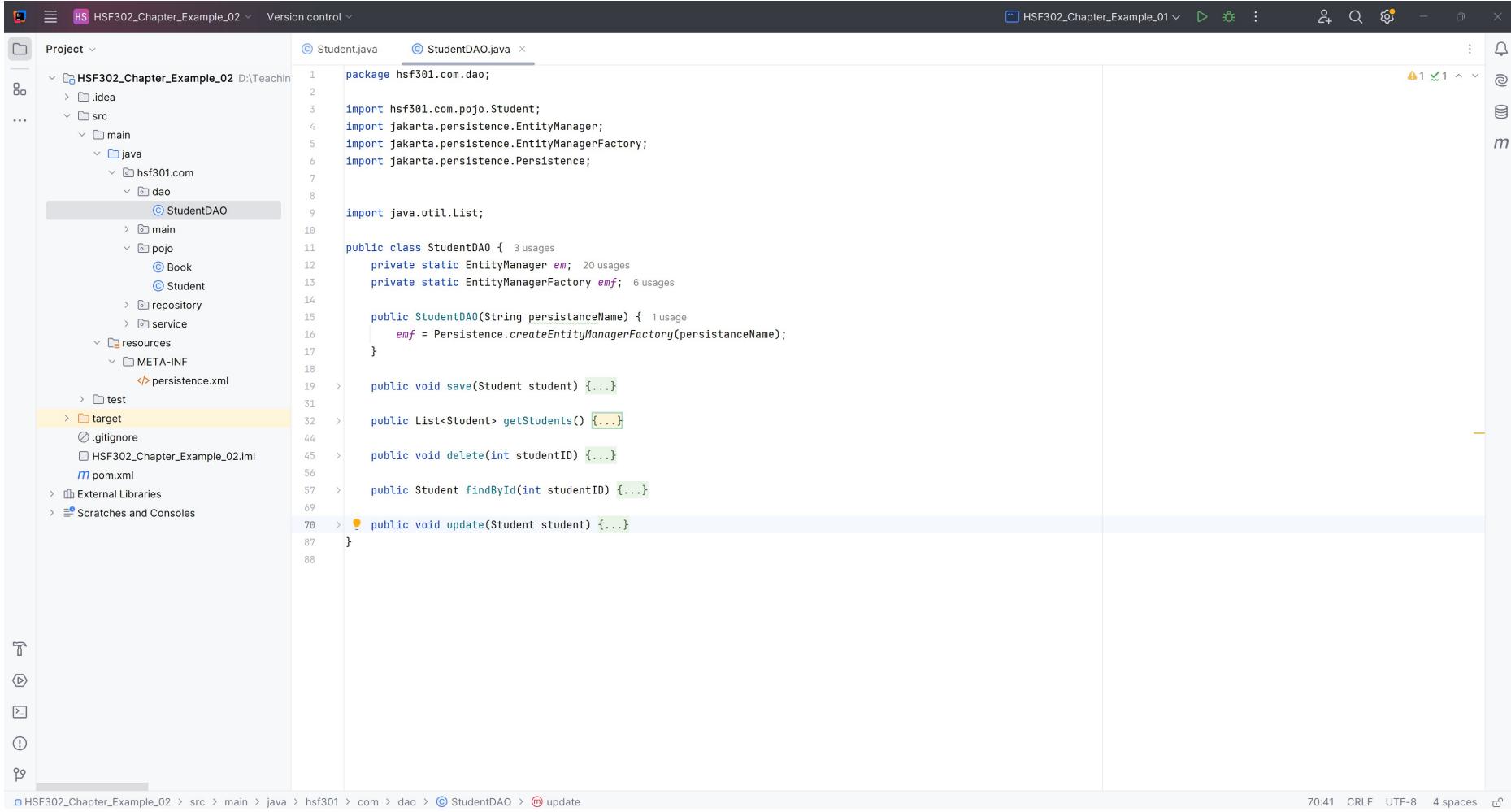
    public Book() { super(); }

    public Book(String title, String author, String isbn) {
        super();
        this.title = title;
        this.author = author;
        this.isbn = isbn;
    }

    public Book(Book book) {
        super();
        this.title = book.title;
        this.author = book.author;
        this.isbn = book.isbn;
    }

    public String getTitle() { return title; }
}
```

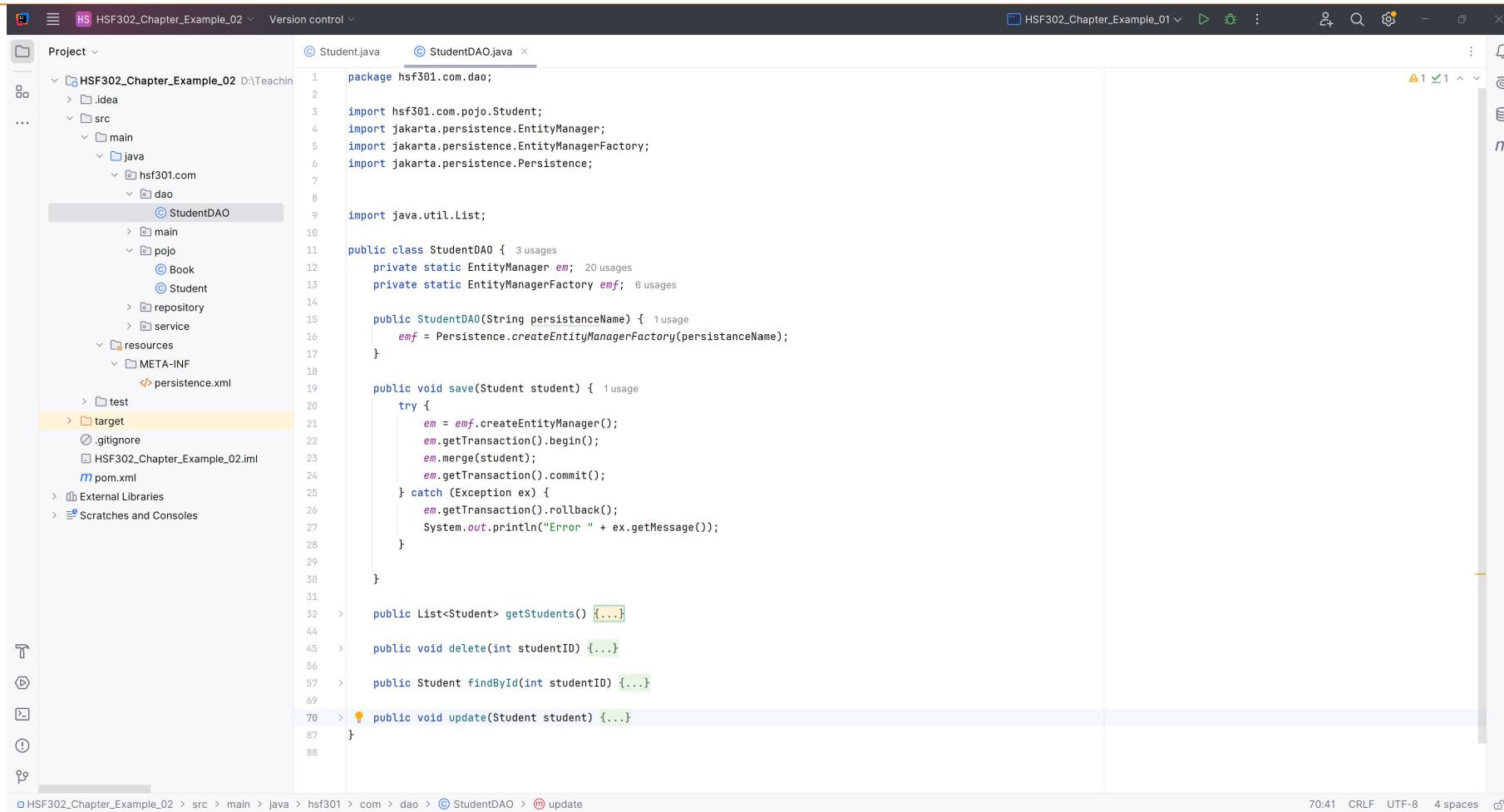
Create StudentDAO



The screenshot shows the IntelliJ IDEA interface with the project `HSF302_Chapter_Example_02` open. The left sidebar displays the project structure, including the `src` directory which contains `main`, `java`, and `dao`. Inside `dao`, the `StudentDAO` class is selected. The right panel shows the code editor with the `StudentDAO.java` file open. The code defines a DAO class for managing students using Jakarta Persistence.

```
package hsf301.com.dao;  
import hsf301.com.pojo.Student;  
import jakarta.persistence.EntityManager;  
import jakarta.persistence.EntityManagerFactory;  
import jakarta.persistence.Persistence;  
  
import java.util.List;  
  
public class StudentDAO {  
    private static EntityManager em; 20 usages  
    private static EntityManagerFactory emf; 6 usages  
  
    public StudentDAO(String persistanceName) {  
        emf = Persistence.createEntityManagerFactory(persistanceName);  
    }  
  
    public void save(Student student) {...}  
    public List<Student> getStudents() {...}  
    public void delete(int studentID) {...}  
    public Student findById(int studentID) {...}  
    public void update(Student student) {...}  
}
```

Save Student in StudentDAO



The screenshot shows the IntelliJ IDEA interface with the project `HSF302_Chapter_Example_02` open. The `StudentDAO.java` file is selected in the editor tab bar. The code implements a DAO interface for managing students using Jakarta Persistence.

```
package hsf301.com.dao;

import hsf301.com.pojo.Student;
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;

import java.util.List;

public class StudentDAO {
    private static EntityManager em;
    private static EntityManagerFactory emf;

    public StudentDAO(String persistanceName) {
        emf = Persistence.createEntityManagerFactory(persistanceName);
    }

    public void save(Student student) {
        try {
            em = emf.createEntityManager();
            em.getTransaction().begin();
            em.merge(student);
            em.getTransaction().commit();
        } catch (Exception ex) {
            em.getTransaction().rollback();
            System.out.println("Error " + ex.getMessage());
        }
    }

    public List<Student> getStudents() {...}

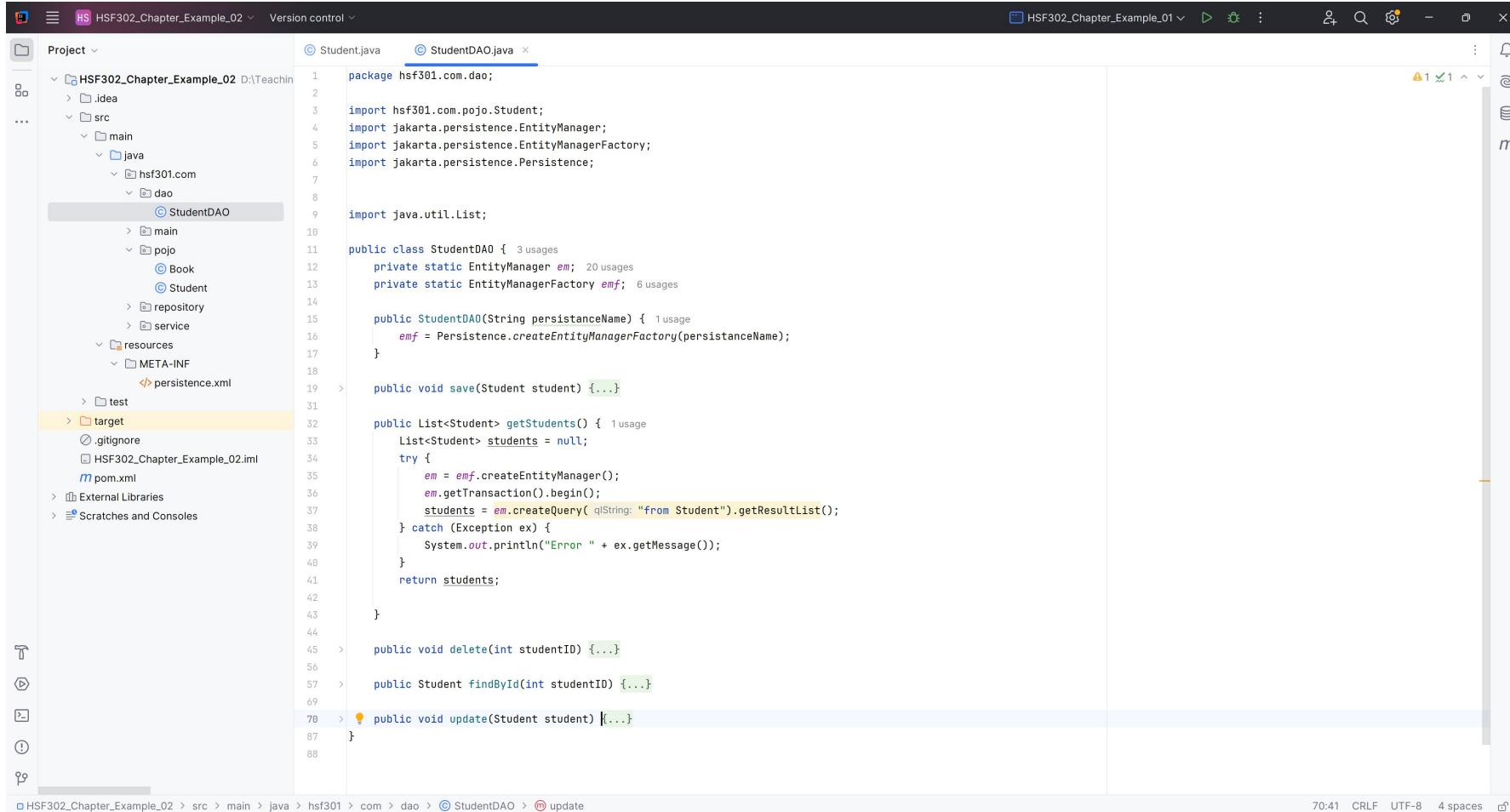
    public void delete(int studentID) {...}

    public Student findById(int studentID) {...}

    public void update(Student student) {...}
}
```

The code uses `EntityManager` and `EntityManagerFactory` from the `jakarta.persistence` package to interact with the database. It includes methods for saving, deleting, finding by ID, and updating students.

Get All Students in StudentDAO



The screenshot shows the IntelliJ IDEA interface with the project `HSF302_Chapter_Example_02` open. The code editor displays `StudentDAO.java`, which contains the following Java code:

```
package hsf301.com.dao;

import hsf301.com.pojo.Student;
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;

import java.util.List;

public class StudentDAO { 3 usages
    private static EntityManager em; 20 usages
    private static EntityManagerFactory emf; 6 usages

    public StudentDAO(String persistanceName) { 1 usage
        emf = Persistence.createEntityManagerFactory(persistanceName);
    }

    public void save(Student student) {...}

    public List<Student> getStudents() { 1 usage
        List<Student> students = null;
        try {
            em = emf.createEntityManager();
            em.getTransaction().begin();
            students = em.createQuery("from Student").getResultList();
        } catch (Exception ex) {
            System.out.println("Error " + ex.getMessage());
        }
        return students;
    }

    public void delete(int studentID) {...}

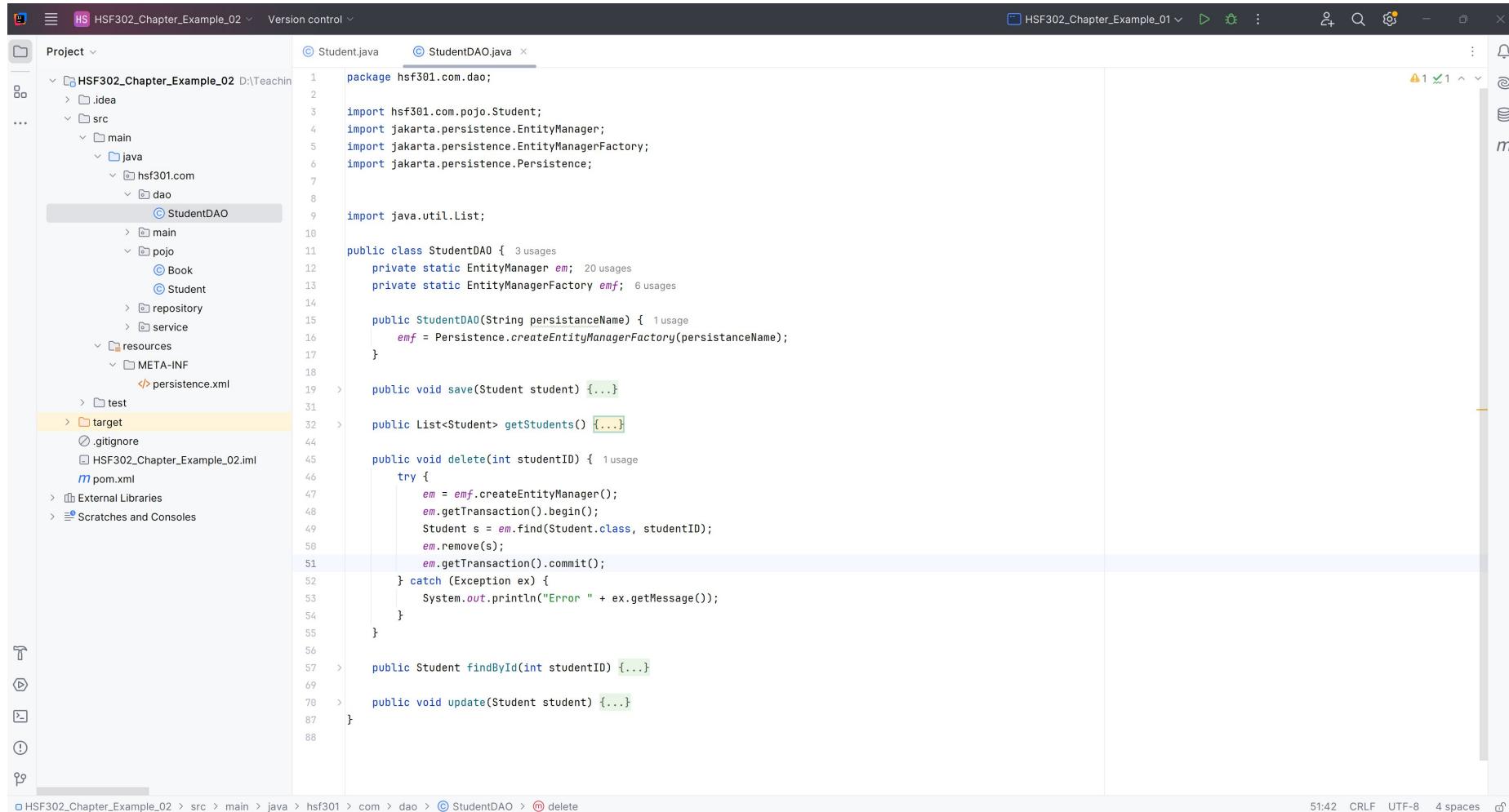
    public Student findById(int studentID) {...}

    public void update(Student student) {...}
}
```

The code editor shows several imports and annotations from the Jakarta Persistence API. The `target` folder in the project structure is highlighted in yellow.

70:41 CRLF UTF-8 4 spaces

Delete Student in StudentDAO



The screenshot shows the IntelliJ IDEA interface with the project `HSF302_Chapter_Example_02` open. The `StudentDAO.java` file is selected in the editor. The code implements a DAO for managing students using Entity Manager. It includes methods for saving, getting all students, deleting a student by ID, finding a student by ID, and updating a student.

```
package hsf301.com.dao;

import hsf301.com.pojo.Student;
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;

import java.util.List;

public class StudentDAO { 3 usages
    private static EntityManager em; 20 usages
    private static EntityManagerFactory emf; 6 usages

    public StudentDAO(String persistanceName) { 1 usage
        emf = Persistence.createEntityManagerFactory(persistanceName);
    }

    public void save(Student student) {...}

    public List<Student> getStudents() {...}

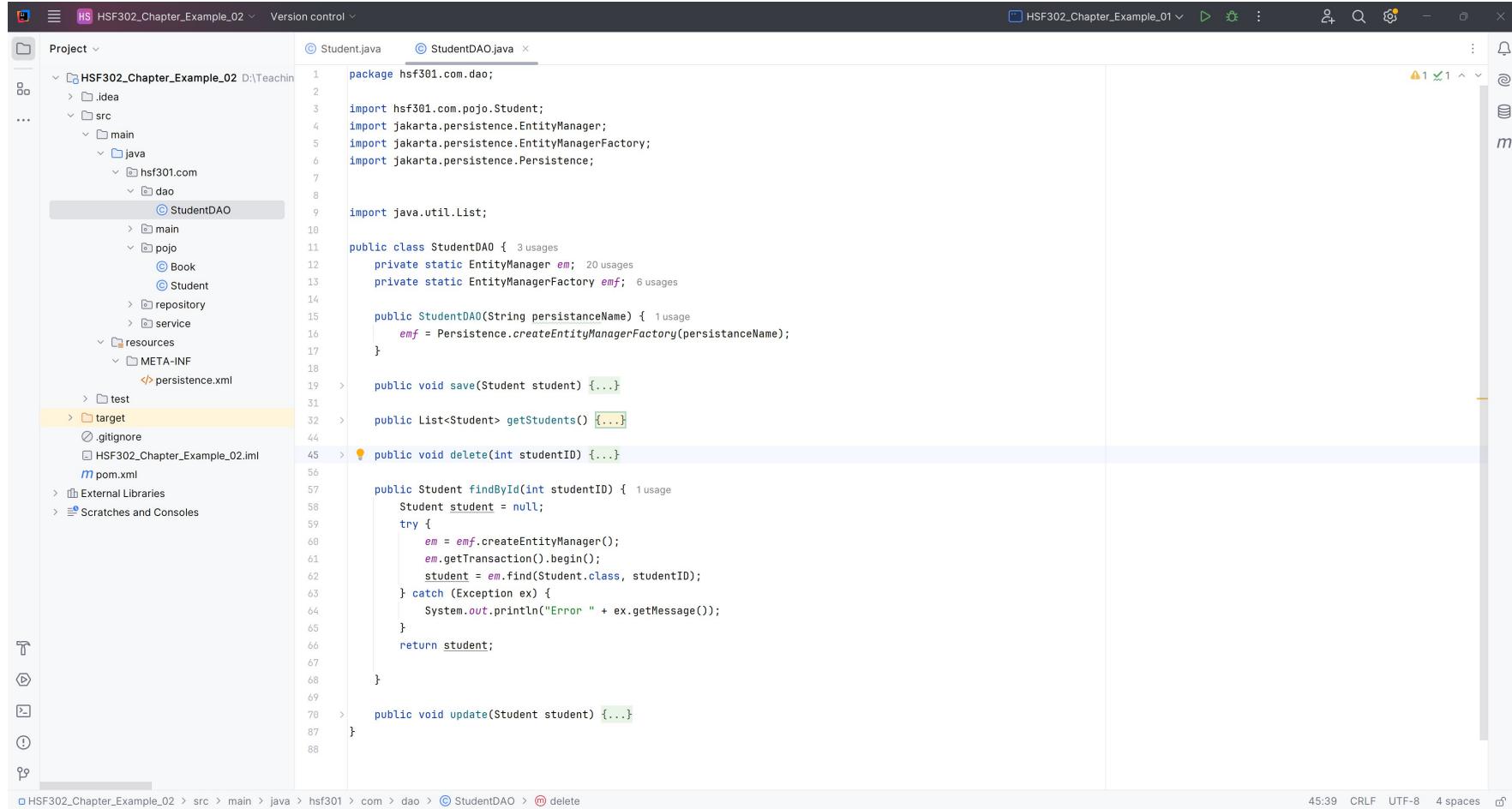
    public void delete(int studentID) { 1 usage
        try {
            em = emf.createEntityManager();
            em.getTransaction().begin();
            Student s = em.find(Student.class, studentID);
            em.remove(s);
            em.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println("Error " + ex.getMessage());
        }
    }

    public Student findById(int studentID) {...}

    public void update(Student student) {...}
}
```

The `target` folder in the project structure is highlighted in yellow. The status bar at the bottom shows the file path `HSF302_Chapter_Example_02 > src > main > java > hsf301 > com > dao > StudentDAO.java`, and the bottom right corner shows the time as 51:42 and file encoding as CRLF.

Get a Student in StudentDAO



The screenshot shows the IntelliJ IDEA interface with the code editor open to the `StudentDAO.java` file. The code implements a DAO interface for managing students using Jakarta Persistence.

```
package hsf301.com.dao;

import hsf301.com.pojo.Student;
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;

import java.util.List;

public class StudentDAO {
    private static EntityManager em; 20 usages
    private static EntityManagerFactory emf; 6 usages

    public StudentDAO(String persistanceName) { 1 usage
        emf = Persistence.createEntityManagerFactory(persistanceName);
    }

    public void save(Student student) {...}

    public List<Student> getStudents() {...}

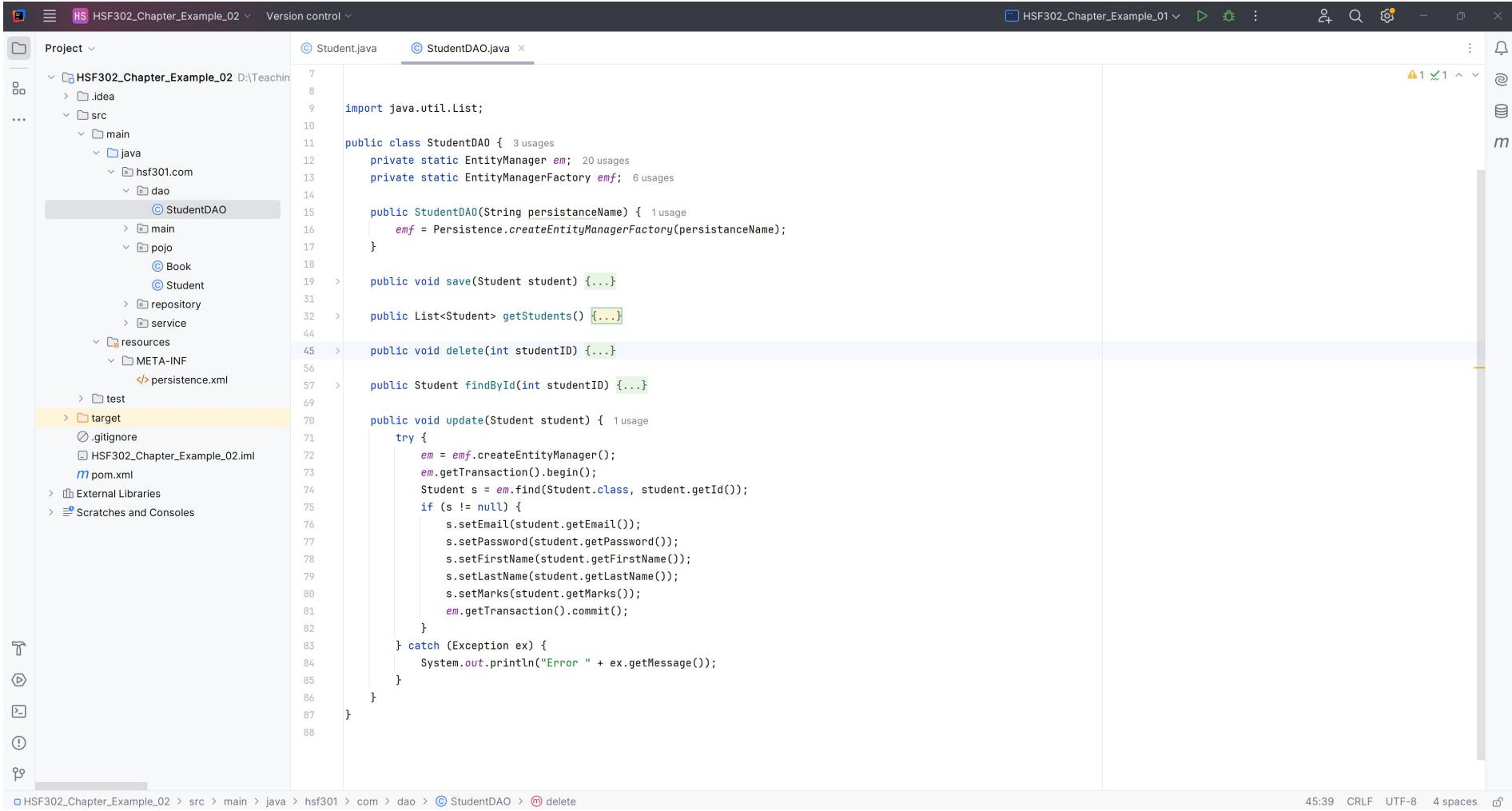
    public void delete(int studentID) {...}

    public Student findById(int studentID) { 1 usage
        Student student = null;
        try {
            em = emf.createEntityManager();
            em.getTransaction().begin();
            student = em.find(Student.class, studentID);
        } catch (Exception ex) {
            System.out.println("Error " + ex.getMessage());
        }
        return student;
    }

    public void update(Student student) {...}
}
```

The code editor shows several imports and annotations from the Jakarta Persistence API. The `StudentDAO` class contains methods for saving, getting all students, deleting, finding by ID, and updating students. It uses a private static `EntityManager` and `EntityManagerFactory` to interact with the database.

UpdateStudent in StudentDAO



The screenshot shows the IntelliJ IDEA interface with the project `HSF302_Chapter_Example_02` open. The `StudentDAO.java` file is selected in the editor tab bar. The code implements a DAO interface for managing students using JPA:

```
import java.util.List;

public class StudentDAO {    3 usages
    private static EntityManager em;    20 usages
    private static EntityManagerFactory emf;    6 usages

    public StudentDAO(String persistanceName) {    1 usage
        emf = Persistence.createEntityManagerFactory(persistanceName);
    }

    public void save(Student student) {...}

    public List<Student> getStudents() {...}

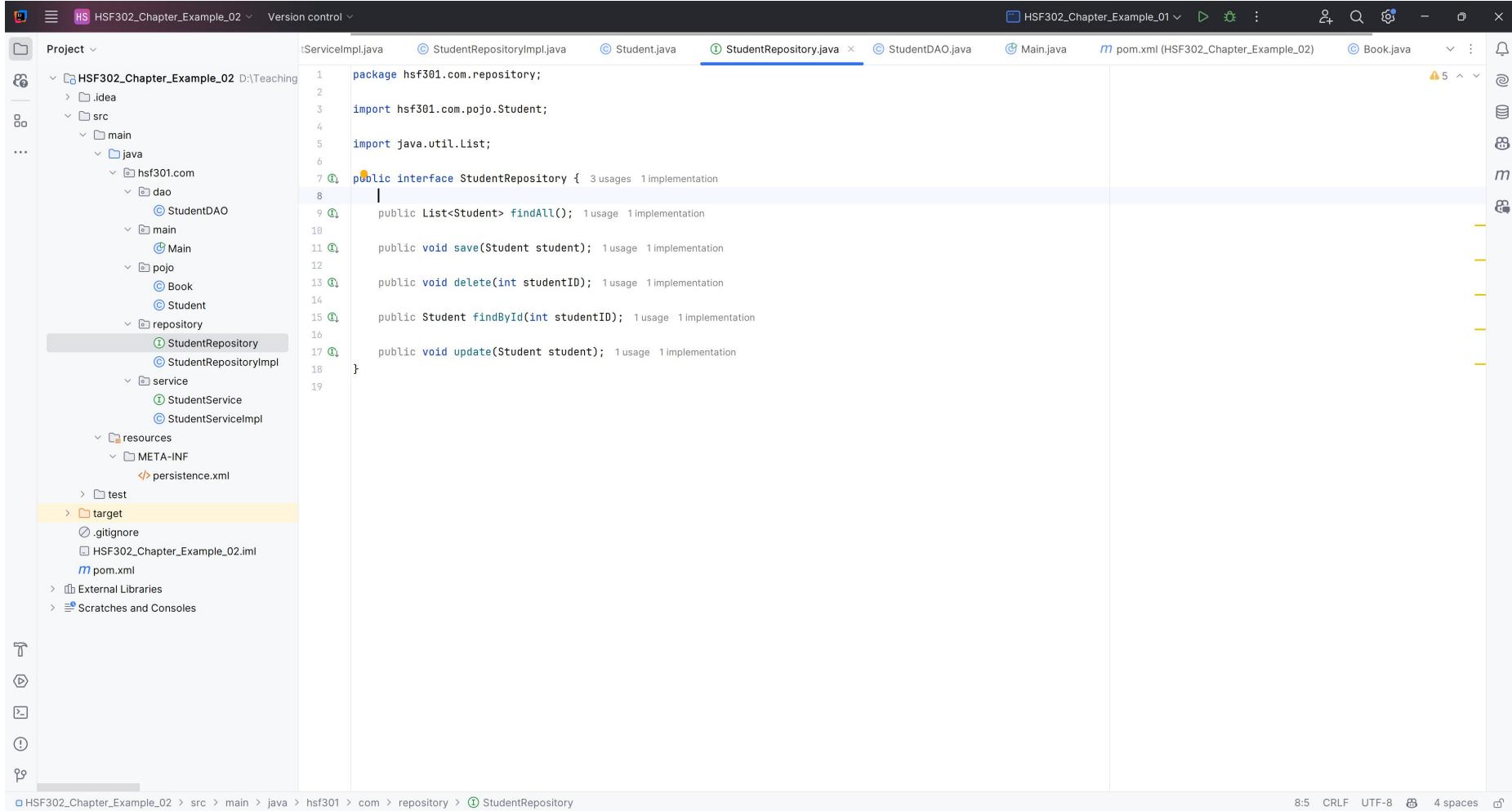
    public void delete(int studentID) {...}

    public Student findById(int studentID) {...}

    public void update(Student student) {    1 usage
        try {
            em = emf.createEntityManager();
            em.getTransaction().begin();
            Student s = em.find(Student.class, student.getId());
            if (s != null) {
                s.setEmail(student.getEmail());
                s.setPassword(student.getPassword());
                s.setFirstName(student.getFirstName());
                s.setLastName(student.getLastName());
                s.setMarks(student.getMarks());
                em.getTransaction().commit();
            }
        } catch (Exception ex) {
            System.out.println("Error " + ex.getMessage());
        }
    }
}
```

The code uses `EntityManager` and `EntityManagerFactory` from the `persistence.xml` configuration file. The `update` method performs a transactional update of a student's details in the database.

Create StudentRepository



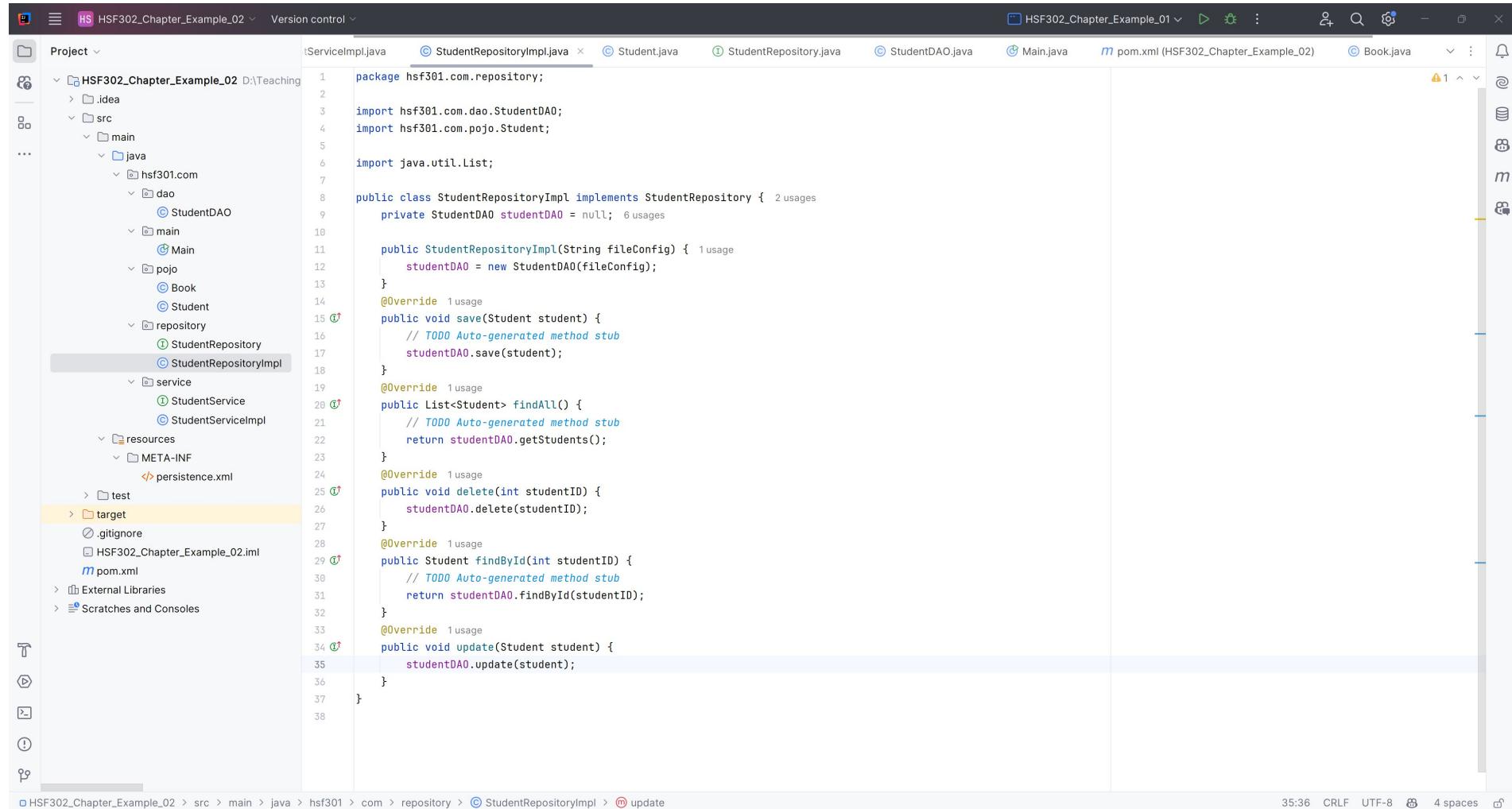
The screenshot shows the IntelliJ IDEA interface with the project `HSF302_Chapter_Example_02` open. The `StudentRepository.java` file is selected in the editor tab bar. The code defines a public interface `StudentRepository` with methods for finding all students, saving a student, deleting a student by ID, finding a student by ID, and updating a student.

```
package hsf301.com.repository;

import hsf301.com.pojo.Student;
import java.util.List;

public interface StudentRepository {
    public List<Student> findAll();
    public void save(Student student);
    public void delete(int studentID);
    public Student findById(int studentID);
    public void update(Student student);
}
```

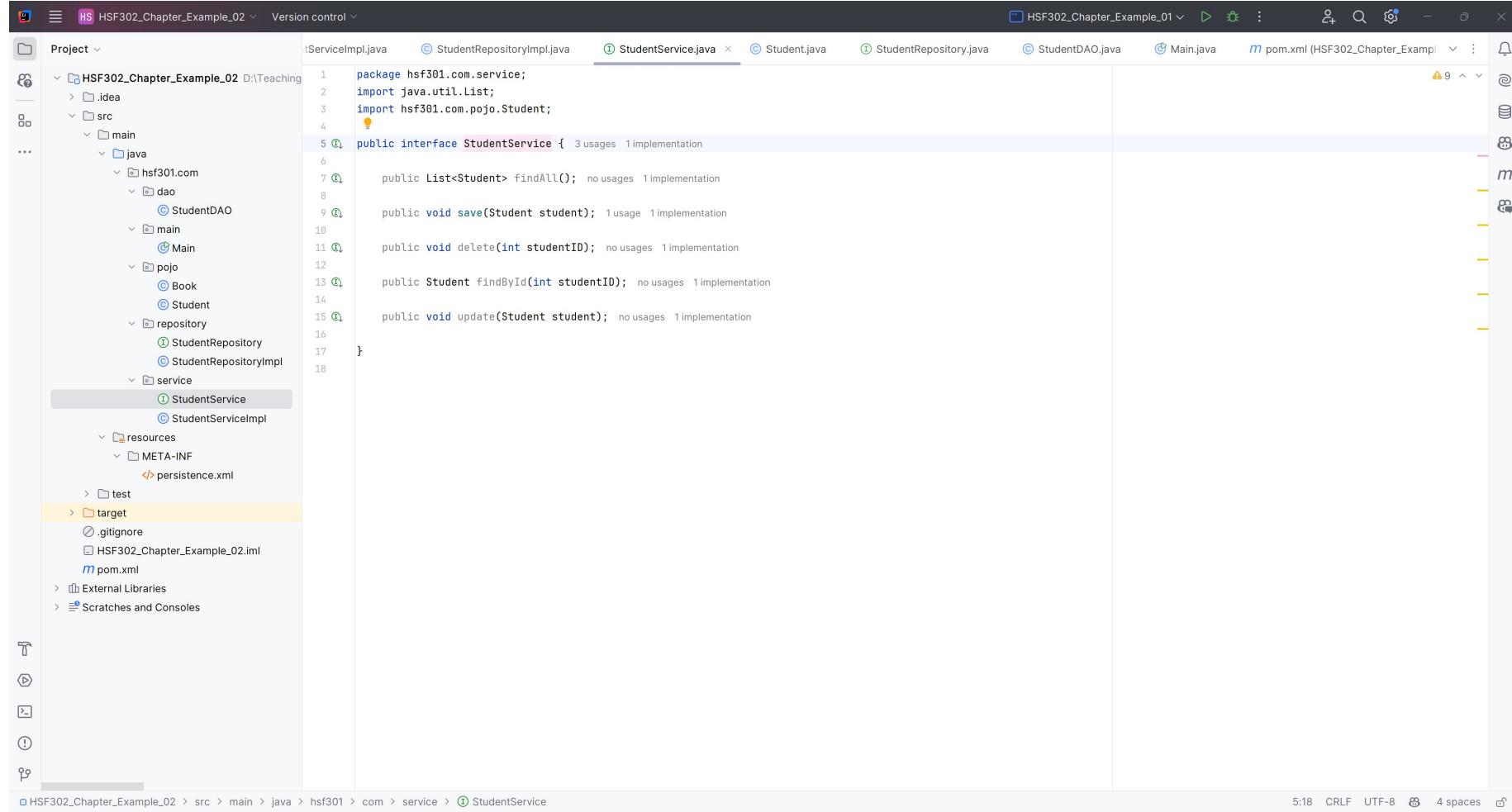
Create StudentRepositoryImpl



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

- Project Structure:** The project is named "HSF302_Chapter_Example_02". The "src" directory contains "main", "java", "resources", "META-INF", and "test" sub-directories. The "java" directory has packages "hsf301.com" and "hsf301.com.repository". Inside "hsf301.com.repository", there are "StudentRepository" and "StudentRepositoryImpl" files.
- Code Editor:** The "StudentRepositoryImpl.java" file is open. It imports "hsf301.com.dao.StudentDAO", "hsf301.com.pojo.Student", and "java.util.List". The class "StudentRepositoryImpl" implements "StudentRepository" and uses "StudentDAO" to perform operations like save, find, delete, and update.
- Status Bar:** The status bar at the bottom shows the file path: "HSF302_Chapter_Example_02 > src > main > java > hsf301 > com > repository > StudentRepositoryImpl > update". It also displays file statistics: 35:36, CRLF, UTF-8, 4 spaces.

Create StudentService

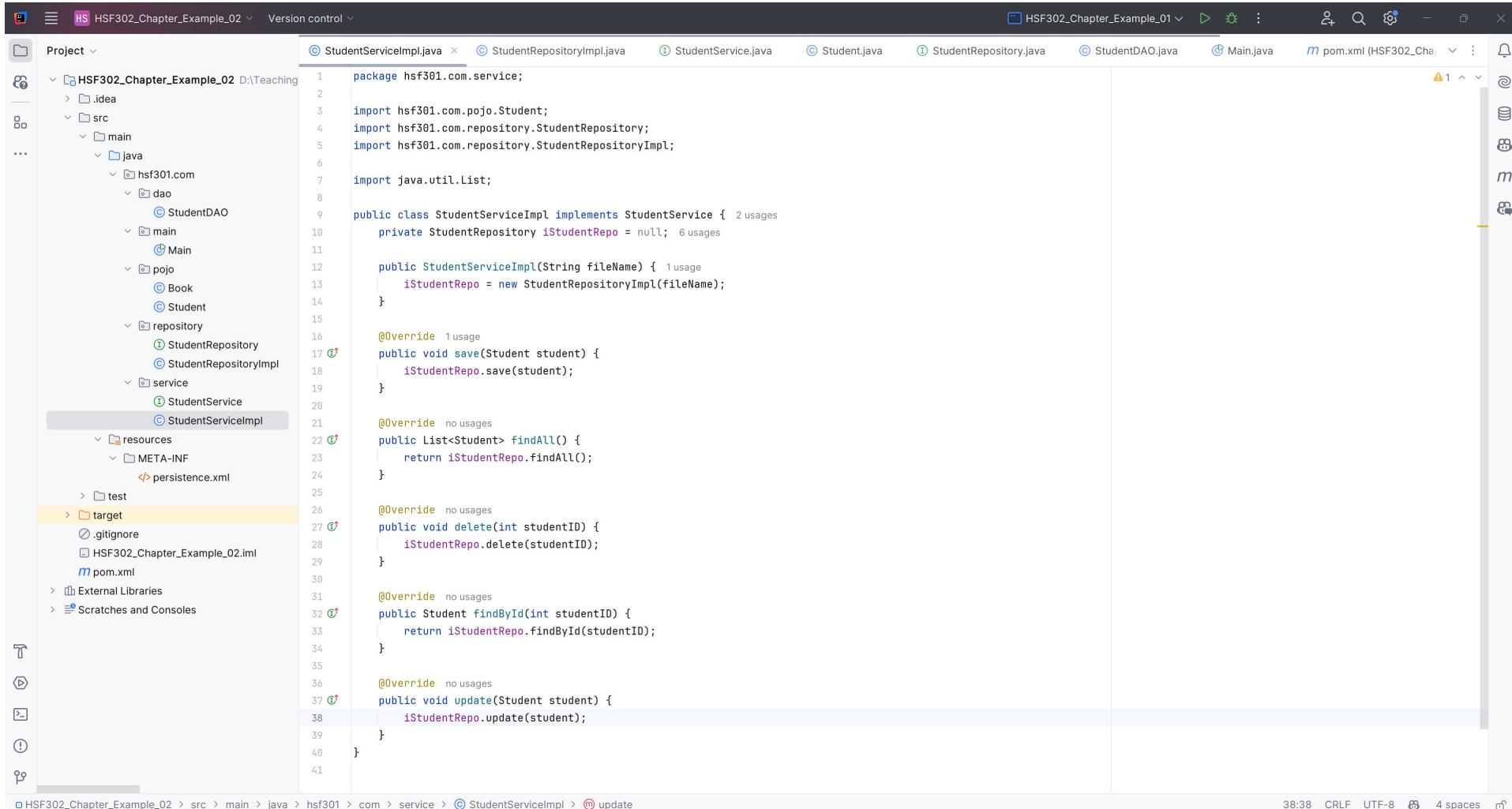


The screenshot shows the IntelliJ IDEA IDE interface with the project `HSF302_Chapter_Example_02` open. The `StudentService.java` file is selected in the editor. The code defines a public interface `StudentService` with the following methods:

```
1 package hsf301.com.service;
2 import java.util.List;
3 import hsf301.com.pojo.Student;
4
5 public interface StudentService {
6     List<Student> findAll();
7     void save(Student student);
8     void delete(int studentID);
9     Student findById(int studentID);
10    void update(Student student);
11 }
```

The `src` directory contains the `main` package, which further contains `java`, `dao`, `main`, `pojo`, and `repository` sub-packages. Within `main.java`, there is a `StudentService` class and an `StudentServiceImpl` class. The `repository` package contains `StudentRepository` and `StudentRepositoryImpl`. The `resources` directory contains a `META-INF` folder with a `persistence.xml` file.

Create StudentServiceImpl



The screenshot shows the IntelliJ IDEA interface with the project `HSF302_Chapter_Example_02` open. The code editor displays the `StudentServiceImpl.java` file, which implements the `StudentService` interface. The code uses dependency injection via a private field `iStudentRepo`. It overrides methods from the interface to interact with a `StudentRepositoryImpl`. The code editor highlights several annotations and method signatures with red and green indicators. The left sidebar shows the project structure with packages like `hsf301.com.service`, `hsf301.com.pojo`, `hsf301.com.repository`, and `hsf301.com.dao`. The bottom status bar indicates the file path, encoding, and other settings.

```
package hsf301.com.service;

import hsf301.com.pojo.Student;
import hsf301.com.repository.StudentRepository;
import hsf301.com.repository.StudentRepositoryImpl;

import java.util.List;

public class StudentServiceImpl implements StudentService {
    private StudentRepository iStudentRepo = null;

    public StudentServiceImpl(String fileName) {
        iStudentRepo = new StudentRepositoryImpl(fileName);
    }

    @Override
    public void save(Student student) {
        iStudentRepo.save(student);
    }

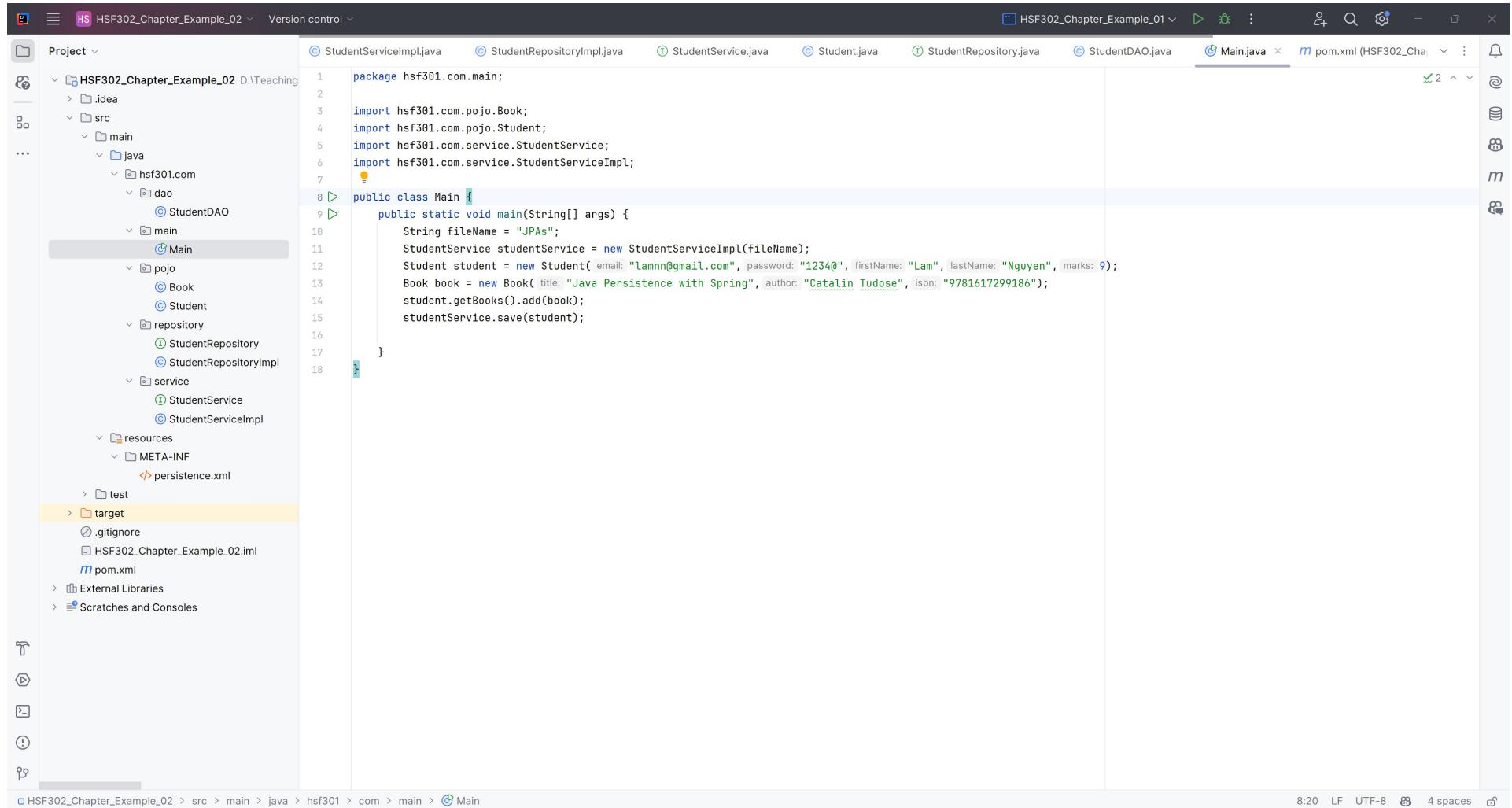
    @Override
    public List<Student> findAll() {
        return iStudentRepo.findAll();
    }

    @Override
    public void delete(int studentID) {
        iStudentRepo.delete(studentID);
    }

    @Override
    public Student findById(int studentID) {
        return iStudentRepo.findById(studentID);
    }

    @Override
    public void update(Student student) {
        iStudentRepo.update(student);
    }
}
```

Create Main function



The screenshot shows the IntelliJ IDEA interface with a Java project named "HSF302_Chapter_Example_02". The "src" directory contains a "main" package with a "java" subdirectory. Inside "java", there is a file named "Main.java". The code in "Main.java" is as follows:

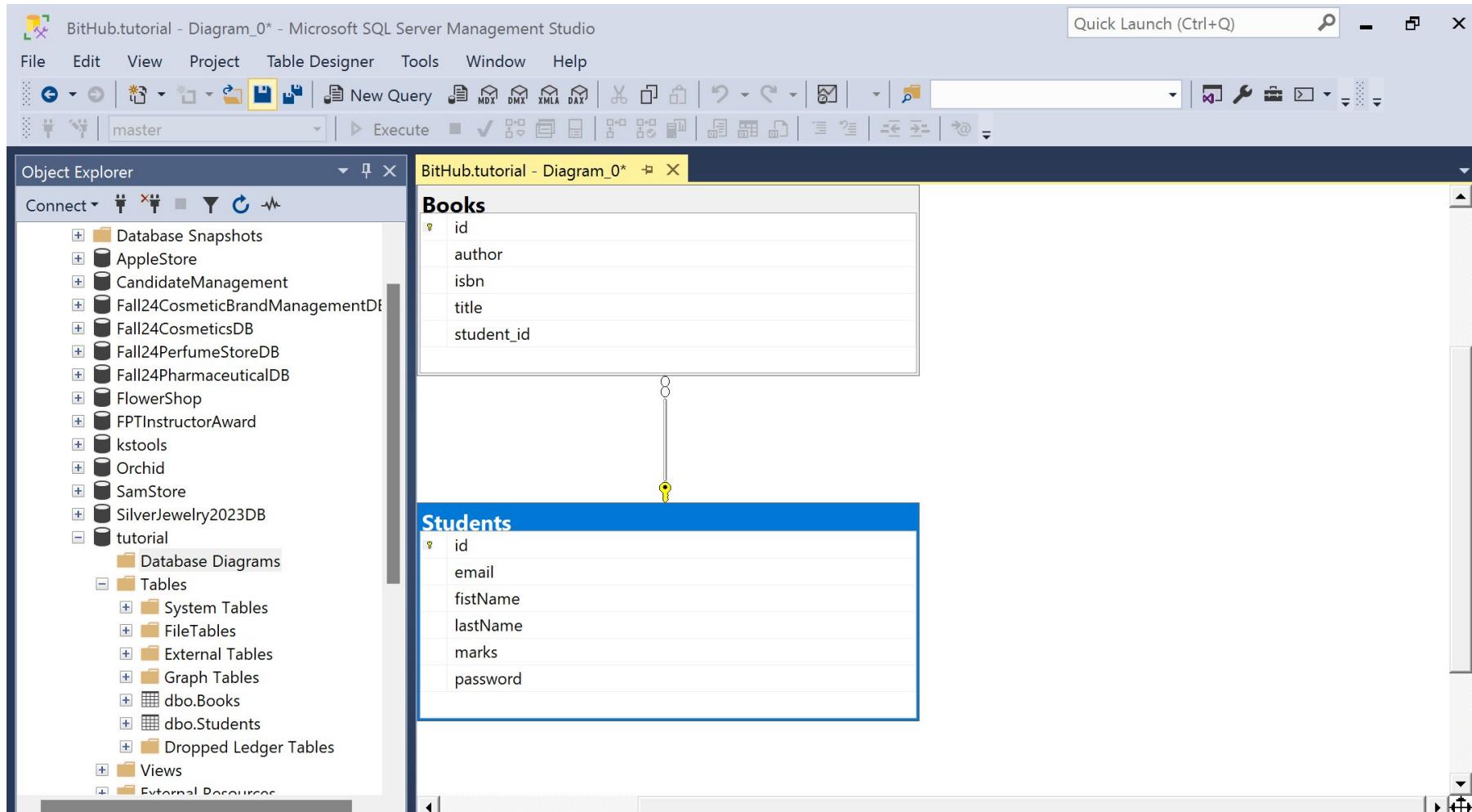
```
package hsf301.com.main;

import hsf301.com.pojo.Book;
import hsf301.com.pojo.Student;
import hsf301.com.service.StudentService;
import hsf301.com.service.StudentServiceImpl;

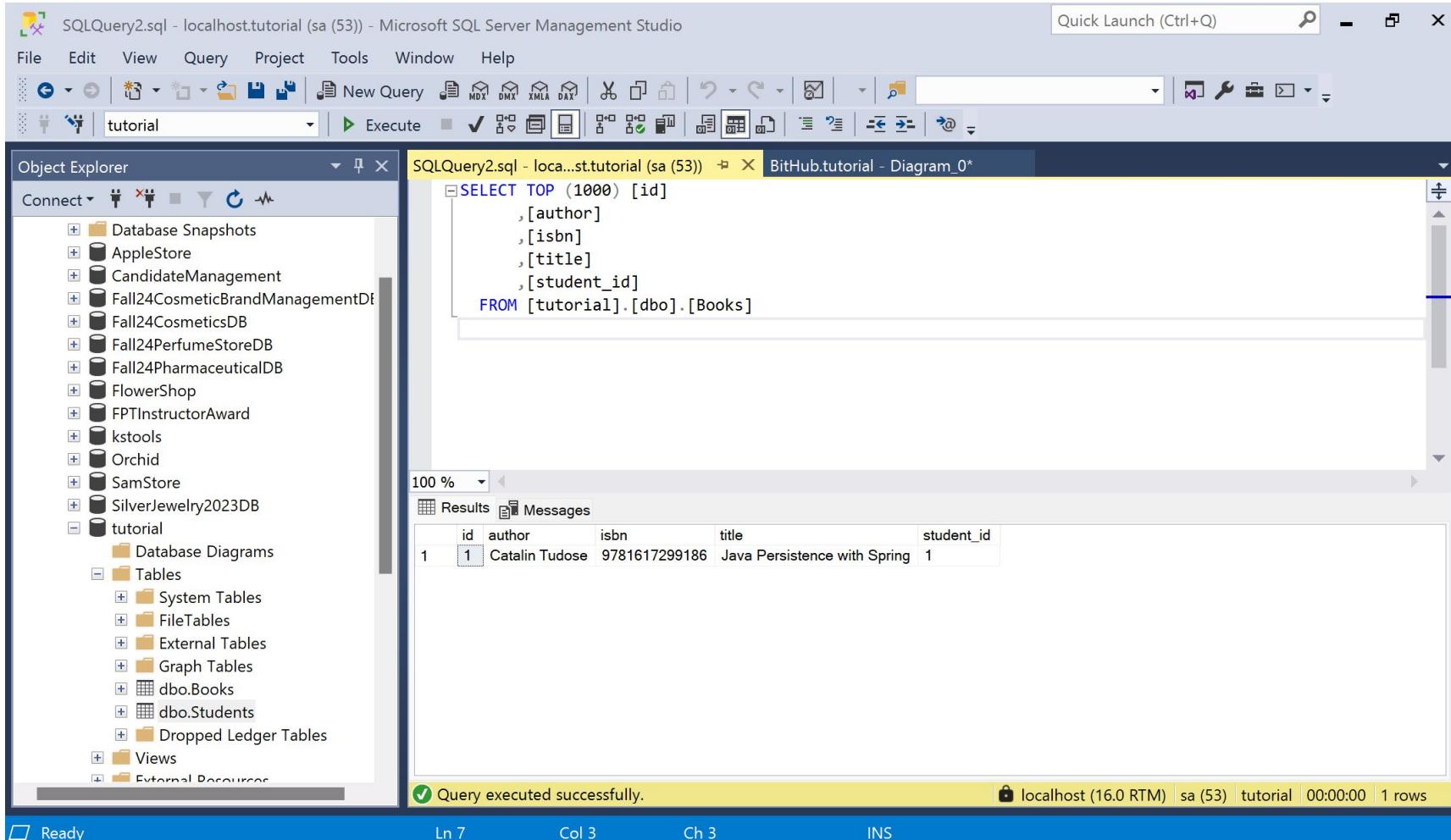
public class Main {
    public static void main(String[] args) {
        String fileName = "JPAs";
        StudentService studentService = new StudentServiceImpl(fileName);
        Student student = new Student( email: "lamnn@gmail.com", password: "1234@", firstName: "Lam", lastName: "Nguyen", marks: 9 );
        Book book = new Book( title: "Java Persistence with Spring", author: "Catalin Tudose", isbn: "9781617299186" );
        student.getBooks().add(book);
        studentService.save(student);
    }
}
```

The "target" directory is highlighted in yellow at the bottom left of the project tree. The status bar at the bottom right shows the time as 8:20, encoding as LF UTF-8, and a code style setting of 4 spaces.

Result



Result



The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. The title bar reads "SQLQuery2.sql - localhost.tutorial (sa (53)) - Microsoft SQL Server Management Studio". The Object Explorer on the left lists various databases, with "tutorial" selected. The central pane displays a T-SQL query:

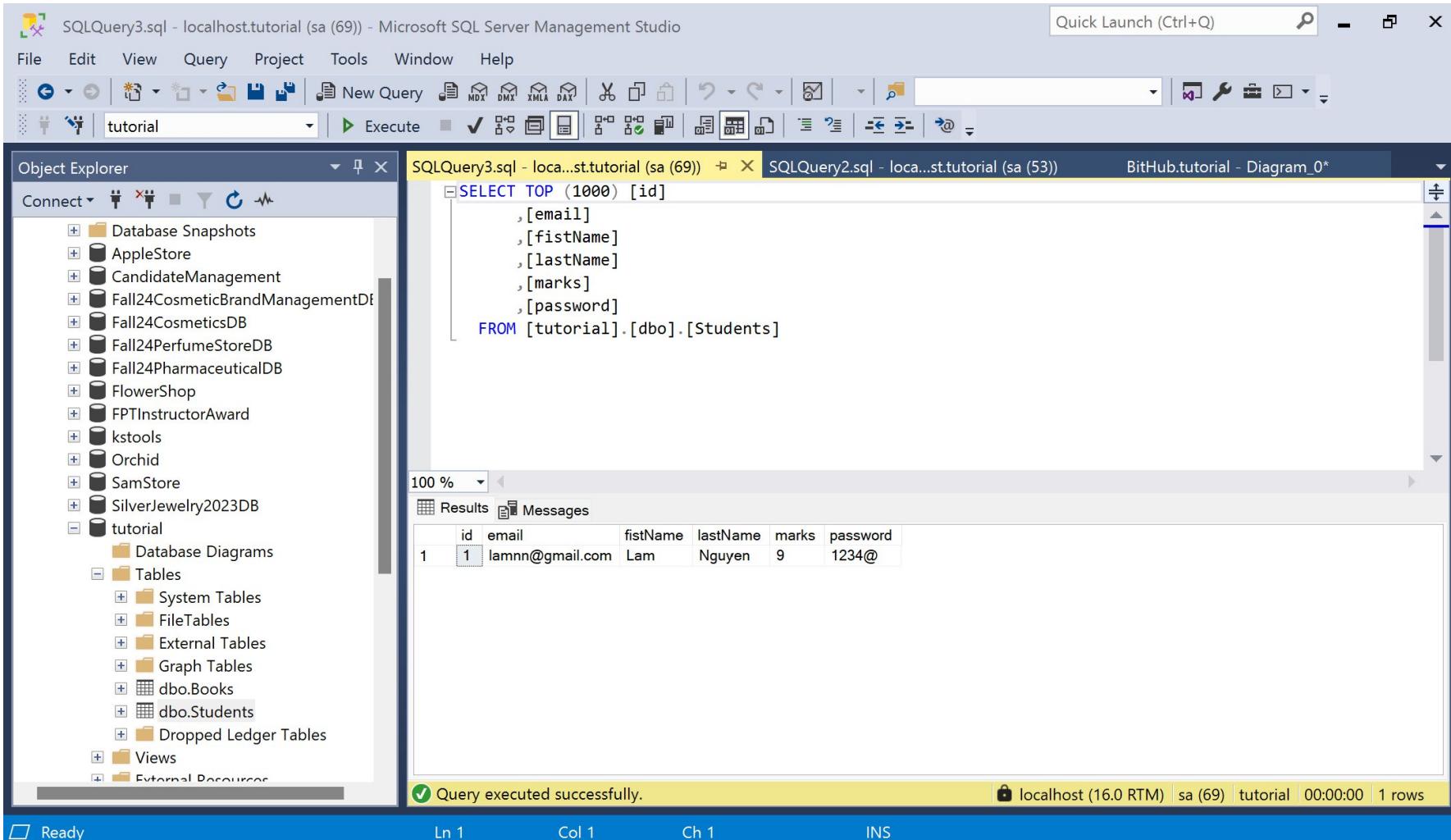
```
SELECT TOP (1000) [id]
      ,[author]
      ,[isbn]
      ,[title]
      ,[student_id]
  FROM [tutorial].[dbo].[Books]
```

The Results pane below shows the output of the query:

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

A status bar at the bottom indicates "Query executed successfully." and "localhost (16.0 RTM) | sa (53) | tutorial | 00:00:00 | 1 rows".

Result



The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. The title bar reads "SQLQuery3.sql - localhost.tutorial (sa (69)) - Microsoft SQL Server Management Studio". The menu bar includes File, Edit, View, Query, Project, Tools, Window, and Help. The toolbar has various icons for database management tasks. The Object Explorer on the left lists several databases, with "tutorial" selected. The "Tables" node under "tutorial" is expanded, showing "dbo.Books", "dbo.Students", and "Dropped Ledger Tables". The central pane displays a query window with the following SQL code:

```
SELECT TOP (1000) [id]
    ,[email]
    ,[firstName]
    ,[lastName]
    ,[marks]
    ,[password]
FROM [tutorial].[dbo].[Students]
```

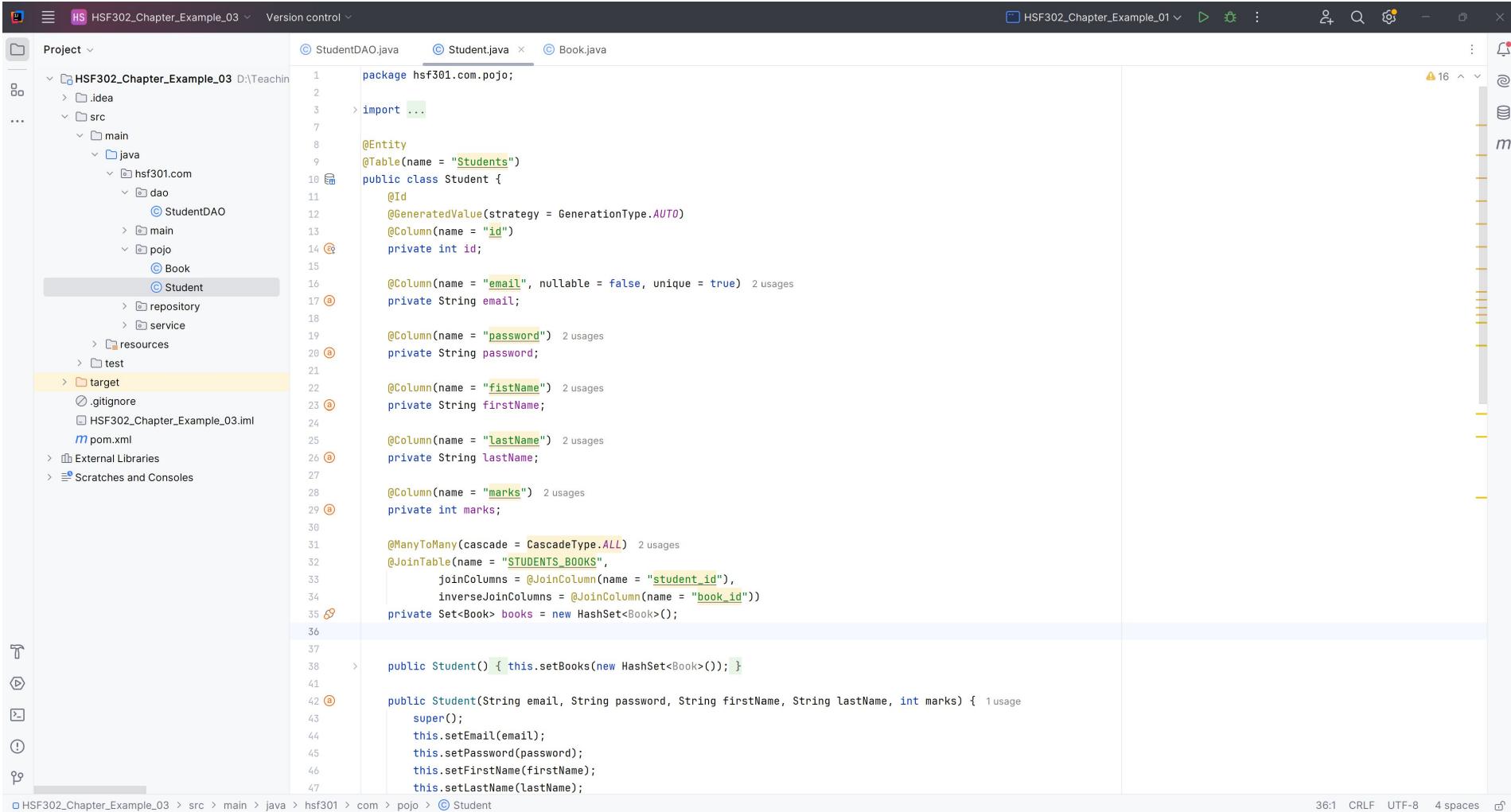
The results pane below shows a table with one row of data:

	id	email	firstName	lastName	marks	password
1	1	lamnn@gmail.com	Lam	Nguyen	9	1234@

A status bar at the bottom indicates "Query executed successfully." and "localhost (16.0 RTM) | sa (69) | tutorial | 00:00:00 | 1 rows".

Demo JPA (Many To Many)

2. Create Student.java in Pojo's Package



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

- Project Structure:** The project is named "HSF302_Chapter_Example_03". The "src" directory contains "main", "java", and "pojo". Inside "pojo", there are "Book.java" and "Student.java". "Student.java" is currently selected.
- Code Editor:** The code editor displays the "Student.java" file. The code defines a class "Student" with annotations for Entity, Table, and Column. It includes fields for id, email, password, firstName, lastName, and marks, along with a many-to-many relationship to "Book" through a join table "STUDENTS_BOOKS".
- Toolbars and Status Bar:** The top bar shows tabs for "StudentDAO.java" and "Book.java". The status bar at the bottom right indicates the code is 36.1% complete, uses CRLF line endings, is in UTF-8 encoding, and has 4 spaces of indentation.

```
package hsf301.com.pojo;

import ...;

@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;

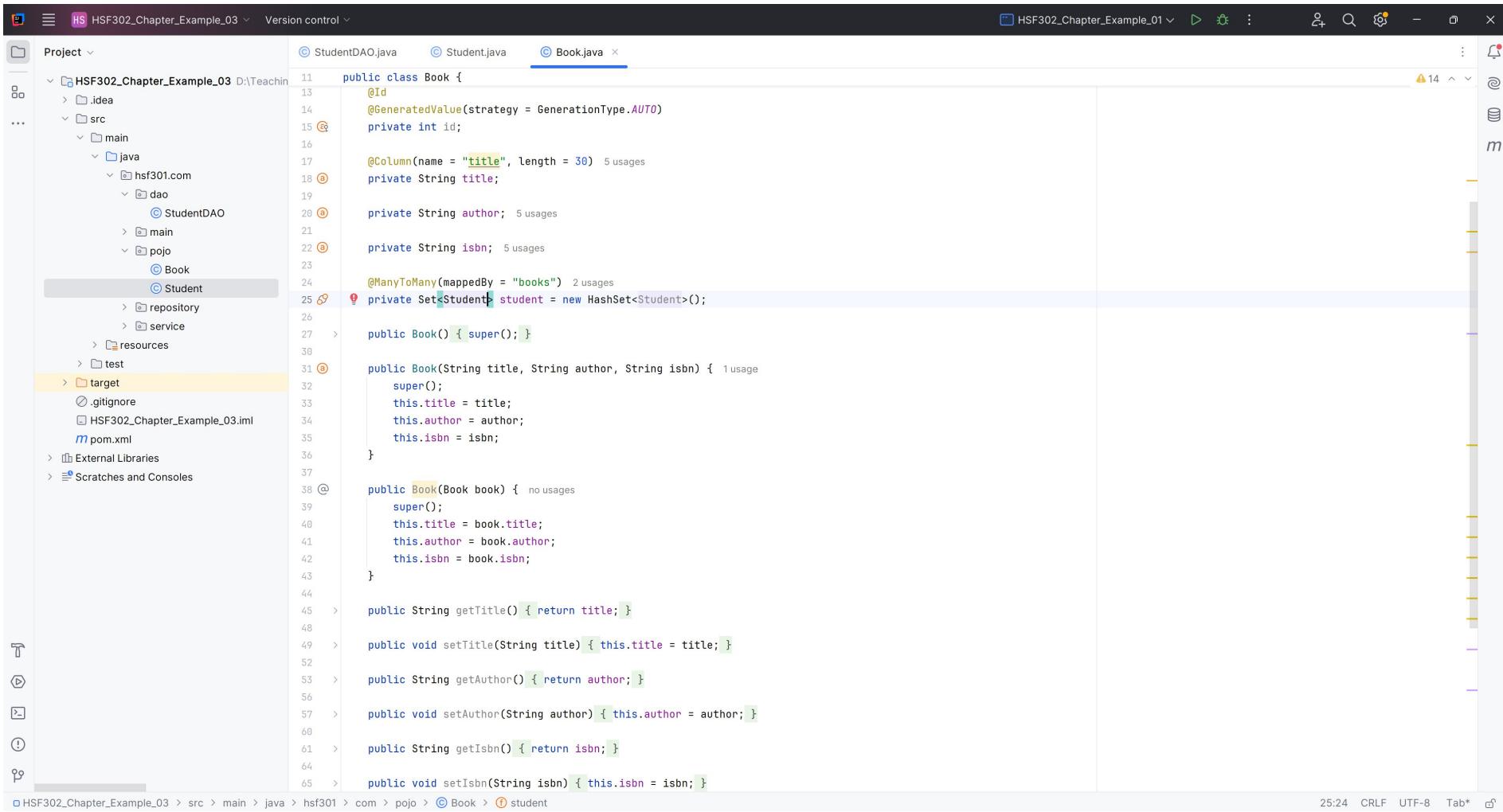
    @Column(name = "marks") 2 usages
    private int marks;

    @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<Book>();

    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);
    }
}
```

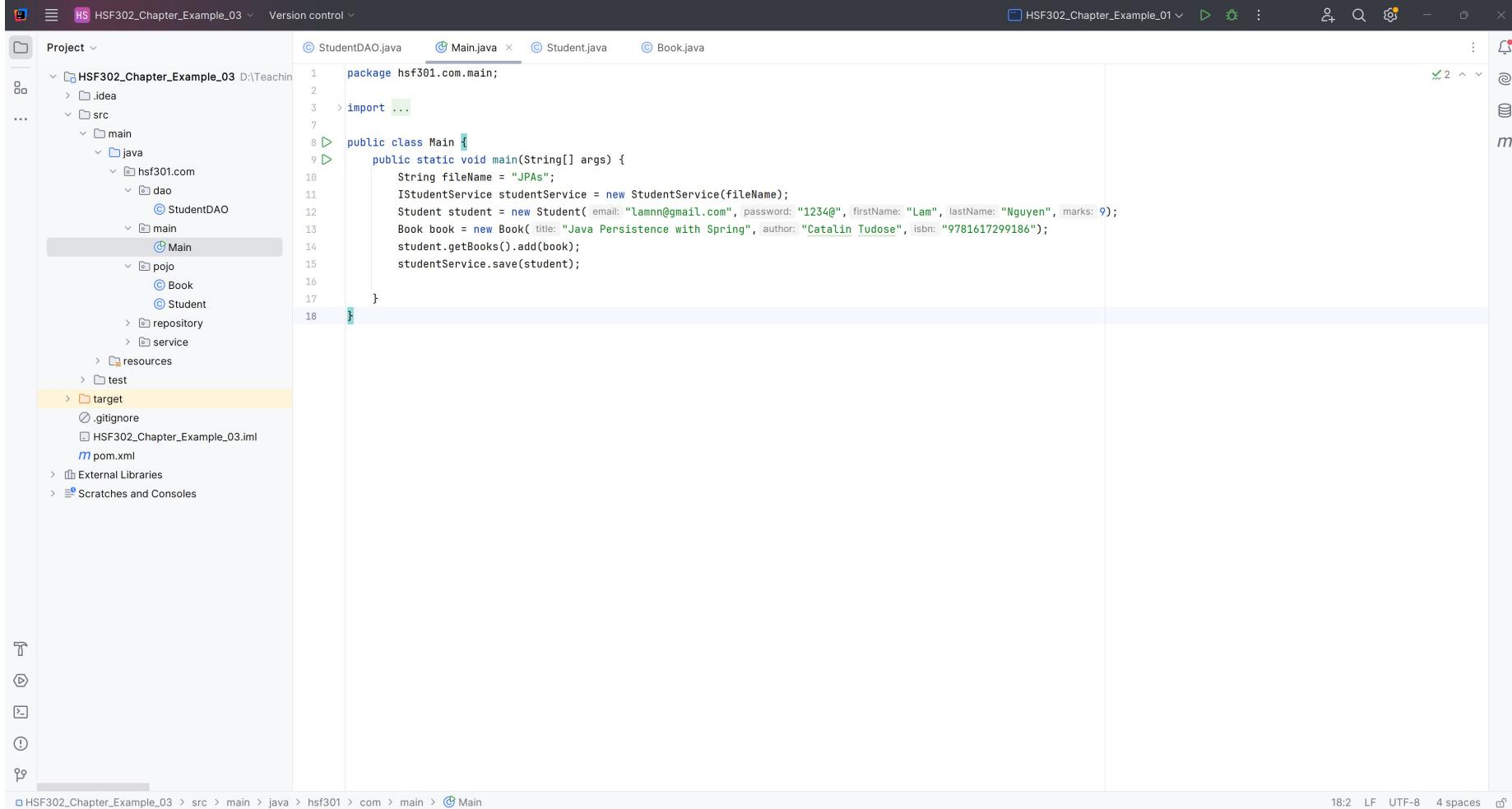
1. Create Book.java in Pojo's Package



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

- Project Structure:** The project is named "HSF302_Chapter_Example_03". The "src" directory contains "main", "java", "hsf301.com", "dao", "main", "pojo", "Book", and "Student" packages.
- Code Editor:** The "Book.java" file is open. The code defines a class "Book" with fields: id (auto-generated), title (length 30), author, and isbn. It has a many-to-many relationship with "Student" via a "books" collection. Constructors, getters, and setters are provided.
- Status Bar:** Shows the file path as "HSF302_Chapter_Example_03 > src > main > java > hsf301 > com > pojo > Book > Book.java", and the status "25:24 CRLF UTF-8 Tab*".

3. Run Program



The screenshot shows the IntelliJ IDEA interface with the project `HSF302_Chapter_Example_03` open. The `Main.java` file is selected in the editor tab bar. The code in `Main.java` is:

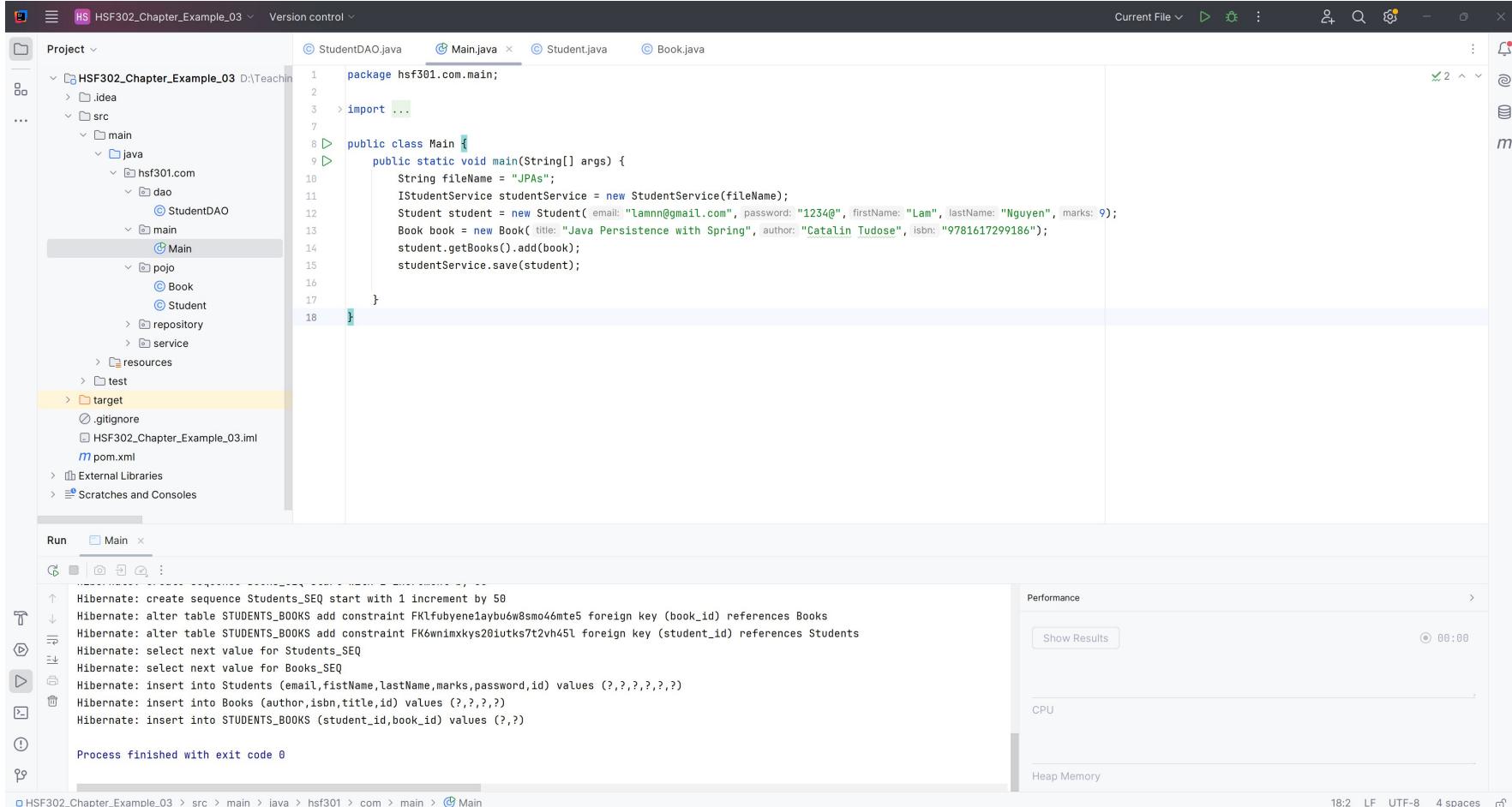
```
1 package hsf301.com.main;
2
3 import ...
4
5 public class Main {
6     public static void main(String[] args) {
7         String fileName = "JPAs";
8         IStudentService studentService = new StudentService(fileName);
9         Student student = new Student( email: "lamnn@gmail.com", password: "1234@", firstName: "Lam", lastName: "Nguyen", marks: 9 );
10        Book book = new Book( title: "Java Persistence with Spring", author: "Catalin Tudose", isbn: "9781617299186" );
11        student.getBooks().add(book);
12        studentService.save(student);
13    }
14
15 }
```

The project structure on the left shows the following directory tree:

- `HSF302_Chapter_Example_03`
 - `.idea`
 - `src`
 - `main`
 - `java`
 - `hsf301.com`
 - `dao`
 - `StudentDAO`
 - `main`
 - `Main` (selected)
 - `pojo`
 - `Book`
 - `Student`
 - `repository`
 - `service`
 - `resources`
 - `test`
 - `target`
 - `.gitignore`
 - `HSF302_Chapter_Example_03.iml`
 - `pom.xml`

The status bar at the bottom indicates the file path `HSF302_Chapter_Example_03 > src > main > java > hsf301 > com > main > Main`, encoding `UTF-8`, and character count `18:2`.

4. Result



The screenshot shows the IntelliJ IDEA IDE interface. The left sidebar displays the project structure for 'HSF302_Chapter_Example_03'. The main editor window shows the 'Main.java' file with the following code:

```
package hsf301.com.main;
import ...
public class Main {
    public static void main(String[] args) {
        String fileName = "JPAs";
        IStudentService studentService = new StudentService(fileName);
        Student student = new Student( email: "lamnn@gmail.com", password: "1234@", firstName: "Lam", lastName: "Nguyen", marks: 9 );
        Book book = new Book( title: "Java Persistence with Spring", author: "Catalin Tudose", isbn: "9781617299186" );
        student.getBooks().add(book);
        studentService.save(student);
    }
}
```

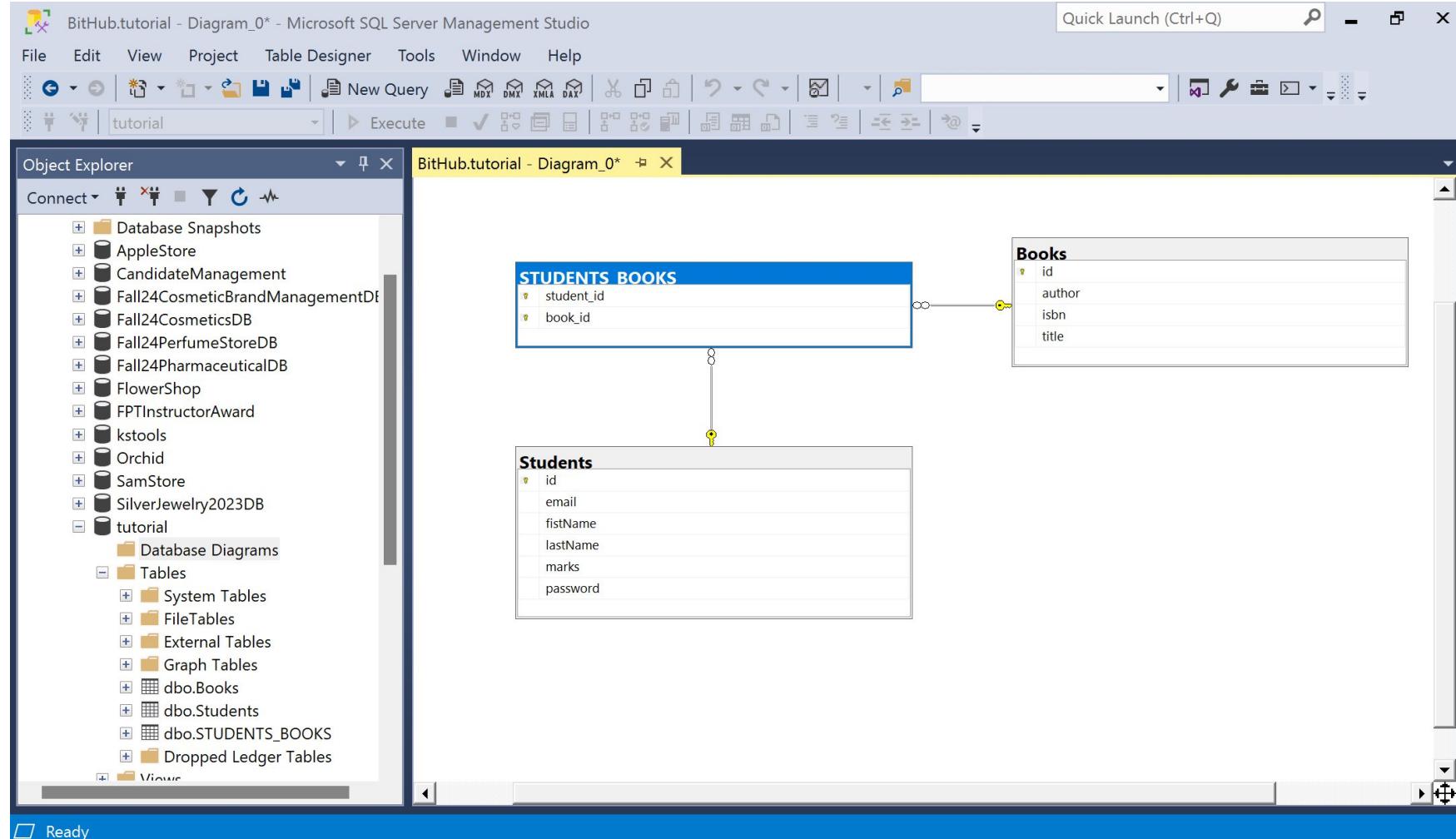
The bottom-left panel shows the run history with the following log output:

```
Hibernate: create sequence Students_SEQ start with 1 increment by 50
Hibernate: alter table STUDENTS_BOOKS add constraint FKlfubyene1aybu6w8smo46mte5 foreign key (book_id) references Books
Hibernate: alter table STUDENTS_BOOKS add constraint FK6wnimxkys20iutks7t2vh45l foreign key (student_id) references Students
Hibernate: select next value for Students_SEQ
Hibernate: select next value for Books_SEQ
Hibernate: insert into Students (email,fistName,lastName,marks,password,id) values (?, ?, ?, ?, ?, ?)
Hibernate: insert into Books (author,isbn,title,id) values (?, ?, ?, ?)
Hibernate: insert into STUDENTS_BOOKS (student_id,book_id) values (?, ?)

Process finished with exit code 0
```

The bottom right panel shows performance monitoring for CPU and Heap Memory.

5. Result



Summary

- ❖ Concepts were introduced:
 - ❖ Overview about JPA
 - ❖ Architecture Overview new features of JPA
 - ❖ Why JPA is selected as develop application?
 - ❖ Explain and demo using Eclipse IDE to create JPA Console App
 - ❖ Create and Run cross-platform Console application with Java connect to MSSQL with Repository Pattern