

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.

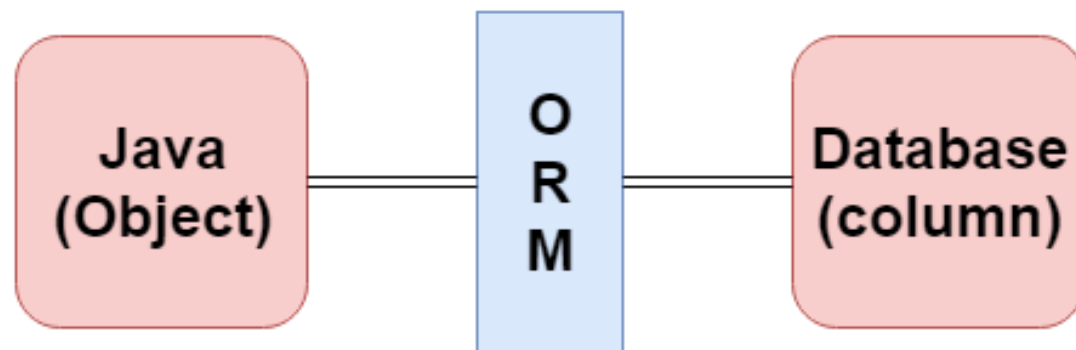
Contents

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

Object Relational Mapping (ORM)

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 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

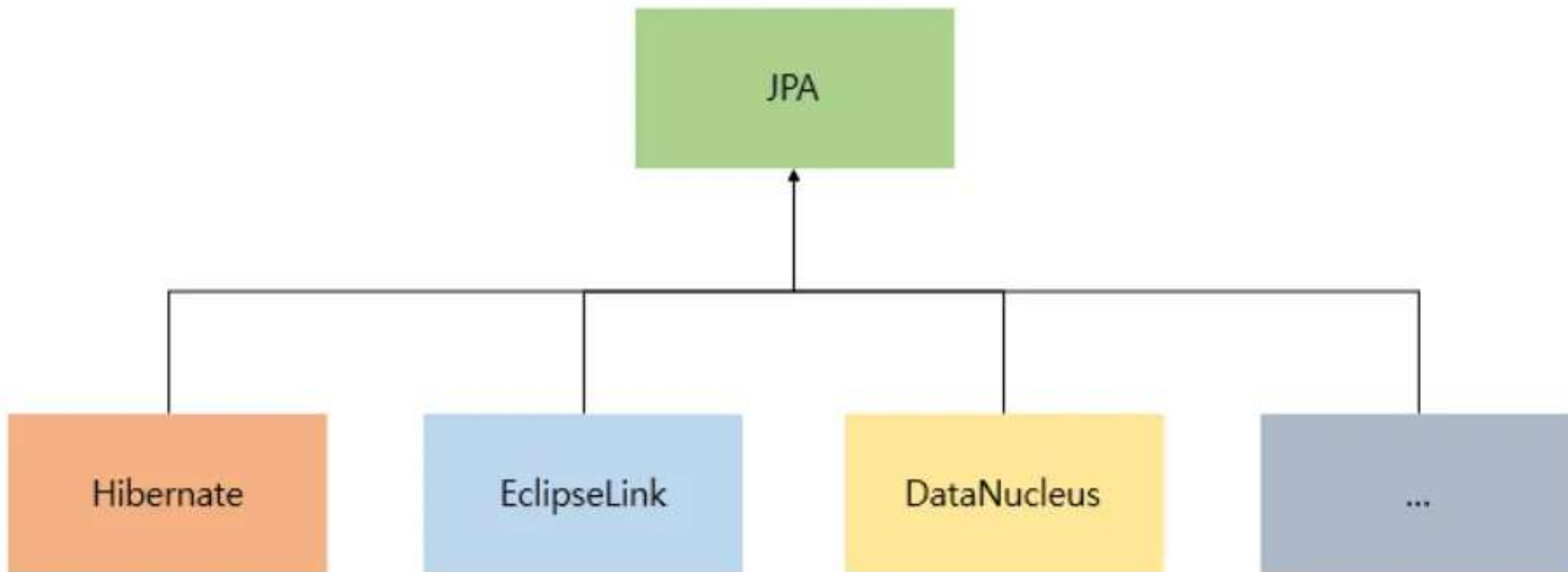
- ❖ **Hibernate:** The most widely used JPA implementation, offering a rich feature set and extensive customization options.
- ❖ **EclipseLink:** Another JPA implementation with a focus on standards compliance and portability.
- ❖ **Spring Data JPA:** An abstraction layer on top of JPA providers, simplifying 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)

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

JPA Implementation Options



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.

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.

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.

JPA Entity Manager

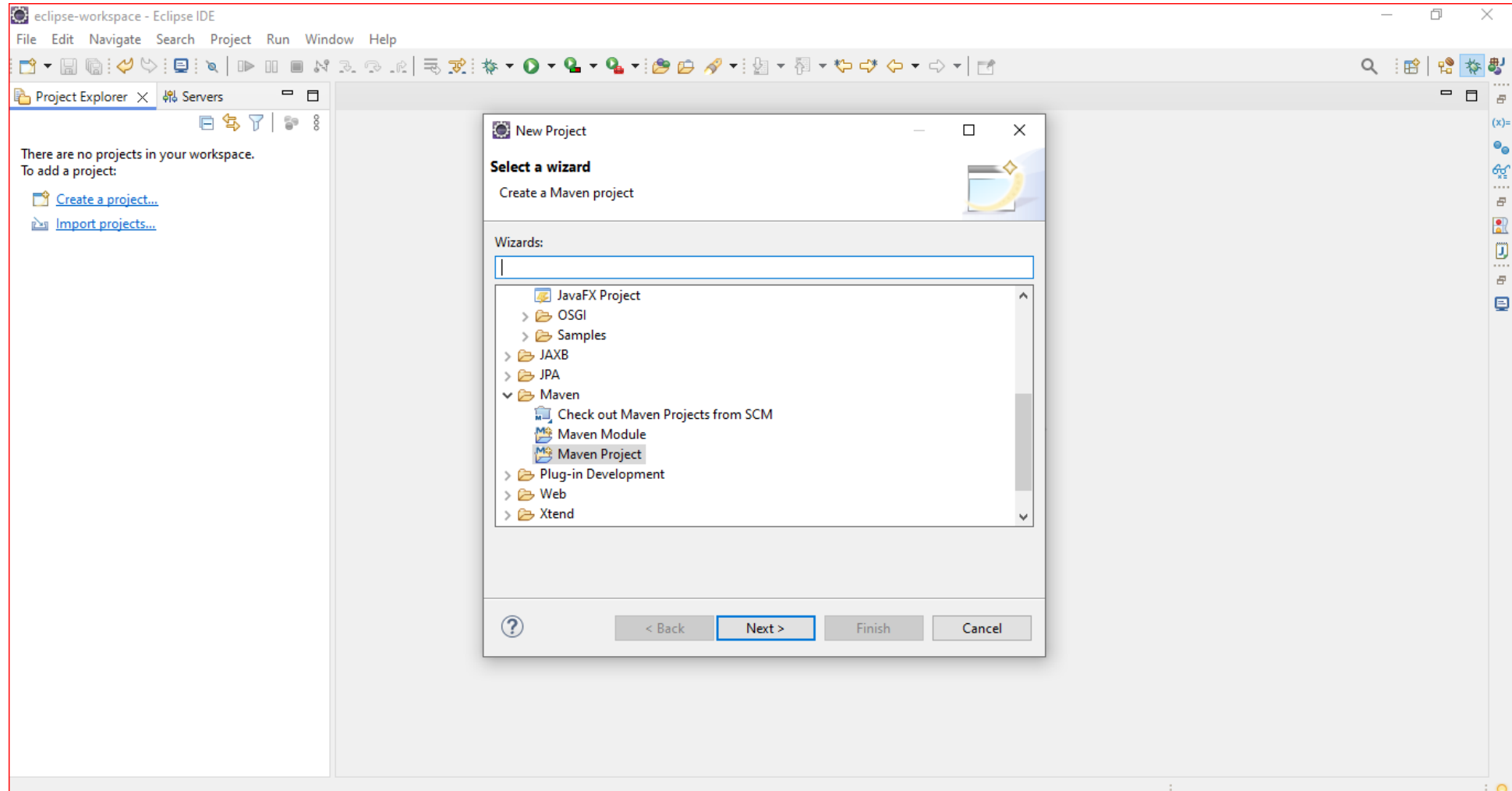
- ❖ The entity manager implements the API and encapsulates all of them within a single interface.
 - **Manages entity lifecycle:** Persist, find, merge, remove
 - **Controls persistence context:** Cache, flush, detach
 - **Executes queries:** JPQL, Criteria API
 - **Handles transactions:** Begin, commit,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

Key Operations of the Entity Manager

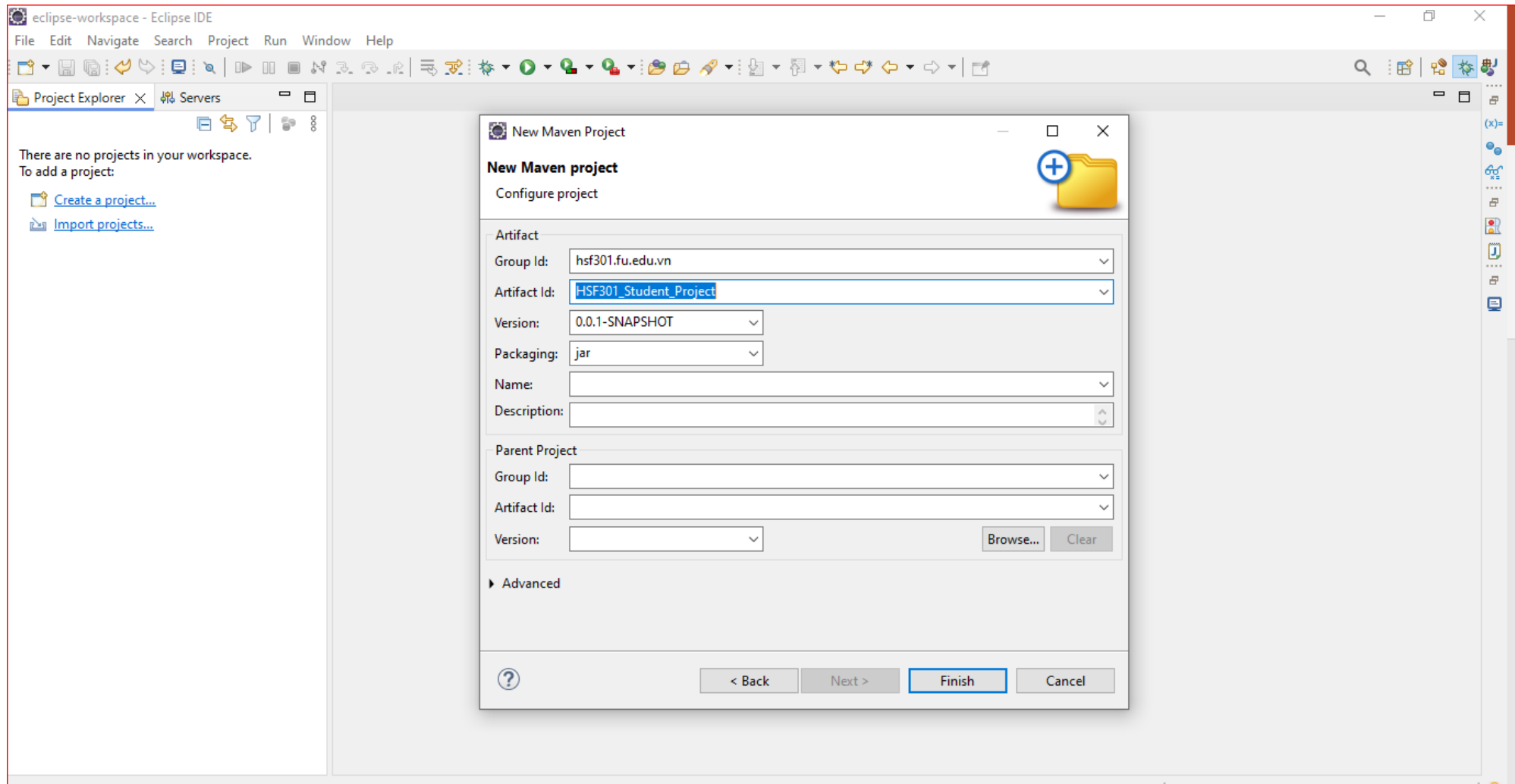
- ❖ **Persisting Entities:** *em.persist(entity)* makes a transient entity instance persistent
- ❖ **Finding Entities:** *em.find(entityClass, primaryKey)* retrieves an entity by its primary key.
- ❖ **Merging Entities:** *em.merge(entity)* merges a detached entity instance into the current persistence 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.

Demo Simple CRUD with JPA

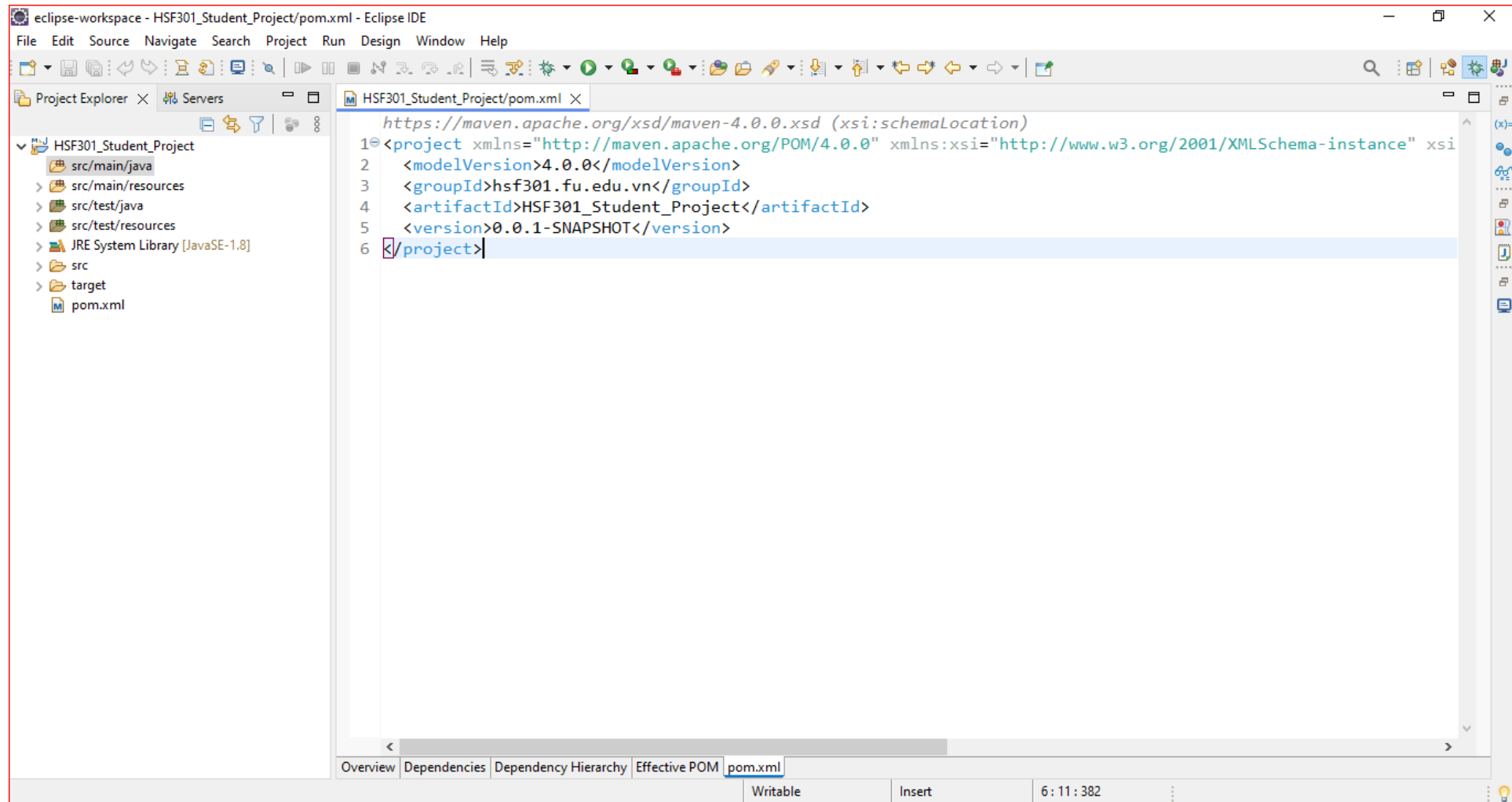
1. Open Eclipse, File | New | Maven Project



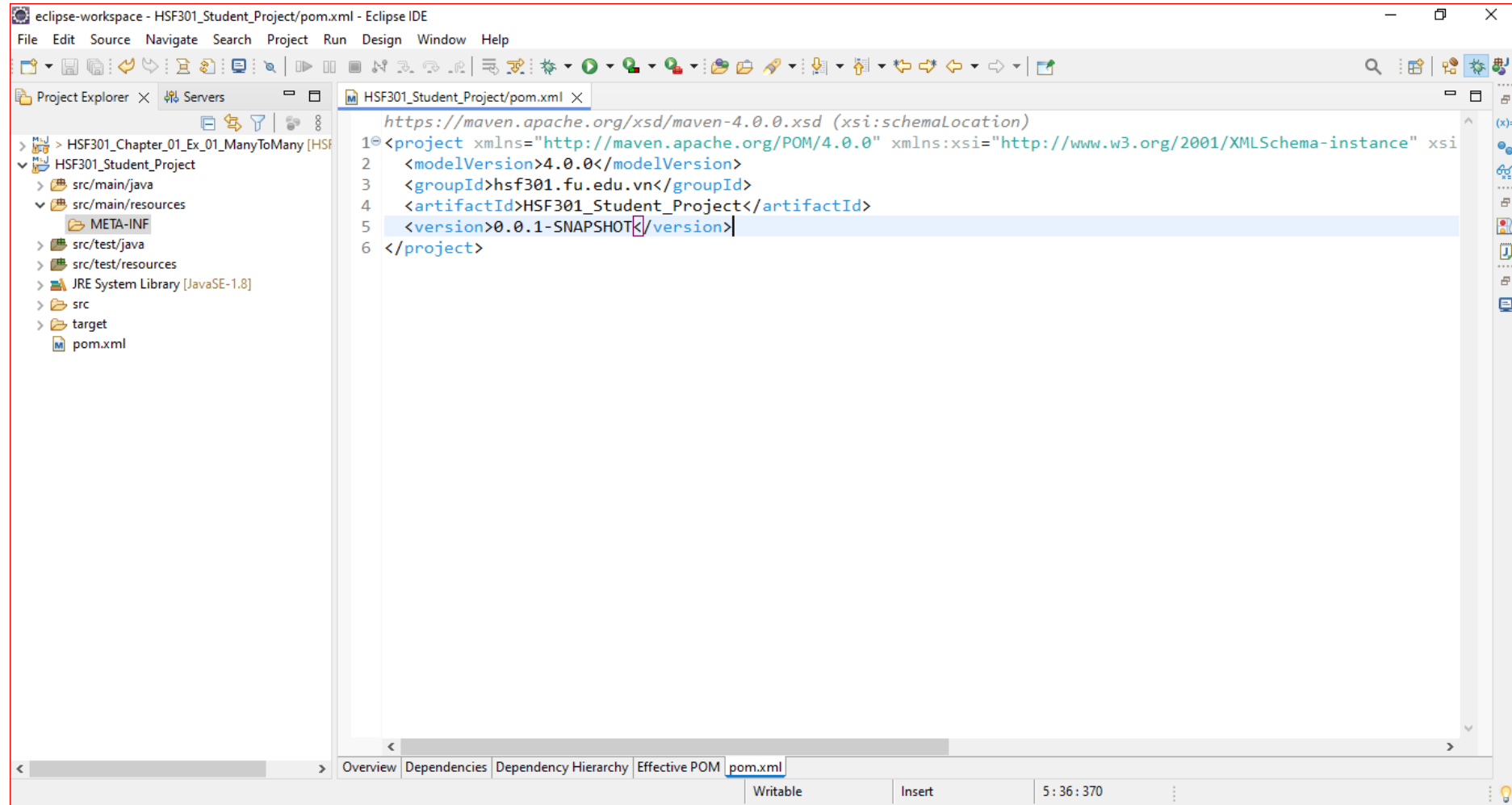
2. Check Create a simple project -> Browse Project -> Next



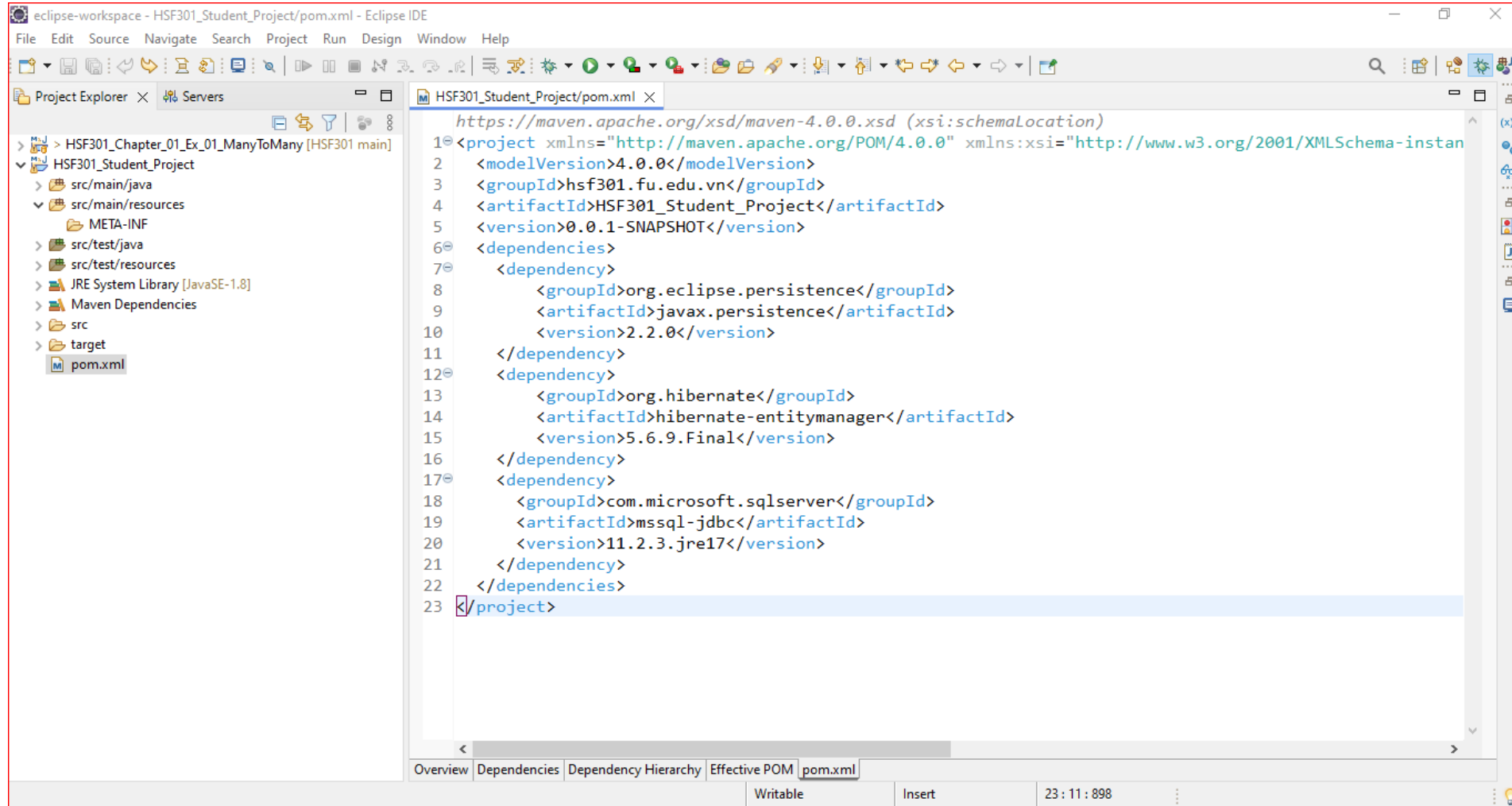
3. The New Project



4. Create folder META-INF in src/main/resources



5. Edit the pom.xml



The screenshot shows the Eclipse IDE interface with the 'HSF301_Student_Project/pom.xml' file open. The Project Explorer on the left shows the project structure, including 'src/main/java', 'src/main/resources', 'META-INF', 'src/test/java', 'src/test/resources', 'JRE System Library [JavaSE-1.8]', 'Maven Dependencies', 'src', 'target', and 'pom.xml'. The main editor displays the XML content of the pom.xml file, which is a Maven project configuration. The file is titled 'HSF301_Student_Project/pom.xml' and has a schema location of 'https://maven.apache.org/xsd/maven-4.0.0.xsd'. The XML content is as follows:

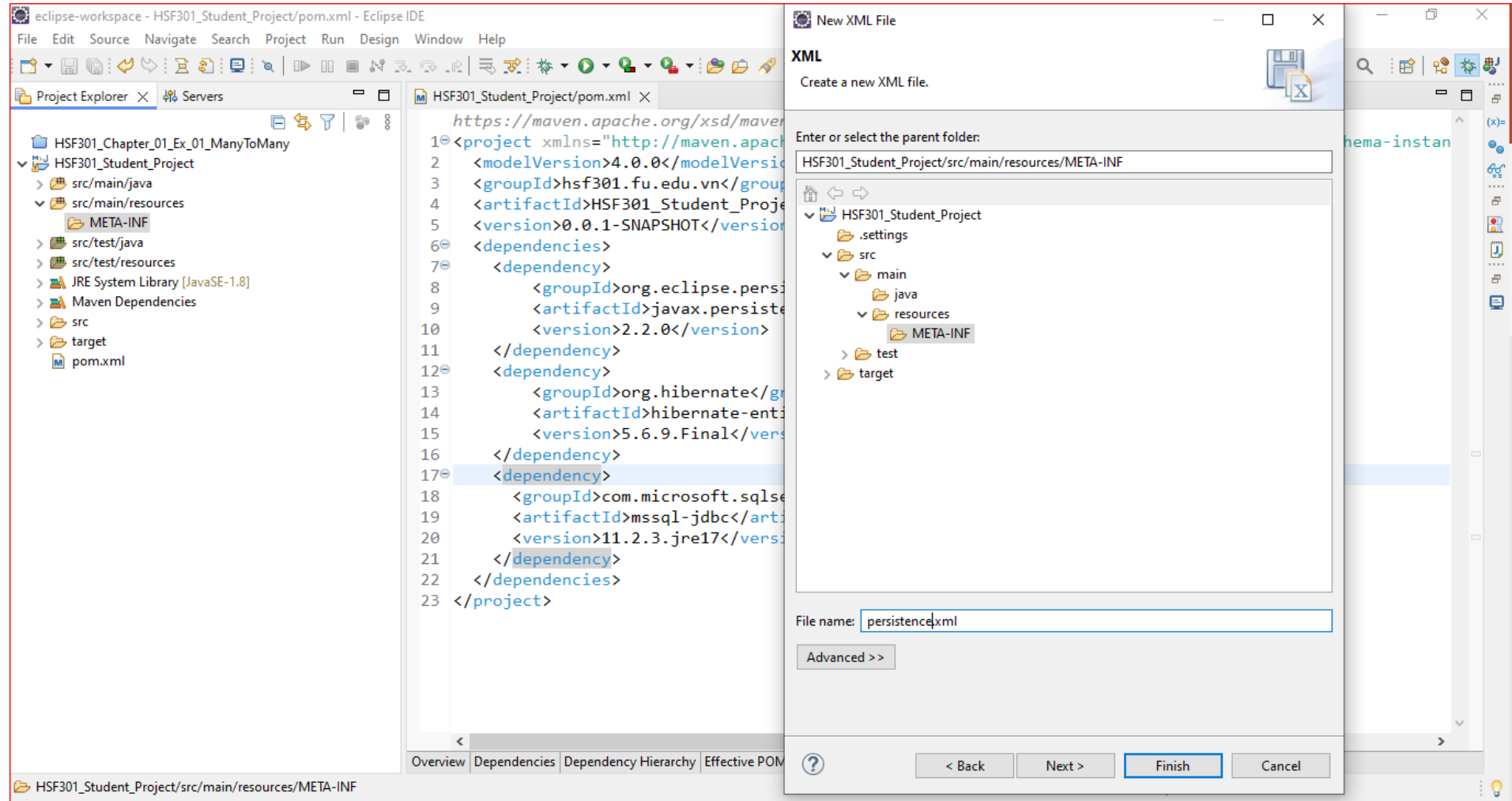
```
1<?xml version="1.0" encoding="UTF-8"?>
2<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4  <modelVersion>4.0.0</modelVersion>
5  <groupId>hsf301.fu.edu.vn</groupId>
6  <artifactId>HSF301_Student_Project</artifactId>
7  <version>0.0.1-SNAPSHOT</version>
8  <dependencies>
9    <dependency>
10     <groupId>org.eclipse.persistence</groupId>
11     <artifactId>javax.persistence</artifactId>
12     <version>2.2.0</version>
13   </dependency>
14   <dependency>
15     <groupId>org.hibernate</groupId>
16     <artifactId>hibernate-entitymanager</artifactId>
17     <version>5.6.9.Final</version>
18   </dependency>
19   <dependency>
20     <groupId>com.microsoft.sqlserver</groupId>
21     <artifactId>mssql-jdbc</artifactId>
22     <version>11.2.3.jre17</version>
23   </dependency>
24 </dependencies>
25 </project>
```

The bottom of the IDE shows the 'Overview' tab selected, with a status bar indicating 'Writable', 'Insert', and '23 : 11 : 898'.

6. Create persistence.xml in META-INF folder

The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer displays the project structure for 'HSF301_Student_Project', with the 'META-INF' folder selected under 'src/main/resources'. The central editor shows the 'pom.xml' file with Maven dependencies for 'org.eclipse.persistence' and 'org.hibernate'. On the right, the 'Select a wizard' dialog is open, showing a list of wizards. The 'XML File' wizard is selected under the 'XML' category. The 'Next >' button is highlighted at the bottom of the dialog.

7. Create persistence.xml in META-INF folder

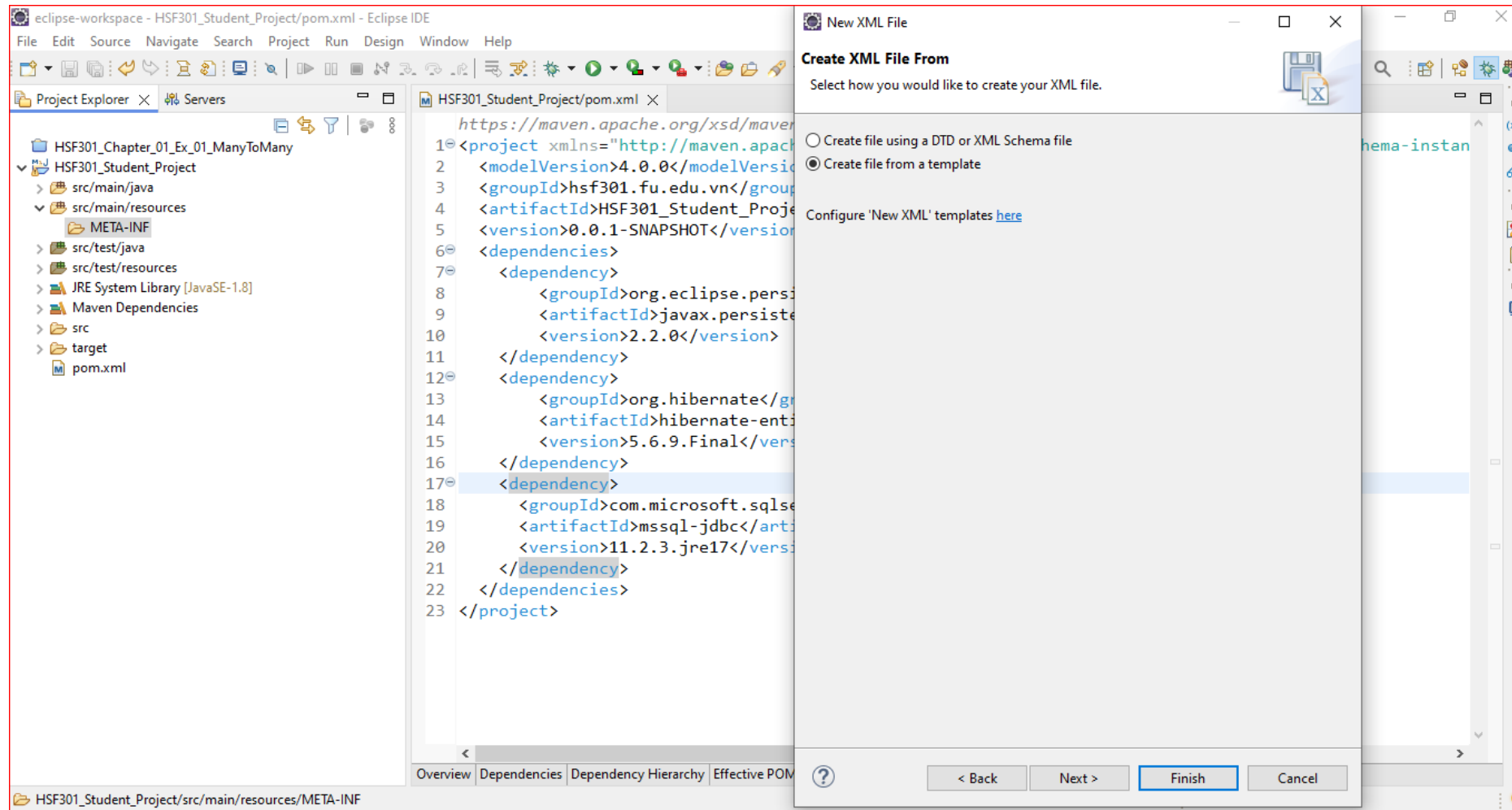


The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer displays the project structure: HSF301_Chapter_01_Ex_01_ManyToMany > HSF301_Student_Project > src/main/resources > META-INF. The main editor shows the pom.xml file with the following content:

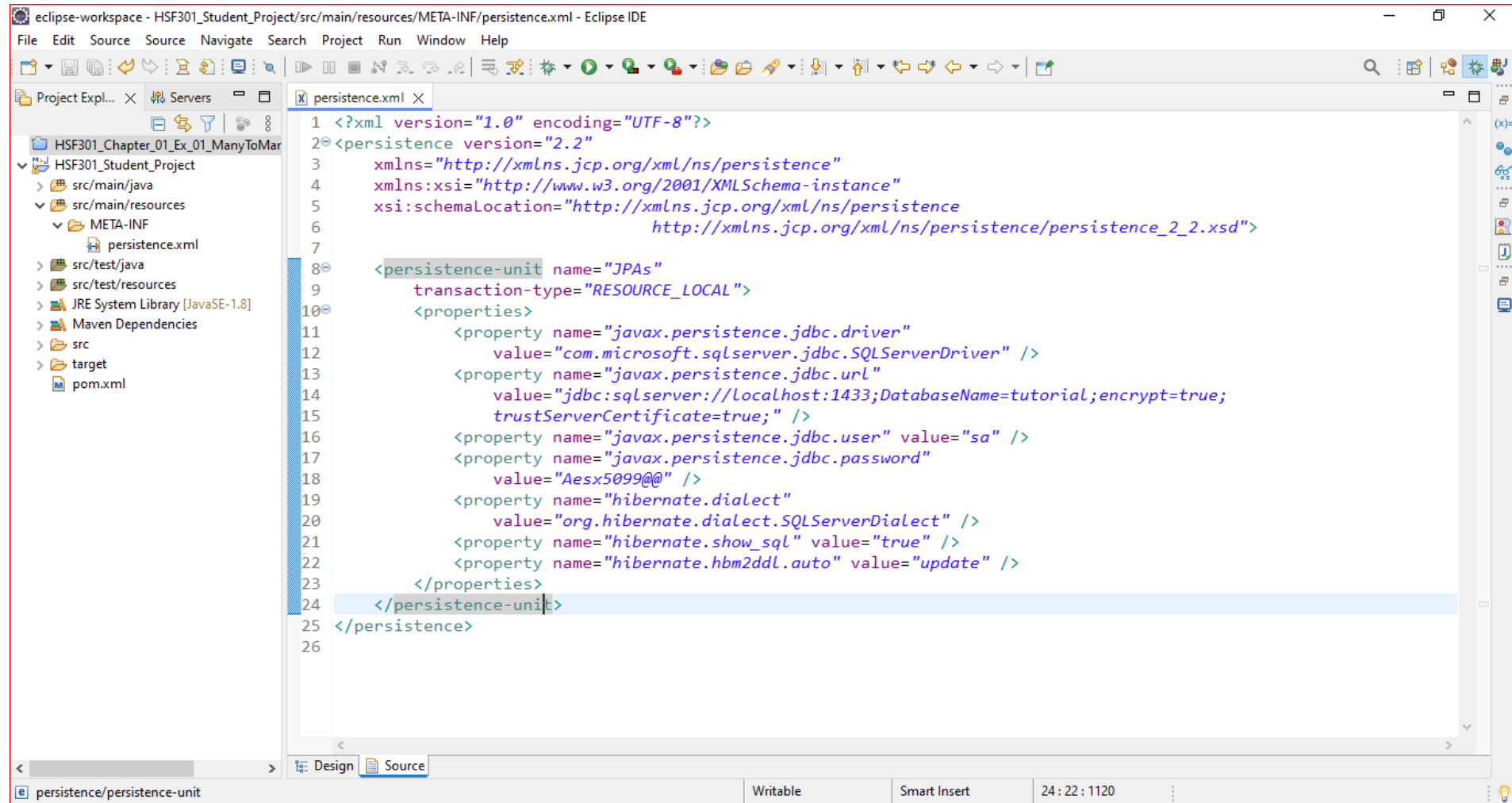
```
1<?xml version="1.0" encoding="UTF-8" ?>
2<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4  <groupId>hsf301.fu.edu.vn</groupId>
5  <artifactId>HSF301_Student_Project</artifactId>
6  <version>0.0.1-SNAPSHOT</version>
7  <dependencies>
8    <dependency>
9      <groupId>org.eclipse.persistence</groupId>
10     <artifactId>javax.persistence</artifactId>
11     <version>2.2.0</version>
12   </dependency>
13   <dependency>
14     <groupId>org.hibernate</groupId>
15     <artifactId>hibernate-entitymanager</artifactId>
16     <version>5.6.9.Final</version>
17   </dependency>
18   <dependency>
19     <groupId>com.microsoft.sqlserver</groupId>
20     <artifactId>mssql-jdbc</artifactId>
21     <version>11.2.3.jre17</version>
22   </dependency>
23 </dependencies>
24 </project>
```

On the right, the 'New XML File' dialog is open. The 'Enter or select the parent folder:' field is set to 'HSF301_Student_Project/src/main/resources/META-INF'. The file name is 'persistence.xml'. The 'Finish' button is highlighted.

8. Create persistence.xml in META-INF folder



9. Edit the persistence.xml in META-INF folder

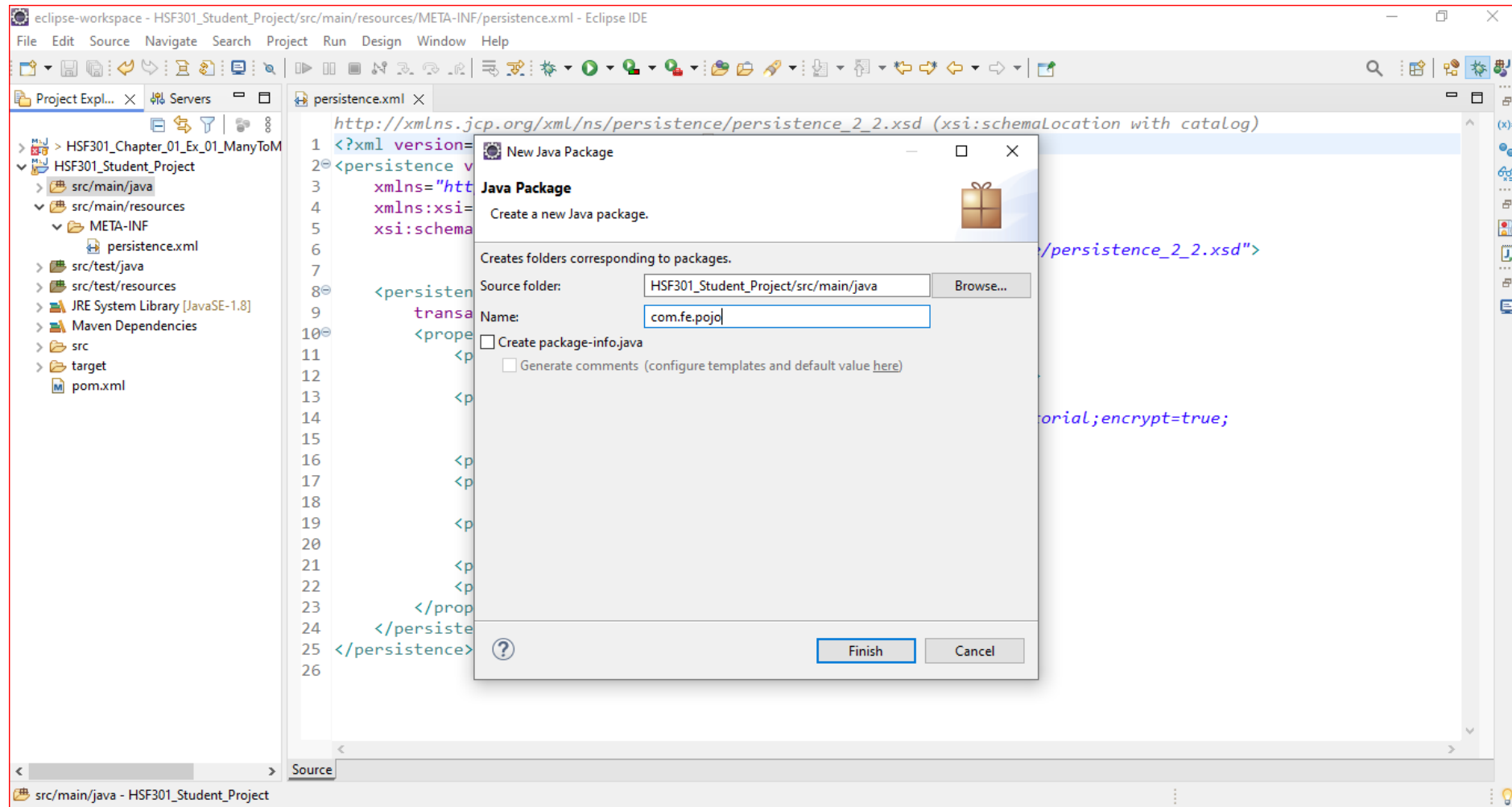


The screenshot shows the Eclipse IDE interface. The Project Explorer on the left displays the project structure: HSF301_Chapter_01_Ex_01_ManyToMar, HSF301_Student_Project, src/main/java, src/main/resources, META-INF, persistence.xml, src/test/java, src/test/resources, JRE System Library [JavaSE-1.8], Maven Dependencies, src, target, and pom.xml. The main editor window shows the persistence.xml file with the following XML content:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.2"
3   xmlns="http://xmlns.jcp.org/xml/ns/persistence"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
6     http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
7
8   <persistence-unit name="JPAs"
9     transaction-type="RESOURCE_LOCAL">
10     <properties>
11       <property name="javax.persistence.jdbc.driver"
12         value="com.microsoft.sqlserver.jdbc.SQLServerDriver" />
13       <property name="javax.persistence.jdbc.url"
14         value="jdbc:sqlserver://localhost:1433;DatabaseName=tutorial;encrypt=true;
15         trustServerCertificate=true;" />
16       <property name="javax.persistence.jdbc.user" value="sa" />
17       <property name="javax.persistence.jdbc.password"
18         value="Aesx5099@@@" />
19       <property name="hibernate.dialect"
20         value="org.hibernate.dialect.SQLServerDialect" />
21       <property name="hibernate.show_sql" value="true" />
22       <property name="hibernate.hbm2ddl.auto" value="update" />
23     </properties>
24   </persistence-unit>
25 </persistence>
26
```

The bottom status bar shows the Design and Source tabs, with the Source tab selected. The status bar also displays "Writable", "Smart Insert", and the time "24 : 22 : 1120".

10. Add com.fe.pojo Package in src/main/java



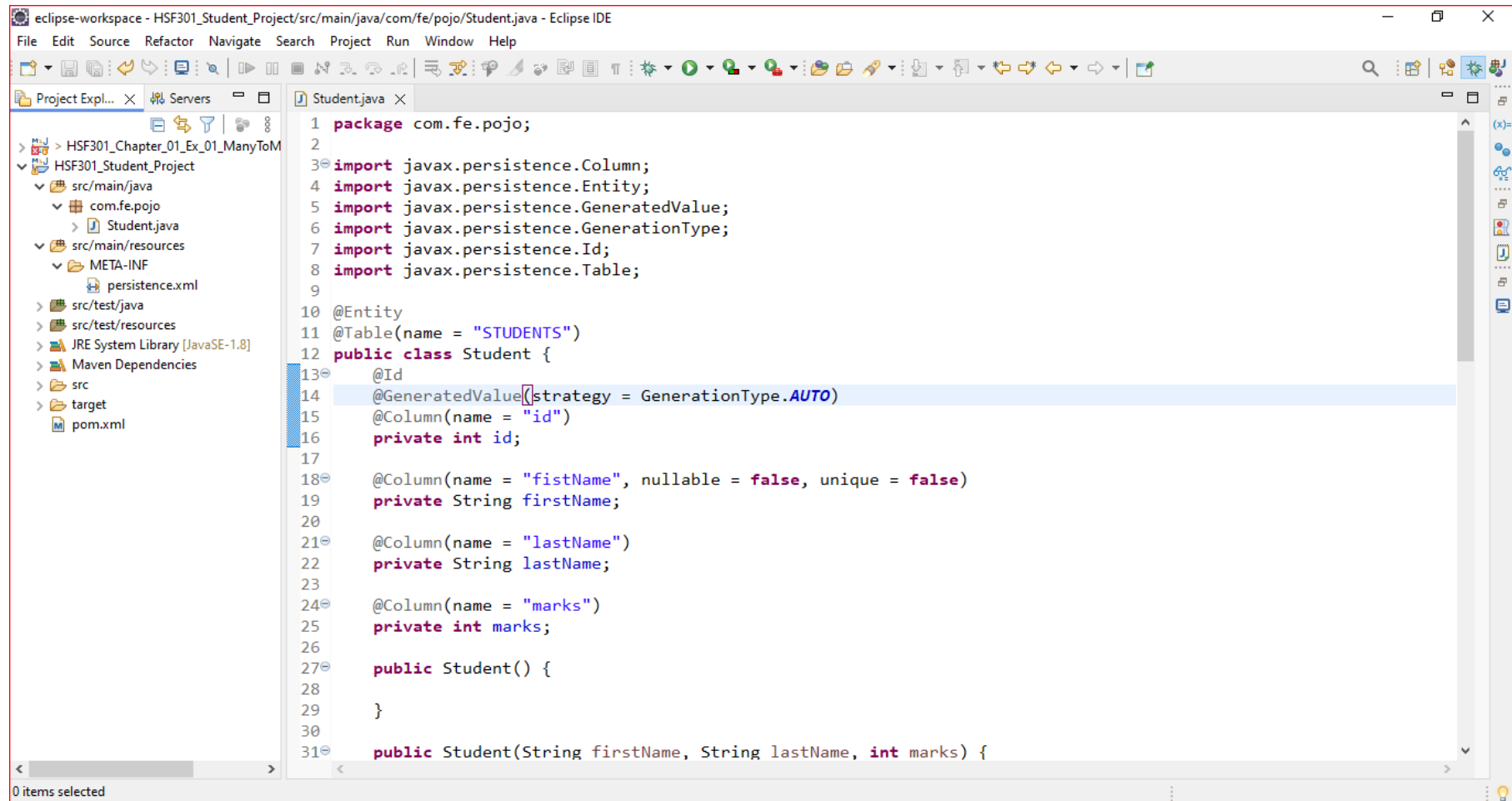
11. Add Student.java in com.fe.pojo package

The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer displays the project structure: HSF301_Student_Project, src/main/java, com.fe.pojo, src/main/resources, META-INF, persistence.xml, src/test/java, src/test/resources, JRE System Library [JavaSE-1.8], Maven Dependencies, src, target, and pom.xml. The central editor shows the 'persistence.xml' file with XML content. A 'New Java Class' dialog box is open in the foreground, allowing the creation of a new class. The dialog fields are as follows:

- Source folder:** HSF301_Student_Project/src/main/java
- Package:** com.fe.pojo
- Enclosing type:** (empty)
- Name:** Student
- Modifiers:** ☒ public, ☐ package, ☐ private, ☐ protected, ☐ abstract, ☐ final, ☐ static
- Superclass:** java.lang.Object
- Interfaces:** (empty)
- Which method stubs would you like to create?**
 - ☐ public static void main(String[] args)
 - ☐ Constructors from superclass
 - ☒ Inherited abstract methods
- Do you want to add comments? (Configure templates and default value [here](#))**
 - ☐ Generate comments

The 'Finish' button is highlighted in blue at the bottom right of the dialog.

12. Edit the Student.java

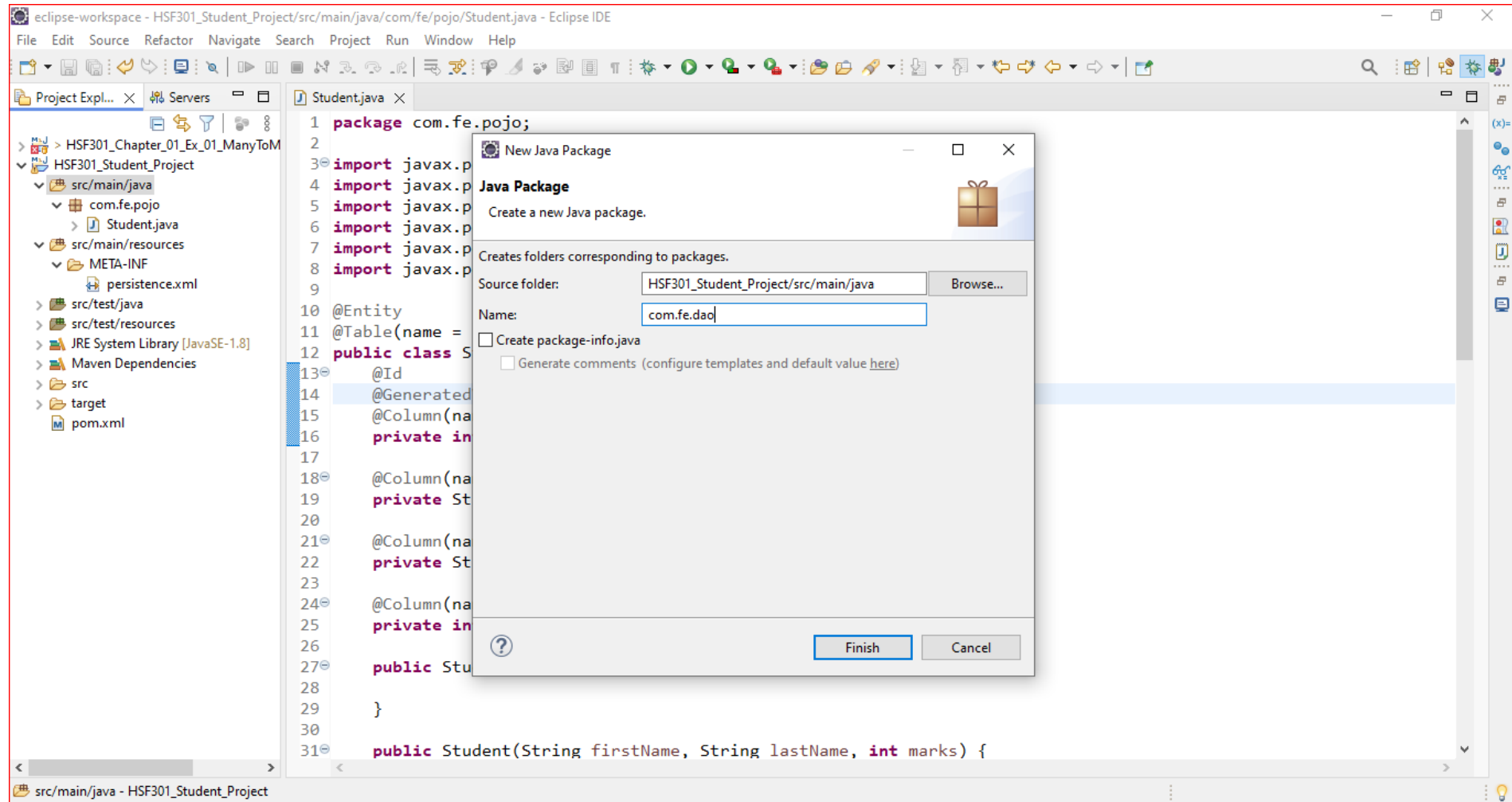


```
eclipse-workspace - HSF301_Student_Project/src/main/java/com/fe/pojo/Student.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

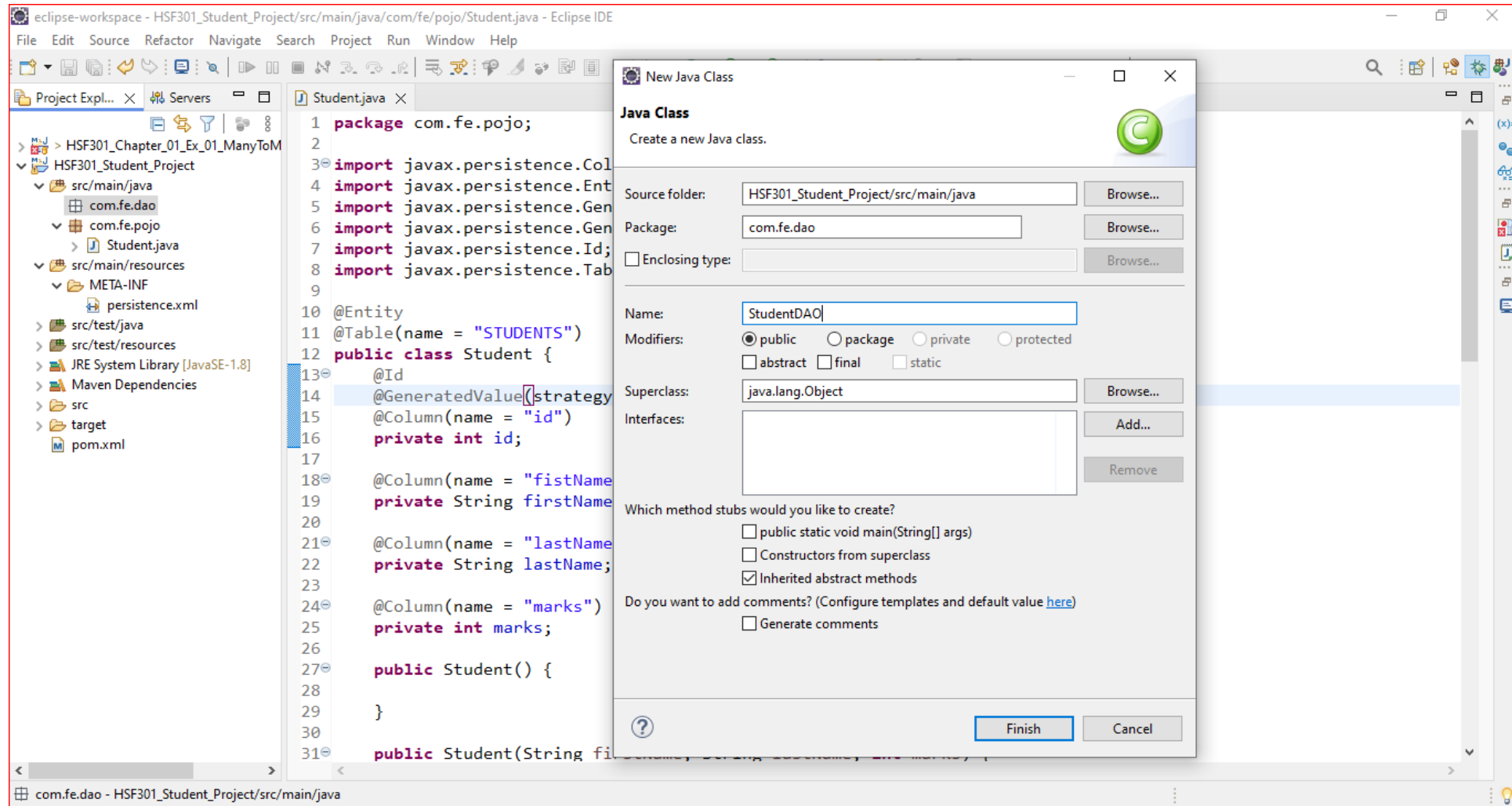
Project Expl... Servers Student.java
> HSF301_Chapter_01_Ex_01_ManyToM
  HSF301_Student_Project
    src/main/java
      com.fe.pojo
        Student.java
    src/main/resources
      META-INF
        persistence.xml
    src/test/java
    src/test/resources
    JRE System Library [JavaSE-1.8]
    Maven Dependencies
    src
    target
    pom.xml

1 package com.fe.pojo;
2
3 import javax.persistence.Column;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.GenerationType;
7 import javax.persistence.Id;
8 import javax.persistence.Table;
9
10 @Entity
11 @Table(name = "STUDENTS")
12 public class Student {
13     @Id
14     @GeneratedValue(strategy = GenerationType.AUTO)
15     @Column(name = "id")
16     private int id;
17
18     @Column(name = "firstName", nullable = false, unique = false)
19     private String firstName;
20
21     @Column(name = "lastName")
22     private String lastName;
23
24     @Column(name = "marks")
25     private int marks;
26
27     public Student() {
28
29     }
30
31     public Student(String firstName, String lastName, int marks) {
```

13. Add com.fe.dao Package in src/main/java



14. Create StudentDAO in com.fe.dao



The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer displays the project structure: HSF301_Chapter_01_Ex_01_ManyToM, HSF301_Student_Project, src/main/java, com.fe.dao, com.fe.pojo, Student.java, src/main/resources, META-INF, persistence.xml, src/test/java, src/test/resources, JRE System Library [JavaSE-1.8], Maven Dependencies, src, target, and pom.xml. The central editor shows the 'Student.java' file with the following code:

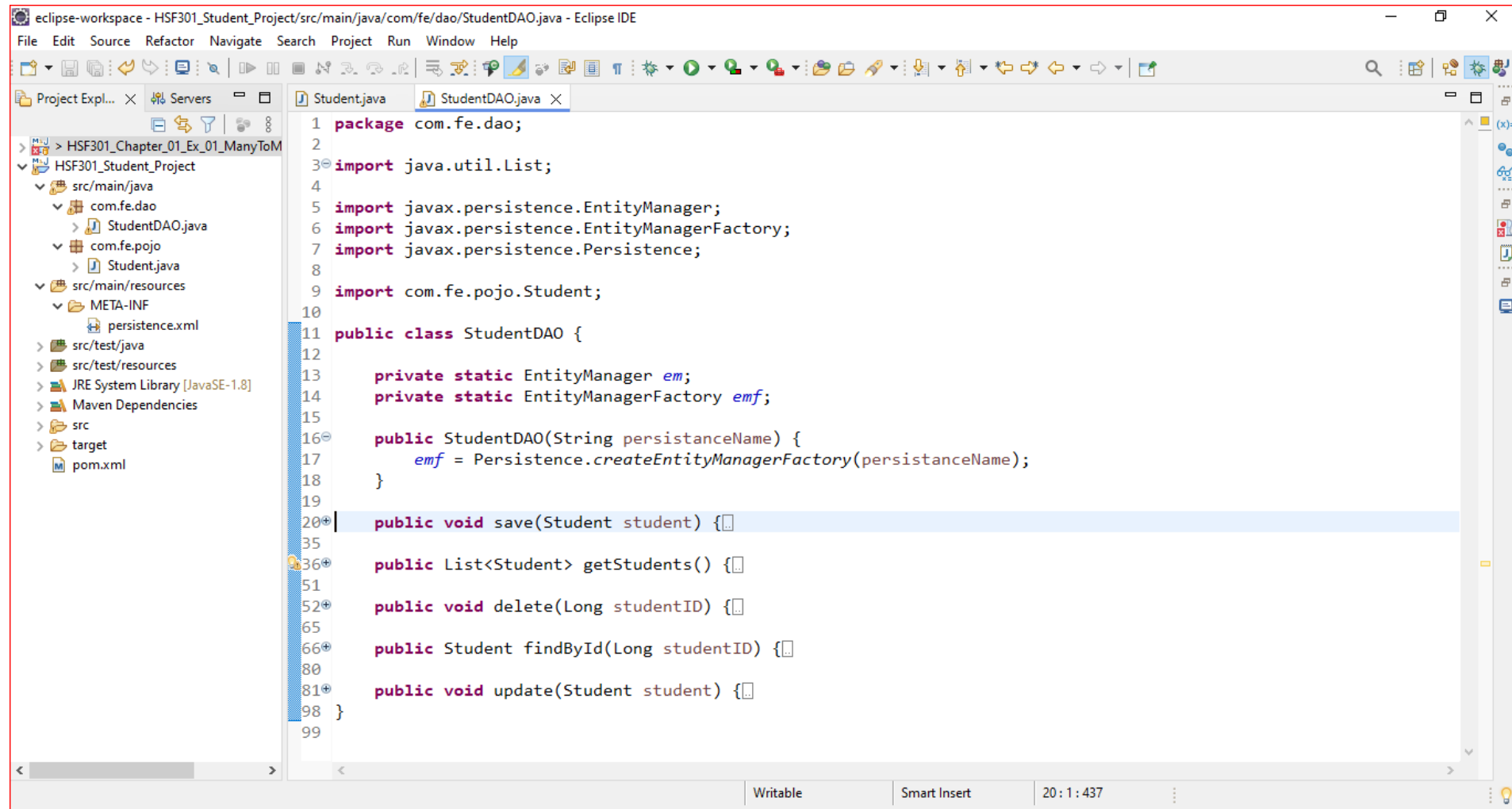
```
1 package com.fe.pojo;
2
3 import javax.persistence.Column;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.GenerationType;
7 import javax.persistence.Id;
8 import javax.persistence.Table;
9
10 @Entity
11 @Table(name = "STUDENTS")
12 public class Student {
13     @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     @Column(name = "id")
16     private int id;
17
18     @Column(name = "firstName")
19     private String firstName;
20
21     @Column(name = "lastName")
22     private String lastName;
23
24     @Column(name = "marks")
25     private int marks;
26
27     public Student() {
28
29     }
30
31     public Student(String firstName, String lastName, int marks) {
```

On the right, the 'New Java Class' dialog is open. It shows the following configuration:

- Source folder: HSF301_Student_Project/src/main/java
- Package: com.fe.dao
- Enclosing type: (empty)
- Name: StudentDAO
- Modifiers: ☒ public, ☐ package, ☐ private, ☐ protected, ☐ abstract, ☐ final, ☐ static
- Superclass: java.lang.Object
- Interfaces: (empty)
- Which method stubs would you like to create?: ☐ public static void main(String[] args), ☐ Constructors from superclass, ☒ Inherited abstract methods
- Do you want to add comments? (Configure templates and default value [here](#)): ☐ Generate comments

The 'Finish' button is highlighted.

15. Edit the StudentDAO in com.fe.dao

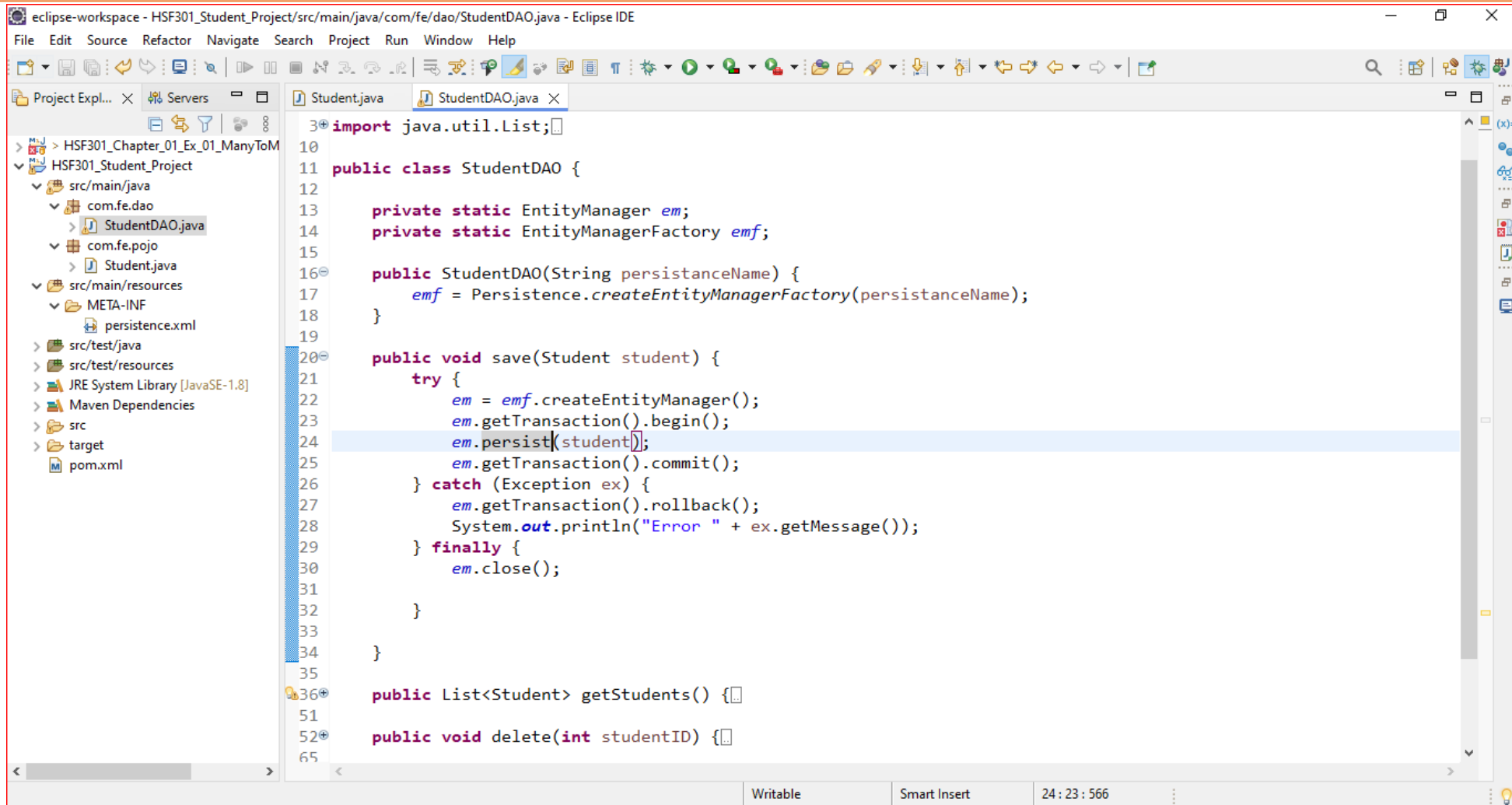


```
eclipse-workspace - HSF301_Student_Project/src/main/java/com/fe/dao/StudentDAO.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

Project Expl... Servers
> HSF301_Chapter_01_Ex_01_ManyToM
  HSF301_Student_Project
    src/main/java
      com.fe.dao
        StudentDAO.java
      com.fe.pojo
        Student.java
    src/main/resources
      META-INF
        persistence.xml
    src/test/java
    src/test/resources
    JRE System Library [JavaSE-1.8]
    Maven Dependencies
    src
    target
    pom.xml

1 package com.fe.dao;
2
3 import java.util.List;
4
5 import javax.persistence.EntityManager;
6 import javax.persistence.EntityManagerFactory;
7 import javax.persistence.Persistence;
8
9 import com.fe.pojo.Student;
10
11 public class StudentDAO {
12
13     private static EntityManager em;
14     private static EntityManagerFactory emf;
15
16     public StudentDAO(String persistenceName) {
17         emf = Persistence.createEntityManagerFactory(persistenceName);
18     }
19
20     public void save(Student student) {}
21
22     public List<Student> getStudents() {}
23
24     public void delete(Long studentID) {}
25
26     public Student findById(Long studentID) {}
27
28     public void update(Student student) {}
29 }
```

16. Save Student Method



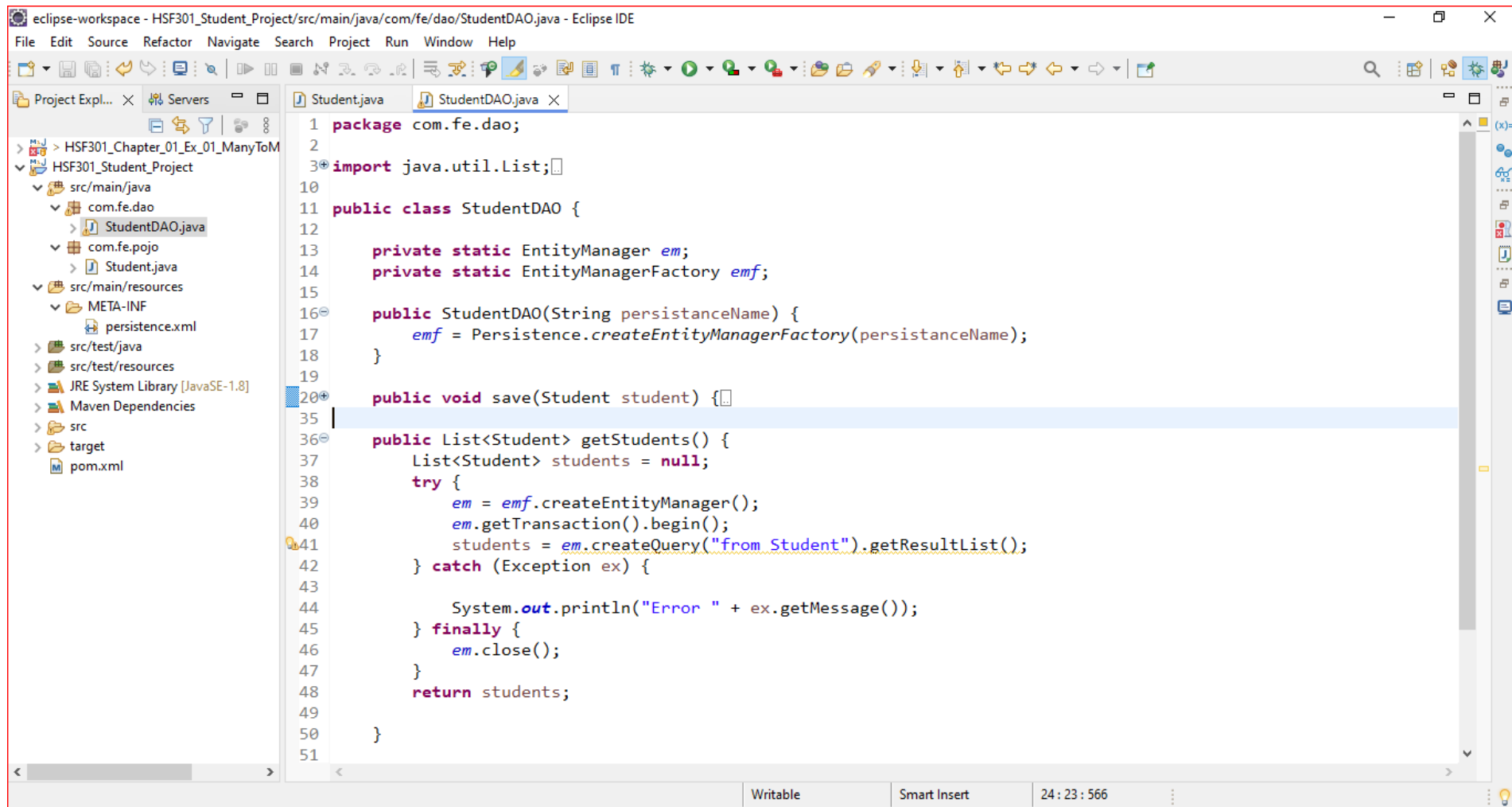
```
eclipse-workspace - HSF301_Student_Project/src/main/java/com/fe/dao/StudentDAO.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

Project Expl... Servers
HSF301_Chapter_01_Ex_01_ManyToM
HSF301_Student_Project
  src/main/java
    com.fe.dao
      StudentDAO.java
    com.fe.pojo
      Student.java
  src/main/resources
    META-INF
      persistence.xml
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  pom.xml

3* import java.util.List;
10
11 public class StudentDAO {
12
13     private static EntityManager em;
14     private static EntityManagerFactory emf;
15
16     public StudentDAO(String persistenceName) {
17         emf = Persistence.createEntityManagerFactory(persistenceName);
18     }
19
20     public void save(Student student) {
21         try {
22             em = emf.createEntityManager();
23             em.getTransaction().begin();
24             em.persist(student);
25             em.getTransaction().commit();
26         } catch (Exception ex) {
27             em.getTransaction().rollback();
28             System.out.println("Error " + ex.getMessage());
29         } finally {
30             em.close();
31         }
32     }
33
34 }
35
36 public List<Student> getStudents() {}
51
52 public void delete(int studentID) {}
65

Writable Smart Insert 24 : 23 : 566
```

17. Get All Students Method



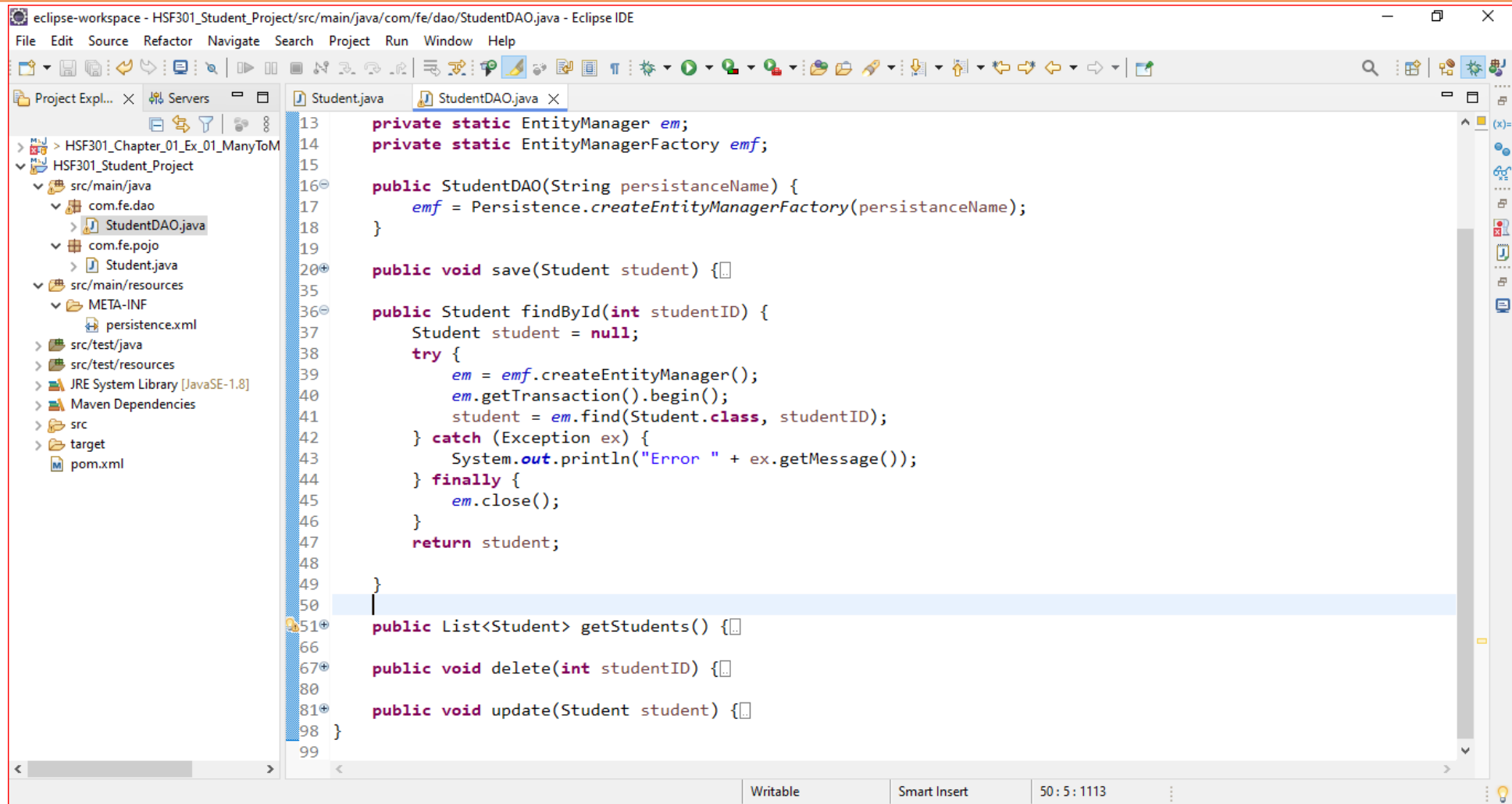
The screenshot shows the Eclipse IDE with the file `StudentDAO.java` open. The code implements the `getStudents` method, which uses JPA to retrieve all students from the database. The method is highlighted with a blue background.

```

1 package com.fe.dao;
2
3 import java.util.List;
4
10
11 public class StudentDAO {
12
13     private static EntityManager em;
14     private static EntityManagerFactory emf;
15
16     public StudentDAO(String persistenceName) {
17         emf = Persistence.createEntityManagerFactory(persistenceName);
18     }
19
20     public void save(Student student) {}
21
35
36     public List<Student> getStudents() {
37         List<Student> students = null;
38         try {
39             em = emf.createEntityManager();
40             em.getTransaction().begin();
41             students = em.createQuery("from Student").getResultList();
42         } catch (Exception ex) {
43
44             System.out.println("Error " + ex.getMessage());
45         } finally {
46             em.close();
47         }
48         return students;
49     }
50
51

```

18. Find a Student Method



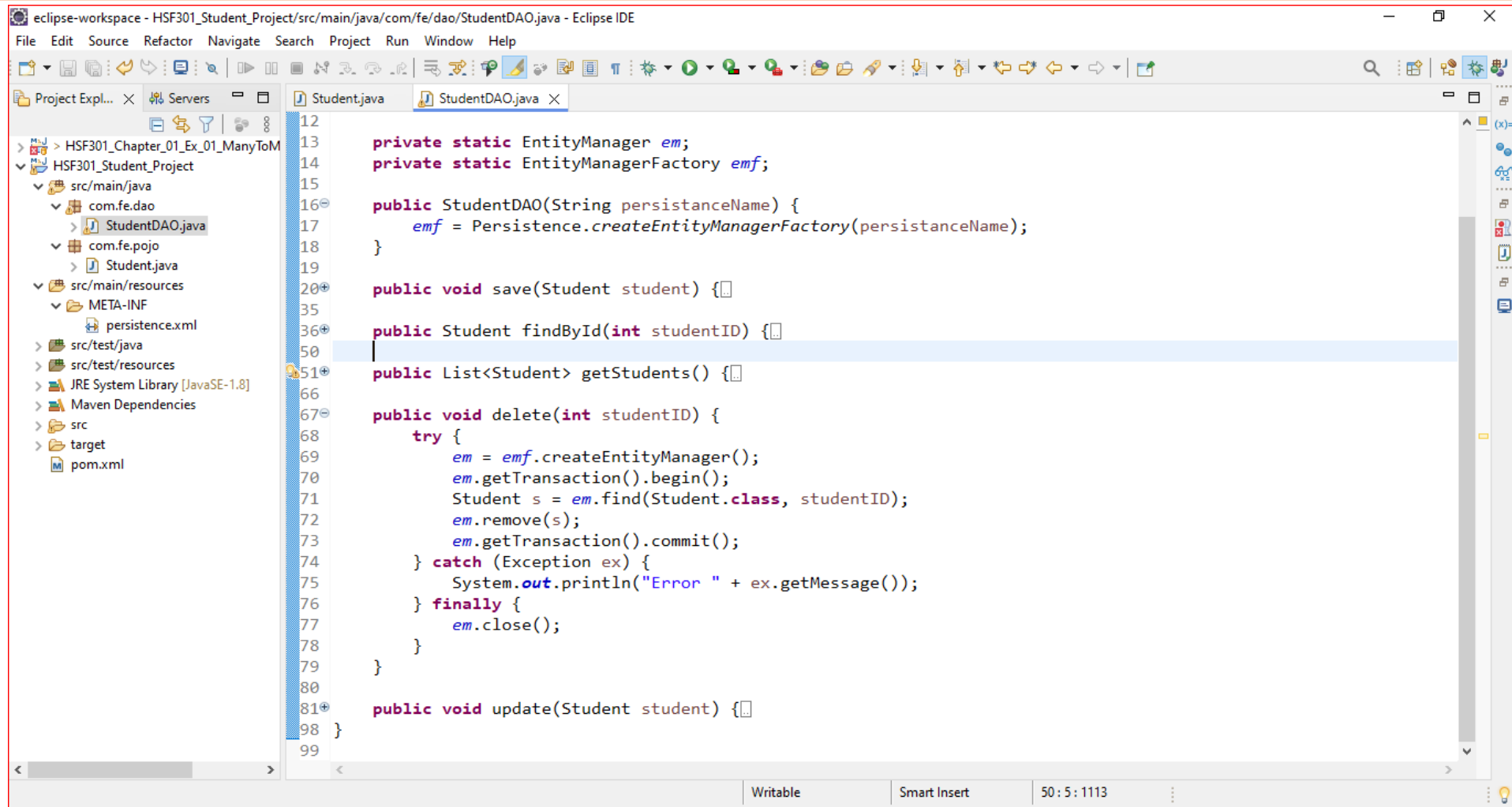
```
eclipse-workspace - HSF301_Student_Project/src/main/java/com/fe/dao/StudentDAO.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

Project Expl... Servers Student.java StudentDAO.java X
HSF301_Chapter_01_Ex_01_ManyToM
HSF301_Student_Project
  src/main/java
    com.fe.dao
      StudentDAO.java
    com.fe.pojo
      Student.java
  src/main/resources
  META-INF
    persistence.xml
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  pom.xml

13 private static EntityManager em;
14 private static EntityManagerFactory emf;
15
16 public StudentDAO(String persistenceName) {
17     emf = Persistence.createEntityManagerFactory(persistenceName);
18 }
19
20 public void save(Student student) {}
21
35
36 public Student findById(int studentID) {
37     Student student = null;
38     try {
39         em = emf.createEntityManager();
40         em.getTransaction().begin();
41         student = em.find(Student.class, studentID);
42     } catch (Exception ex) {
43         System.out.println("Error " + ex.getMessage());
44     } finally {
45         em.close();
46     }
47     return student;
48 }
49
50
51 public List<Student> getStudents() {}
52
66
67 public void delete(int studentID) {}
68
80
81 public void update(Student student) {}
82
98 }
99

Writable Smart Insert 50:5:1113
```

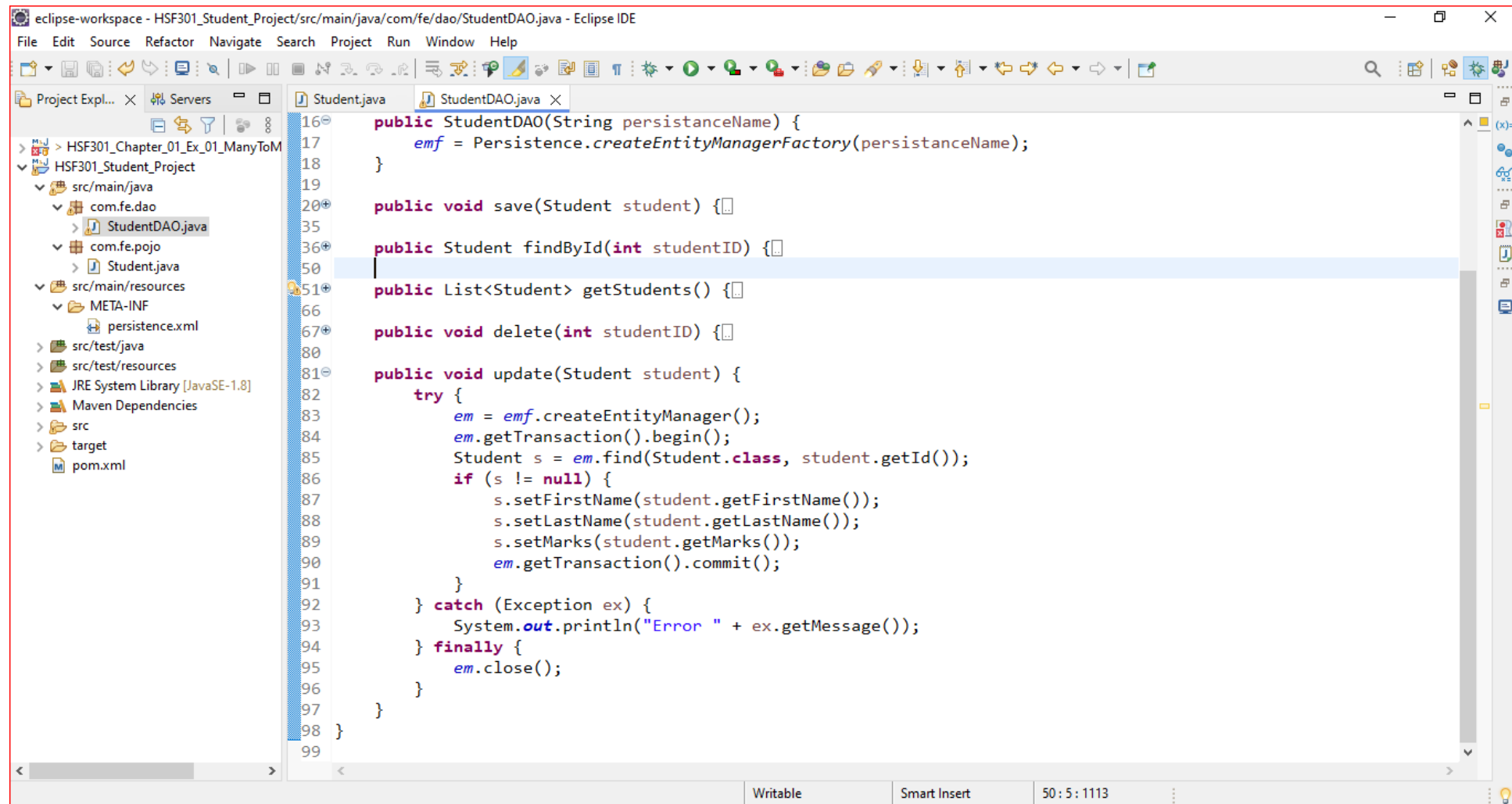
19. Delete Student Method



The screenshot shows the Eclipse IDE with the file `StudentDAO.java` open. The code defines a `StudentDAO` class with several methods. The `delete` method is highlighted, showing its implementation using `EntityManager` and `EntityManagerFactory` to remove a student from the database.

```
12 private static EntityManager em;  
13 private static EntityManagerFactory emf;  
14  
15 public StudentDAO(String persistenceName) {  
16     emf = Persistence.createEntityManagerFactory(persistenceName);  
17 }  
18  
19 public void save(Student student) {}  
20  
21 public Student findById(int studentID) {}  
22  
23 public List<Student> getStudents() {}  
24  
25 public void delete(int studentID) {  
26     try {  
27         em = emf.createEntityManager();  
28         em.getTransaction().begin();  
29         Student s = em.find(Student.class, studentID);  
30         em.remove(s);  
31         em.getTransaction().commit();  
32     } catch (Exception ex) {  
33         System.out.println("Error " + ex.getMessage());  
34     } finally {  
35         em.close();  
36     }  
37 }  
38  
39 public void update(Student student) {}  
40 }  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99
```

20. Update Student Method

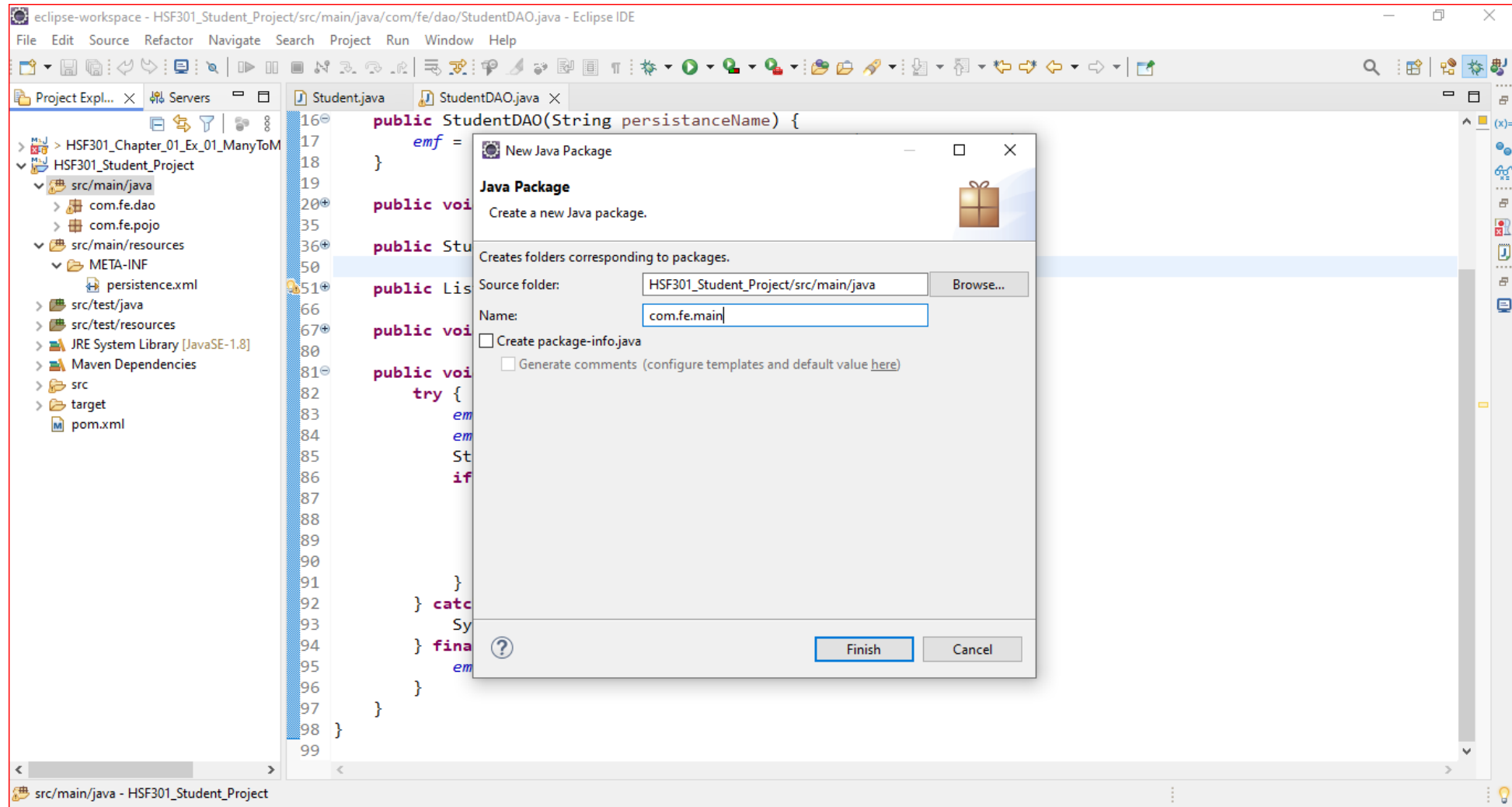


```
eclipse-workspace - HSF301_Student_Project/src/main/java/com/fe/dao/StudentDAO.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

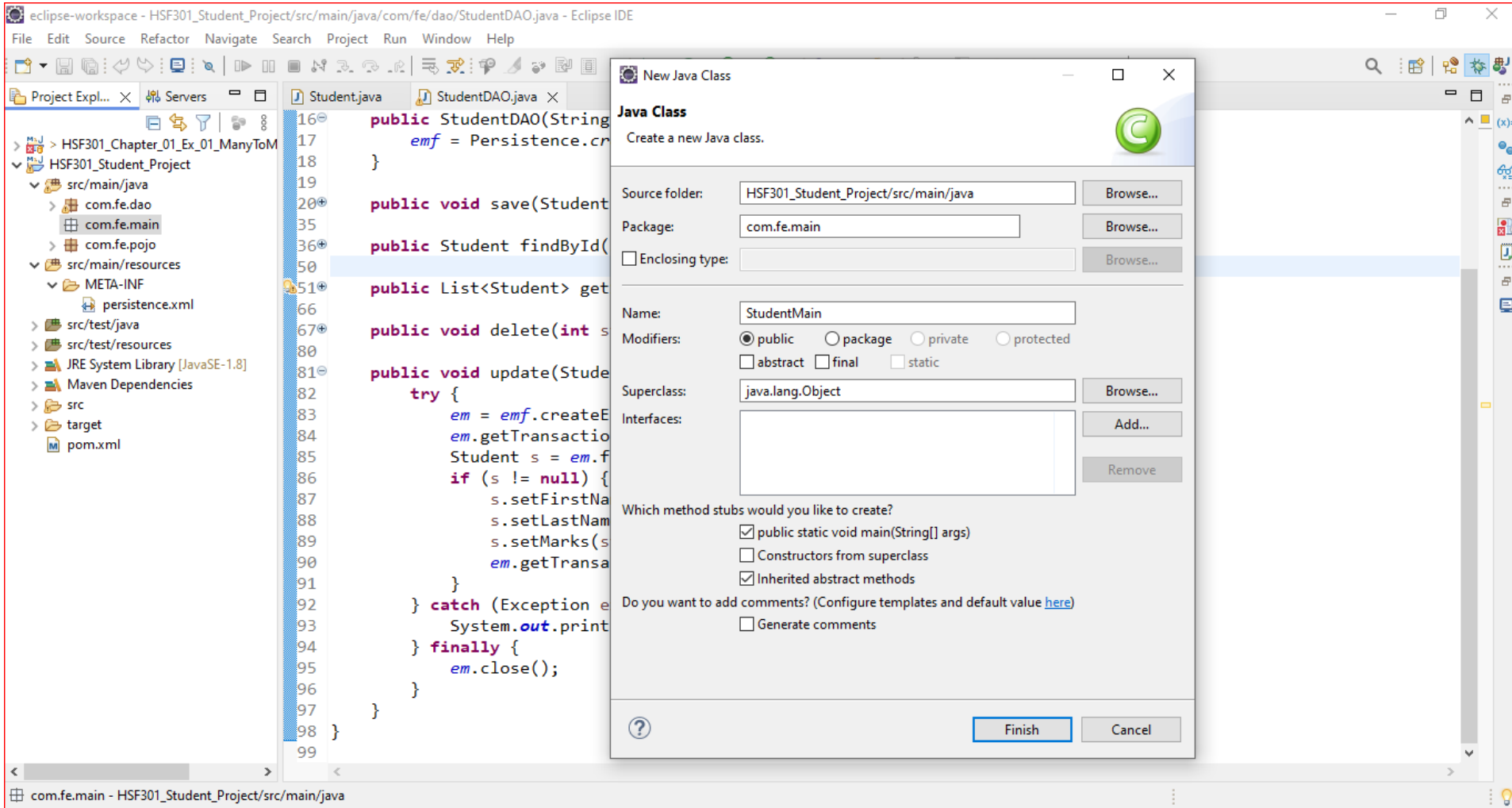
Project Explorer:
HSF301_Chapter_01_Ex_01_ManyToM
HSF301_Student_Project
  src/main/java
    com.fe.dao
      StudentDAO.java
    com.fe.pojo
      Student.java
  src/main/resources
    META-INF
      persistence.xml
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  pom.xml

StudentDAO.java:
16 public StudentDAO(String persistenceName) {
17     emf = Persistence.createEntityManagerFactory(persistenceName);
18 }
19
20 public void save(Student student) {}
21
22 public Student findById(int studentID) {}
23
24 public List<Student> getStudents() {}
25
26 public void delete(int studentID) {}
27
28 public void update(Student student) {
29     try {
30         em = emf.createEntityManager();
31         em.getTransaction().begin();
32         Student s = em.find(Student.class, student.getId());
33         if (s != null) {
34             s.setFirstName(student.getFirstName());
35             s.setLastName(student.getLastName());
36             s.setMarks(student.getMarks());
37             em.getTransaction().commit();
38         }
39     } catch (Exception ex) {
40         System.out.println("Error " + ex.getMessage());
41     } finally {
42         em.close();
43     }
44 }
45 }
```

21. Add com.fe.main Package in src/main/java



22. Create StudentMain class in com.fe.main Package



23. Edit the StudentMain in com.fe.main

The screenshot shows the Eclipse IDE interface. The left sidebar displays the Project Explorer with the following structure:

- HSF301_Chapter_01_Ex_01_ManyToM
 - HSF301_Student_Project
 - src/main/java
 - com.fe.dao
 - com.fe.main
 - StudentMain.java (selected)
 - com.fe.pojo
 - src/main/resources
 - META-INF
 - persistence.xml
 - src/test/java
 - src/test/resources
 - JRE System Library [JavaSE-1.8]
 - Maven Dependencies
 - src
 - target
 - pom.xml

The main editor shows the code for `StudentMain.java`:

```
1 package com.fe.main;
2
3 public class StudentMain {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         System.out.println("++++++MENU++++++");
8         System.out.println("+ 1. Get Students  +");
9         System.out.println("+ 2. Add Student  +");
10        System.out.println("+ 3. Delete Student +");
11        System.out.println("+ 4. Update Student +");
12        System.out.println("+ 5. Delete Student +");
13        System.out.println("+ 6. Get a Student +");
14        System.out.println("+ 0. QUIT      +");
15        System.out.println("++++++END++++++");
16    }
17 }
18
19
```

The right sidebar shows the Console window with the following output:

```
<terminated> StudentMain [Java Application] D:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full
++++++MENU++++++
+ 1. Get Students  +
+ 2. Add Student  +
+ 3. Delete Student +
+ 4. Update Student +
+ 5. Delete Student +
+ 6. Get a Student +
+ 0. QUIT      +
++++++END++++++
```

24. Edit the StudentMain in com.fe.main

The screenshot shows the Eclipse IDE with the following components:

- Project Explorer:** Shows the project structure for HSF301_Student_Project. The package `com.fe.main` is expanded, showing `StudentMain.java`.
- Editor:** Displays the code for `StudentMain.java`. The code includes a menu, a scanner for user input, and a switch statement to handle different actions (Add, Delete, Update, Get, Quit).
- Console:** Shows the output of the application. It displays the menu, the prompt "Please enter a number!", and various Hibernate logs indicating the application is running successfully.

```

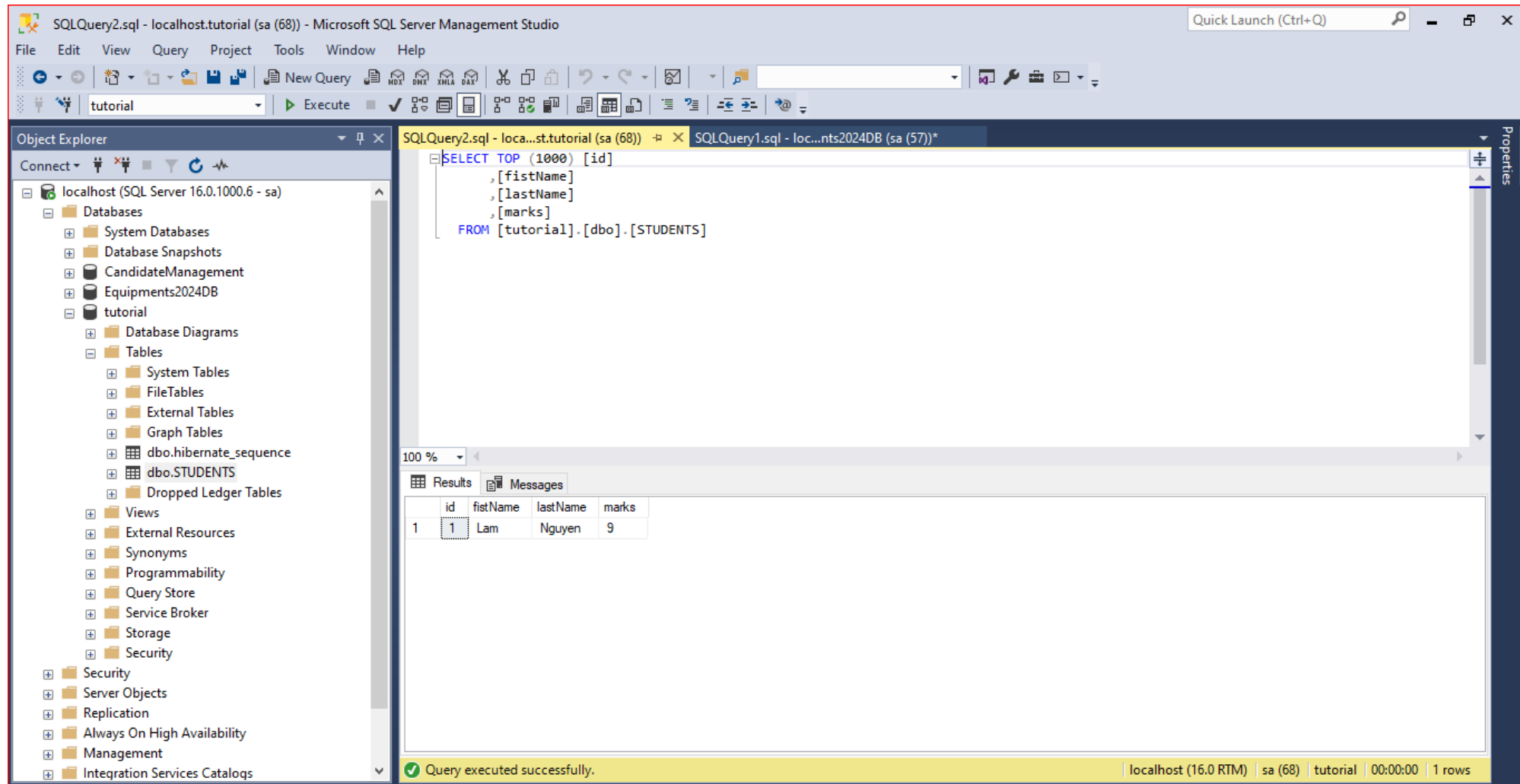
19 System.out.println("+ 0. QUIT +");
20 System.out.println("++++++END++++++");
21
22 int inputKey = -1;
23 while (inputKey != 0) {
24     Scanner console = new Scanner(System.in);
25     System.out.println("Please enter a number !");
26     inputKey = console.nextInt();
27     StudentDAO studentDAO = new StudentDAO("JPAs");
28     Student student = new Student("Lam", "Nguyen", 9);
29     switch (inputKey) {
30         case 0:
31             break;
32         case 1:
33             studentDAO.save(student);
34             break;
35         case 2:
36             studentDAO.delete(1);
37             break;
38         case 3:
39             student = new Student(1, "Sang", "Nguyen", 9);
40             studentDAO.update(student);
41         case 4:
42             studentDAO.findById(1);
43             break;
44         default:
45             System.out.println("Please choice menu !");
46     }
47 }
48
49 }
    
```

Console Output:

```

StudentMain [Java Application] [pid: 17148]
++++++MENU++++++
+ 1. Add Student +
+ 2. Delete Student +
+ 3. Update Student +
+ 4. Get a Student +
+ 0. QUIT +
++++++END++++++
Please enter a number !
1
Apr 14, 2024 10:26:51 PM org.hibernate.jpahi
INFO: HHH000204: Processing PersistenceUnitIn
Apr 14, 2024 10:26:52 PM org.hibernate.Versio
INFO: HHH000412: Hibernate ORM core version 5
Apr 14, 2024 10:26:52 PM org.hibernate.annota
INFO: HCANN000001: Hibernate Commons Annotati
Apr 14, 2024 10:26:52 PM org.hibernate.engine
WARN: HHH10001002: Using Hibernate built-in c
Apr 14, 2024 10:26:53 PM org.hibernate.engine
INFO: HHH10001005: using driver [com.microsof
Apr 14, 2024 10:26:53 PM org.hibernate.engine
INFO: HHH10001001: Connection properties: {pa
Apr 14, 2024 10:26:53 PM org.hibernate.engine
INFO: HHH10001003: Autocommit mode: false
Apr 14, 2024 10:26:53 PM org.hibernate.engine
INFO: HHH000115: Hibernate connection pool si
Apr 14, 2024 10:26:53 PM org.hibernate.dialec
INFO: HHH000400: Using dialect: org.hibernate
Apr 14, 2024 10:26:54 PM org.hibernate.resour
INFO: HHH10001501: Connection obtained from J
    
```

25. Result



The screenshot displays the Microsoft SQL Server Management Studio interface. The title bar indicates the active window is 'SQLQuery2.sql - localhost.tutorial (sa (68))'. The menu bar includes File, Edit, View, Query, Project, Tools, Window, and Help. The toolbar contains various icons for file operations, query execution, and server management. The Object Explorer on the left shows the database structure for 'localhost (SQL Server 16.0.1000.6 - sa)', including System Databases, Database Snapshots, CandidateManagement, Equipments2024DB, and the 'tutorial' database. The 'tutorial' database is expanded, showing Database Diagrams, Tables (System Tables, FileTables, External Tables, Graph Tables, and user tables like dbo.hibernate_sequence and dbo.STUDENTS), Views, External Resources, Synonyms, Programmability, Query Store, Service Broker, Storage, and Security. The Query Editor in the center shows a SQL query:

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

 The Results pane at the bottom shows the query output as a table with 1 row and 4 columns: id, firstName, lastName, and marks. The status bar at the bottom indicates 'Query executed successfully.' and 'localhost (16.0 RTM) | sa (68) | tutorial | 00:00:00 | 1 rows'.

id	firstName	lastName	marks
1	Lam	Nguyen	9

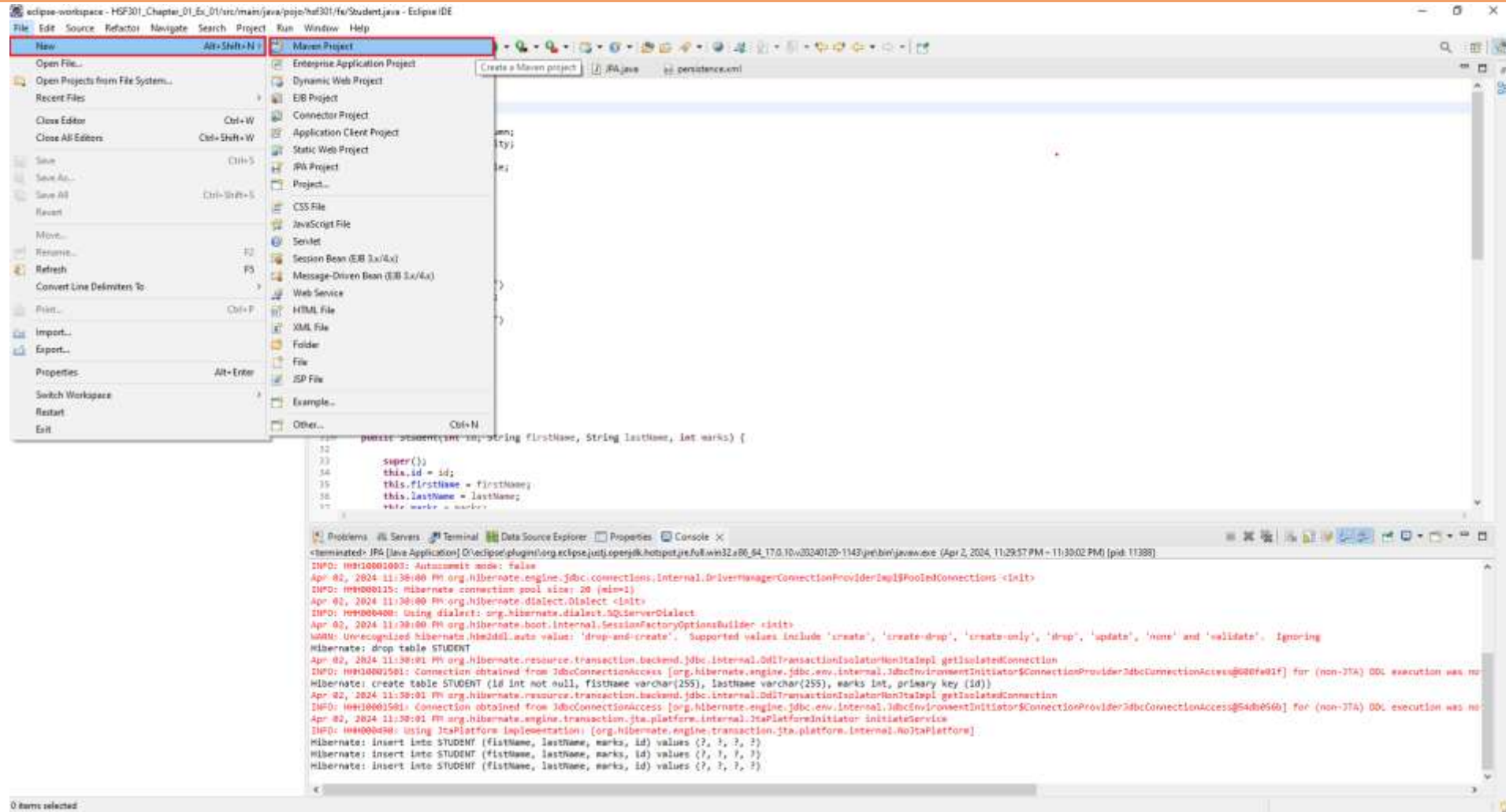
Mapping in JPA

Relationships Annotations in JPA

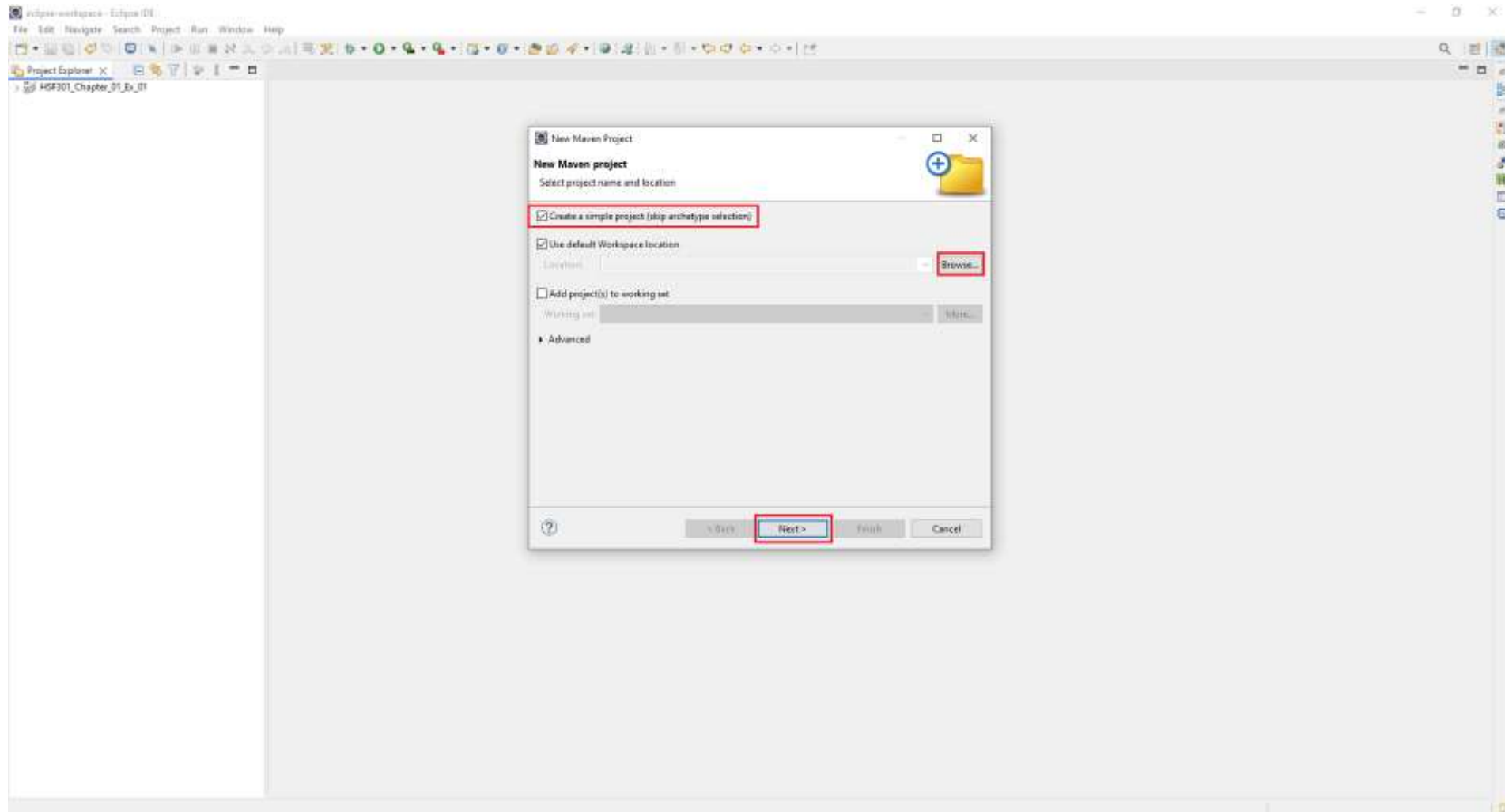
- ❖ **@ManyToOne:** This annotation defines a many-to-one relationship between two entities.
- ❖ **@OneToMany:** This annotation defines a one-to-many relationship between two entities.
- ❖ **@OneToOne:** This annotation defines a one-to-one relationship between two entities.
- ❖ **@ManyToMany:** This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

Demo JPA (One To Many)

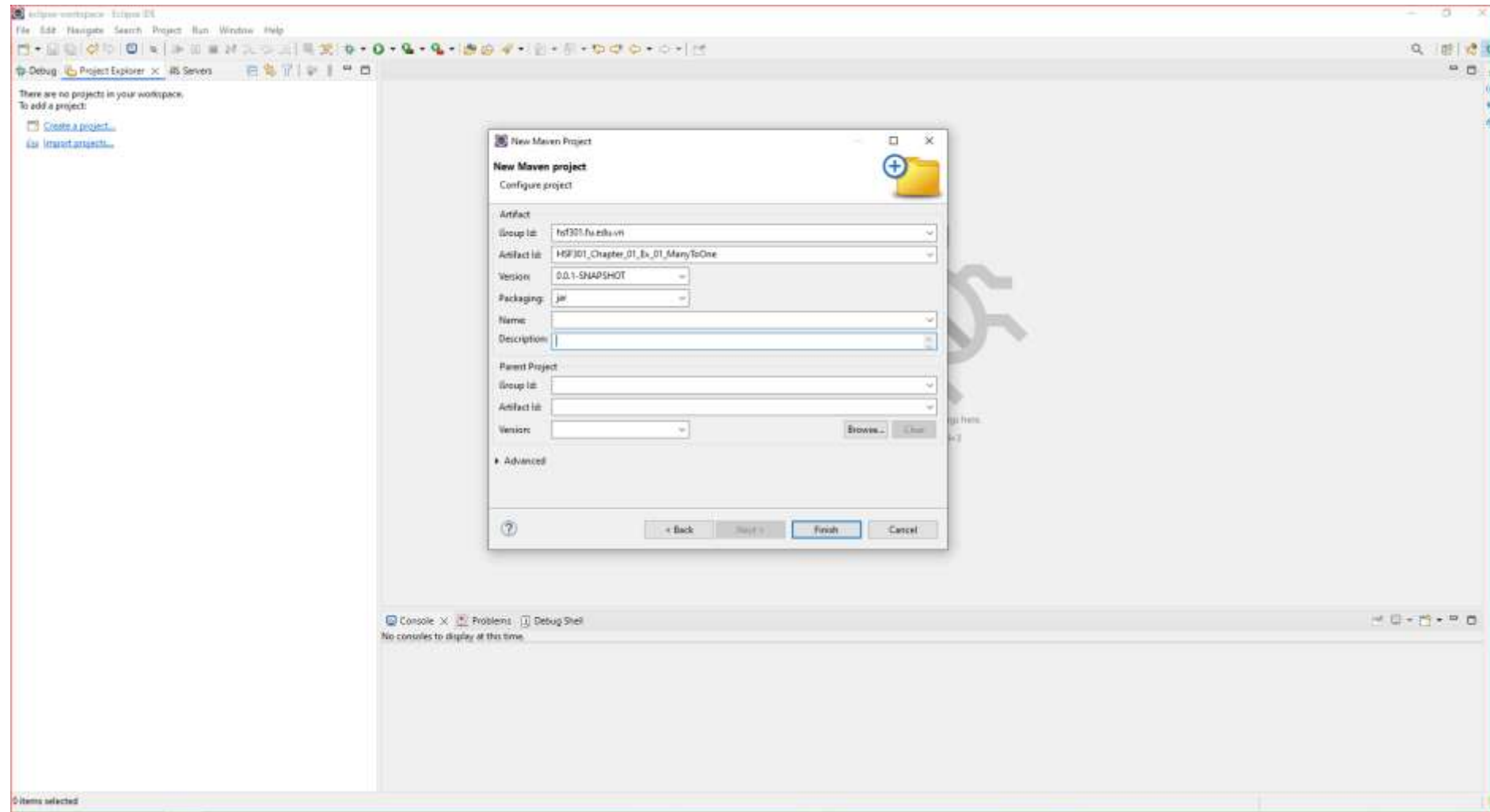
1. Open Eclipse, File | New | Maven Project



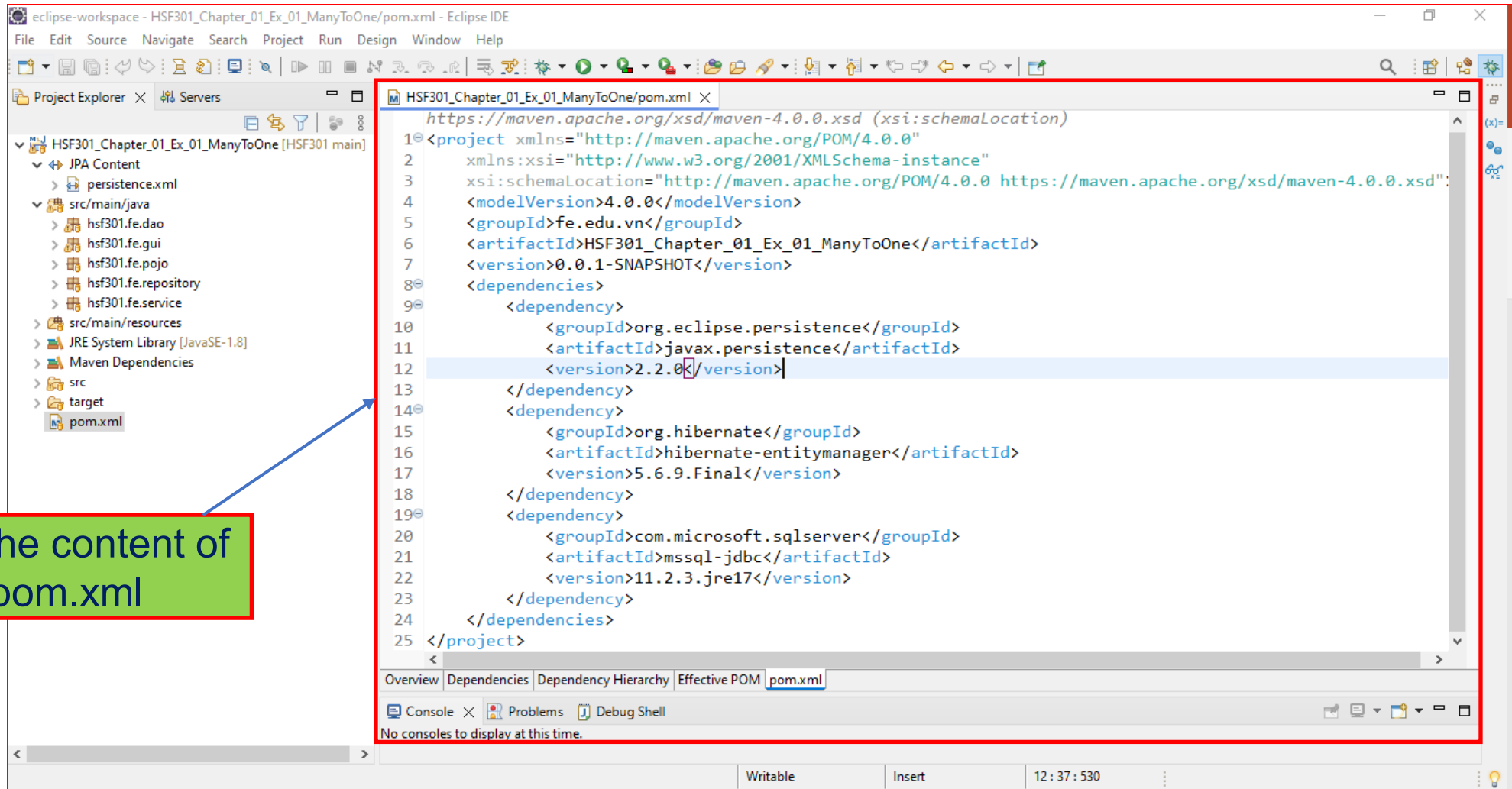
2. Check Create a simple project -> Browse Project -> Next



3. Fill the information Project -> Click Finish



4. Structure of Maven Project

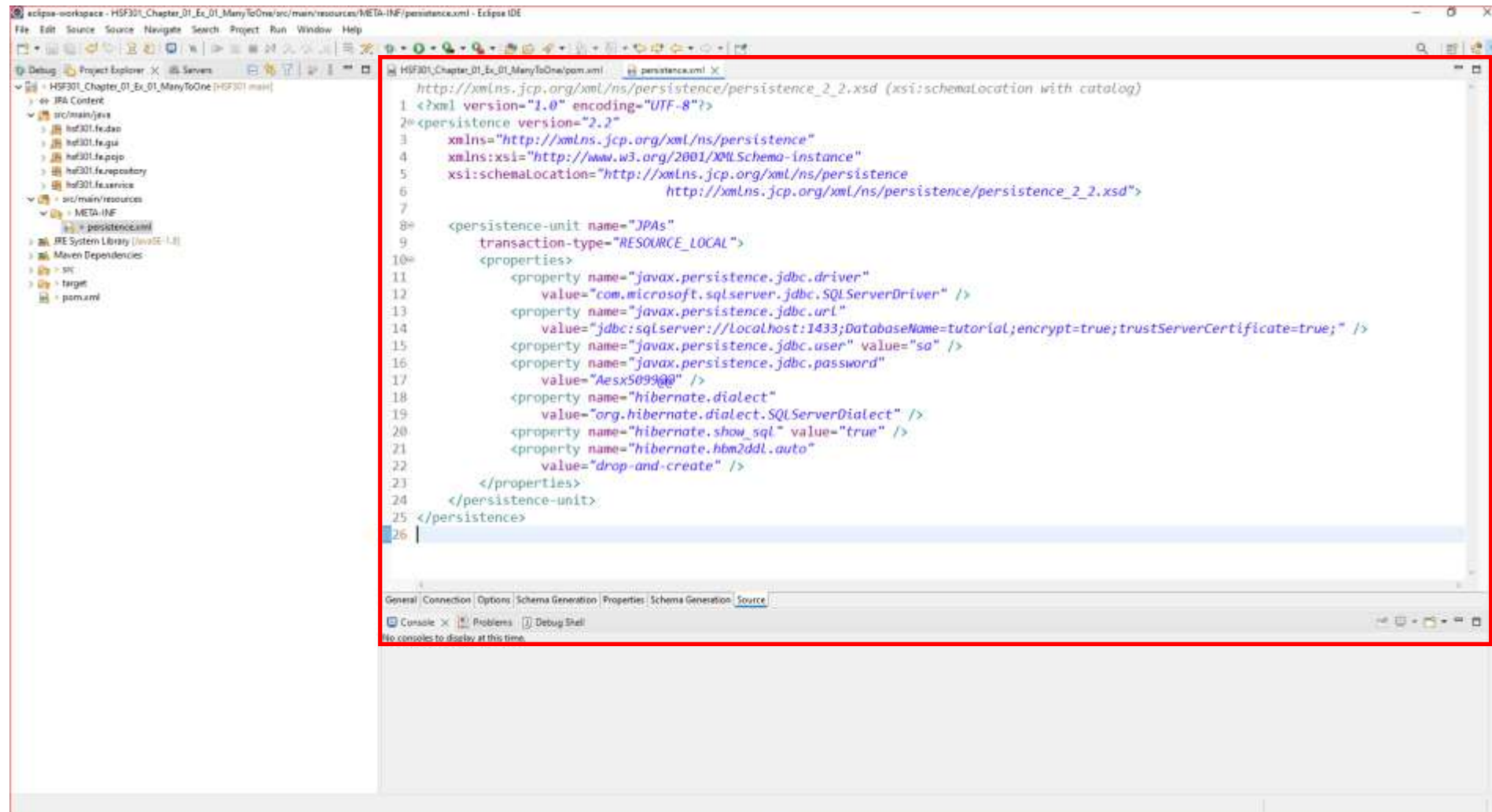


The screenshot shows the Eclipse IDE with a Maven project named `HSF301_Chapter_01_Ex_01_ManyToOne`. The `pom.xml` file is open in the editor, displaying the following XML content:

```
1<?xml version="1.0" encoding="UTF-8"?>
2<project xmlns="http://maven.apache.org/POM/4.0.0"
3  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5  <modelVersion>4.0.0</modelVersion>
6  <groupId>fe.edu.vn</groupId>
7  <artifactId>HSF301_Chapter_01_Ex_01_ManyToOne</artifactId>
8  <version>0.0.1-SNAPSHOT</version>
9  <dependencies>
10    <dependency>
11      <groupId>org.eclipse.persistence</groupId>
12      <artifactId>javax.persistence</artifactId>
13      <version>2.2.0</version>
14    </dependency>
15    <dependency>
16      <groupId>org.hibernate</groupId>
17      <artifactId>hibernate-entitymanager</artifactId>
18      <version>5.6.9.Final</version>
19    </dependency>
20    <dependency>
21      <groupId>com.microsoft.sqlserver</groupId>
22      <artifactId>mssql-jdbc</artifactId>
23      <version>11.2.3.jre17</version>
24    </dependency>
25  </dependencies>
26</project>
```

A red box highlights the `pom.xml` file in the Project Explorer and the corresponding XML content in the editor. A blue arrow points from a green box with the text "Edit the content of pom.xml" to the `pom.xml` file in the Project Explorer.

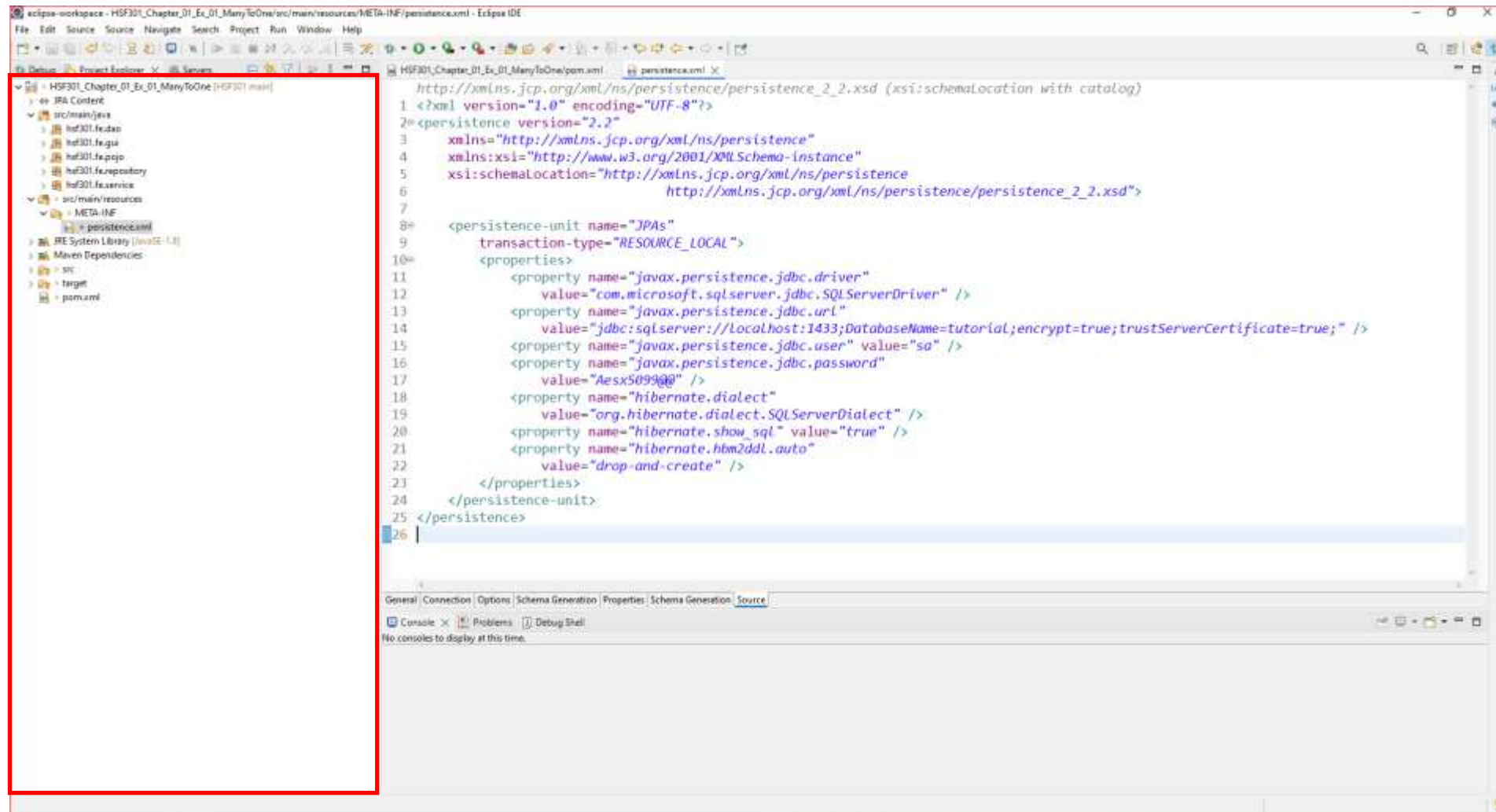
5. Create persistence.xml in META-INF folder



The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer displays the project structure, with the META-INF folder highlighted under the src/main/resources directory. The main editor window shows the content of persistence.xml, which is an XML file defining a persistence unit named 'JPAs'. The XML includes namespace declarations, a schema location, and a list of properties for the persistence unit, such as the JDBC driver, URL, user, password, dialect, and hibernate settings. The file is named persistence.xml and is located in the META-INF folder.

```
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd (xsi:schemaLocation with catalog)
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.2"
3   xmlns="http://xmlns.jcp.org/xml/ns/persistence"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
6     http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
7
8   <persistence-unit name="JPAs"
9     transaction-type="RESOURCE_LOCAL">
10     <properties>
11       <property name="javax.persistence.jdbc.driver"
12         value="com.microsoft.sqlserver.jdbc.SQLServerDriver" />
13       <property name="javax.persistence.jdbc.url"
14         value="jdbc:sqlserver://localhost:1433;DatabaseName=tutorial;encrypt=true;trustServerCertificate=true;" />
15       <property name="javax.persistence.jdbc.user" value="sa" />
16       <property name="javax.persistence.jdbc.password"
17         value="Aesx509%00" />
18       <property name="hibernate.dialect"
19         value="org.hibernate.dialect.SQLServerDialect" />
20       <property name="hibernate.show_sql" value="true" />
21       <property name="hibernate.hbm2ddl.auto"
22         value="drop-and-create" />
23     </properties>
24   </persistence-unit>
25 </persistence>
26
```

6. Create structure of project



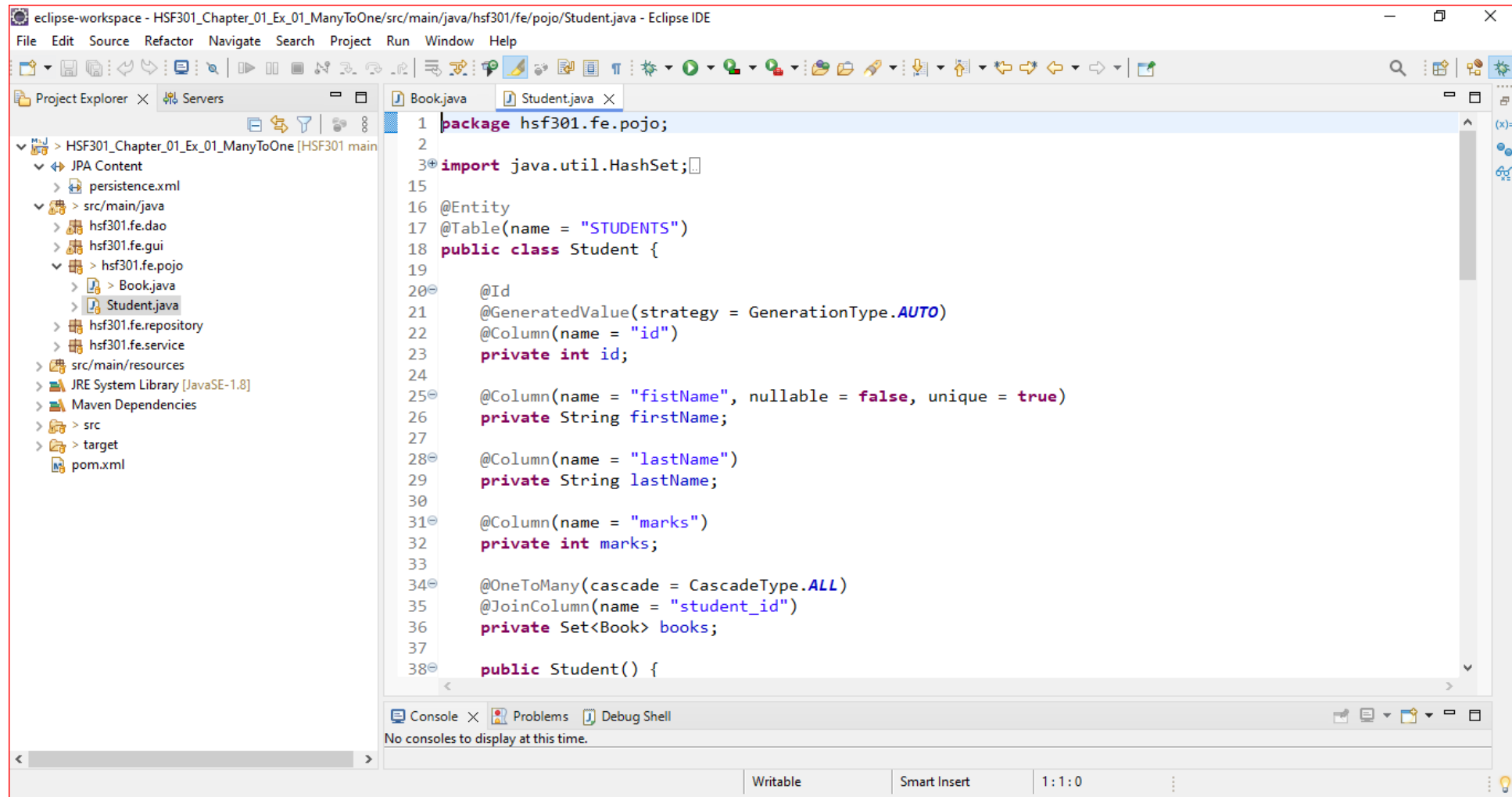
7. Create Books in Pojo

```

1 package hsf301.fe.pojo;
2
3 import javax.persistence.CascadeType;
4
5 @Entity
6 @Table(name = "BOOKS")
7 public class Book {
8
9     @Id
10    @GeneratedValue(strategy = GenerationType.AUTO)
11    private Long id;
12
13    @Column(name = "title", length = 30)
14    private String title;
15
16    private String author;
17
18    private String isbn;
19
20    @ManyToOne(cascade = CascadeType.ALL)
21    @JoinColumn(name = "student_id")
22    private Student student;
23
24    public Book() {
25        super();
26    }
27
28 }

```

8. Create Students in Pojo



```
eclipse-workspace - HSF301_Chapter_01_Ex_01_ManyToOne/src/main/java/hsf301/fe/pojo/Student.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

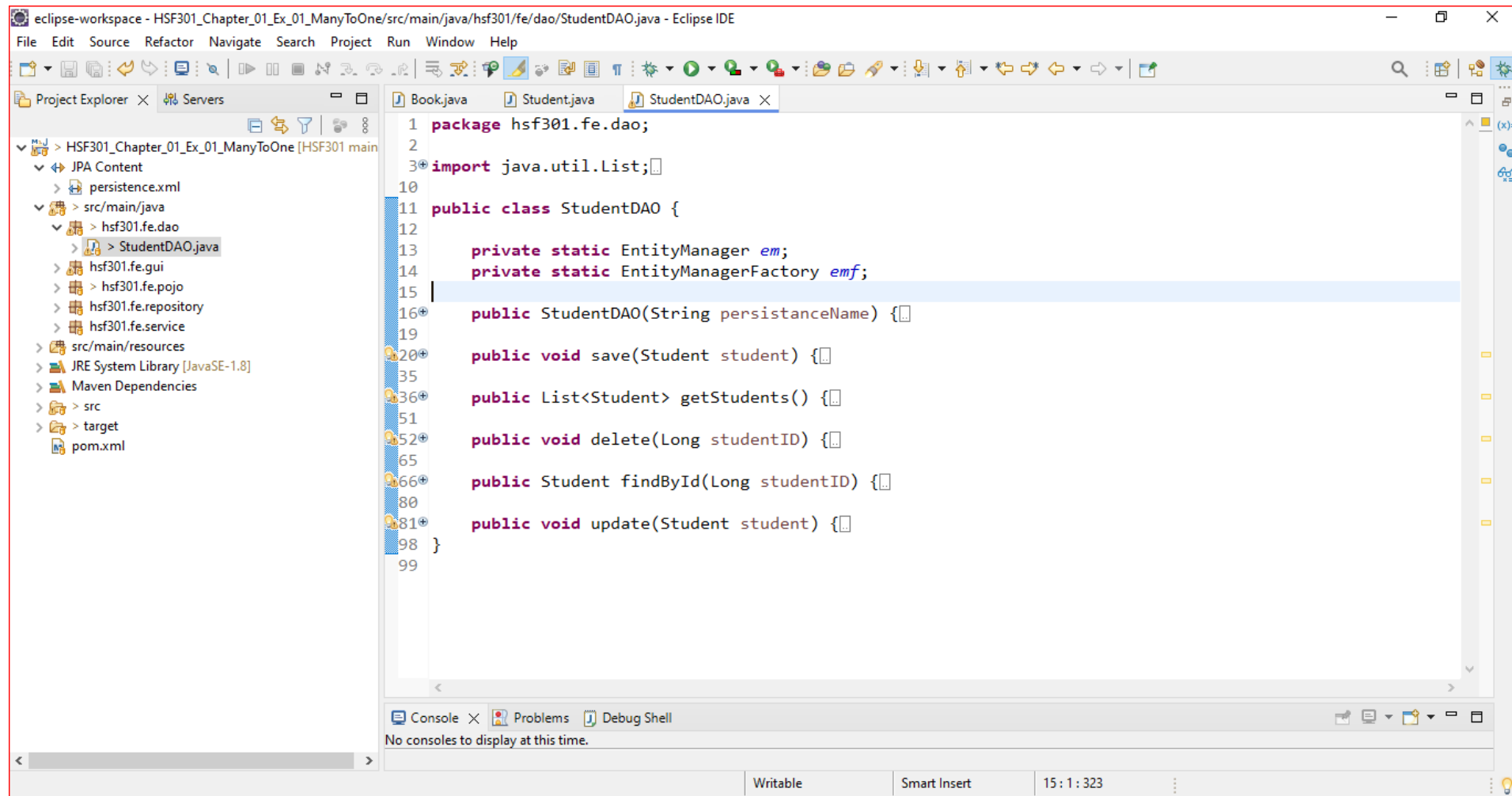
Project Explorer Servers
  HSF301_Chapter_01_Ex_01_ManyToOne [HSF301 main]
    JPA Content
      persistence.xml
    src/main/java
      hsf301.fe.dao
      hsf301.fe.gui
      hsf301.fe.pojo
        Book.java
        Student.java
      hsf301.fe.repository
      hsf301.fe.service
    src/main/resources
    JRE System Library [JavaSE-1.8]
    Maven Dependencies
    src
    target
    pom.xml

1 package hsf301.fe.pojo;
2
3 import java.util.HashSet;
4
15
16 @Entity
17 @Table(name = "STUDENTS")
18 public class Student {
19
20     @Id
21     @GeneratedValue(strategy = GenerationType.AUTO)
22     @Column(name = "id")
23     private int id;
24
25     @Column(name = "firstName", nullable = false, unique = true)
26     private String firstName;
27
28     @Column(name = "lastName")
29     private String lastName;
30
31     @Column(name = "marks")
32     private int marks;
33
34     @OneToMany(cascade = CascadeType.ALL)
35     @JoinColumn(name = "student_id")
36     private Set<Book> books;
37
38     public Student() {
```

Console Problems Debug Shell
No consoles to display at this time.

Writable Smart Insert 1:1:0

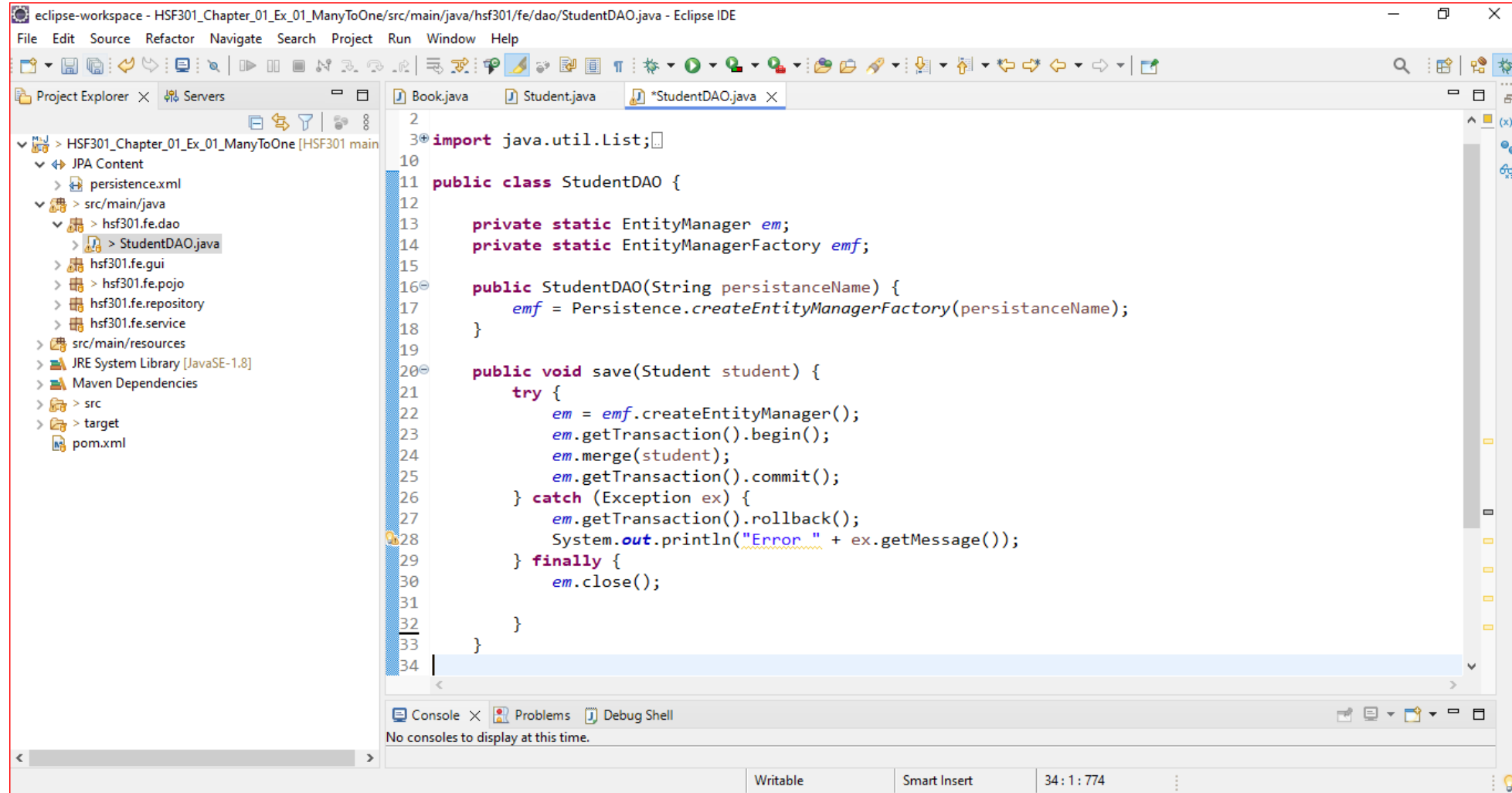
9. Create StudentDAO



The screenshot shows the Eclipse IDE interface. The Project Explorer on the left displays the project structure: HSF301_Chapter_01_Ex_01_ManyToOne [HSF301 main] > JPA Content > persistence.xml > src/main/java > hsf301.fe.dao > StudentDAO.java. The main editor window shows the code for StudentDAO.java. The code defines a package, imports List, and implements the StudentDAO interface with methods for save, getStudents, delete, findById, and update.

```
1 package hsf301.fe.dao;
2
3 import java.util.List;
4
5
6
7
8
9
10
11 public class StudentDAO {
12
13     private static EntityManager em;
14     private static EntityManagerFactory emf;
15
16     public StudentDAO(String persistenceName) {}
17
18     public void save(Student student) {}
19
20     public List<Student> getStudents() {}
21
22     public void delete(Long studentID) {}
23
24     public Student findById(Long studentID) {}
25
26     public void update(Student student) {}
27 }
```

10. Save Student in StudentDAO



```
eclipse-workspace - HSF301_Chapter_01_Ex_01_ManyToOne/src/main/java/hsf301/fe/dao/StudentDAO.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

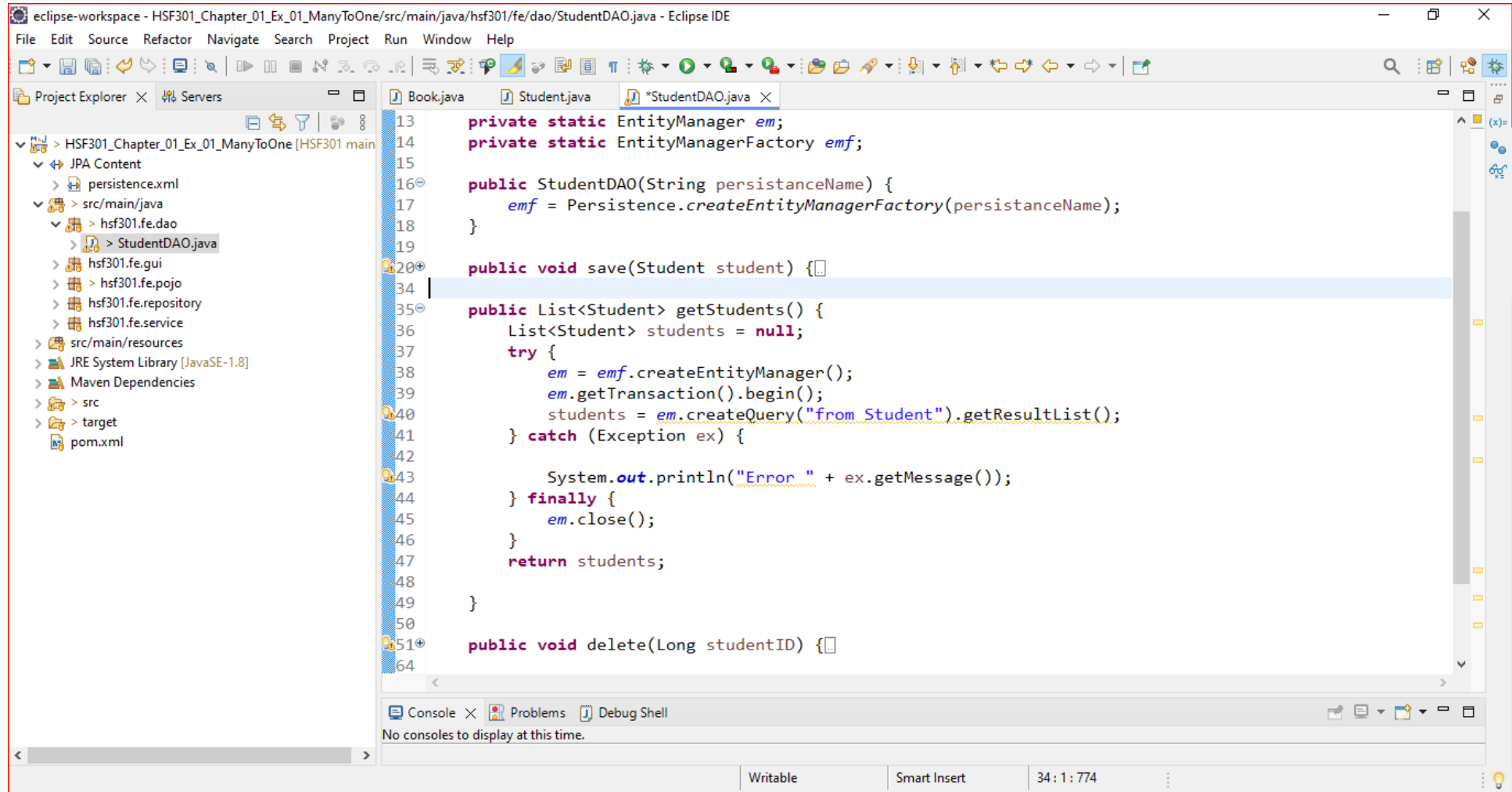
Project Explorer Servers
  HSF301_Chapter_01_Ex_01_ManyToOne [HSF301 main]
    JPA Content
      persistence.xml
    src/main/java
      hsf301.fe.dao
        StudentDAO.java
      hsf301.fe.gui
      hsf301.fe.pojo
      hsf301.fe.repository
      hsf301.fe.service
    src/main/resources
    JRE System Library [JavaSE-1.8]
    Maven Dependencies
    src
    target
    pom.xml

Book.java Student.java *StudentDAO.java
2
3 import java.util.List;
10
11 public class StudentDAO {
12
13     private static EntityManager em;
14     private static EntityManagerFactory emf;
15
16     public StudentDAO(String persistenceName) {
17         emf = Persistence.createEntityManagerFactory(persistenceName);
18     }
19
20     public void save(Student student) {
21         try {
22             em = emf.createEntityManager();
23             em.getTransaction().begin();
24             em.merge(student);
25             em.getTransaction().commit();
26         } catch (Exception ex) {
27             em.getTransaction().rollback();
28             System.out.println("Error " + ex.getMessage());
29         } finally {
30             em.close();
31         }
32     }
33 }
34

Console Problems Debug Shell
No consoles to display at this time.

Writable Smart Insert 34:1:774
```


11. Get All Students in StudentDAO

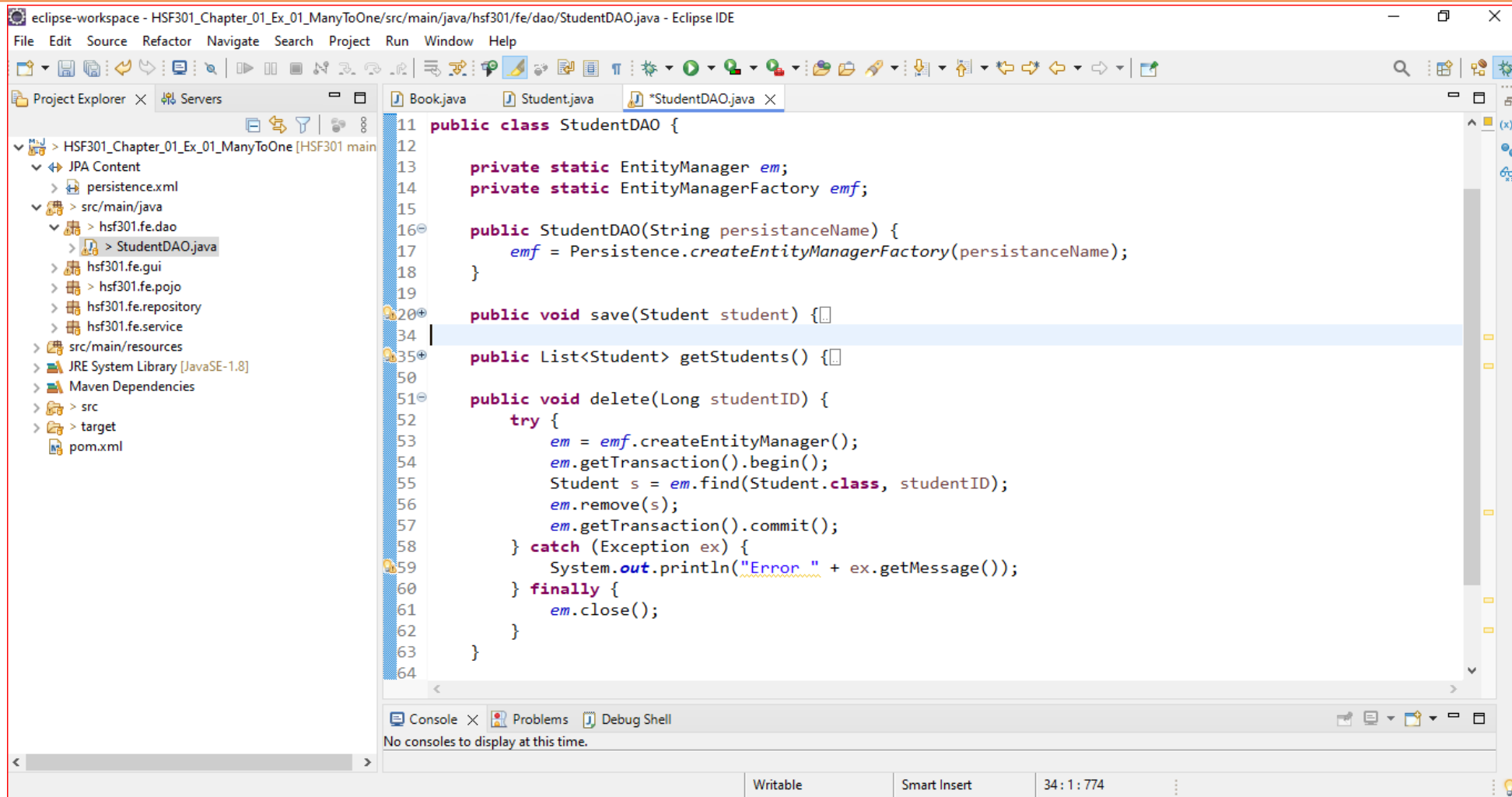


```
eclipse-workspace - HSF301_Chapter_01_Ex_01_ManyToOne/src/main/java/hsf301/fe/dao/StudentDAO.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer Servers
HSF301_Chapter_01_Ex_01_ManyToOne [HSF301 main]
  JPA Content
    persistence.xml
  src/main/java
    hsf301.fe.dao
      StudentDAO.java
    hsf301.fe.gui
    hsf301.fe.pojo
    hsf301.fe.repository
    hsf301.fe.service
  src/main/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  pom.xml

Book.java Student.java *StudentDAO.java
13 private static EntityManager em;
14 private static EntityManagerFactory emf;
15
16 public StudentDAO(String persistenceName) {
17     emf = Persistence.createEntityManagerFactory(persistenceName);
18 }
19
20 public void save(Student student) {}
34
35 public List<Student> getStudents() {
36     List<Student> students = null;
37     try {
38         em = emf.createEntityManager();
39         em.getTransaction().begin();
40         students = em.createQuery("from Student").getResultList();
41     } catch (Exception ex) {
42
43         System.out.println("Error " + ex.getMessage());
44     } finally {
45         em.close();
46     }
47     return students;
48 }
49
50
51 public void delete(Long studentID) {}
64

Console Problems Debug Shell
No consoles to display at this time.
Writable Smart Insert 34: 1: 774
```

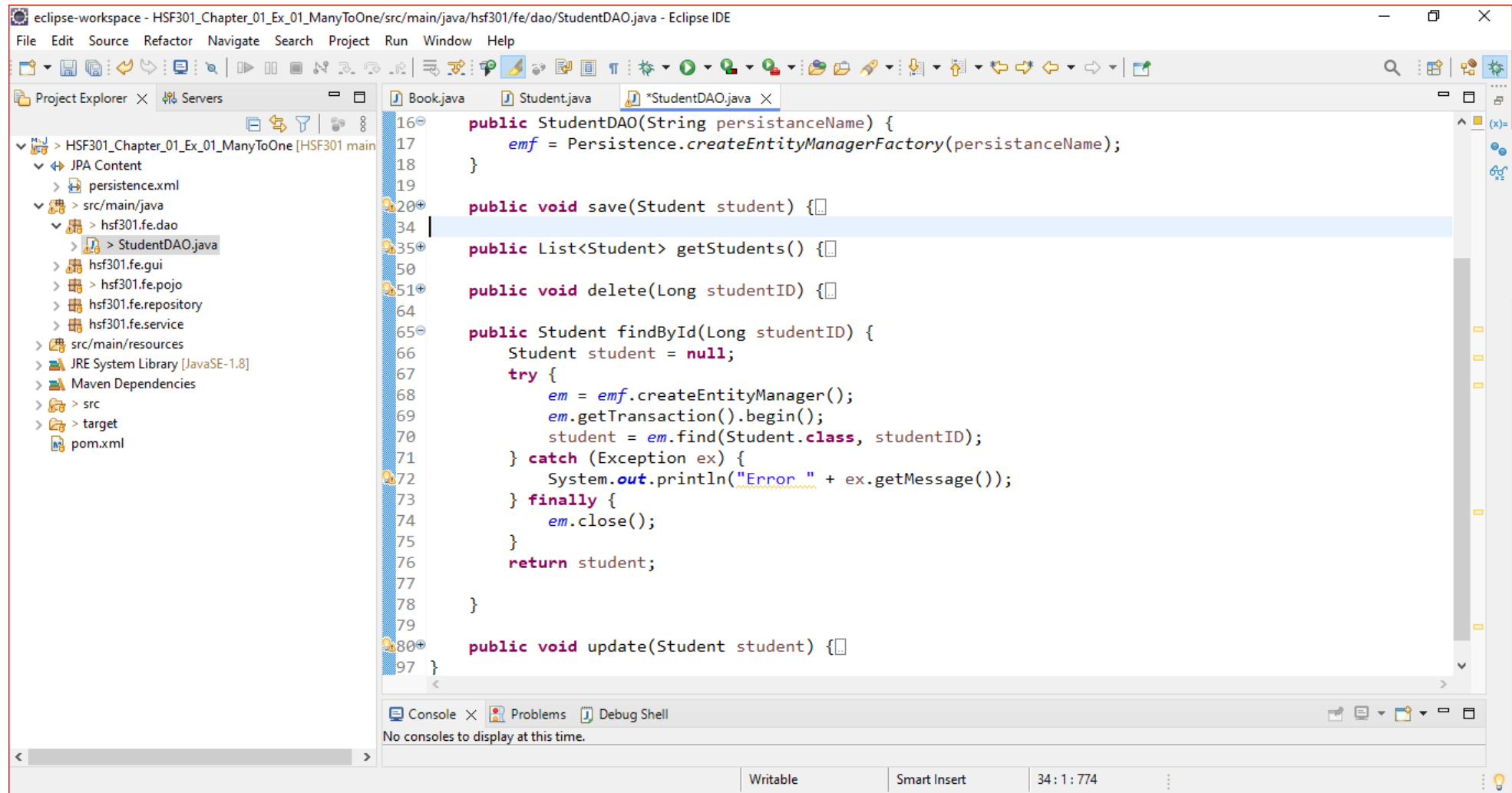
12. Delete Student in StudentDAO



The screenshot shows the Eclipse IDE with the file `StudentDAO.java` open. The code defines a `StudentDAO` class with a static `EntityManager` and `EntityManagerFactory`. It includes methods for `save`, `getStudents`, and `delete`. The `delete` method uses a try-catch block to handle exceptions and prints an error message if one occurs.

```
11 public class StudentDAO {
12
13     private static EntityManager em;
14     private static EntityManagerFactory emf;
15
16     public StudentDAO(String persistenceName) {
17         emf = Persistence.createEntityManagerFactory(persistenceName);
18     }
19
20     public void save(Student student) {}
21
22     public List<Student> getStudents() {}
23
24     public void delete(Long studentID) {
25         try {
26             em = emf.createEntityManager();
27             em.getTransaction().begin();
28             Student s = em.find(Student.class, studentID);
29             em.remove(s);
30             em.getTransaction().commit();
31         } catch (Exception ex) {
32             System.out.println("Error " + ex.getMessage());
33         } finally {
34             em.close();
35         }
36     }
37 }
```

13. Get a Student in StudentDAO



```
eclipse-workspace - HSF301_Chapter_01_Ex_01_ManyToOne/src/main/java/hsf301/fe/dao/StudentDAO.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

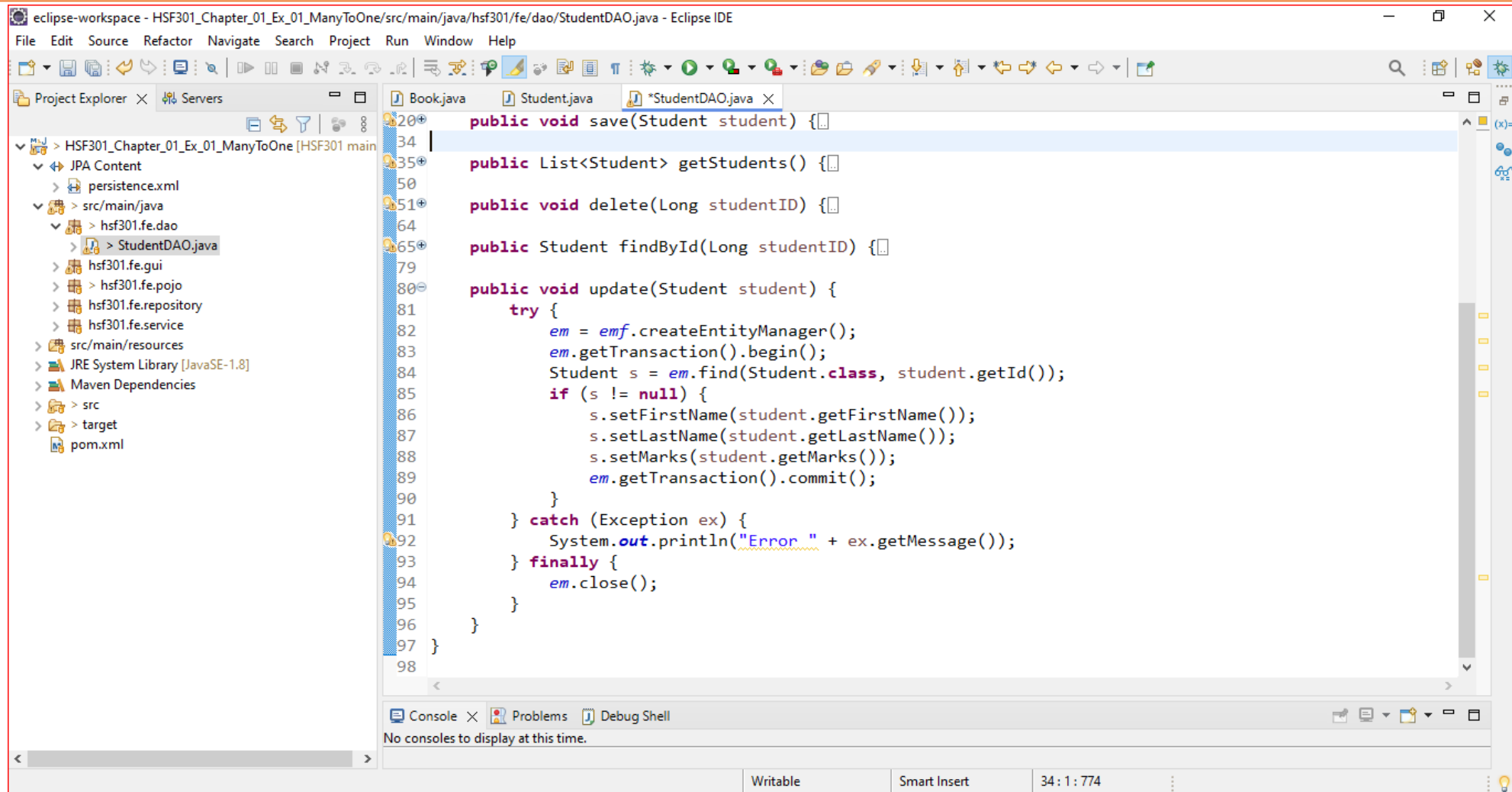
Project Explorer
  HSF301_Chapter_01_Ex_01_ManyToOne [HSF301 main]
    JPA Content
      persistence.xml
    src/main/java
      hsf301.fe.dao
        StudentDAO.java
      hsf301.fe.gui
      hsf301.fe.pojo
      hsf301.fe.repository
      hsf301.fe.service
    src/main/resources
    JRE System Library [JavaSE-1.8]
    Maven Dependencies
    src
    target
    pom.xml

Book.java Student.java *StudentDAO.java
16 public StudentDAO(String persistenceName) {
17     emf = Persistence.createEntityManagerFactory(persistenceName);
18 }
19
20 public void save(Student student) {}
34
35 public List<Student> getStudents() {}
50
51 public void delete(Long studentID) {}
64
65 public Student findById(Long studentID) {
66     Student student = null;
67     try {
68         em = emf.createEntityManager();
69         em.getTransaction().begin();
70         student = em.find(Student.class, studentID);
71     } catch (Exception ex) {
72         System.out.println("Error " + ex.getMessage());
73     } finally {
74         em.close();
75     }
76     return student;
77 }
78
79
80 public void update(Student student) {}
97 }
```

Console Problems Debug Shell
No consoles to display at this time.

Writable Smart Insert 34 : 1 : 774

14. Get a Student in StudentDAO



```
eclipse-workspace - HSF301_Chapter_01_Ex_01_ManyToOne/src/main/java/hsf301/fe/dao/StudentDAO.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

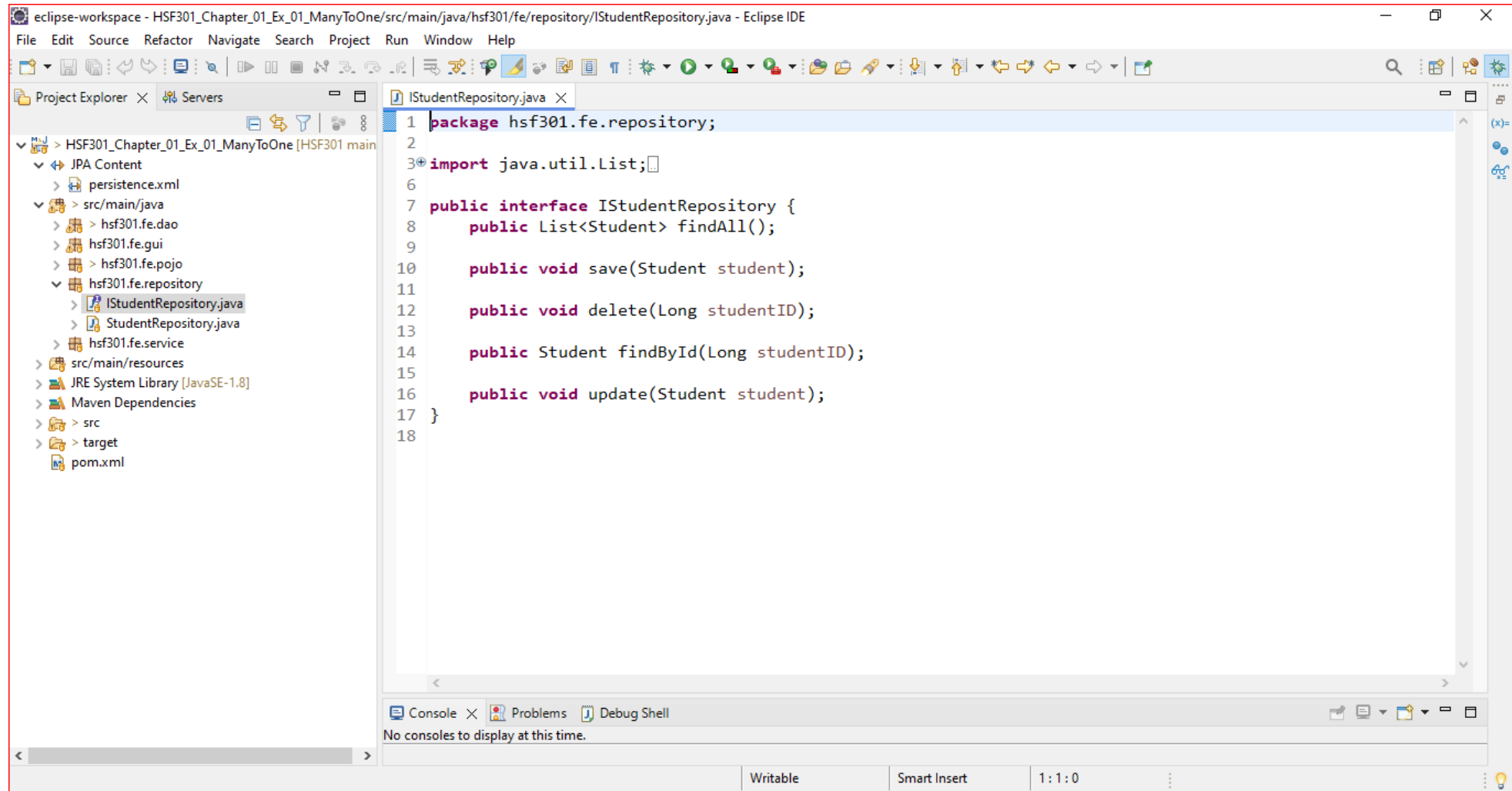
Project Explorer
  HSF301_Chapter_01_Ex_01_ManyToOne [HSF301 main]
    JPA Content
      persistence.xml
    src/main/java
      hsf301.fe.dao
        StudentDAO.java
      hsf301.fe.gui
      hsf301.fe.pojo
      hsf301.fe.repository
      hsf301.fe.service
    src/main/resources
    JRE System Library [JavaSE-1.8]
    Maven Dependencies
    src
    target
    pom.xml

StudentDAO.java
20+ public void save(Student student) {}
34
35+ public List<Student> getStudents() {}
50
51+ public void delete(Long studentID) {}
64
65+ public Student findById(Long studentID) {}
79
80+ public void update(Student student) {
81     try {
82         em = emf.createEntityManager();
83         em.getTransaction().begin();
84         Student s = em.find(Student.class, student.getId());
85         if (s != null) {
86             s.setFirstName(student.getFirstName());
87             s.setLastName(student.getLastName());
88             s.setMarks(student.getMarks());
89             em.getTransaction().commit();
90         }
91     } catch (Exception ex) {
92         System.out.println("Error " + ex.getMessage());
93     } finally {
94         em.close();
95     }
96 }
97 }
98

Console Problems Debug Shell
No consoles to display at this time.

Writable Smart Insert 34:1:774
```

15. Create IStudentRepository

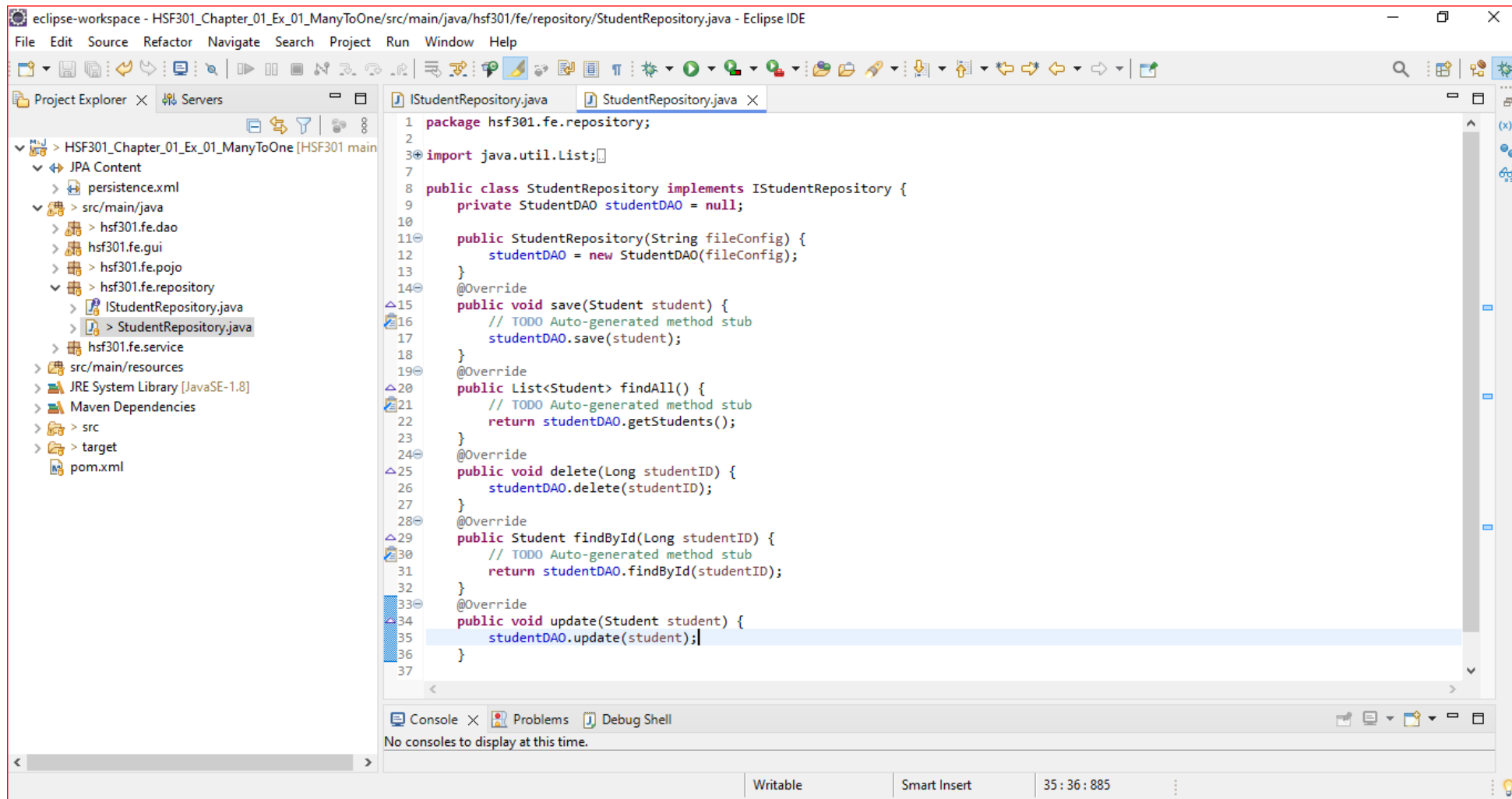


The screenshot shows the Eclipse IDE interface. The Project Explorer on the left displays the project structure for 'HSF301_Chapter_01_Ex_01_ManyToOne'. The 'hsf301.fe.repository' package is selected, and the 'IStudentRepository.java' file is highlighted. The main editor window shows the code for the 'IStudentRepository' interface. The code includes a package declaration, an import statement for 'java.util.List', and five methods: 'findAll()', 'save()', 'delete()', 'findById()', and 'update()'.

```
1 package hsf301.fe.repository;
2
3 import java.util.List;
4
5
6
7 public interface IStudentRepository {
8     public List<Student> findAll();
9
10    public void save(Student student);
11
12    public void delete(Long studentID);
13
14    public Student findById(Long studentID);
15
16    public void update(Student student);
17 }
18
```

The bottom of the IDE shows the Console, Problems, and Debug Shell tabs, all of which are empty.

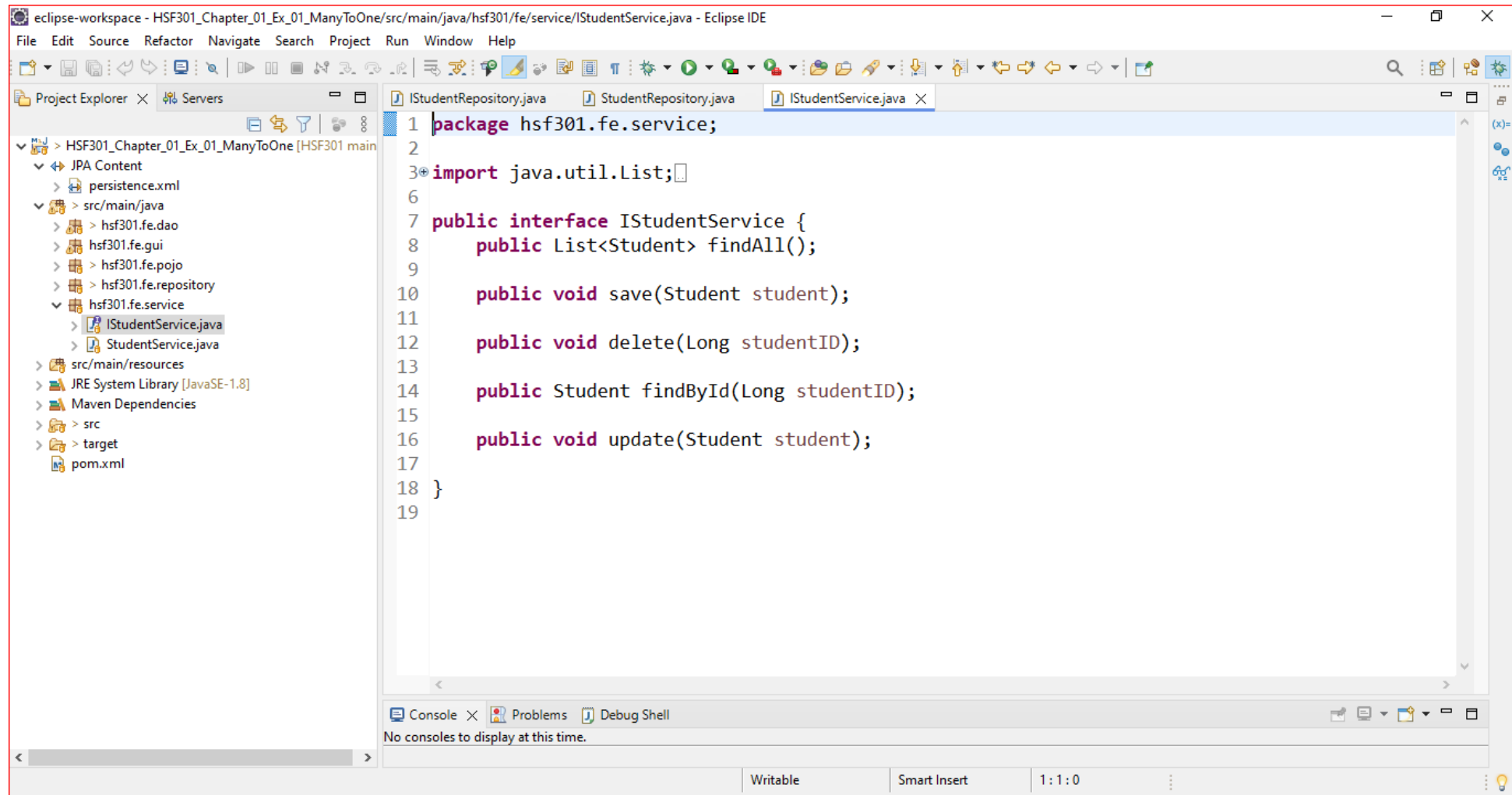
16. Create StudentRepository



The screenshot shows the Eclipse IDE interface. The Project Explorer on the left displays the project structure, with the file `StudentRepository.java` selected under the `hsf301.fe.repository` package. The main editor window shows the code for `StudentRepository.java`, which implements the `IStudentRepository` interface. The code includes package declarations, imports, and several methods: `StudentRepository(String fileConfig)`, `save(Student student)`, `findAll()`, `delete(Long studentID)`, `findById(Long studentID)`, and `update(Student student)`. The `update` method is currently selected and highlighted. The status bar at the bottom indicates the file is writable and shows the cursor position at line 35, column 36.

```
1 package hsf301.fe.repository;
2
3 import java.util.List;
4
5
6
7
8 public class StudentRepository implements IStudentRepository {
9     private StudentDAO studentDAO = null;
10
11     public StudentRepository(String fileConfig) {
12         studentDAO = new StudentDAO(fileConfig);
13     }
14
15     @Override
16     public void save(Student student) {
17         // TODO Auto-generated method stub
18         studentDAO.save(student);
19     }
20
21     @Override
22     public List<Student> findAll() {
23         // TODO Auto-generated method stub
24         return studentDAO.getStudents();
25     }
26
27     @Override
28     public void delete(Long studentID) {
29         studentDAO.delete(studentID);
30     }
31
32     @Override
33     public Student findById(Long studentID) {
34         // TODO Auto-generated method stub
35         return studentDAO.findById(studentID);
36     }
37
38     @Override
39     public void update(Student student) {
40         studentDAO.update(student);
41     }
42 }
```

17. Create IStudentService



The screenshot shows the Eclipse IDE interface. The Project Explorer on the left displays the project structure for 'HSF301_Chapter_01_Ex_01_ManyToOne'. The main editor window shows the code for 'IStudentService.java'. The code defines a package, imports a utility class, and declares a public interface with five methods.

```
eclipse-workspace - HSF301_Chapter_01_Ex_01_ManyToOne/src/main/java/hsf301/fe/service/IStudentService.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

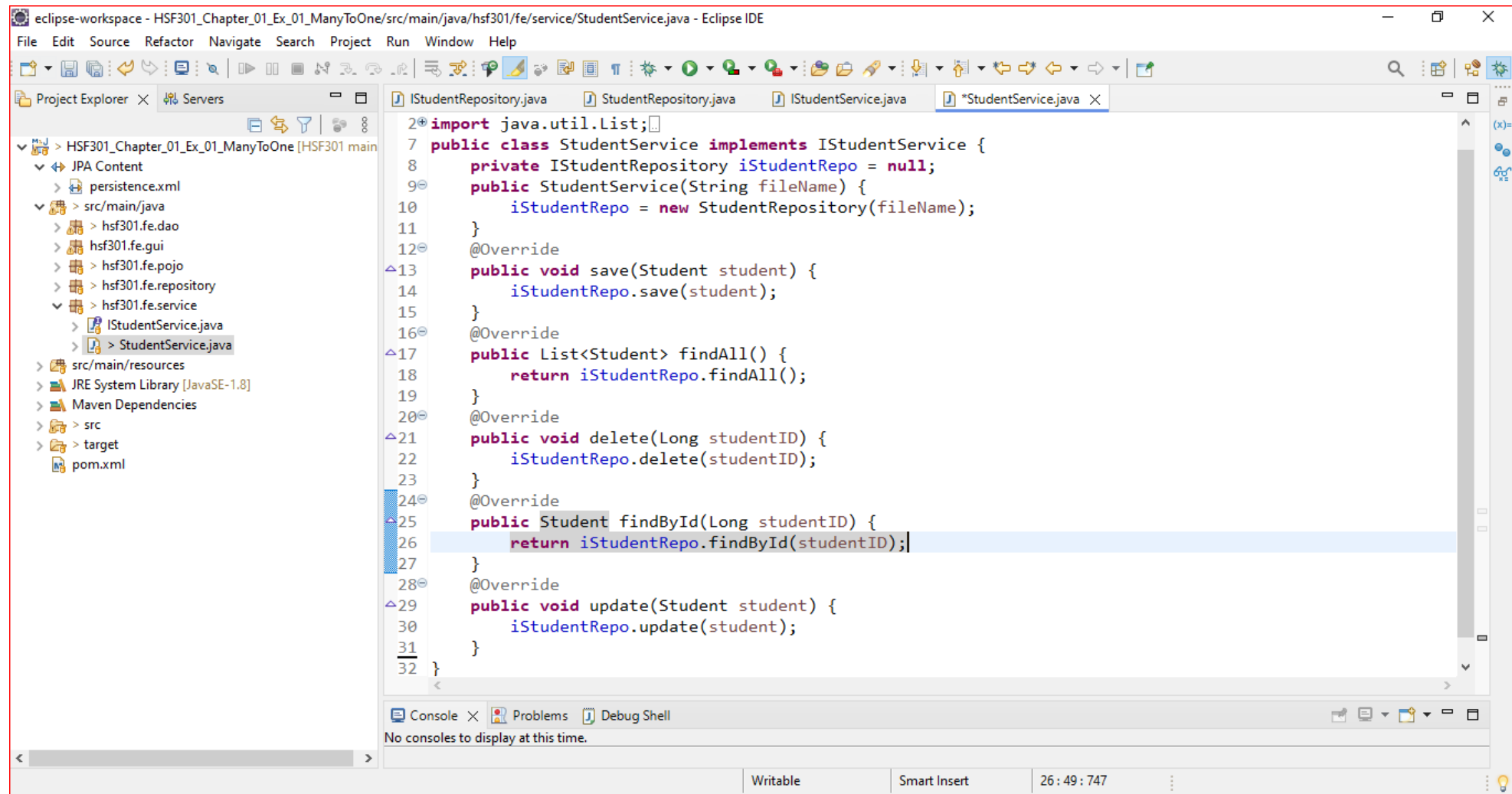
Project Explorer × Servers
  HSF301_Chapter_01_Ex_01_ManyToOne [HSF301 main]
    JPA Content
      persistence.xml
    src/main/java
      hsf301.fe.dao
      hsf301.fe.gui
      hsf301.fe.pojo
      hsf301.fe.repository
      hsf301.fe.service
        IStudentService.java
        StudentService.java
      src/main/resources
      JRE System Library [JavaSE-1.8]
      Maven Dependencies
      src
      target
      pom.xml

IStudentRepository.java StudentRepository.java IStudentService.java ×
1 package hsf301.fe.service;
2
3 import java.util.List;
6
7 public interface IStudentService {
8     public List<Student> findAll();
9
10    public void save(Student student);
11
12    public void delete(Long studentID);
13
14    public Student findById(Long studentID);
15
16    public void update(Student student);
17 }
18
19

Console × Problems Debug Shell
No consoles to display at this time.

Writable Smart Insert 1:1:0
```

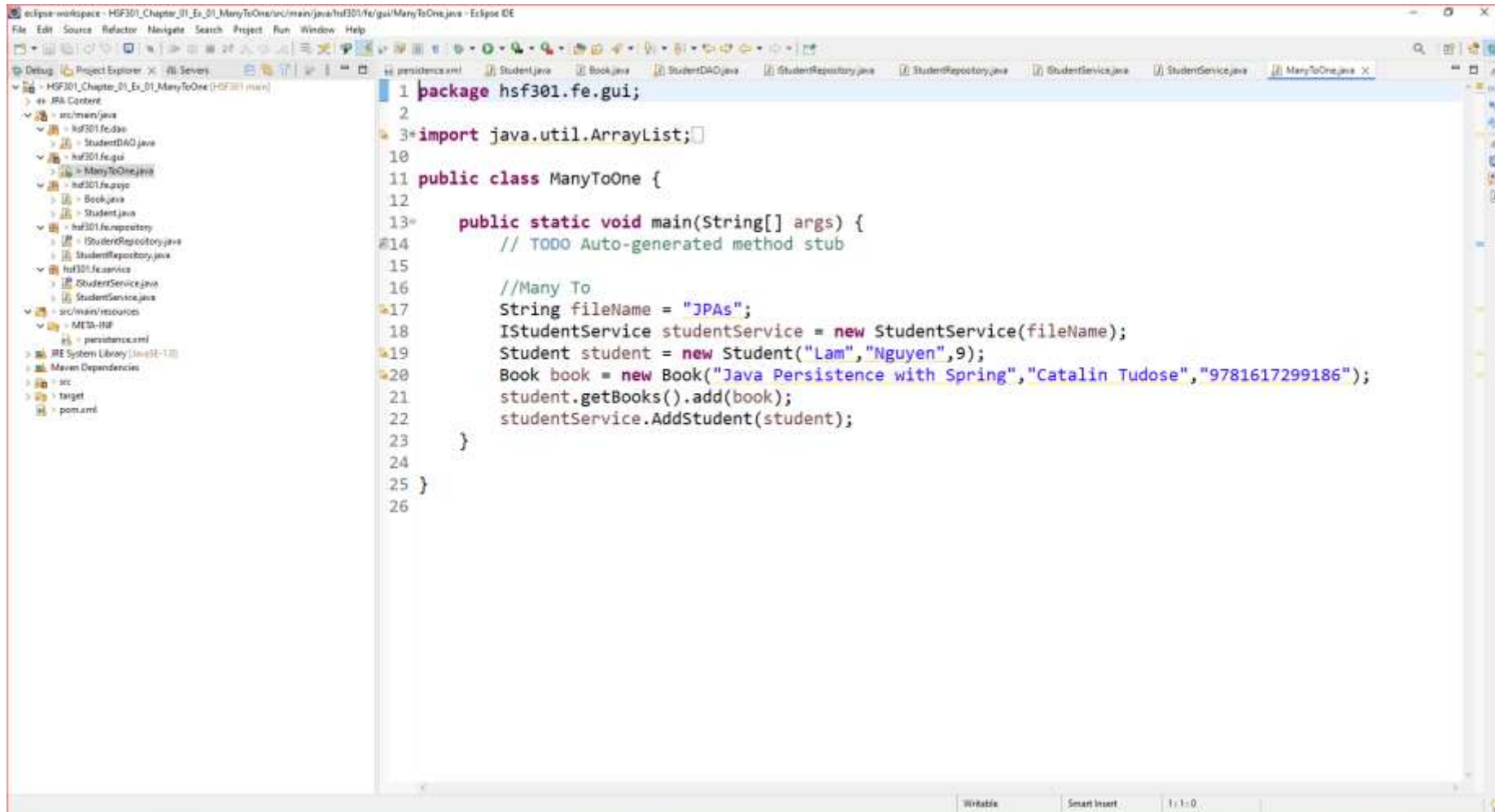

18. Create StudentService



The screenshot shows the Eclipse IDE interface. The Project Explorer on the left displays the project structure for 'HSF301_Chapter_01_Ex_01_ManyToOne'. The main editor window shows the 'StudentService.java' file, which implements the 'IStudentService' interface. The code includes imports for 'java.util.List' and 'java.util.ArrayList', and defines several methods: 'save', 'findAll', 'delete', 'findById', and 'update'. The 'findById' method is currently selected and highlighted.

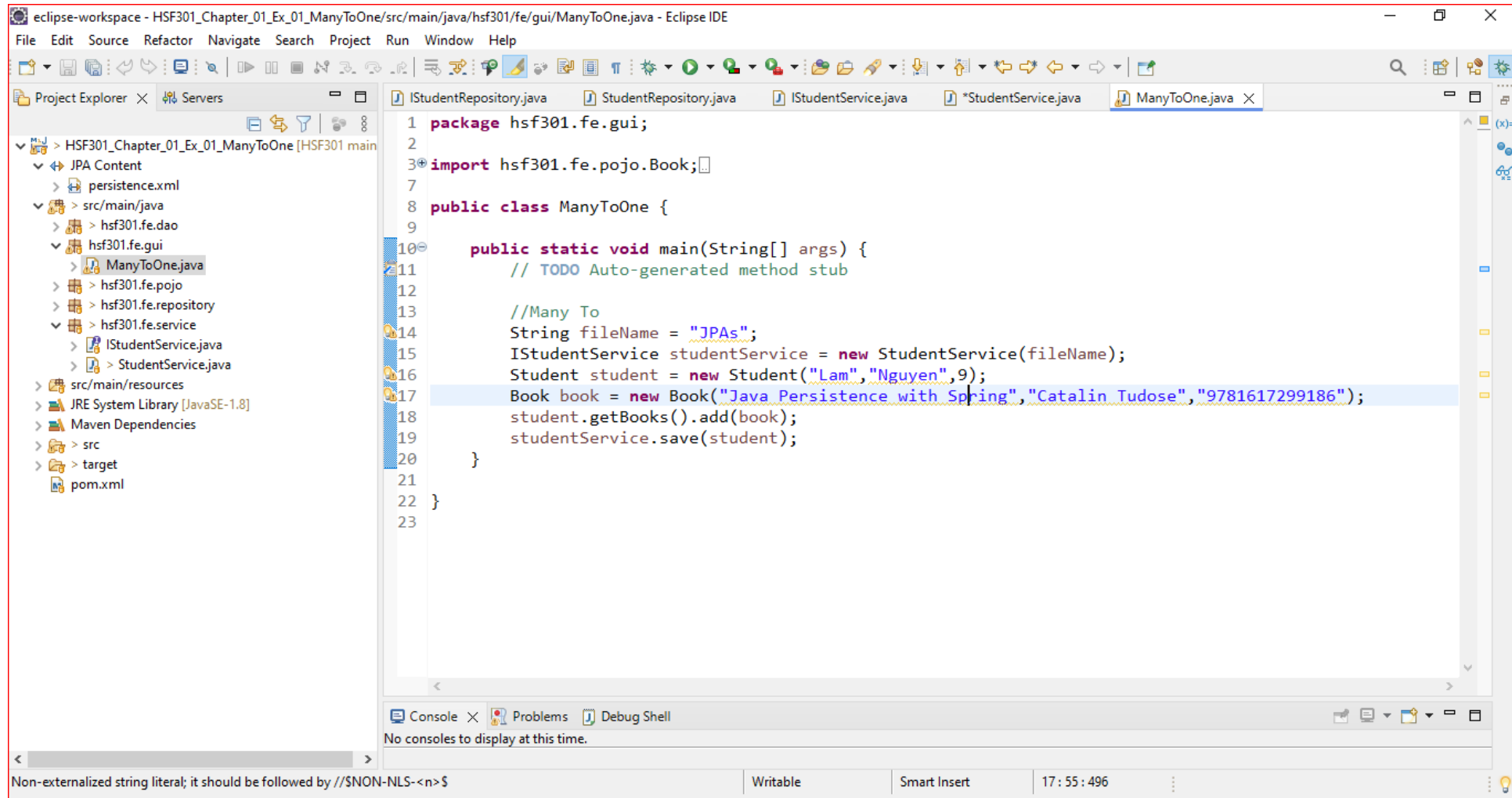
```
2 import java.util.List;
3 import java.util.ArrayList;
4 
5 public class StudentService implements IStudentService {
6     private IStudentRepository iStudentRepo = null;
7     public StudentService(String fileName) {
8         iStudentRepo = new StudentRepository(fileName);
9     }
10    @Override
11    public void save(Student student) {
12        iStudentRepo.save(student);
13    }
14    @Override
15    public List<Student> findAll() {
16        return iStudentRepo.findAll();
17    }
18    @Override
19    public void delete(Long studentID) {
20        iStudentRepo.delete(studentID);
21    }
22    @Override
23    public Student findById(Long studentID) {
24        return iStudentRepo.findById(studentID);
25    }
26    @Override
27    public void update(Student student) {
28        iStudentRepo.update(student);
29    }
30 }
```


19. Create Main function



```
1 package hsf301.fe.gui;
2
3 import java.util.ArrayList;
4
5
6
7
8
9
10
11 public class ManyToOne {
12
13     public static void main(String[] args) {
14         // TODO Auto-generated method stub
15
16         //Many To
17         String fileName = "JPAs";
18         IStudentService studentService = new StudentService(fileName);
19         Student student = new Student("Lam", "Nguyen", 9);
20         Book book = new Book("Java Persistence with Spring", "Catalin Tudose", "9781617299186");
21         student.getBooks().add(book);
22         studentService.AddStudent(student);
23     }
24 }
25
26
```

20. Run Program



```
eclipse-workspace - HSF301_Chapter_01_Ex_01_ManyToOne/src/main/java/hsf301/fe/gui/ManyToOne.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer Servers
  HSF301_Chapter_01_Ex_01_ManyToOne [HSF301 main]
    JPA Content
      persistence.xml
    src/main/java
      hsf301.fe.dao
      hsf301.fe.gui
        ManyToOne.java
      hsf301.fe.pojo
      hsf301.fe.repository
      hsf301.fe.service
        IStudentService.java
        StudentService.java
    src/main/resources
    JRE System Library [JavaSE-1.8]
    Maven Dependencies
    src
    target
    pom.xml

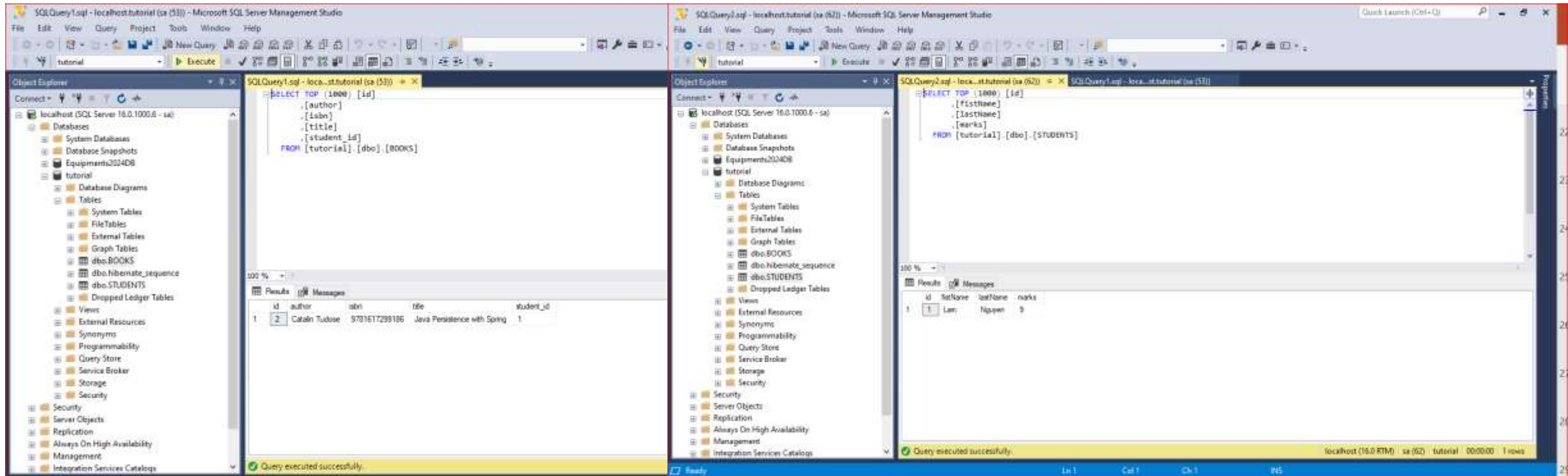
1 package hsf301.fe.gui;
2
3 import hsf301.fe.pojo.Book;
4
5
6
7
8 public class ManyToOne {
9
10     public static void main(String[] args) {
11         // TODO Auto-generated method stub
12
13         //Many To
14         String fileName = "JPAs";
15         IStudentService studentService = new StudentService(fileName);
16         Student student = new Student("Lam", "Nguyen", 9);
17         Book book = new Book("Java Persistence with Spring", "Catalin Tudose", "9781617299186");
18         student.getBooks().add(book);
19         studentService.save(student);
20     }
21 }
22
23

Console Problems Debug Shell
No consoles to display at this time.

Non-externalized string literal; it should be followed by //$NON-NLS-1$
```



22. Result



The screenshot displays two instances of Microsoft SQL Server Management Studio (SSMS) side-by-side, both connected to a local SQL Server instance (sa).

Left Window (SQLQuery1.sql - localhost:tutorial (sa (53))):

- Query:**

```
SELECT TOP (1000) [id]
, [author]
, [isbn]
, [title]
, [student_id]
FROM [tutorial].[dbo].[BOOKS]
```
- Results:**

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

Right Window (SQLQuery2.sql - localhost:tutorial (sa (52))):

- Query:**

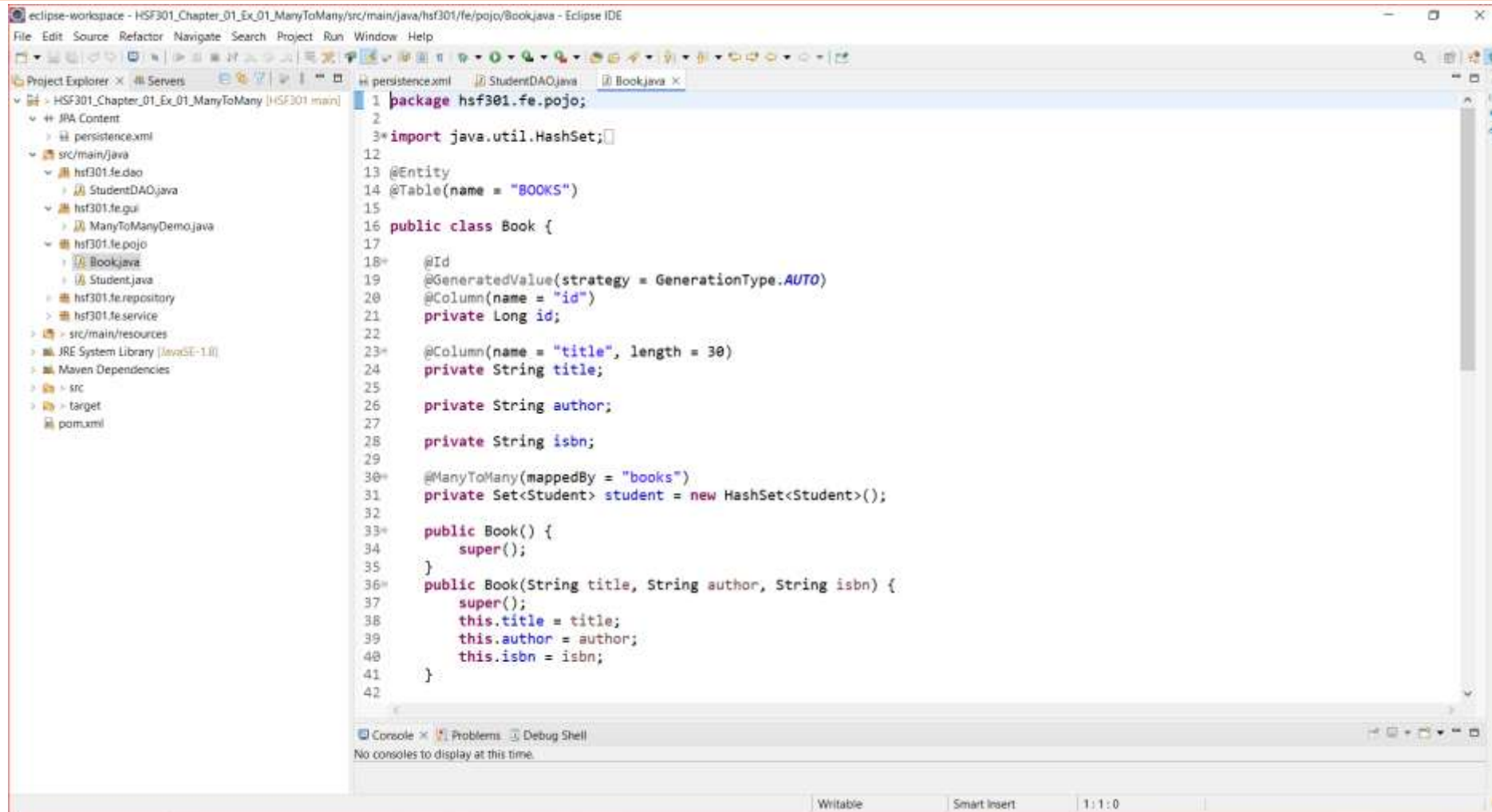
```
SELECT TOP (1000) [id]
, [firstName]
, [lastName]
, [marks]
FROM [tutorial].[dbo].[STUDENTS]
```
- Results:**

id	firstName	lastName	marks
1	Lam	Nguyen	9

Both queries were executed successfully, as indicated by the status bar and the 'Query executed successfully' message in the Results pane.

Demo JPA (Many To Many)

1. Create Books in Pojo's Package



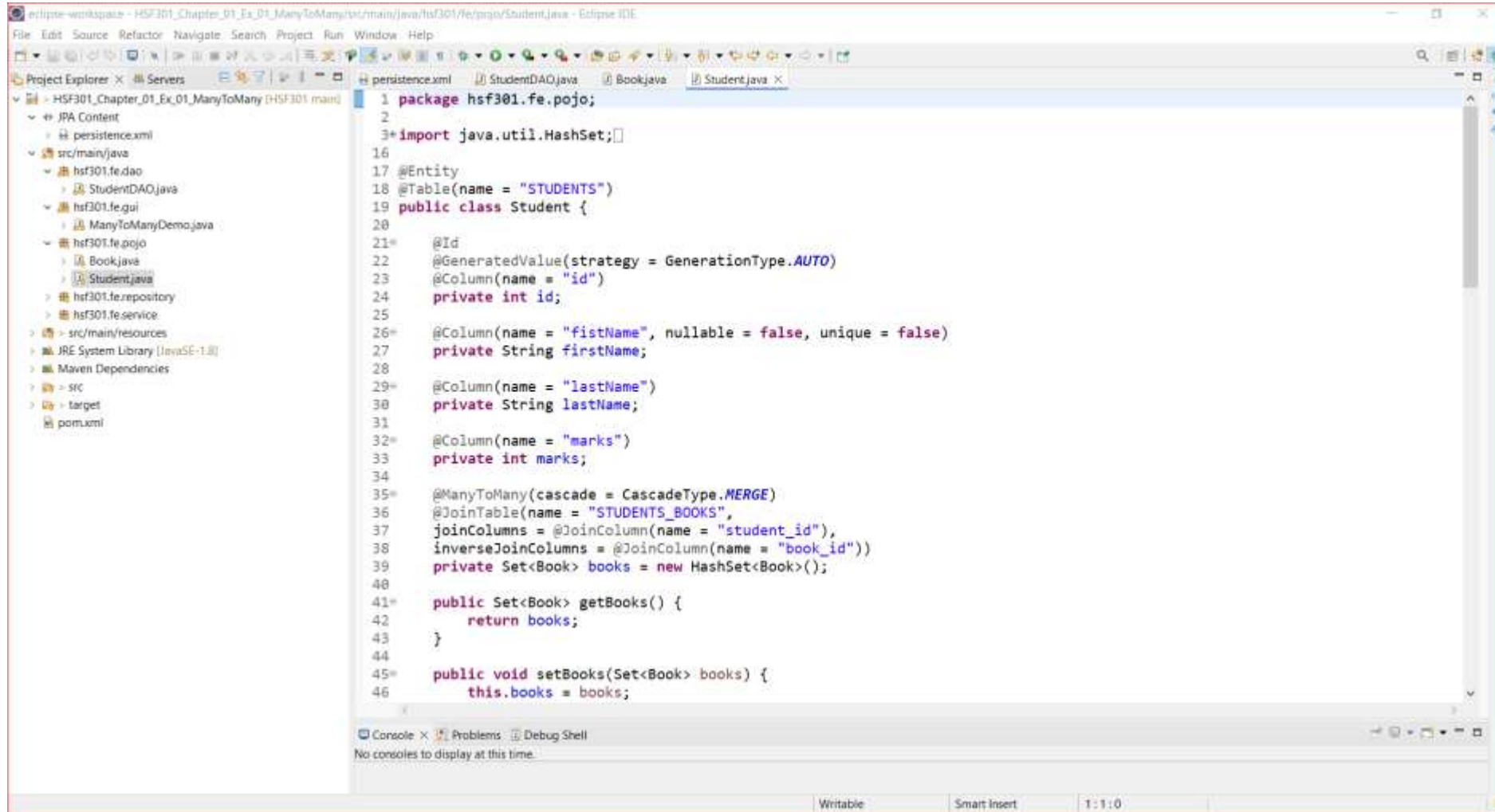
The screenshot shows the Eclipse IDE with the following details:

- Project Explorer:** Shows a project named 'HSF301_Chapter_01_Ex_01_ManyToMany' with a package 'hsf301.fe.pojo' containing 'Book.java'.
- Code Editor:** Displays the content of 'Book.java'.

```
1 package hsf301.fe.pojo;
2
3 import java.util.HashSet;
4
5
6
7
8
9
10
11
12
13 @Entity
14 @Table(name = "BOOKS")
15
16 public class Book {
17
18     @Id
19     @GeneratedValue(strategy = GenerationType.AUTO)
20     @Column(name = "id")
21     private Long id;
22
23     @Column(name = "title", length = 30)
24     private String title;
25
26     private String author;
27
28     private String isbn;
29
30     @ManyToMany(mappedBy = "books")
31     private Set<Student> student = new HashSet<Student>();
32
33     public Book() {
34         super();
35     }
36
37     public Book(String title, String author, String isbn) {
38         super();
39         this.title = title;
40         this.author = author;
41         this.isbn = isbn;
42     }
43 }
```

The bottom of the IDE shows the 'Console' tab with the message: 'No consoles to display at this time.'

2. Create Students in Pojo's Package

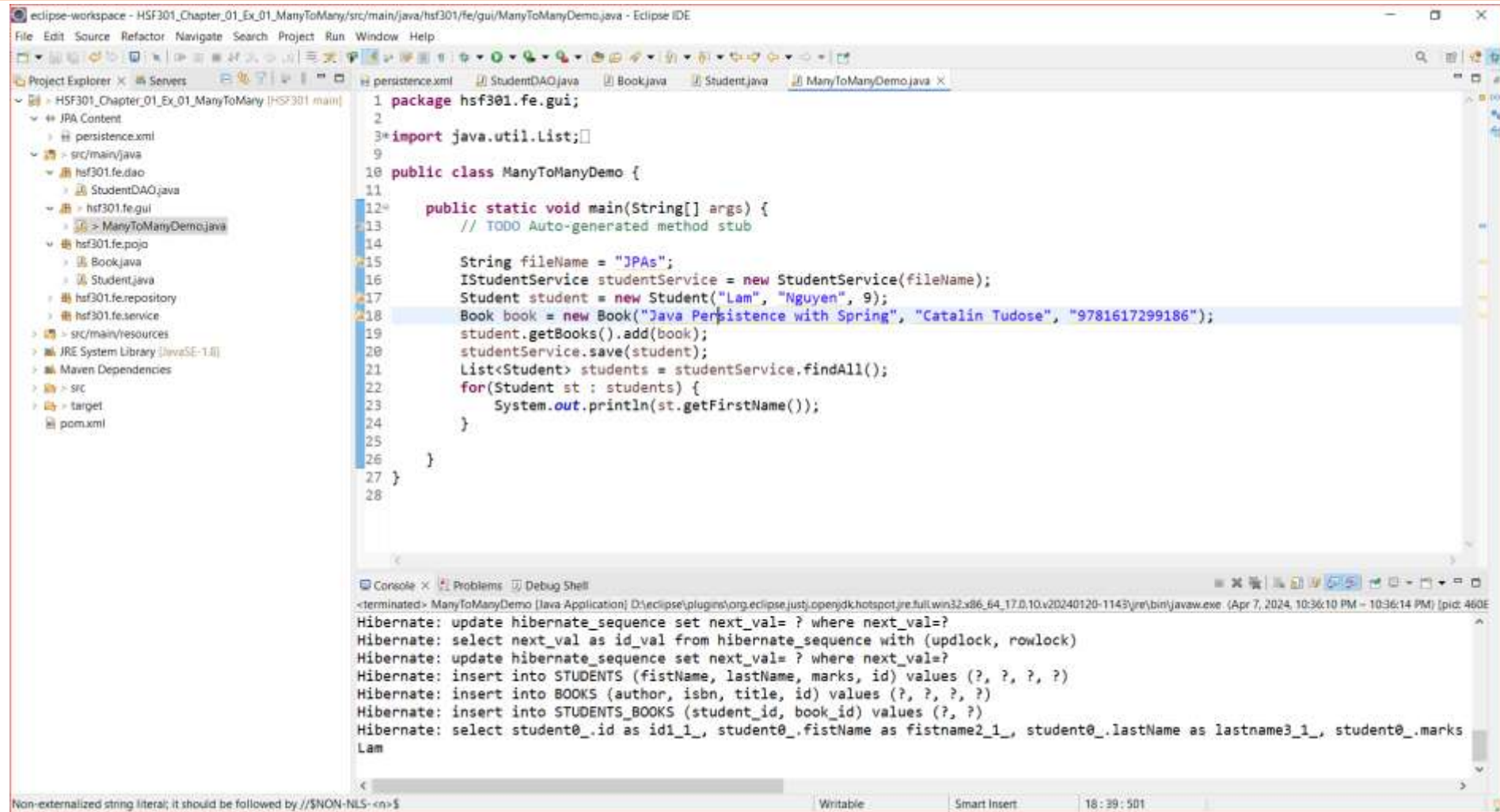


```

1 package hsf301.fe.pojo;
2
3 import java.util.HashSet;
4
5
6
7 @Entity
8 @Table(name = "STUDENTS")
9 public class Student {
10
11     @Id
12     @GeneratedValue(strategy = GenerationType.AUTO)
13     @Column(name = "id")
14     private int id;
15
16     @Column(name = "firstName", nullable = false, unique = false)
17     private String firstName;
18
19     @Column(name = "lastName")
20     private String lastName;
21
22     @Column(name = "marks")
23     private int marks;
24
25     @ManyToMany(cascade = CascadeType.MERGE)
26     @JoinTable(name = "STUDENTS_BOOKS",
27         joinColumns = @JoinColumn(name = "student_id"),
28         inverseJoinColumns = @JoinColumn(name = "book_id"))
29     private Set<Book> books = new HashSet<Book>();
30
31     public Set<Book> getBooks() {
32         return books;
33     }
34
35     public void setBooks(Set<Book> books) {
36         this.books = books;
37     }
38 }

```

3. Run Program



The screenshot shows the Eclipse IDE with the file `ManyToManyDemo.java` open. The code defines a `ManyToManyDemo` class with a `main` method that interacts with a database using Hibernate. The console output shows the execution of the program, including database updates and insertions.

```

1 package hsf301.fe.gui;
2
3 import java.util.List;
4
5
6
7
8
9
10 public class ManyToManyDemo {
11
12     public static void main(String[] args) {
13         // TODO Auto-generated method stub
14
15         String fileName = "JPAs";
16         IStudentService studentService = new StudentService(fileName);
17         Student student = new Student("Lam", "Nguyen", 9);
18         Book book = new Book("Java Persistence with Spring", "Catalin Tudose", "9781617299186");
19         student.getBooks().add(book);
20         studentService.save(student);
21         List<Student> students = studentService.findAll();
22         for(Student st : students) {
23             System.out.println(st.getFirstName());
24         }
25     }
26 }
27
28

```

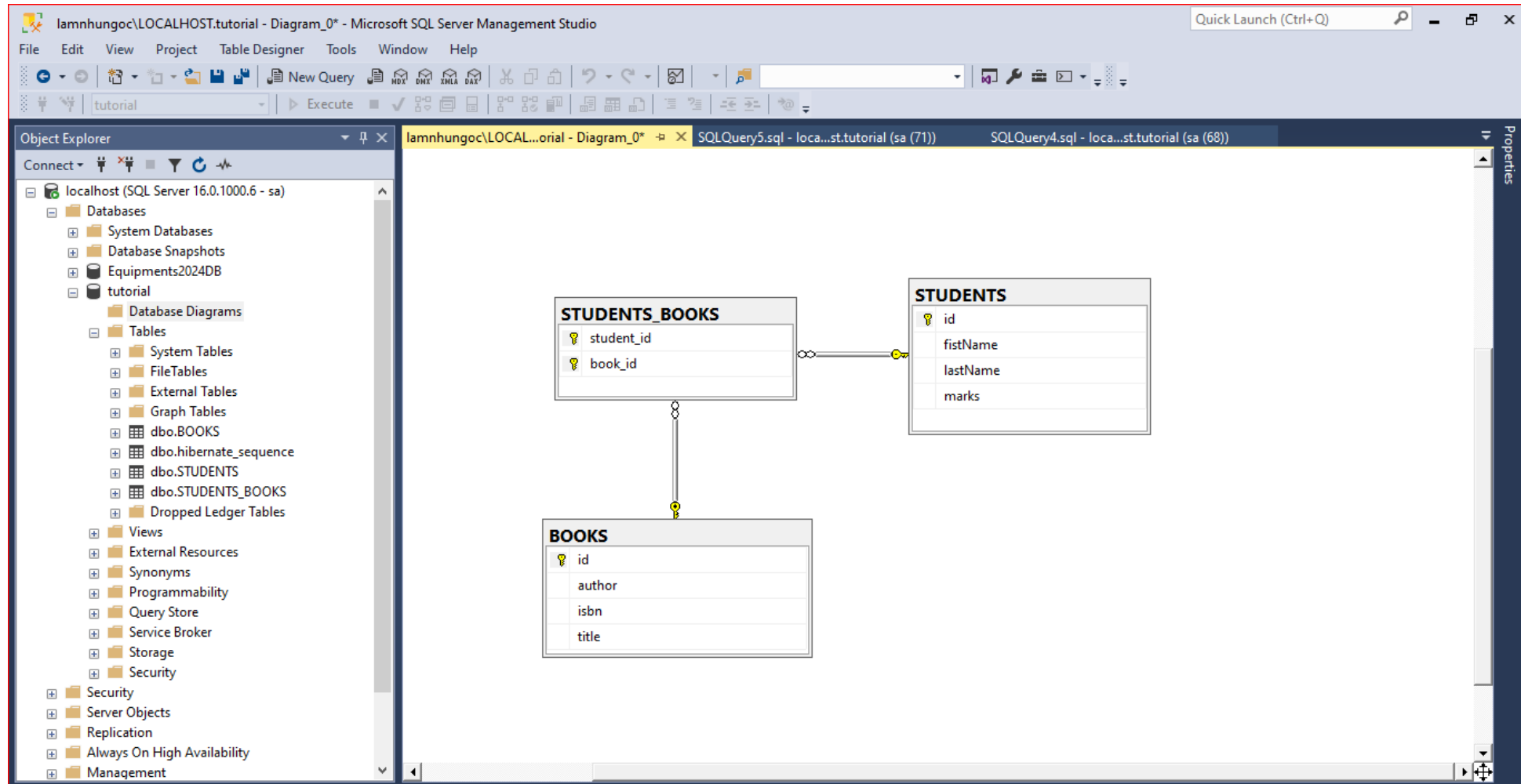
```

<terminated> ManyToManyDemo [Java Application] D:\eclipse\plugins\org.eclipse.just.openjdk.hotspot.jre.full.win32.x86_64.17.0.10.v20240120-1143\jre\bin\javaw.exe. (Apr 7, 2024, 10:36:14 PM) [pid: 460E]
Hibernate: update hibernate_sequence set next_val= ? where next_val=?
Hibernate: select next_val as id_val from hibernate_sequence with (updlock, rowlock)
Hibernate: update hibernate_sequence set next_val= ? where next_val=?
Hibernate: insert into STUDENTS (firstName, lastName, marks, id) values (?, ?, ?, ?)
Hibernate: insert into BOOKS (author, isbn, title, id) values (?, ?, ?, ?)
Hibernate: insert into STUDENTS_BOOKS (student_id, book_id) values (?, ?)
Hibernate: select student0_.id as id1_1_, student0_.firstName as firstname2_1_, student0_.lastName as lastname3_1_, student0_.marks
Lam

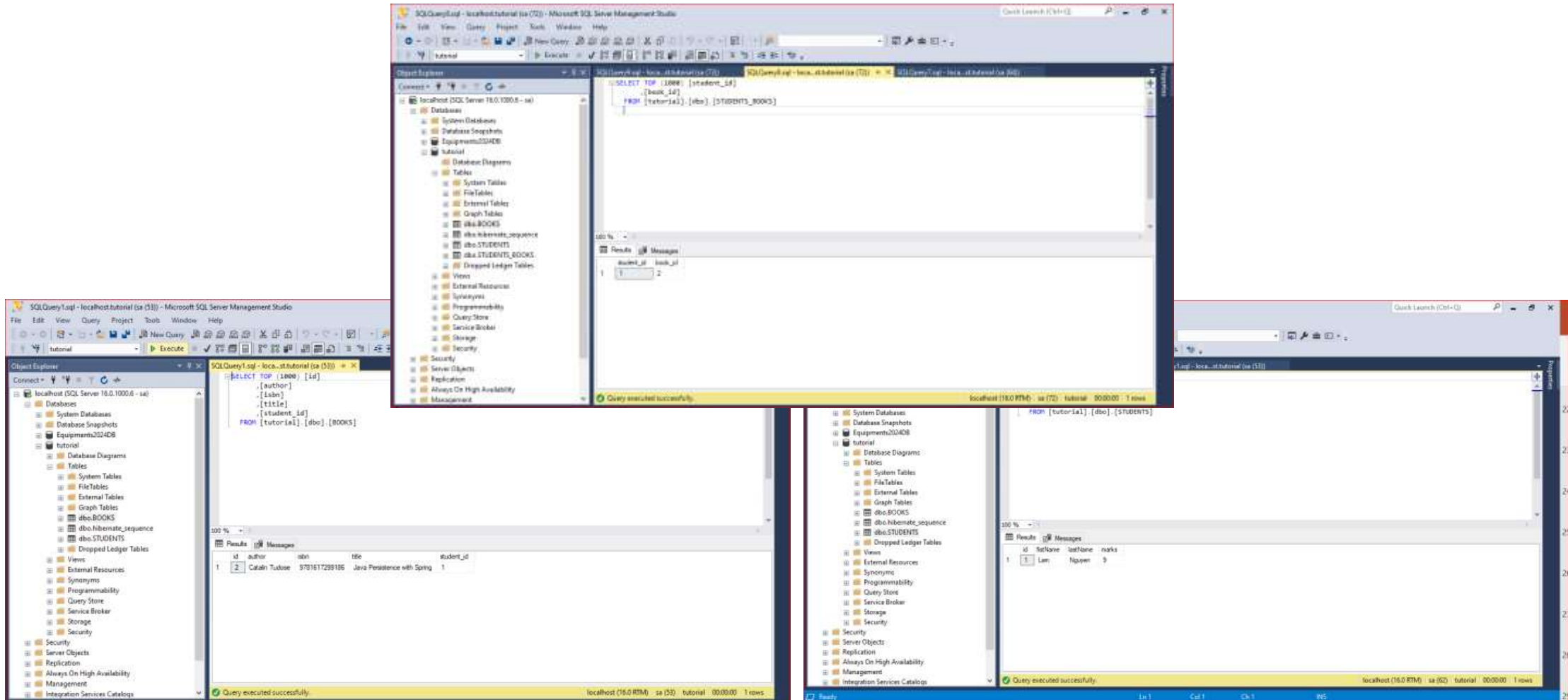
```

Non-externalized string literal; it should be followed by //\$NON-NLS-1\$

4. Result



5. Result



The image displays three screenshots of the Microsoft SQL Server Enterprise Manager interface, showing the execution of SQL queries and the resulting data.

Top Screenshot: Shows a query window with the following SQL code:

```
SELECT TOP (1000) [student_id]
FROM [tutorial].[dbo].[STUDENTS_BOOKS]
```

The results pane shows a single row of data:

student_id	book_id
1	2

Bottom Left Screenshot: Shows a query window with the following SQL code:

```
SELECT TOP (1000) [id]
FROM [tutorial].[dbo].[BOOKS]
```

The results pane shows a single row of data:

id	author	isbn	title	student_id
1	Catalin Tudone	9781617299106	Java Persistence with Spring	1

Bottom Right Screenshot: Shows a query window with the following SQL code:

```
FROM [tutorial].[dbo].[STUDENTS]
```

The results pane shows a single row of data:

id	firstName	lastName	marks
1	Lam	Nguyen	9

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