FPT Education
FPT UNIVERSITY

# Hibernate Architecture, Configuration

## Objectives

- **Understand ORM:** Grasp the core concepts and benefits of object-relational mapping.
- **Map Entities:** Efficiently map Java classes to database tables and manage relationships.
- **Control Transactions:** Ensure data integrity with transactions and concurrecy handling.
- **Integrate with Spring:** Leverage Spring Framework for dependency injection and simplified data access.
- **Gain Practical Experience:** Build projects and experiment with advanced features.

## Contents

## What is Hibernate ?

- Hibernate is a powerful object-relational mapping (ORM) framework for the Java programming language.
- It acts as a bridge between the object-oriented world of Java and the relational world of databases, making it easier for developers to work with persistent data.
- Hibernate simplifies Java persistence, allowing developers to focus on the business logic of their applications rather than the intricacies of database interactions.

## Key Features of Hibernate

- **ORM:** Maps Java objects to relational database tables, simplifying data access.
- **JPA Implementation:** Adheres to the JPA standard, ensuring portability and flexibility.
- **HQL (Hibernate Query Language):** Powerful object-oriented query language for retrieving and manipulating data.
- **Lazy Loading:** Loads associated data on demand, minimizing data transfer and enhancing responsiveness.
- **Transaction Management:** Ensures data consistency and integrity through transaction support.
- **Inheritance Mapping:** Handles various inheritance scenarios, mapping Java class hierarchies to database tables.

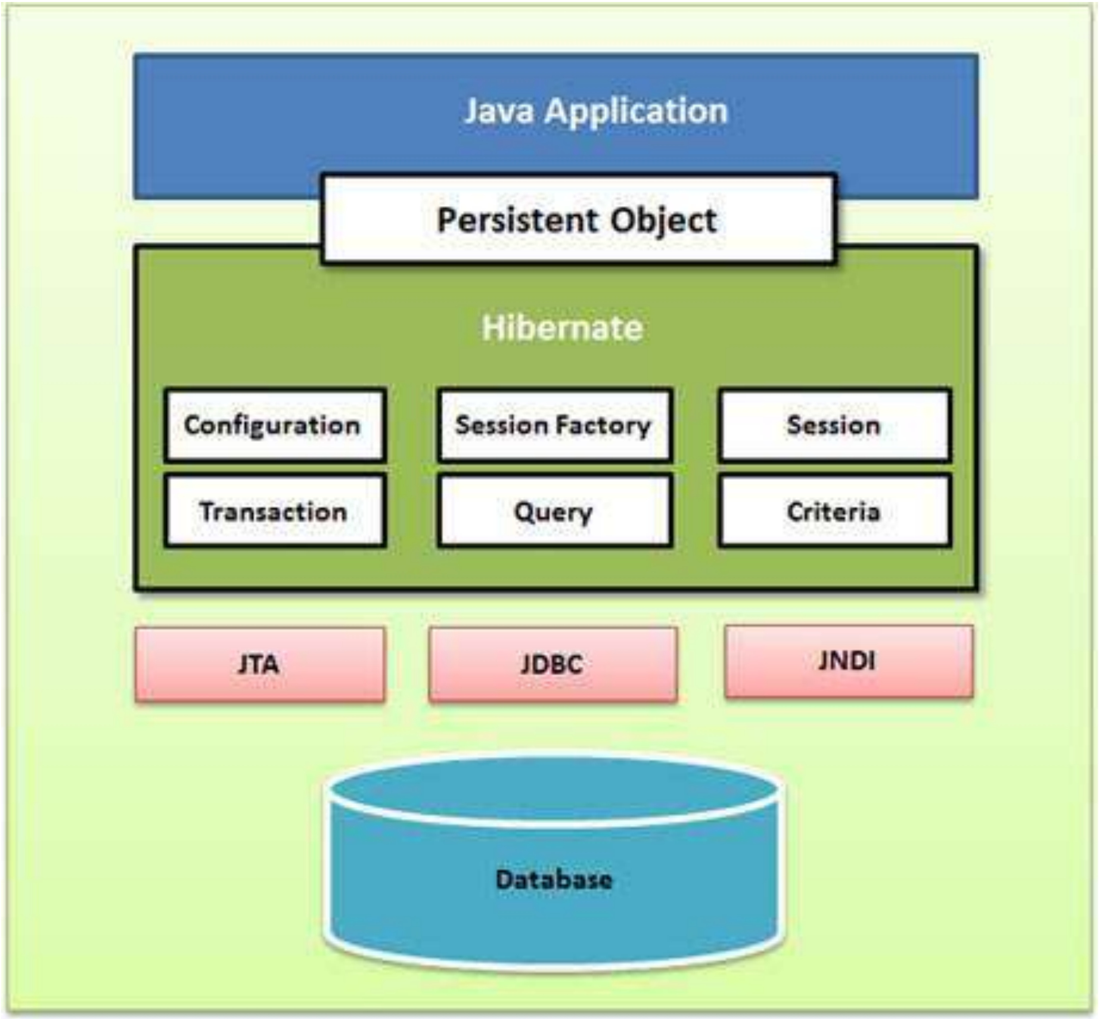
## Benefits of Using Hibernate

- **Faster Development:** Spend less time on database code, more time on logic features
- **Cleaner Code:** Write concise, object-oriented code instead of complex SQL.
- **Improved Maintainability:** Easier to understand, update, and refactor your cod ebase.
- **Database Flexibility:** Switch between different databases without major code changes.

# Benefits of Using Hibernate

- **Performance Boost:** Caching and lazy loading optimize data access for faster applications.
- **Data Integrity:** Built-in mechanisms ensure data consistency and prevent errors.
- **Industry Standard:** Widely used in the Java ecosystem, making you job-ready.
- **Spring Integration:** Works seamlessly with Spring, the leading Java framework .

## Hibernate Architecture



# Hibernate Architecture

- **Session Factory:** Manages configuration and creates Sessions
- **Session:** Provides data access methods and interacts with persistent object
- **Persistent Objects:** Java objects representing data stored in the database.
- **Transaction Management:** Ensures data consistency and integrity.
- **Connection Provider:** Handles database connections and pooling.
- **Query API:** Supports HQL, Criteria API, and native SQL for flexible querying
- **Caching:** Optimizes performance with first-level and second-level caches.
- **Event System:** Allows customization of persistence lifecycle events.
- **Dialects:** Generates database-specific SQL for portability.

FPT Education
FPT UNIVERSITY

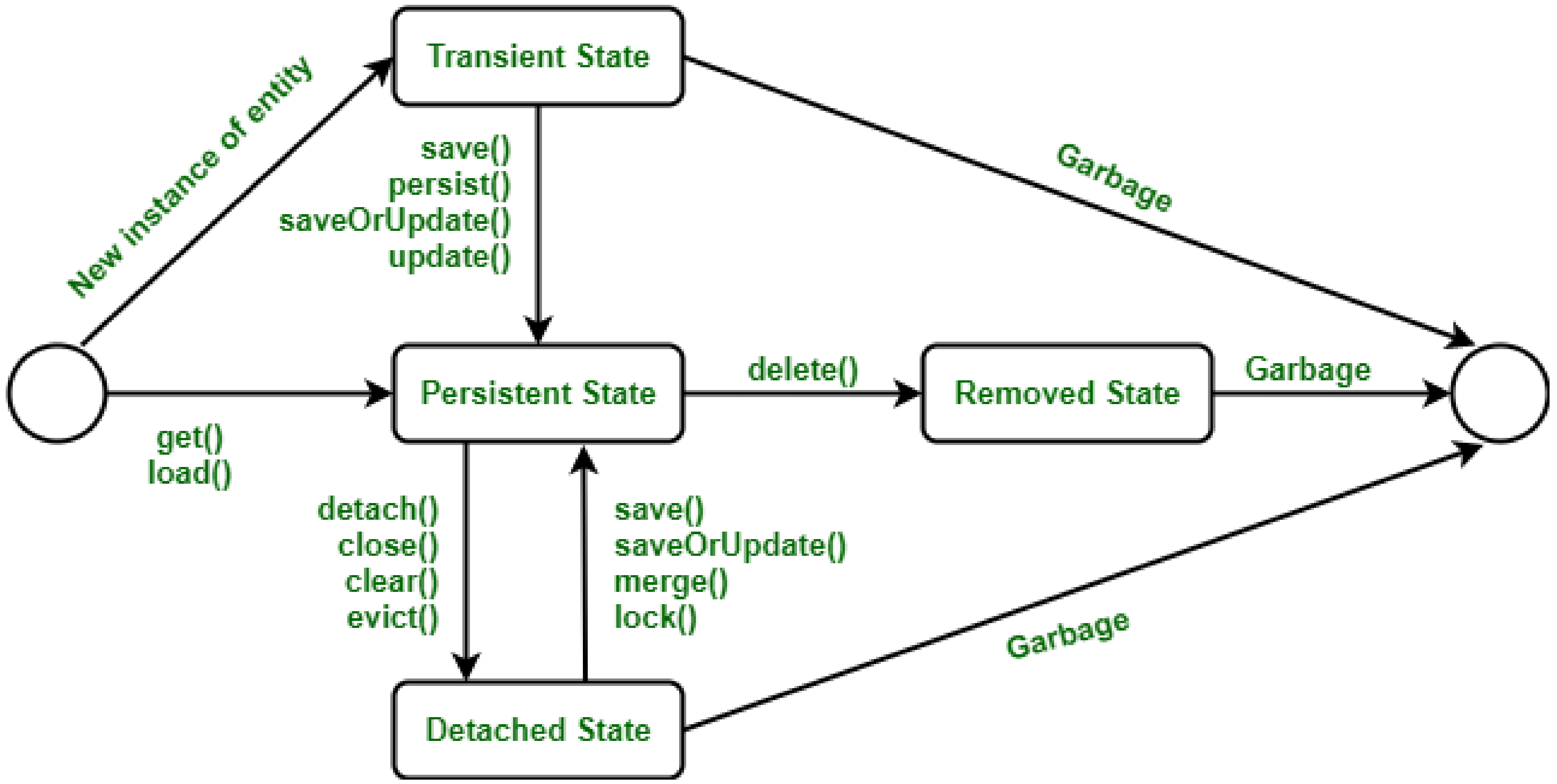
## Hibernate Architecture

- **SessionFactory**
  - The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data.
  - The **org.hibernate.SessionFactory** interface provides factory method to get the object of Session.
- **Session**
  - The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data.
  - The **org.hibernate.Session** interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

## Hibernate Architecture

- **Transaction**
  - The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.
- **ConnectionProvider**
  - It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.
- **TransactionFactory**
  - It is a factory of Transaction. It is optional.

# Hibernate Lifecycle



# Hibernate Lifecycle

- The Hibernate lifecycle refers to the various states an entity instance goes through during its interaction with the persistence framework. Understanding this lifecycle is crucial for effectively managing data and ensuring data integrity in your applications.

- Understanding the Hibernate lifecycle is essential for writing efficient and reliable persistence code. By managing entity states and transitions effectively, you can ensure data integrity and optimize the performance of your applications.

# Hibernate Lifecycle

## 1. Transient State

- An entity instance is in a transient state when it's newly created using the new operator and not yet associated with a Hibernate session.
- Changes made to a transient instance are not tracked by Hibernate and won't be persisted to the database.

## Hibernate Lifecycle

**2. Persistent State**

- An entity instance transitions to the persistent state when it's associated with a Hibernate session. This can happen through:
  - **persist() method:** Explicitly makes an instance persistent.
  - **Cascading:** If an associated entity is persisted and cascading is enabled, the current instance becomes persistent as well.
  - **Querying:** When an entity is retrieved from the database using find(), createQuery (), or other query methods.
- Changes made to a persistent instance are tracked by Hibernate and will be synchro nized with the database upon flushing or transaction commit.

## Hibernate Lifecycle

### 3. Detached State

- An entity instance becomes detached when it's no longer associated with a Hibernate session. This can occur when:
  - **Session is closed:** Closing the session detaches all persistent instances as sociated with it.
  - **detach() method:** Explicitly detaches an instance from the session.
  - **Serialization:** Serializing a persistent instance detaches it.
- Changes made to a detached instance are not tracked by Hibernate and won't be automatically persisted.

## Hibernate Lifecycle

### 4. Removed State

- An entity instance enters the removed state when it's marked for deletion using the remove() method.
- The actual deletion from the database occurs upon flushing or transaction commit.

# JPA vs Hibernate

| JPA | Hibernate |
|---|---|
| Java Persistence API (JP.A) defines the management of relational data in the Java applications. | Hibernate is an Object-Relational Mapping (ORM) tool which is used to save the state of Java object into the database. |
| It is just a specification. Various ORM tools implement it for data persistence. | It is one of the most frequently used JPA implementation. |
| It is defined in **javax.persistence** package. | It is defined in **org.hibernate** package. |
| The **EntityManagerFactory** interface is used to interact with the entity manager factory for the persistence unit. Thus, it provides an entity manager. | It uses **SessionFactory** interface to create Session instances. |
| It uses **EntityManager** interface to create, read, and delete operations for instances of mapped entity classes. This interface interacts with the persistence context. | It uses **Session** interface to create, read, and delete operations for instances of mapped entity classes. It behaves as a runtime interface between a Java application and Hibernate. |
| It uses **Java Persistence Query Language (JPQL)** as an object-oriented query language to perform database operations. | It uses **Hibernate Query Language (HQL)** as an object-oriented query language to perform database operations. |

## Hibernate Configuration

- As Hibernate can operate in different environments, it requires a wide range of configuration parameters. These configurations contain the mapping information that provides different functionalities to Java classes. Generally, we provide database related mappings in the configuration file. Hibernate facilitates to provide the configurations either in an XML file (like hibernate.cfg.xml) or properties file (like hibernate.properties).

- An instance of Configuration class allows specifying properties and mappings to applications. This class also builds an immutable **SessionFactory**.

# Properties of Hibernate Configuration

## Hibernate JDBC Properties

| Property | Description |
|---|---|
| hibernate.connection.driver_class | It represents the JDBC driver class. |
| hibernate.connection.url | It represents the JDBC URL. |
| hibernate.connection.username | It represents the database username. |
| hibernate.connection.password | It represents the database password. |
| Hibernate.connection.pool_size | It represents the maximum number of connections available in the connection pool. |

## Properties of Hibernate Configuration

- **Hibernate Datasource Properties**

| **Property** | **Description** |
|---|---|
| hibernate.connection.datasource | It represents datasource JNDI name which is used by Hibernate for database properties. |
| hibernate.jndi.url | It is optional. It represents the URL of the JNDI provider. |
| hibernate.jndi.class | It is optional. It represents the class of the JNDI InitialContextFactory. |

## Properties of Hibernate Configuration

| Property | Description |
|---|---|
| hibernate.dialect | It represents the type of database used in hibernate to generate SQL statements for a particular relational database. |
| hibernate.show_sql | It is used to display the executed SQL statements to console. |
| hibernate.format_sql | It is used to print the SQL in the log and console. |
| hibernate.default_catalog | It qualifies unqualified table names with the given catalog in generated SQL. |
| hibernate.default_schema | It qualifies unqualified table names with the given schema in generated SQL. |
| hibernate.session_factory_name | The SessionFactory interface automatically bound to this name in JNDI after it has been created. |

# Properties of Hibernate Configuration

| Property | Description |
|---|---|
| hibernate.default_entity_mode | It sets a default mode for entity representation for all sessions opened from this SessionFactory |
| hibernate.order_updates | It orders SQL updates on the basis of the updated primary key. |
| hibernate.use_identifier_rollback | If enabled, the generated identifier properties will be reset to default values when objects are deleted. |
| hibernate.generate_statistics | If enabled, the Hibernate will collect statistics useful for performance tuning. |
| hibernate.use_sql_comments | If enabled, the Hibernate generate comments inside the SQL. It is used to make debugging easier. |

## Hibernate Cache Properties

| Property | Description |
|---|---|
| hibernate.cache.provider_class | It represents the classname of a custom CacheProvider. |
| hibernate.cache.use_minimal_puts | It is used to optimize the second-level cache. It minimizes writes, at the cost of more frequent reads. |
| hibernate.cache.use_query_cache | It is used to enable the query cache. |
| hibernate.cache.use_second_level_cache | It is used to disable the second-level cache, which is enabled by default for classes which specify a mapping. |
| hibernate.cache.query_cache_factory | It represents the classname of a custom QueryCache interface. |
| hibernate.cache.region_prefix | It specifies the prefix which is used for second-level cache region names. |
| hibernate.cache.use_structured_entries | It facilitates Hibernate to store data in the second-level cache in a more human-friendly format. |

# Properties of Hibernate Configuration

- Hibernate Transaction Properties

| Property | Description |
|---|---|
| hibernate.transaction.factory_class | It represents the classname of a TransactionFactory which is used with Hibernate Transaction API. |
| hibernate.transaction.manager_lookup_class | It represents the classname of a TransactionManagerLookup. It is required when JVM-level caching is enabled. |
| hibernate.transaction.flush_before_completion | If it is enabled, the session will be automatically flushed during the before completion phase of the transaction. |
| hibernate.transaction.auto_close_session | If it is enabled, the session will be automatically closed during the after completion phase of the transaction. |

# Properties of Hibernate Configuration

- Other Hibernate Properties

| Property | Description |
|---|---|
| hibernate.connection.provider_class | It represents the classname of a custom ConnectionProvider which provides JDBC connections to Hibernate. |
| hibernate.connection.isolation | It is used to set the JDBC transaction isolation level. |
| hibernate.connection.autocommit | It enables auto-commit for JDBC pooled connections. However, it is not recommended. |
| hibernate.connection.release_mode | It specifies when Hibernate should release JDBC connections. |
| hibernate.current_session_context_class | It provides a custom strategy for the scoping of the "current" Session. |
| hibernate.hbm2ddl.auto | It automatically generates a schema in the database with the creation of SessionFactory. |

FPT Education
FPT UNIVERSITY

Java

## Annotations in Hibernate

## Commonly Used Annotations

- **@Entity:** Marks a class as a persistent entity, indicating that it represents data stored in a database table.
- **@Table:** Specifies the name of the database table to which the entity is mapped.
- **@Id:** Identifies the primary key property of the entity.
- **@GeneratedValue:** Configures the strategy for generating identifier values (e.g., AUTO, SEQUENCE, IDENTITY).
- **@Column:** Provides details about the mapping of a property to a database column, s uch as the column name, data type, and nullability.

# Commonly Used Annotations

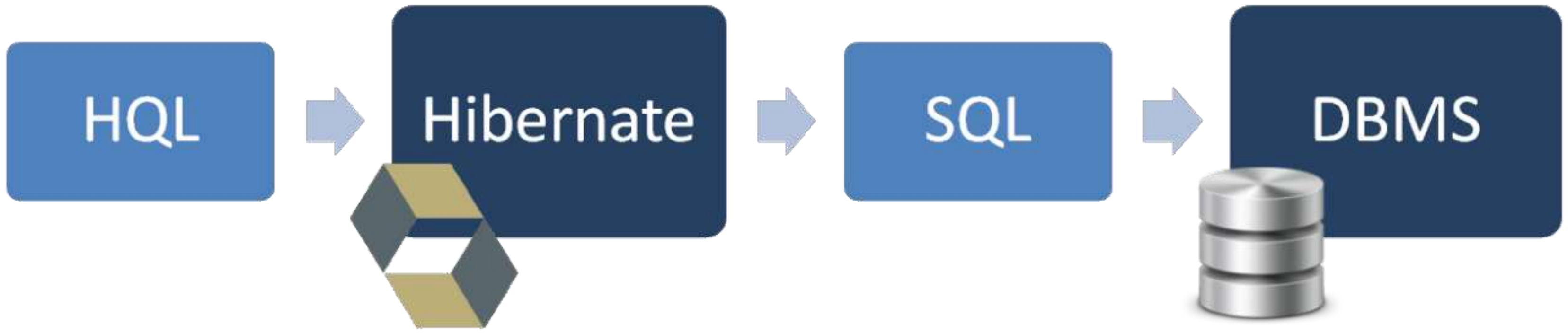
- **@Basic:** Marks a property as a basic type (e.g., String, int, Date).
- **@Transient:** Excludes a property from persistence.
- **@Embedded:** Map embeddable components as value types.
- **@Temporal:** Specifies the temporal precision of a date/time property.
- **@Enumerated:** Configures the mapping of an enum type.
- **@Lob:** Marks a property as a large object (BLOB or CLOB).
- **@Version:** Enables optimistic locking with a version property.
- **@CreationTimestamp, @UpdateTimestamp:** Automatically set timestamps for creation and update events.

## Relationships Annotations in Hibernate

- **@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.

# Hibernate Query Language (HQL)

- Hibernate Query Language (HQL) is same as SQL (Structured Query Language) but it doesn't depends on the table of the database. Instead of table name, we use class name in HQL. So it is database independent query language.



## Advantage of HQL

- Database independent
- Supports polymorphic queries
- Easy to learn for Java Programmer

## Query Interface

- It is an object oriented representation of Hibernate Query. The object of Query can be obtained by calling the createQuery() method Session interface.
  - **public int executeUpdate()** is used to execute the update or delete query.
  - **public List list()** returns the result of the ralation as a list.
  - **public Query setFirstResult(int rowno)** specifies the row number from where record will be retrieved.

## Query Interface

- **public Query setMaxResult(int rowno)** specifies the no. of records to be retrieved from the relation (table).
- **public Query setParameter(int position, Object value)** it sets the value to the JDBC style query parameter.
- **public Query setParameter(String name, Object value)** it sets the value to a named query parameter.

## Example of HQL paging

- Query query=session.createQuery("from Student");
- query.setFirstResult(5);
- query.setMaxResult(10);
- List list=query.list();//will return the records from 5 to 10th number

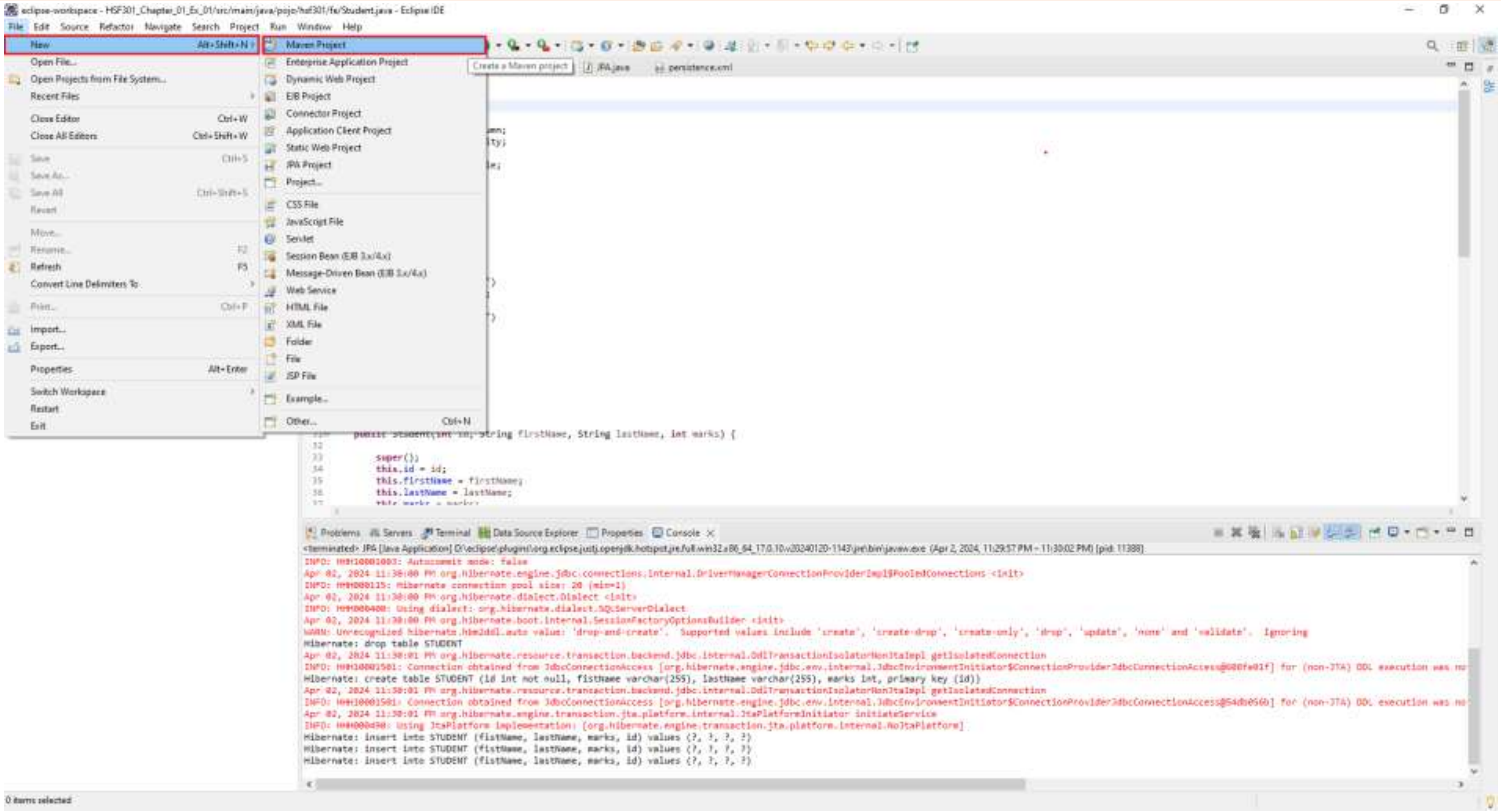
# Example of HQL update query

1. *Transaction tx=session.beginTransaction();*
2. *Query q=session.createQuery("update Student set lastName=:n where id=:i");*
3. *q.setParameter("n",“Sang");*
4. *q.setParameter("i",1);*
5. ***int** status=q.executeUpdate();*
6. *System.out.println(status);*
7. *tx.commit();*

FPT Education
FPT UNIVERSITY

## Demo Hibernate (One To Many)

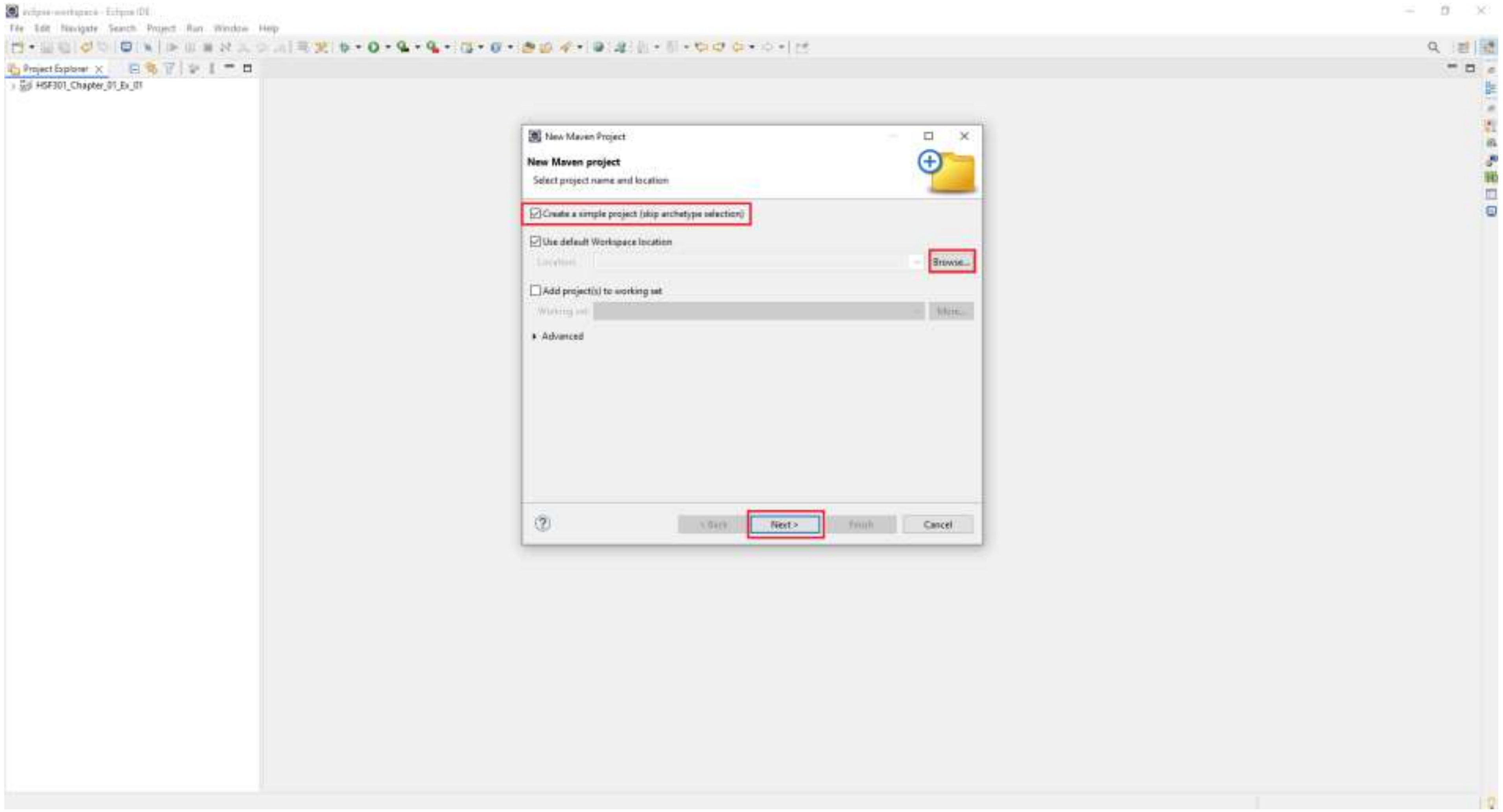
# 1. Open Eclipse, File | New | Maven Project



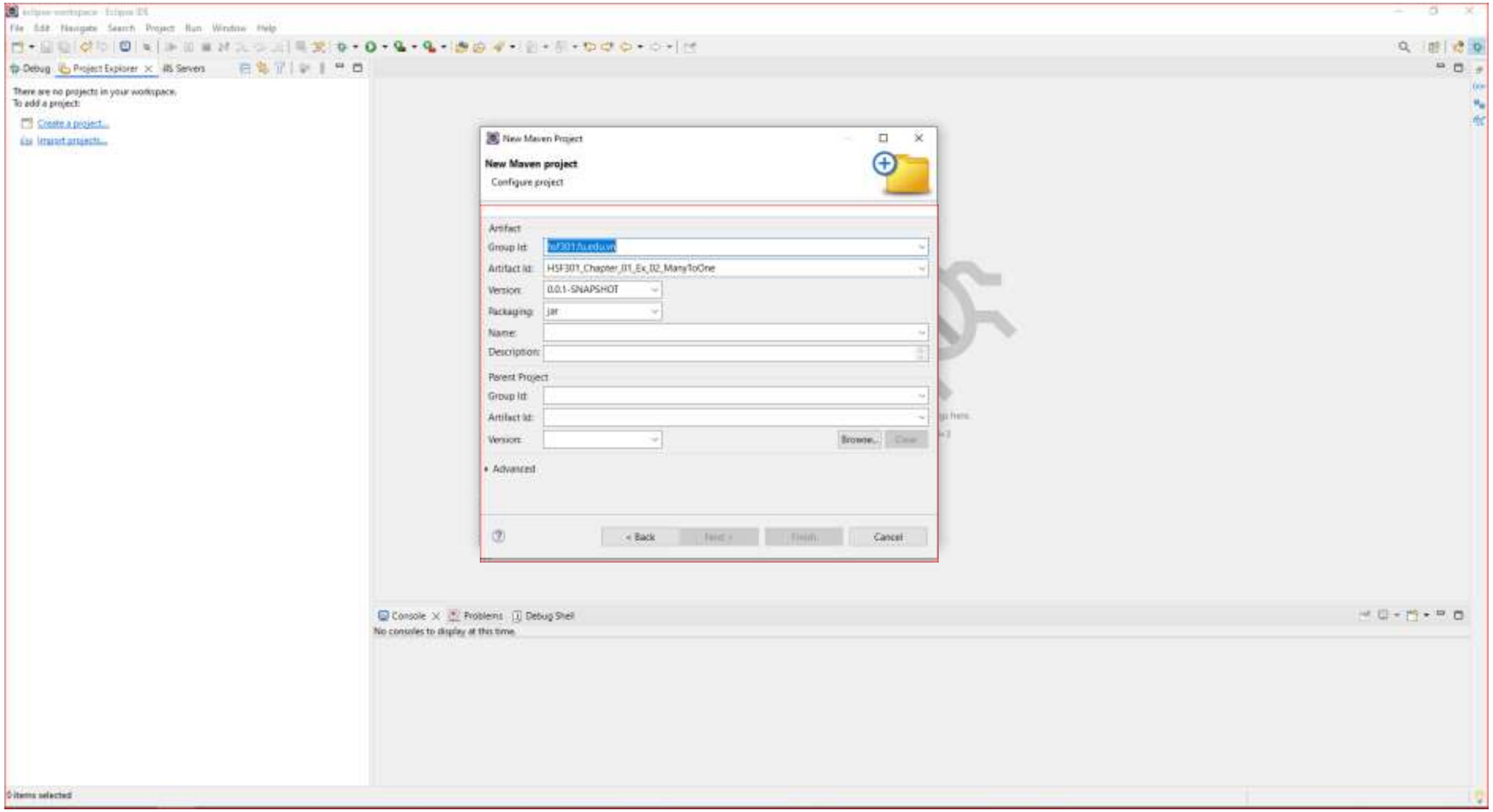
```
    public Student(int id, String firstName, String lastName, int marks) {

        super();
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.marks = marks;
```

```
<terminated> JPA [Java Application] D:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (Apr 2, 2024, 11:29:57 PM – 11:30:02 PM) [pid: 11388]
INFO: HHH10001003: Autocommit mode: false
Apr 02, 2024 11:30:00 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
Apr 02, 2024 11:30:00 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.SQLServerDialect
Apr 02, 2024 11:30:00 PM org.hibernate.boot.internal.SessionFactoryOptionsBuilder <init>
WARN: Unrecognized hibernate.hbm2ddl.auto value: 'drop-and-create'.  Supported values include 'create', 'create-drop', 'create-only', 'drop', 'update', 'none' and 'validate'.  Ignoring
Hibernate: drop table STUDENT
Apr 02, 2024 11:30:01 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@600fe01f] for (non-JTA) DDL execution was no
Hibernate: create table STUDENT (id int not null, fistName varchar(255), lastName varchar(255), marks int, primary key (id))
Apr 02, 2024 11:30:01 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@54db056b] for (non-JTA) DDL execution was no
Apr 02, 2024 11:30:01 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
Hibernate: insert into STUDENT (fistName, lastName, marks, id) values (?, ?, ?, ?)
Hibernate: insert into STUDENT (fistName, lastName, marks, id) values (?, ?, ?, ?)
Hibernate: insert into STUDENT (fistName, lastName, marks, id) values (?, ?, ?, ?)
```

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


eclipse-workspace - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Project Explorer
HSF301_Chapter_01_Ex_01
New Maven Project
New Maven project
Select project name and location
Create a simple project (skip archetype selection)
Use default Workspace location
Location:
Browse...
Add project(s) to working set
Working set:
More...
Advanced
< Back
Next >
Finish
Cancel

## 3. Fill the information Project -> Click Finish


New Maven Project

**New Maven project**

Configure project

Artifact

Group Id: hsf301.fu.edu.vn

Artifact Id: HSF301_Chapter_01_Ex_02_ManyToOne

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name:

Description:

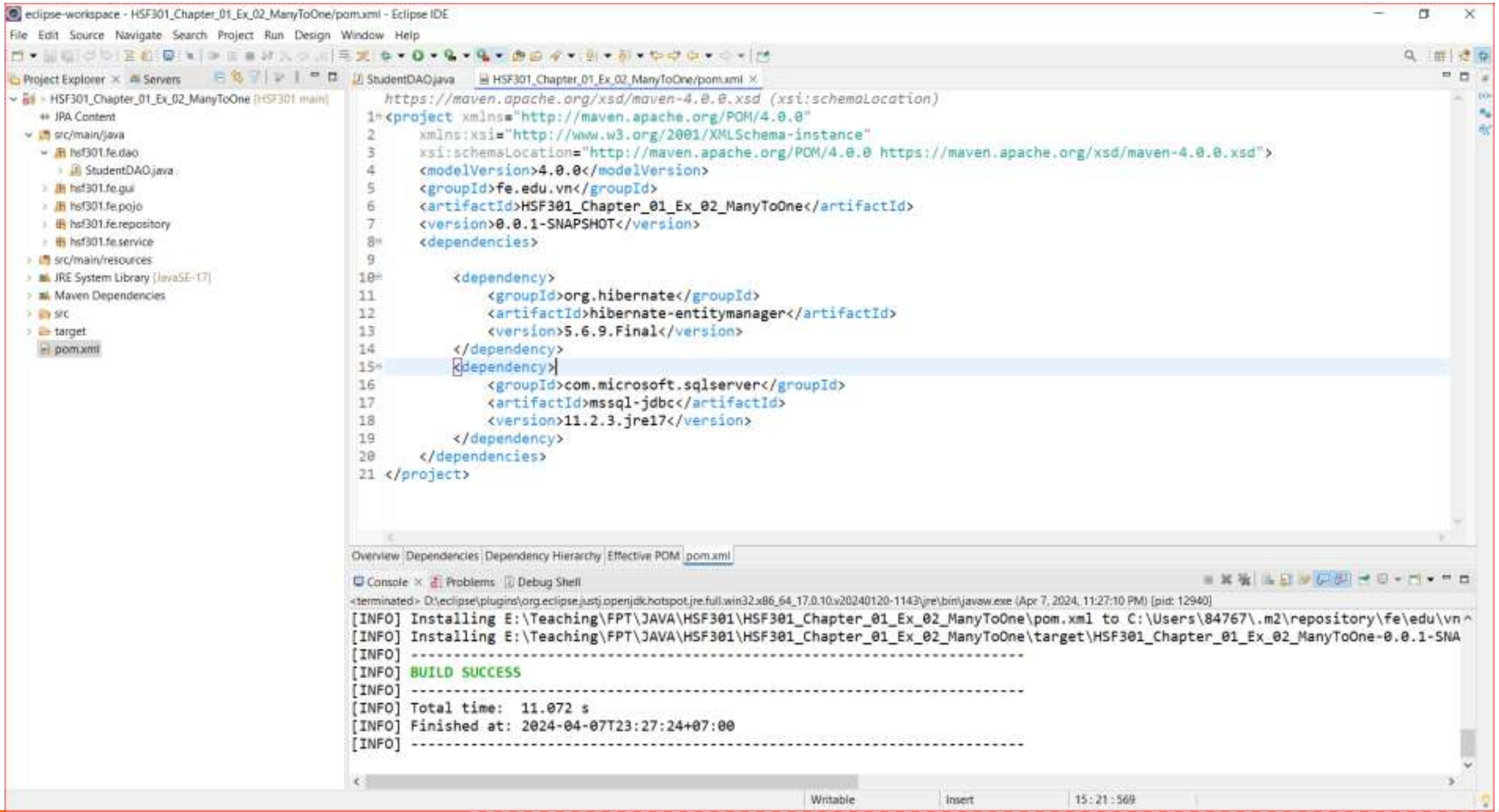
Parent Project

Group Id:

Artifact Id:

Version:

## 4. Structure of Maven Project



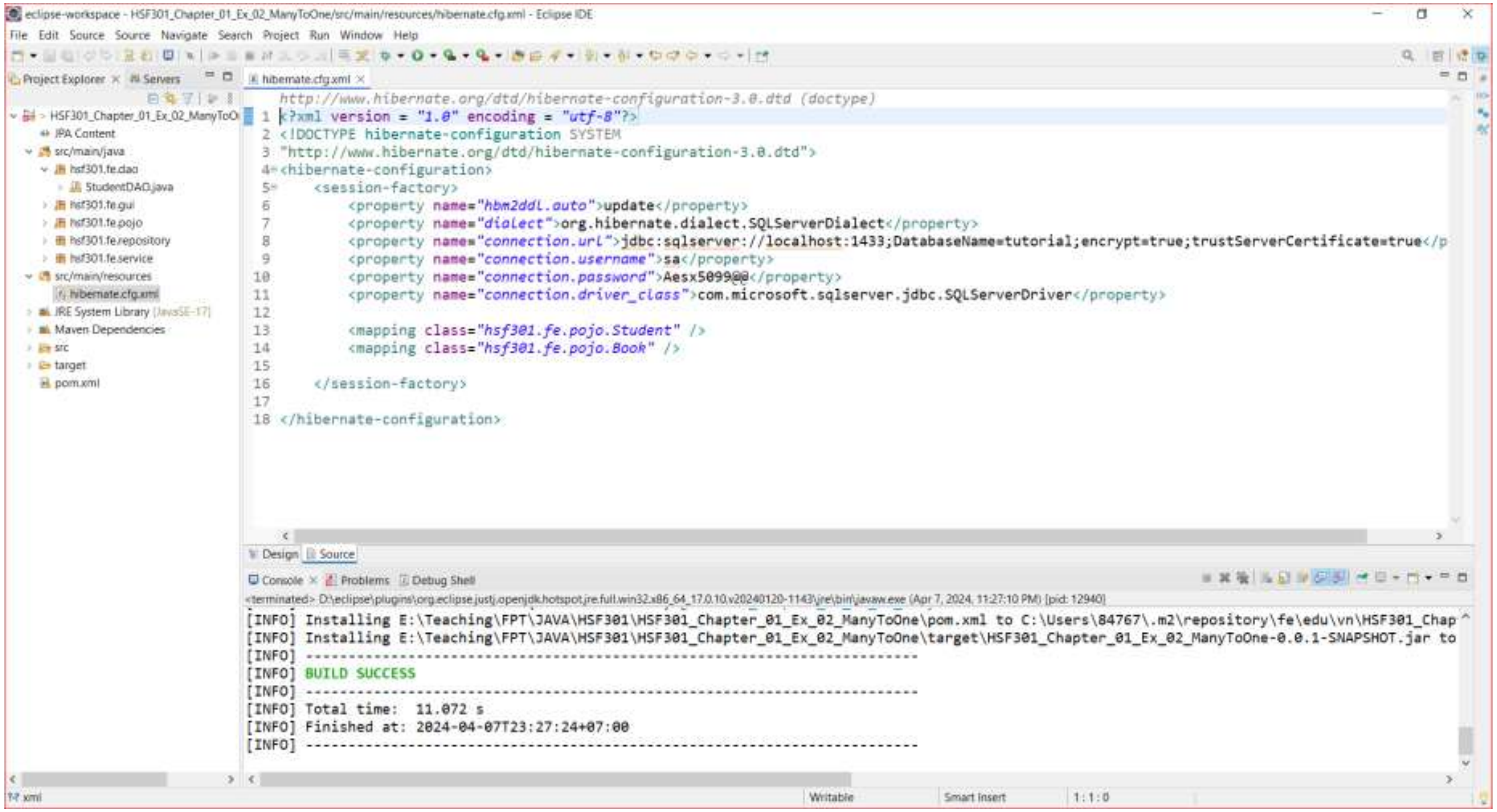
eclipse-workspace - HSF301_Chapter_01_Ex_02_ManyToOne/pom.xml - Eclipse IDE

```
HSF301_Chapter_01_Ex_02_ManyToOne [HSF301 main]
    JPA Content
  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-17]
  Maven Dependencies
  src
  target
  pom.xml
```

```xml
https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>fe.edu.vn</groupId>
    <artifactId>HSF301_Chapter_01_Ex_02_ManyToOne</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <dependencies>

        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-entitymanager</artifactId>
            <version>5.6.9.Final</version>
        </dependency>
        <dependency>
            <groupId>com.microsoft.sqlserver</groupId>
            <artifactId>mssql-jdbc</artifactId>
            <version>11.2.3.jre17</version>
        </dependency>
    </dependencies>
</project>
```

```
<terminated> D:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (Apr 7, 2024, 11:27:10 PM) [pid: 12940]
[INFO] Installing E:\Teaching\FPT\JAVA\HSF301\HSF301_Chapter_01_Ex_02_ManyToOne\pom.xml to C:\Users\84767\.m2\repository\fe\edu\vn
[INFO] Installing E:\Teaching\FPT\JAVA\HSF301\HSF301_Chapter_01_Ex_02_ManyToOne\target\HSF301_Chapter_01_Ex_02_ManyToOne-0.0.1-SNA
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  11.072 s
[INFO] Finished at: 2024-04-07T23:27:24+07:00
[INFO] ------------------------------------------------------------------------
```

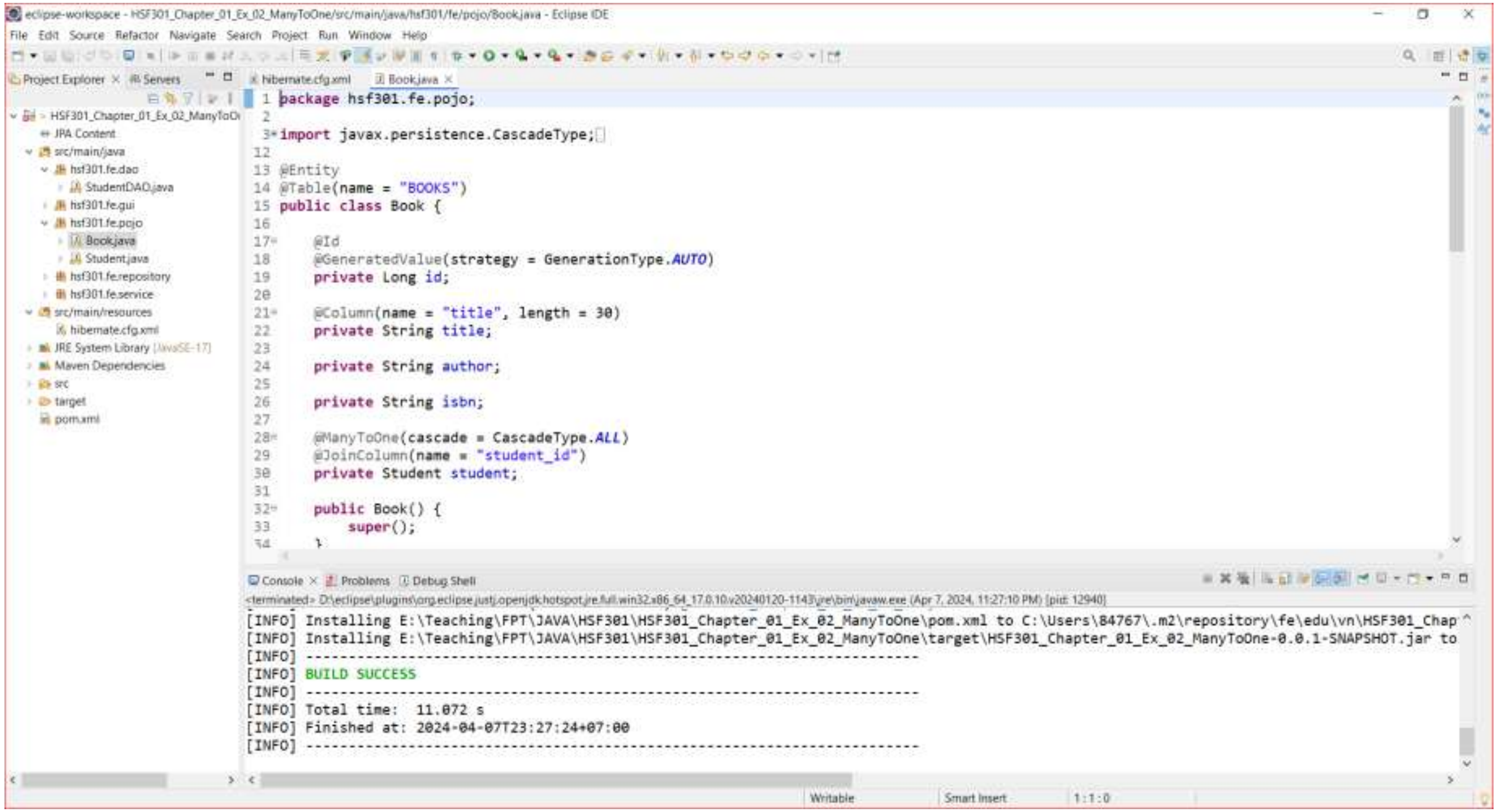
# 5. Create hibernate.cfg.xml



```xml
http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd (doctype)
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.SQLServerDialect</property>
        <property name="connection.url">jdbc:sqlserver://localhost:1433;DatabaseName=tutorial;encrypt=true;trustServerCertificate=true</p
        <property name="connection.username">sa</property>
        <property name="connection.password">Aesx5099@@</property>
        <property name="connection.driver_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>

        <mapping class="hsf301.fe.pojo.Student" />
        <mapping class="hsf301.fe.pojo.Book" />

    </session-factory>

</hibernate-configuration>
```

```
<terminated> D:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (Apr 7, 2024, 11:27:10 PM) [pid: 12940]
[INFO] Installing E:\Teaching\FPT\JAVA\HSF301\HSF301_Chapter_01_Ex_02_ManyToOne\pom.xml to C:\Users\84767\.m2\repository\fe\edu\vn\HSF301_Chap
[INFO] Installing E:\Teaching\FPT\JAVA\HSF301\HSF301_Chapter_01_Ex_02_ManyToOne\target\HSF301_Chapter_01_Ex_02_ManyToOne-0.0.1-SNAPSHOT.jar to
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  11.072 s
[INFO] Finished at: 2024-04-07T23:27:24+07:00
[INFO] ------------------------------------------------------------------------
```

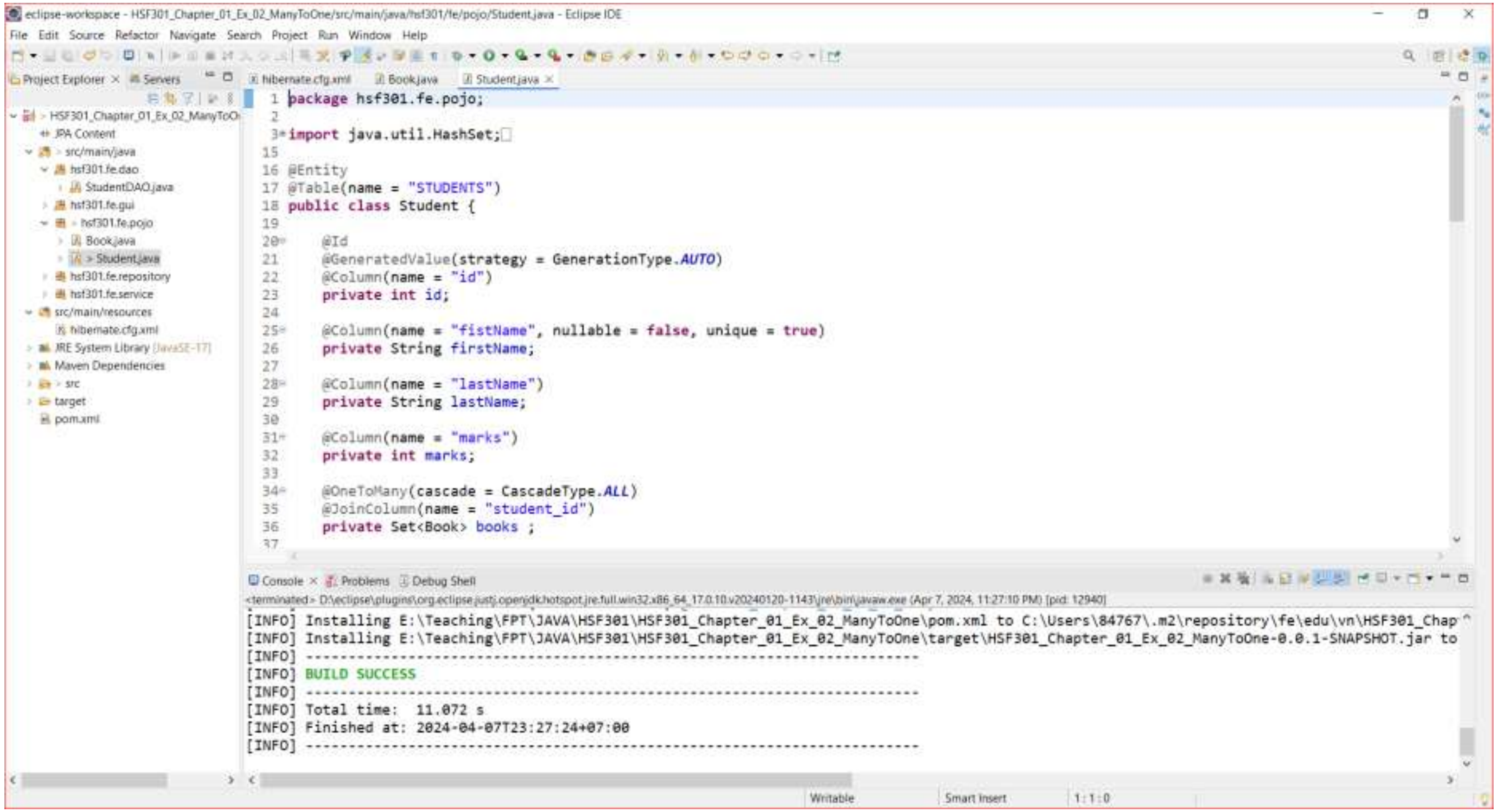
## 6. Create Books in Pojo



```java
package hsf301.fe.pojo;

import javax.persistence.CascadeType;

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

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long 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();
    }
```

```
<terminated> D:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (Apr 7, 2024, 11:27:10 PM) [pid: 12940]
[INFO] Installing E:\Teaching\FPT\JAVA\HSF301\HSF301_Chapter_01_Ex_02_ManyToOne\pom.xml to C:\Users\84767\.m2\repository\fe\edu\vn\HSF301_Chap
[INFO] Installing E:\Teaching\FPT\JAVA\HSF301\HSF301_Chapter_01_Ex_02_ManyToOne\target\HSF301_Chapter_01_Ex_02_ManyToOne-0.0.1-SNAPSHOT.jar to
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  11.072 s
[INFO] Finished at: 2024-04-07T23:27:24+07:00
[INFO] ------------------------------------------------------------------------
```

## 7. Create Students in Pojo

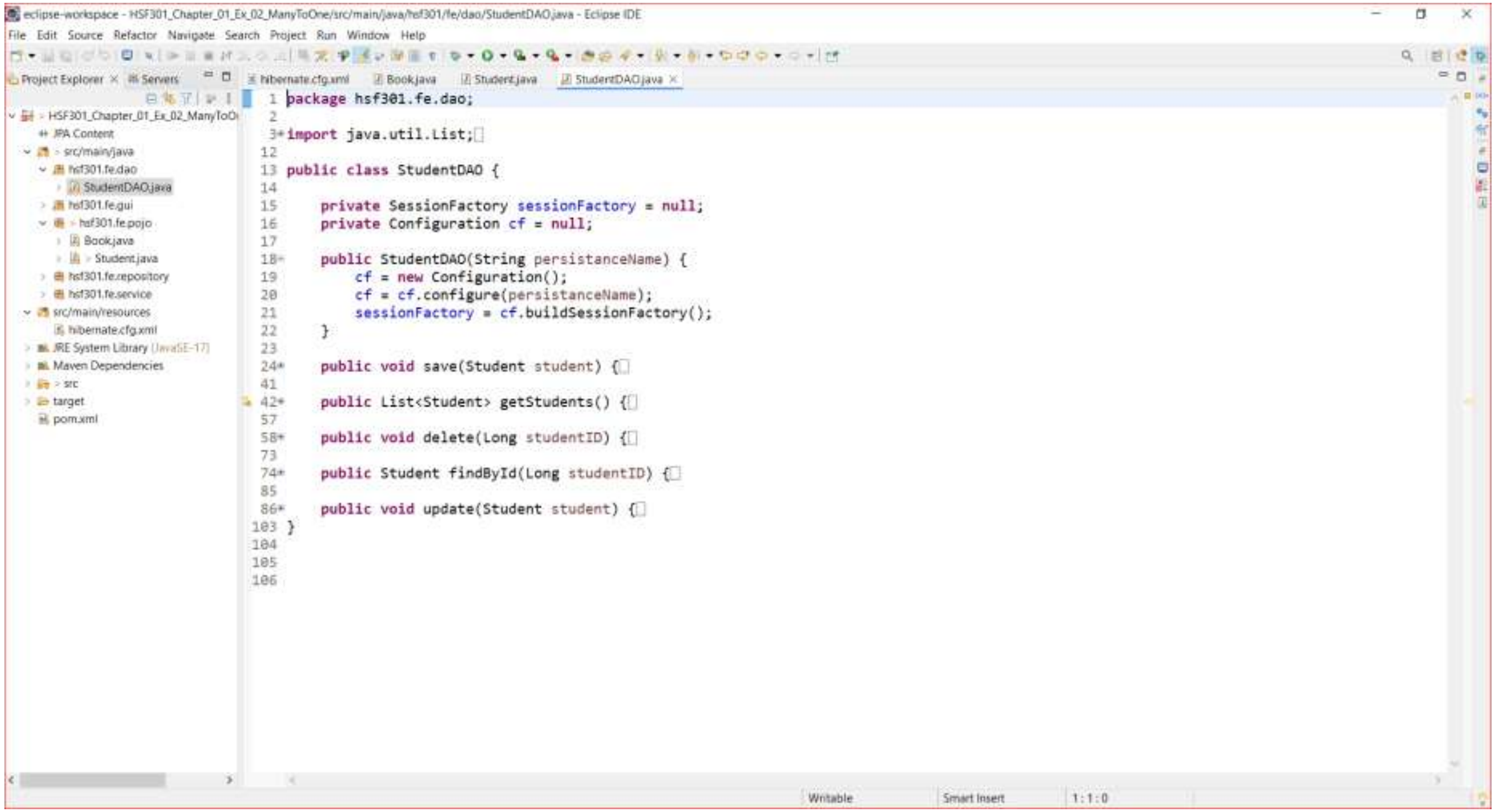


```java
package hsf301.fe.pojo;

import java.util.HashSet;

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

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

    @Column(name = "fistName", nullable = false, unique = true)
    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 ;
```

```
<terminated> D:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (Apr 7, 2024, 11:27:10 PM) [pid: 12940]
[INFO] Installing E:\Teaching\FPT\JAVA\HSF301\HSF301_Chapter_01_Ex_02_ManyToOne\pom.xml to C:\Users\84767\.m2\repository\fe\edu\vn\HSF301_Chap
[INFO] Installing E:\Teaching\FPT\JAVA\HSF301\HSF301_Chapter_01_Ex_02_ManyToOne\target\HSF301_Chapter_01_Ex_02_ManyToOne-0.0.1-SNAPSHOT.jar to
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  11.072 s
[INFO] Finished at: 2024-04-07T23:27:24+07:00
[INFO] ------------------------------------------------------------------------
```

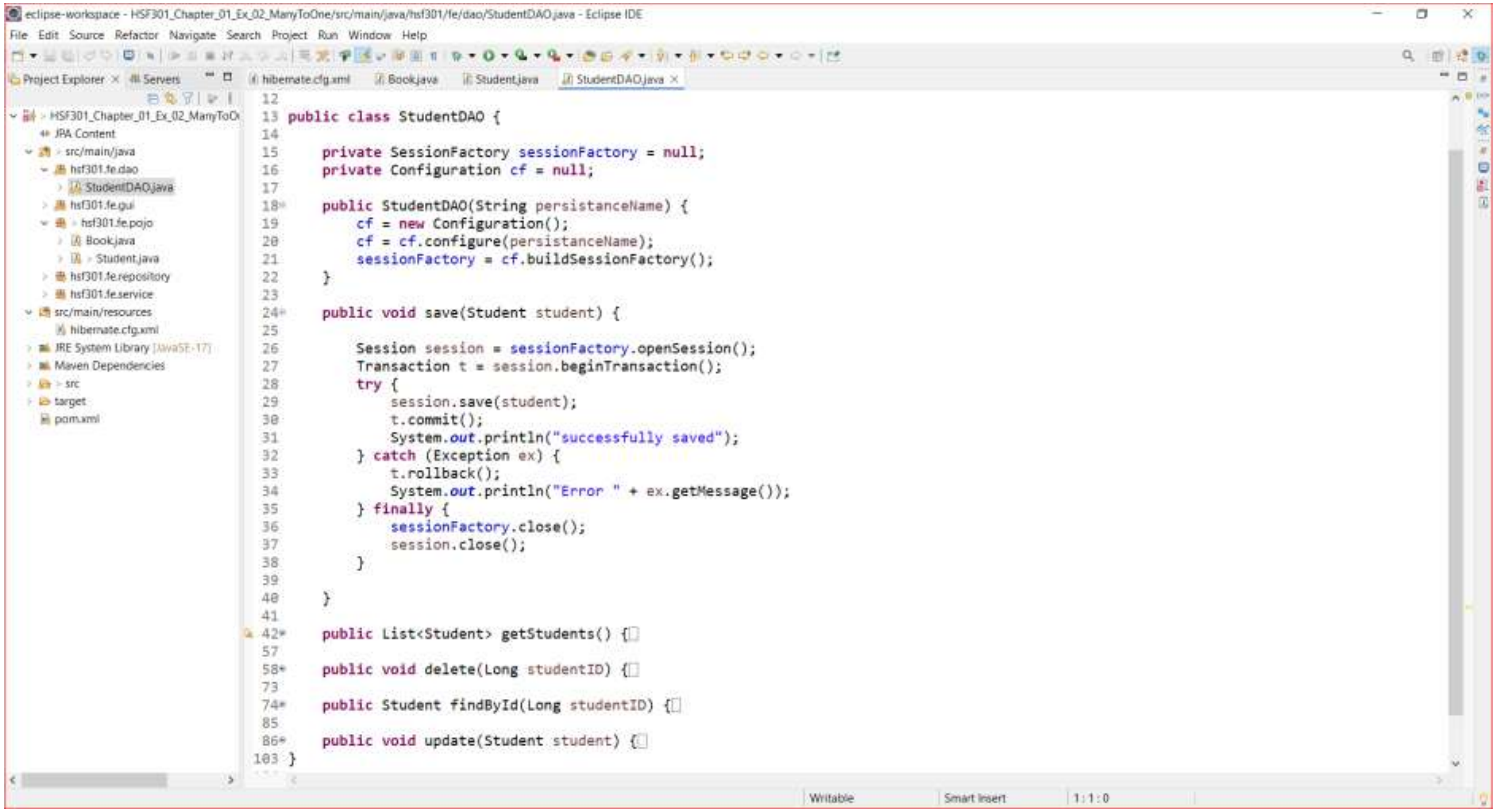
FPT Education
FPT UNIVERSITY

## 8. Create StudentDAO



```java
package hsf301.fe.dao;

import java.util.List;

public class StudentDAO {

    private SessionFactory sessionFactory = null;
    private Configuration cf = null;

    public StudentDAO(String persistanceName) {
        cf = new Configuration();
        cf = cf.configure(persistanceName);
        sessionFactory = cf.buildSessionFactory();
    }

    public void save(Student student) {

    public List<Student> getStudents() {

    public void delete(Long studentID) {

    public Student findById(Long studentID) {

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

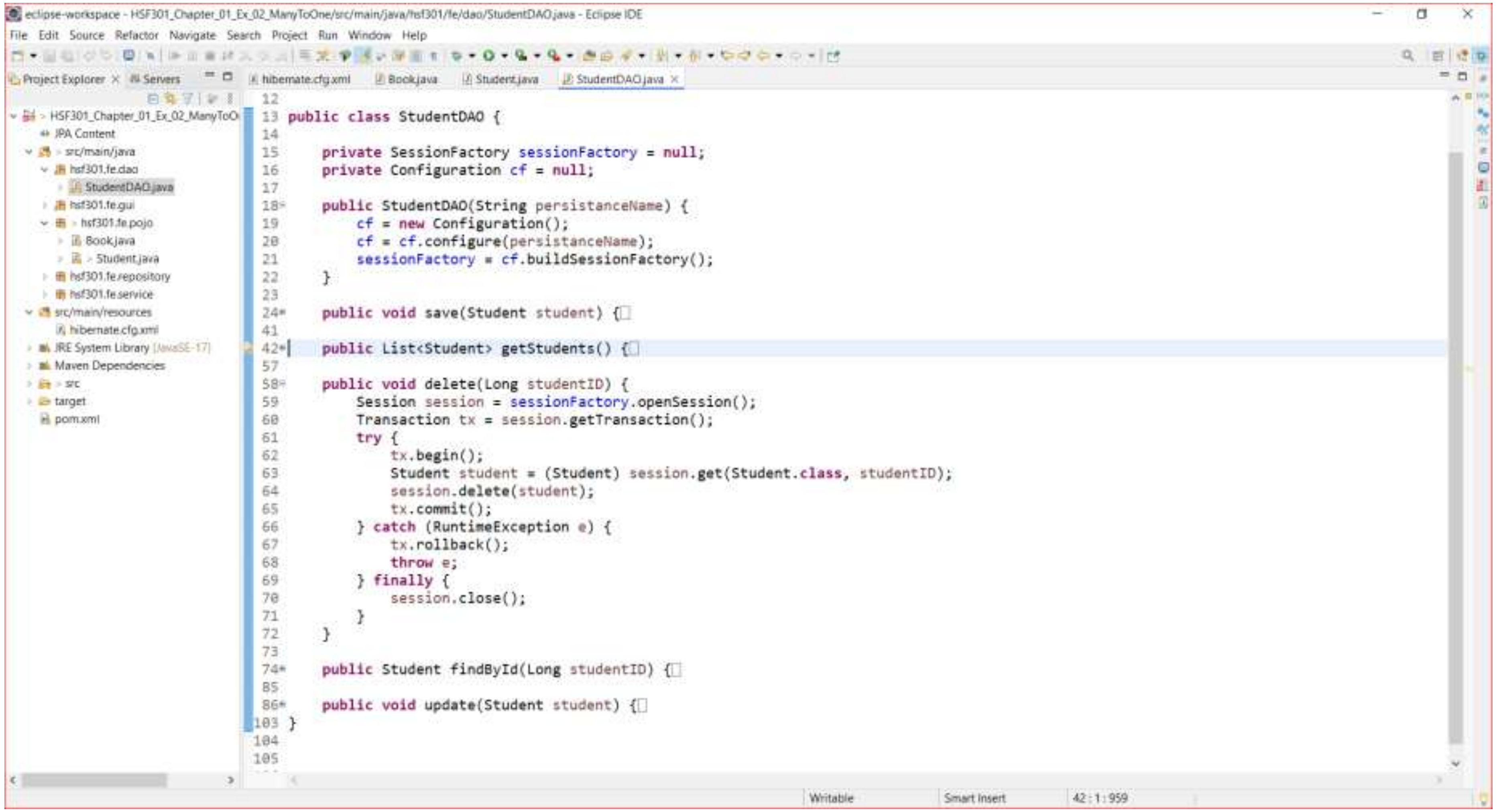
# 9. Save Student in StudentDAO



```java

public class StudentDAO {

    private SessionFactory sessionFactory = null;
    private Configuration cf = null;

    public StudentDAO(String persistanceName) {
        cf = new Configuration();
        cf = cf.configure(persistanceName);
        sessionFactory = cf.buildSessionFactory();
    }

    public void save(Student student) {

        Session session = sessionFactory.openSession();
        Transaction t = session.beginTransaction();
        try {
            session.save(student);
            t.commit();
            System.out.println("successfully saved");
        } catch (Exception ex) {
            t.rollback();
            System.out.println("Error " + ex.getMessage());
        } finally {
            sessionFactory.close();
            session.close();
        }

    }

    public List<Student> getStudents() {

    public void delete(Long studentID) {

    public Student findById(Long studentID) {

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

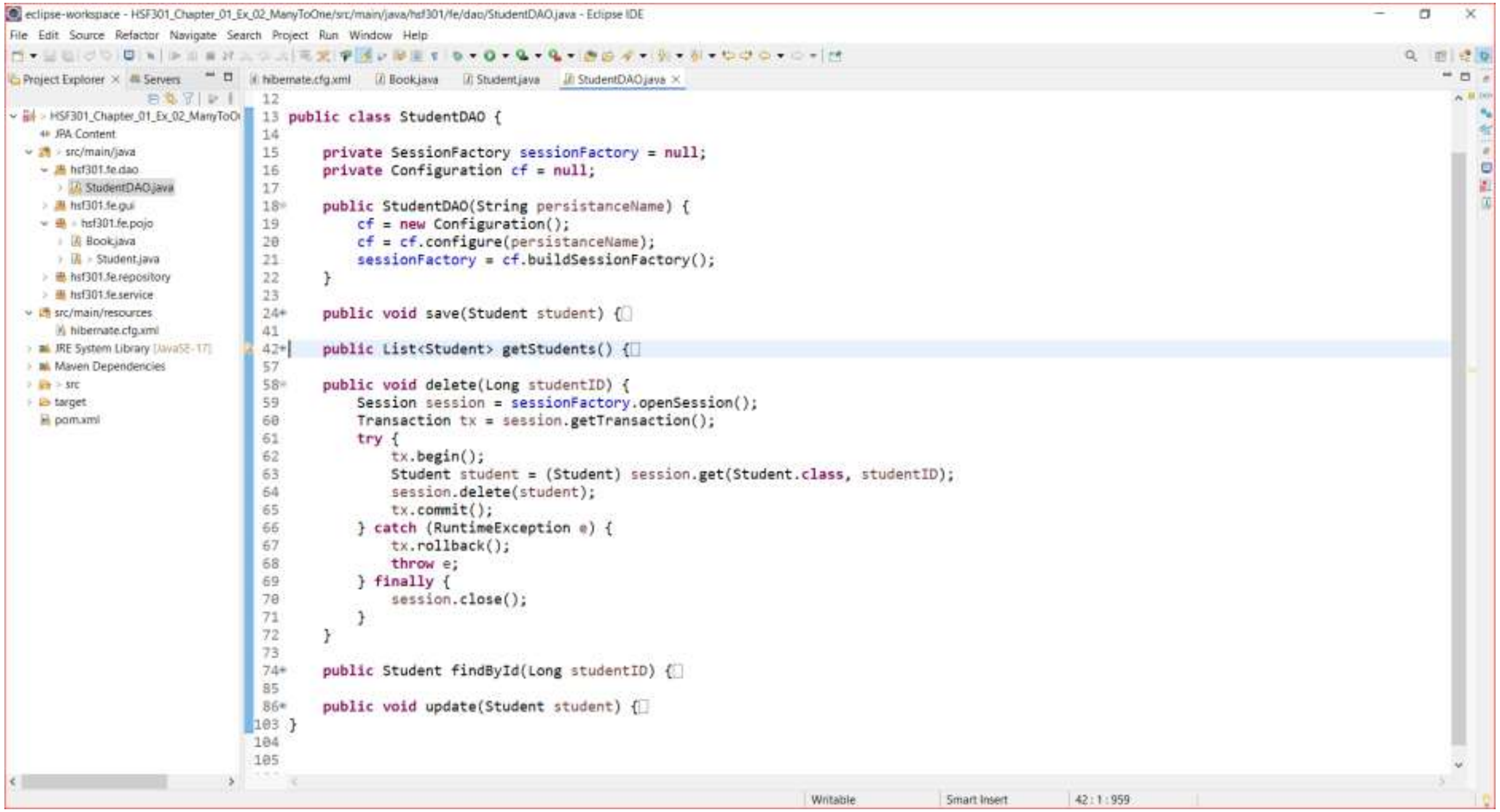
FPT Education
FPT UNIVERSITY

## 10. Get All Student in StudentDAO



```java
public class StudentDAO {

    private SessionFactory sessionFactory = null;
    private Configuration cf = null;

    public StudentDAO(String persistanceName) {
        cf = new Configuration();
        cf = cf.configure(persistanceName);
        sessionFactory = cf.buildSessionFactory();
    }

    public void save(Student student) {

    public List<Student> getStudents() {

    public void delete(Long studentID) {
        Session session = sessionFactory.openSession();
        Transaction tx = session.getTransaction();
        try {
            tx.begin();
            Student student = (Student) session.get(Student.class, studentID);
            session.delete(student);
            tx.commit();
        } catch (RuntimeException e) {
            tx.rollback();
            throw e;
        } finally {
            session.close();
        }
    }

    public Student findById(Long studentID) {

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

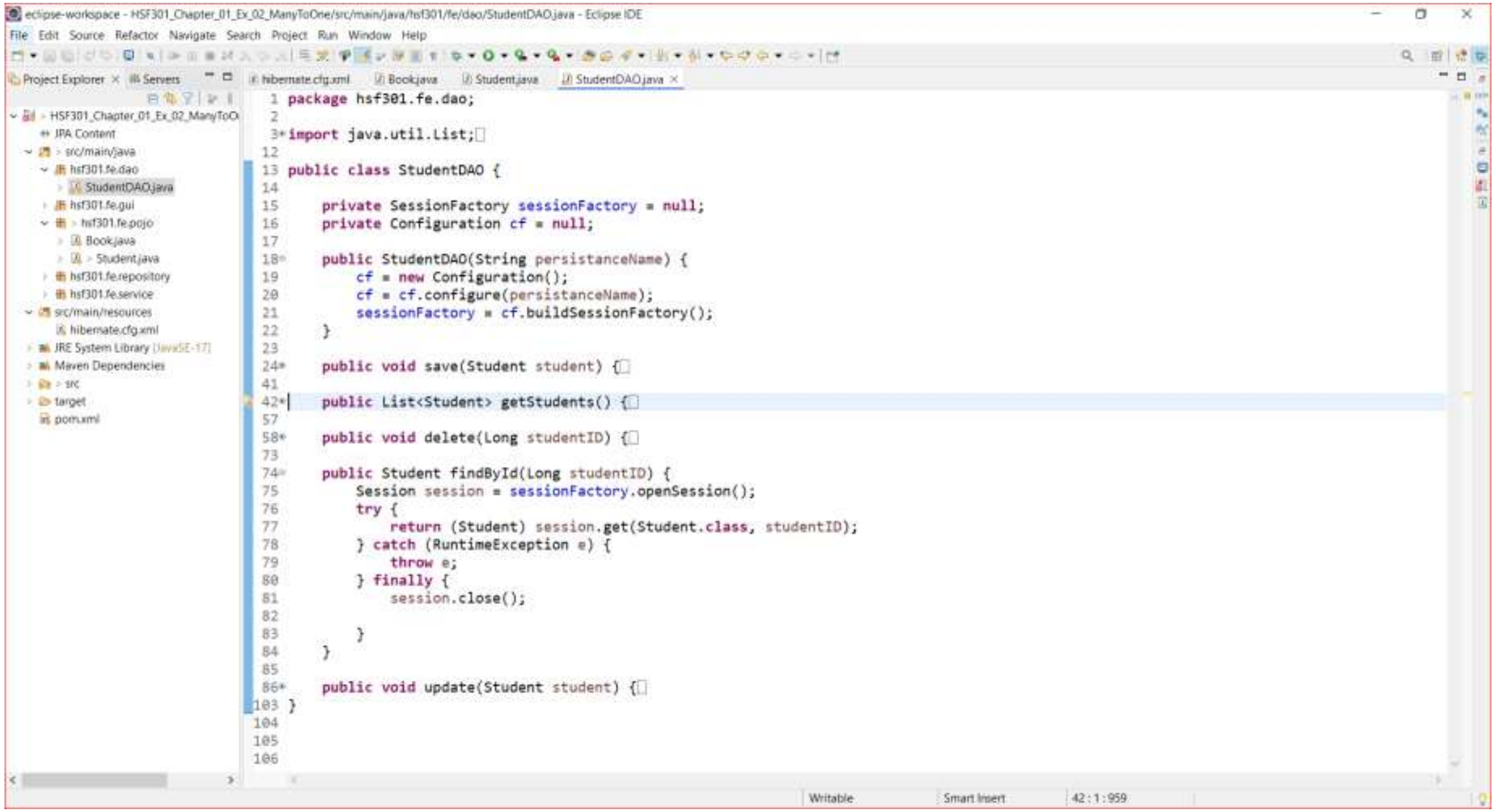
# 11. Delete Student in StudentDAO



eclipse-workspace - HSF301_Chapter_01_Ex_02_ManyToOne/src/main/java/hsf301/fe/dao/StudentDAO.java - Eclipse IDE

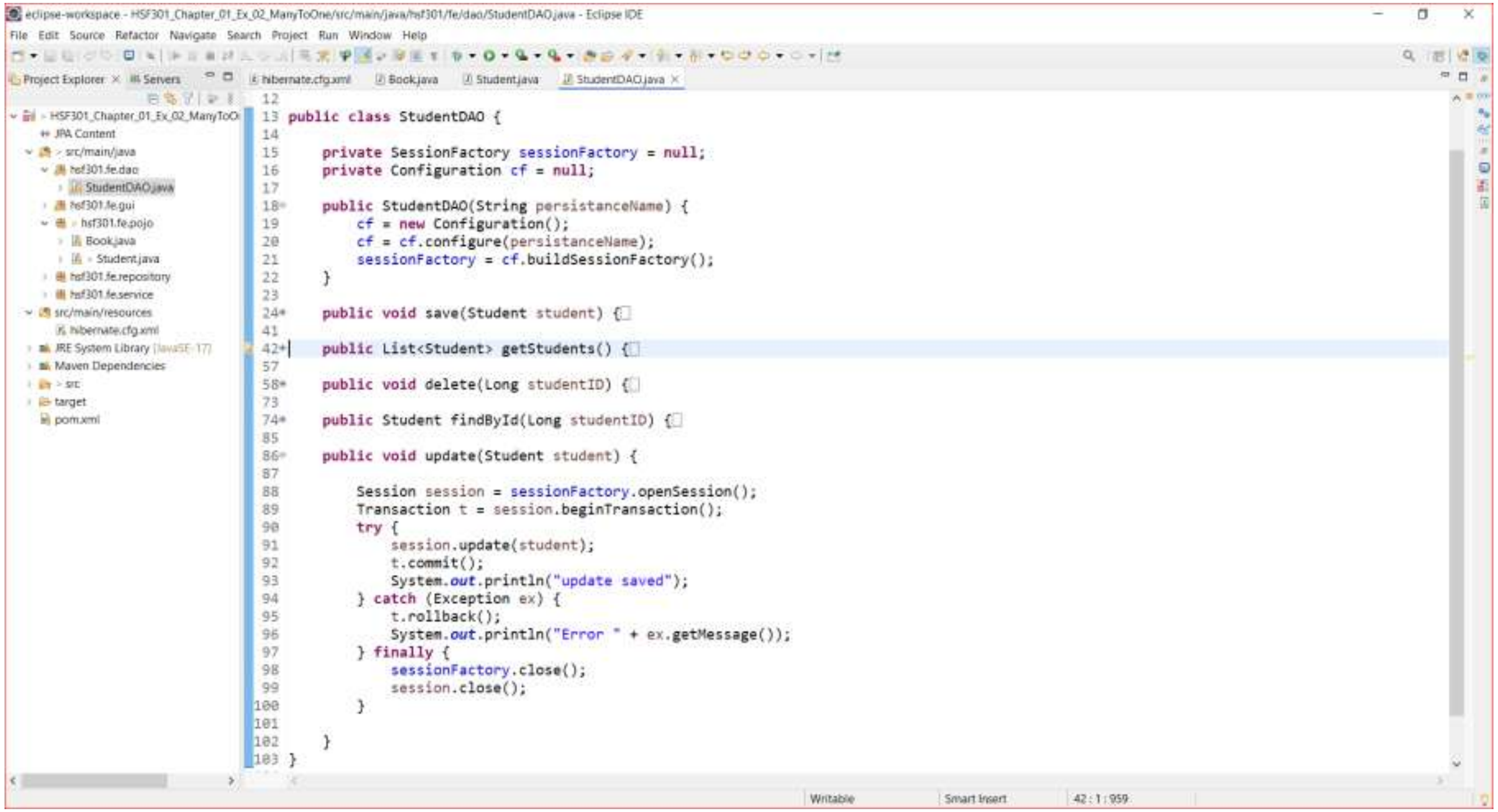
```java
public class StudentDAO {

    private SessionFactory sessionFactory = null;
    private Configuration cf = null;

    public StudentDAO(String persistanceName) {
        cf = new Configuration();
        cf = cf.configure(persistanceName);
        sessionFactory = cf.buildSessionFactory();
    }

    public void save(Student student) {

    public List<Student> getStudents() {

    public void delete(Long studentID) {
        Session session = sessionFactory.openSession();
        Transaction tx = session.getTransaction();
        try {
            tx.begin();
            Student student = (Student) session.get(Student.class, studentID);
            session.delete(student);
            tx.commit();
        } catch (RuntimeException e) {
            tx.rollback();
            throw e;
        } finally {
            session.close();
        }
    }

    public Student findById(Long studentID) {

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

## 12. Find A Student in StudentDAO



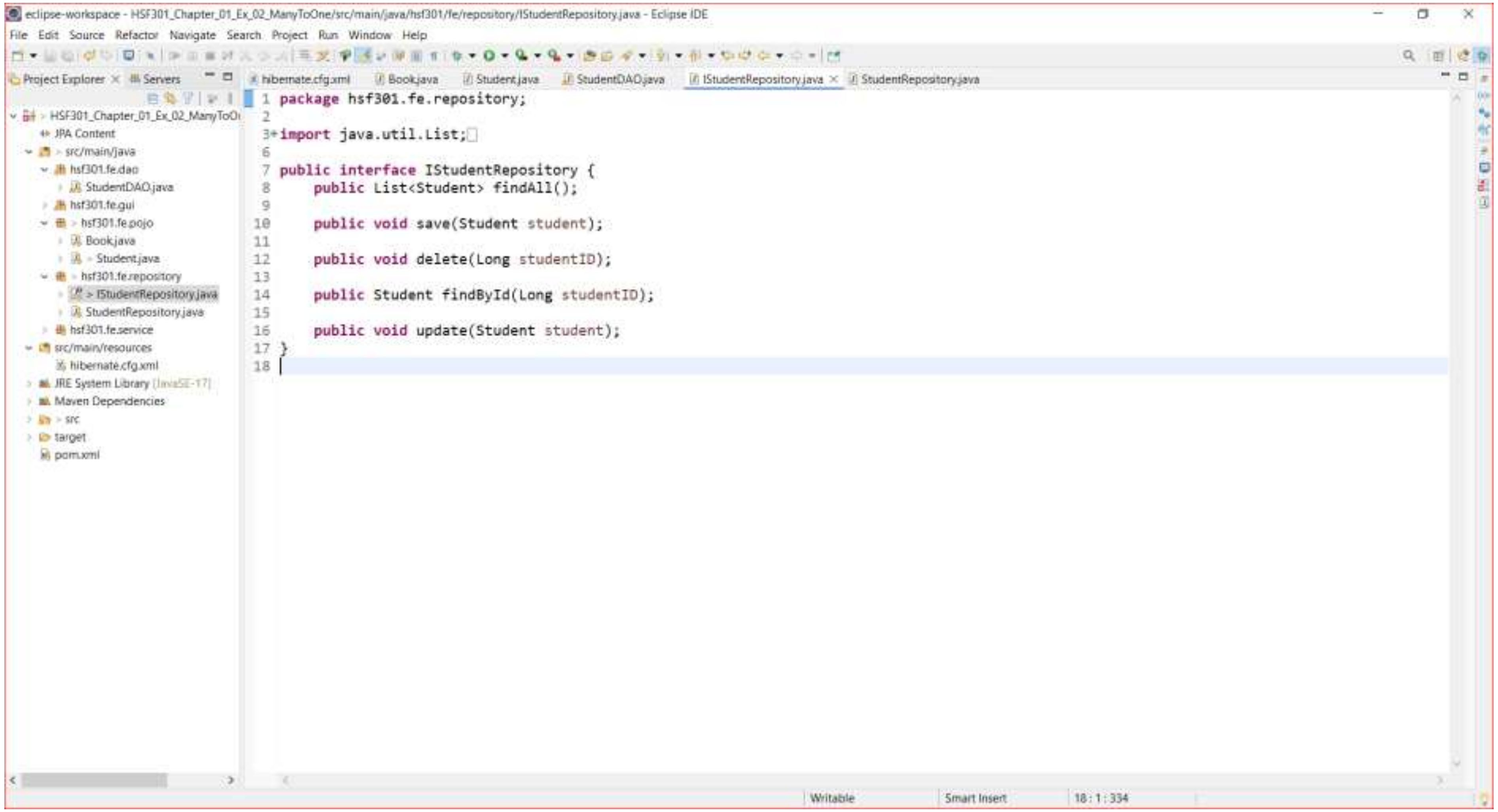
```java
package hsf301.fe.dao;

import java.util.List;

public class StudentDAO {

    private SessionFactory sessionFactory = null;
    private Configuration cf = null;

    public StudentDAO(String persistanceName) {
        cf = new Configuration();
        cf = cf.configure(persistanceName);
        sessionFactory = cf.buildSessionFactory();
    }

    public void save(Student student) {

    public List<Student> getStudents() {

    public void delete(Long studentID) {

    public Student findById(Long studentID) {
        Session session = sessionFactory.openSession();
        try {
            return (Student) session.get(Student.class, studentID);
        } catch (RuntimeException e) {
            throw e;
        } finally {
            session.close();

        }
    }

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

## 13. Update a Student in StudentDAO



```java
public class StudentDAO {

    private SessionFactory sessionFactory = null;
    private Configuration cf = null;

    public StudentDAO(String persistanceName) {
        cf = new Configuration();
        cf = cf.configure(persistanceName);
        sessionFactory = cf.buildSessionFactory();
    }

    public void save(Student student) {

    public List<Student> getStudents() {

    public void delete(Long studentID) {

    public Student findById(Long studentID) {

    public void update(Student student) {

        Session session = sessionFactory.openSession();
        Transaction t = session.beginTransaction();
        try {
            session.update(student);
            t.commit();
            System.out.println("update saved");
        } catch (Exception ex) {
            t.rollback();
            System.out.println("Error " + ex.getMessage());
        } finally {
            sessionFactory.close();
            session.close();
        }

    }
}
```

# 14. Create IStudentRepository

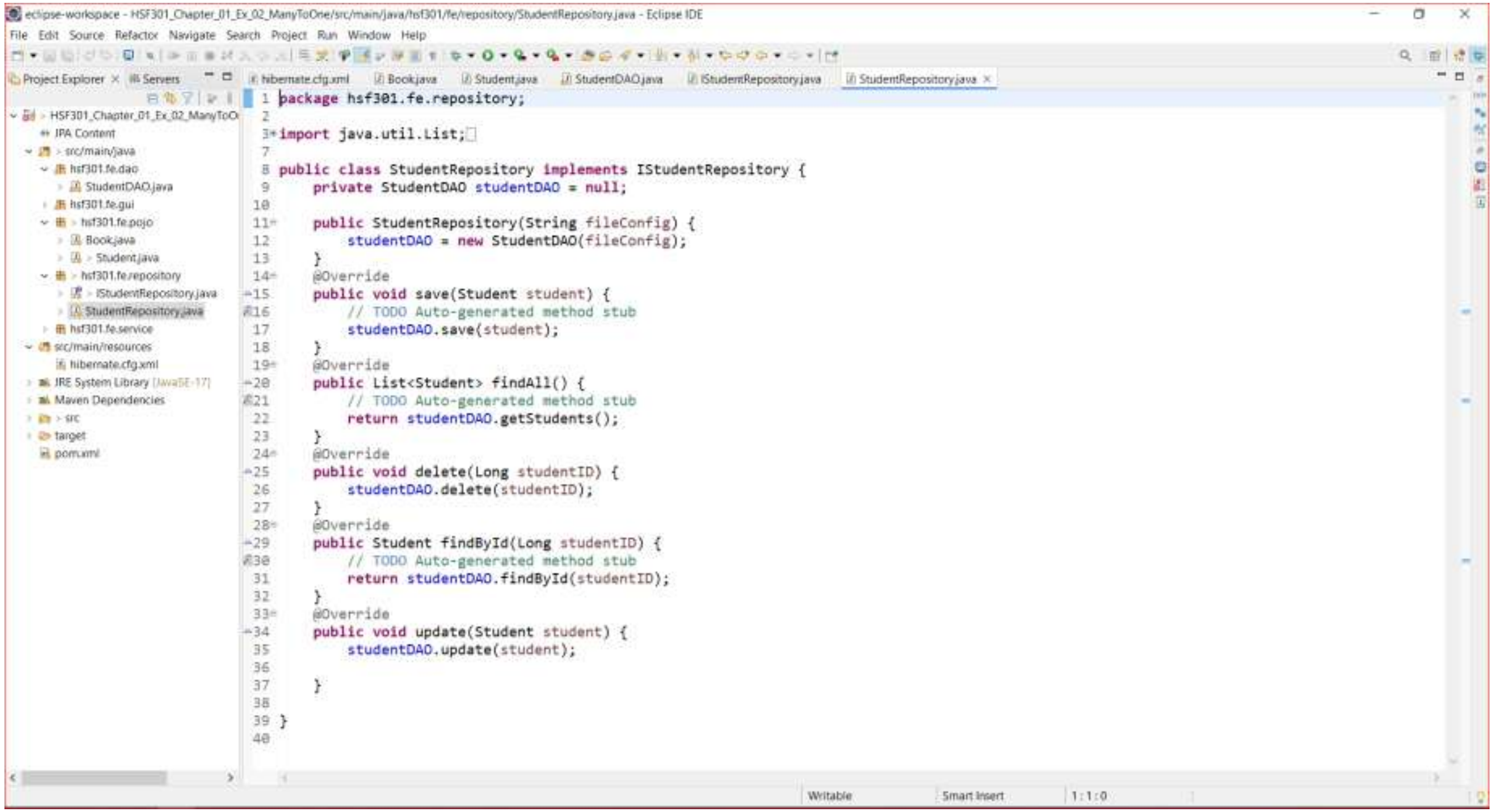


eclipse-workspace - HSF301_Chapter_01_Ex_02_ManyToOne/src/main/java/hsf301/fe/repository/IStudentRepository.java - Eclipse IDE

```java
package hsf301.fe.repository;

import java.util.List;

public interface IStudentRepository {
    public List<Student> findAll();

    public void save(Student student);

    public void delete(Long studentID);

    public Student findById(Long studentID);

    public void update(Student student);
}
```

FPT Education
FPT UNIVERSITY

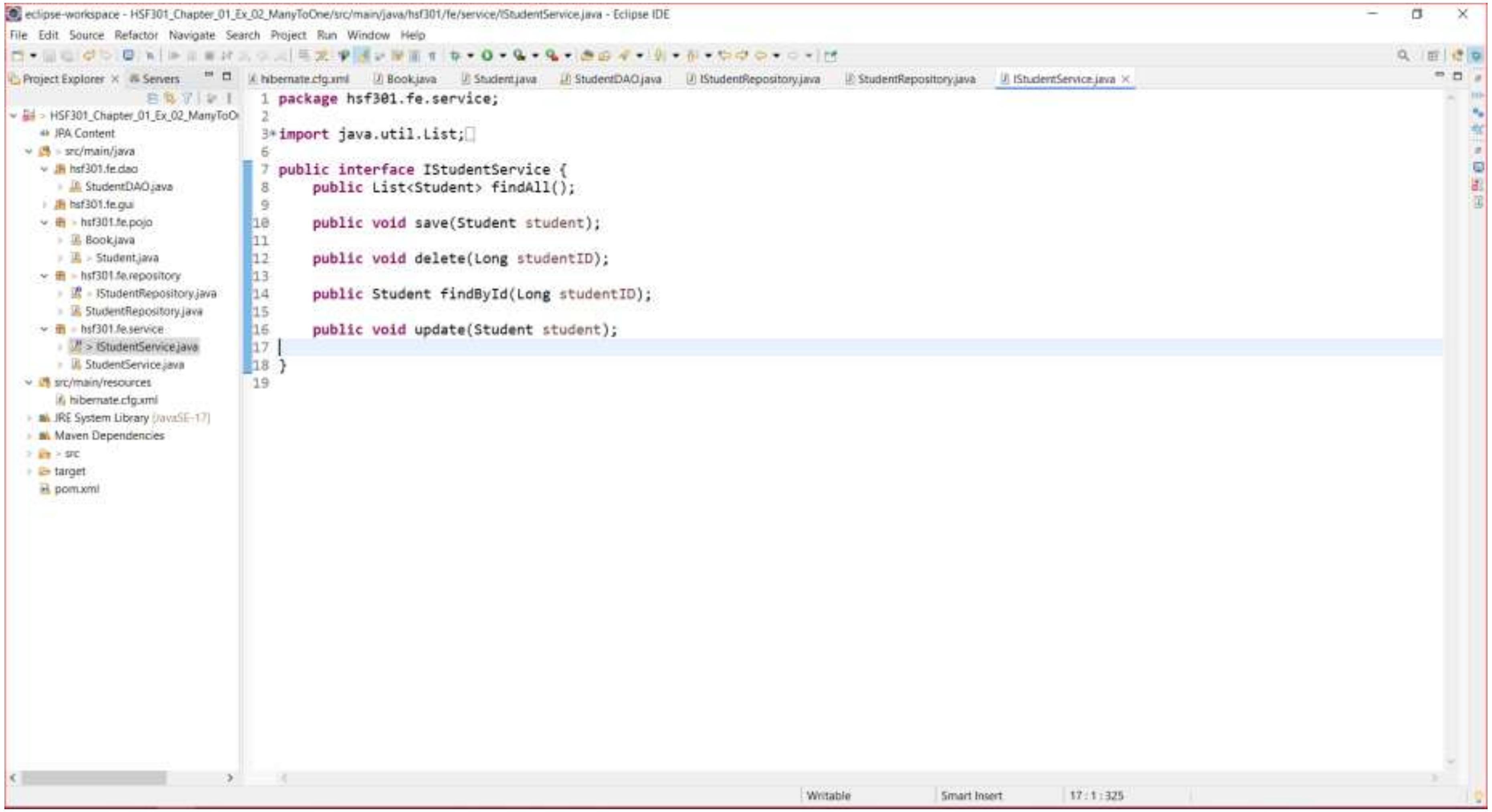
## 15. Create StudentRepository



eclipse-workspace - HSF301_Chapter_01_Ex_02_ManyToOne/src/main/java/hsf301/fe/repository/StudentRepository.java - Eclipse IDE

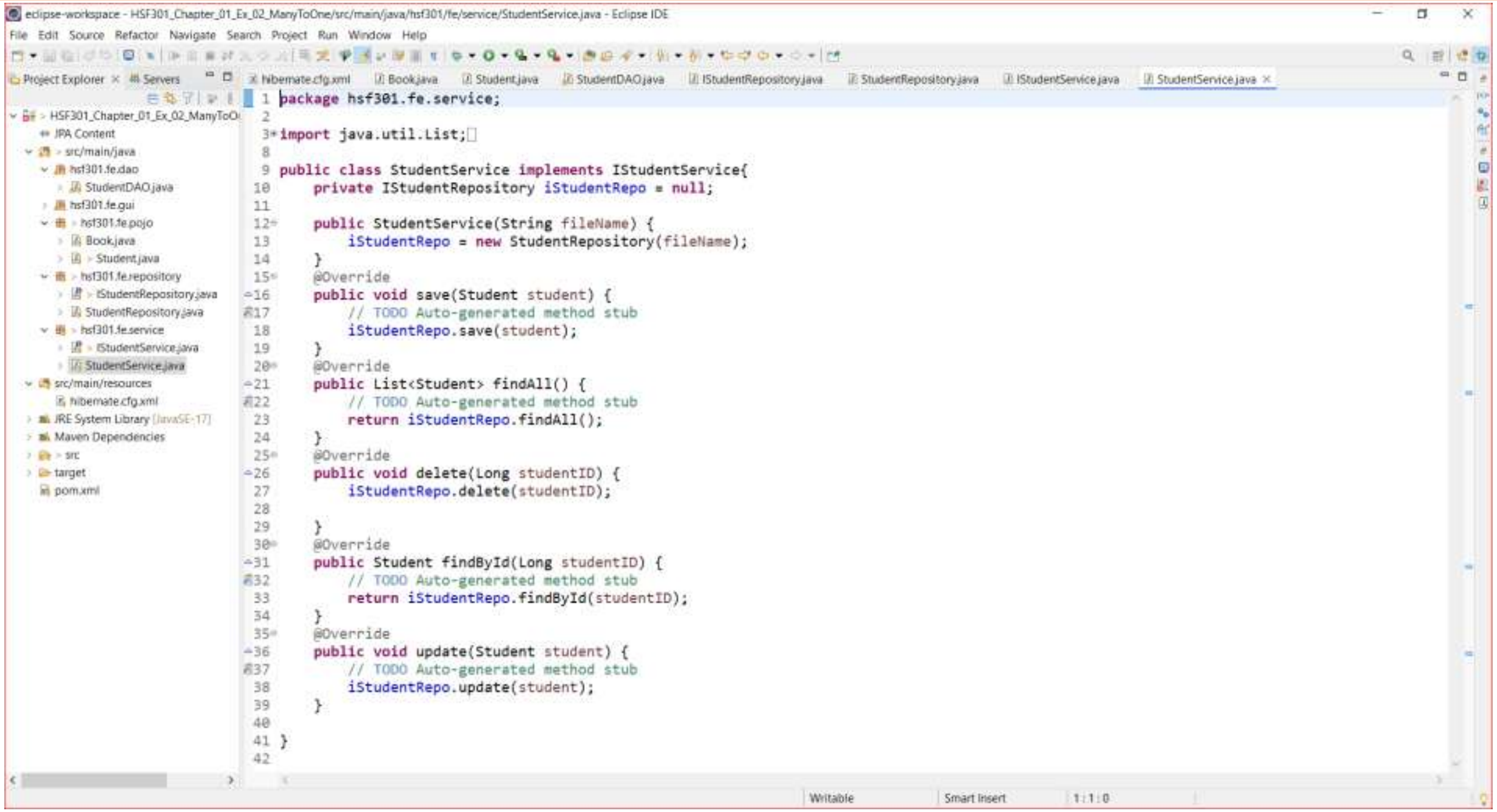
```java
package hsf301.fe.repository;

import java.util.List;

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

    public StudentRepository(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(Long studentID) {
        studentDAO.delete(studentID);
    }
    @Override
    public Student findById(Long studentID) {
        // TODO Auto-generated method stub
        return studentDAO.findById(studentID);
    }
    @Override
    public void update(Student student) {
        studentDAO.update(student);

    }

}
```

## 16. Create IStudentService


eclipse-workspace - HSF301_Chapter_01_Ex_02_ManyToOne/src/main/java/hsf301/fe/service/IStudentService.java - Eclipse IDE

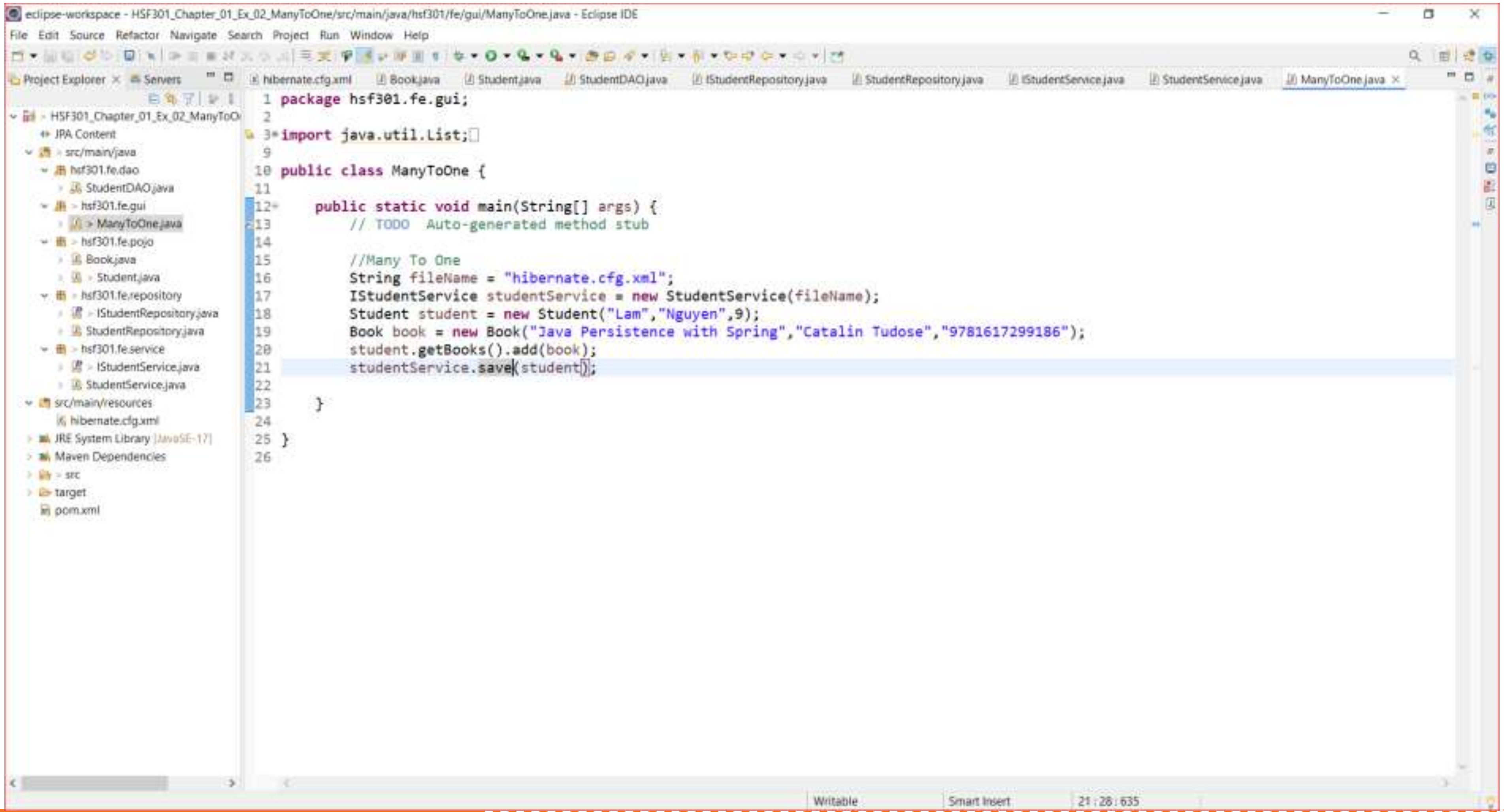
```java
package hsf301.fe.service;

import java.util.List;

public interface IStudentService {
    public List<Student> findAll();

    public void save(Student student);

    public void delete(Long studentID);

    public Student findById(Long studentID);

    public void update(Student student);

}
```

# 17. Create StudentService



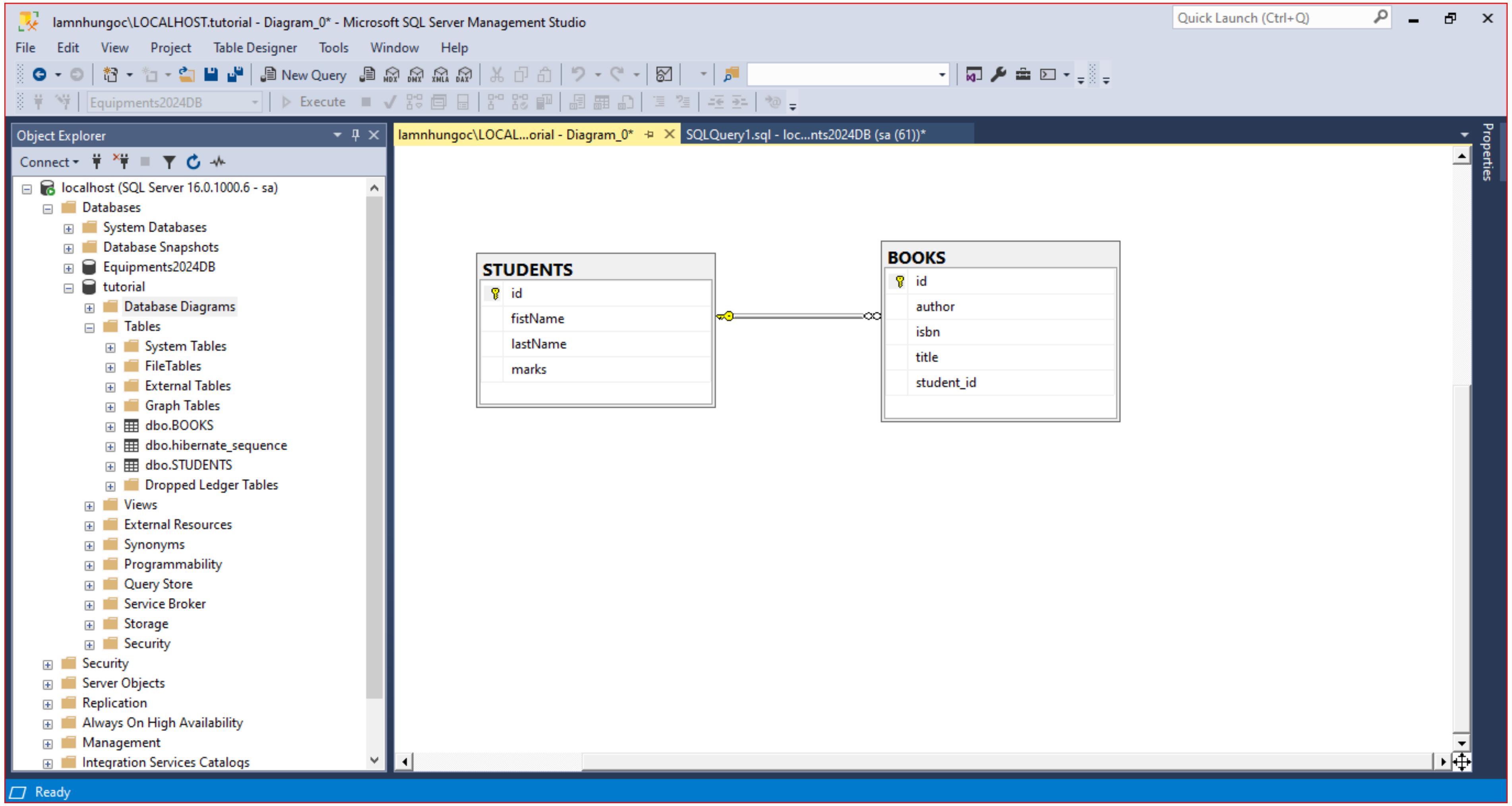
```java
package hsf301.fe.service;

import java.util.List;

public class StudentService implements IStudentService{
    private IStudentRepository iStudentRepo = null;

    public StudentService(String fileName) {
        iStudentRepo = new StudentRepository(fileName);
    }
    @Override
    public void save(Student student) {
        // TODO Auto-generated method stub
        iStudentRepo.save(student);
    }
    @Override
    public List<Student> findAll() {
        // TODO Auto-generated method stub
        return iStudentRepo.findAll();
    }
    @Override
    public void delete(Long studentID) {
        iStudentRepo.delete(studentID);

    }
    @Override
    public Student findById(Long studentID) {
        // TODO Auto-generated method stub
        return iStudentRepo.findById(studentID);
    }
    @Override
    public void update(Student student) {
        // TODO Auto-generated method stub
        iStudentRepo.update(student);
    }

}
```

## 18. Create Main function

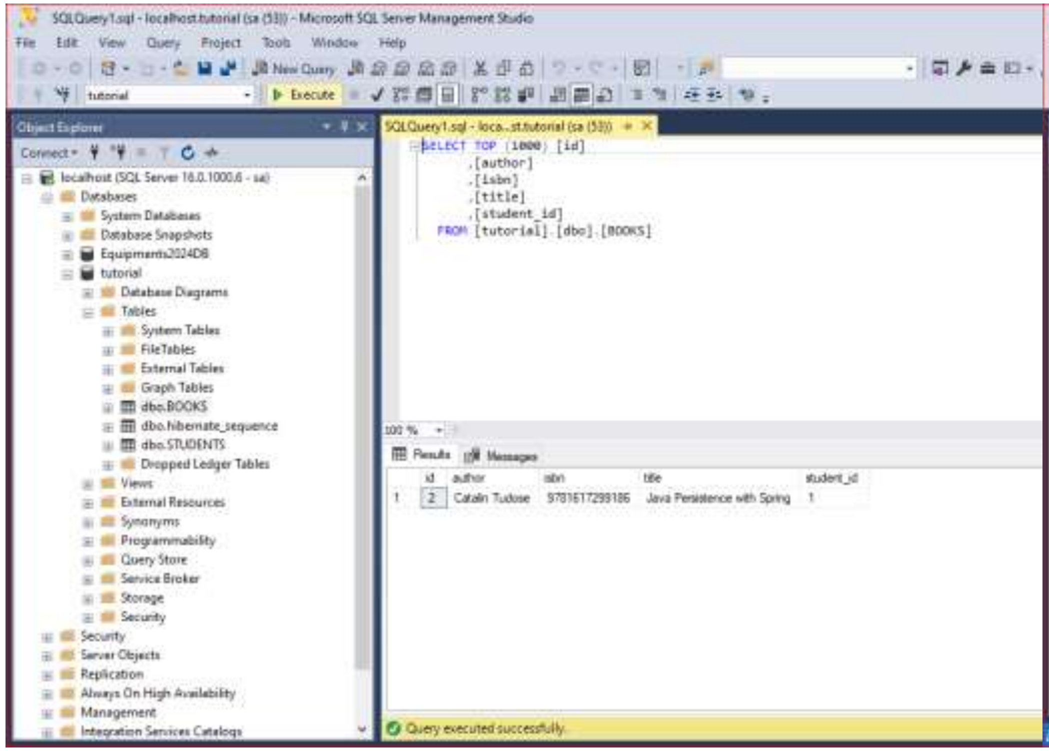


```
package hsf301.fe.gui;

import java.util.List;

public class ManyToOne {

    public static void main(String[] args) {
        // TODO  Auto-generated method stub

        //Many To One
        String fileName = "hibernate.cfg.xml";
        IStudentService studentService = new StudentService(fileName);
        Student student = new Student("Lam","Nguyen",9);
        Book book = new Book("Java Persistence with Spring","Catalin Tudose","9781617299186");
        student.getBooks().add(book);
        studentService.save(student);

    }
}
```

## 19. Result


lamnhungoc\LOCALHOST.tutorial - Diagram_0* - Microsoft SQL Server Management Studio
File
Edit
View
Project
Table Designer
Tools
Window
Help
New Query
Equipments2024DB
Execute
Object Explorer
Connect
localhost (SQL Server 16.0.1000.6 - sa)
Databases
System Databases
Database Snapshots
Equipments2024DB
tutorial
Database Diagrams
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.BOOKS
dbo.hibernate_sequence
dbo.STUDENTS
Dropped Ledger Tables
Views
External Resources
Synonyms
Programmability
Query Store
Service Broker
Storage
Security
Security
Server Objects
Replication
Always On High Availability
Management
Integration Services Catalogs
lamnhungoc\LOCAL...orial - Diagram_0*
SQLQuery1.sql - loc...nts2024DB (sa (61))*
STUDENTS
id
fistName
lastName
marks
BOOKS
id
author
isbn
title
student_id
Ready
Properties

# 20. Result

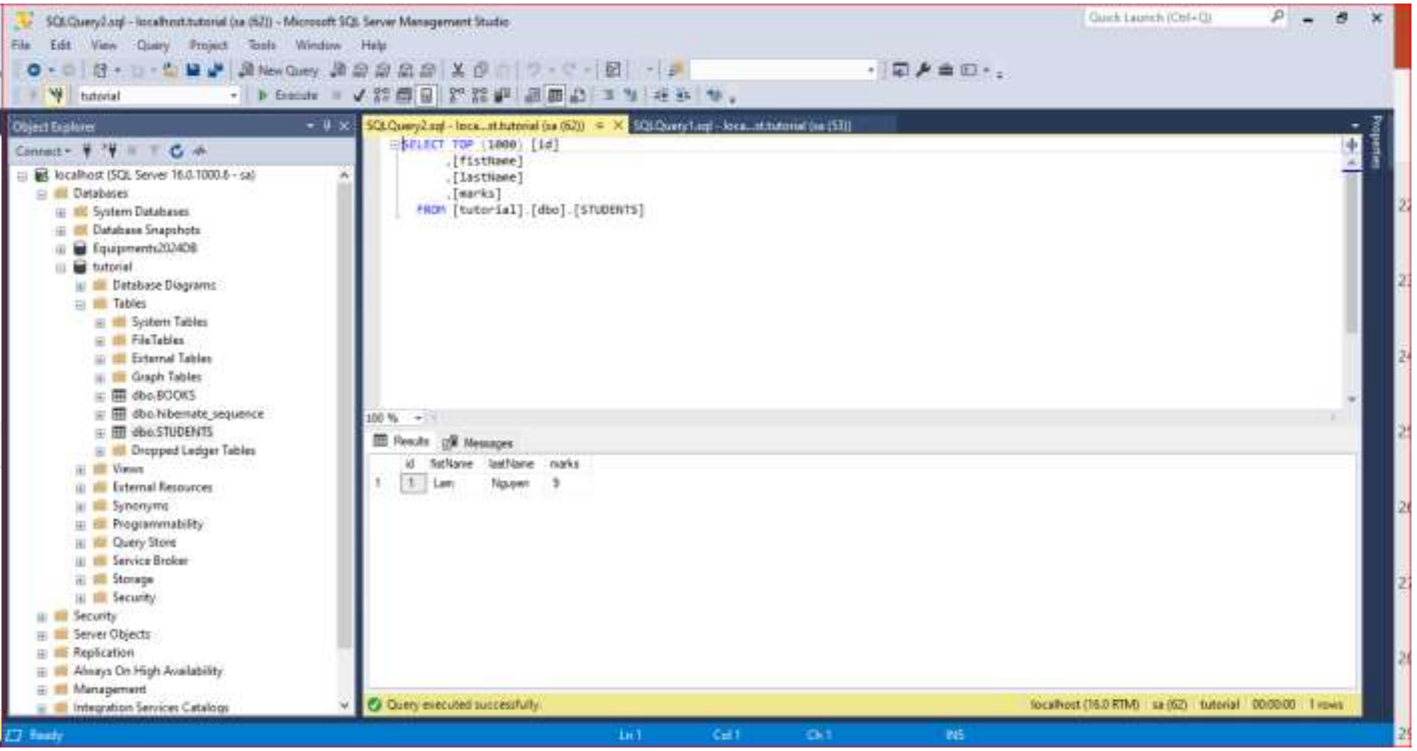


SQLQuery1.sql - localhost.tutorial (sa (53)) - Microsoft SQL Server Management Studio

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

|  | id | author | isbn | title | student_id |
|---|---|---|---|---|---|
| 1 | 2 | Catalin Tudose | 9781617299186 | Java Persistence with Spring | 1 |

Query executed successfully.



SQLQuery2.sql - localhost.tutorial (sa (62)) - Microsoft SQL Server Management Studio

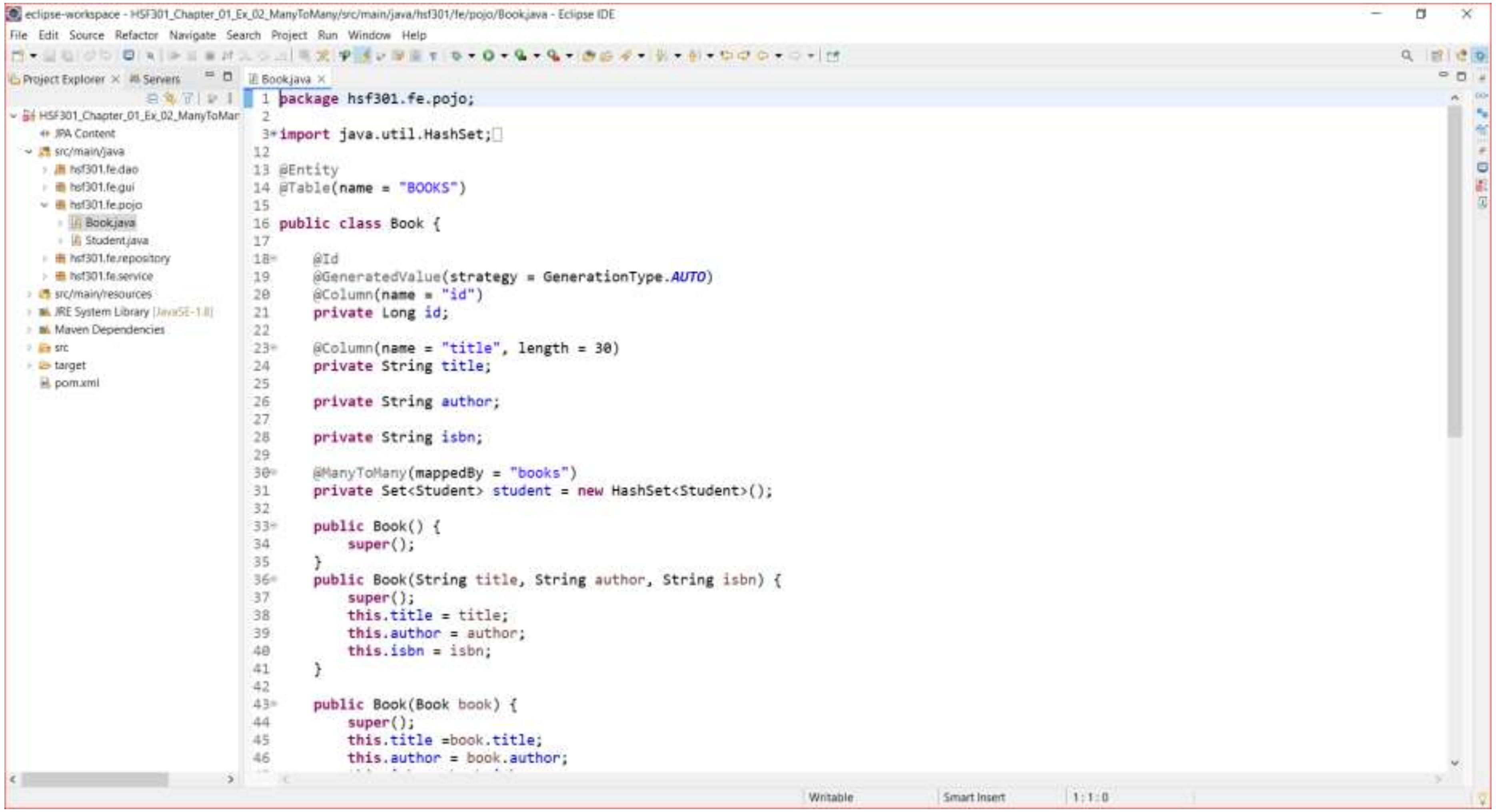
```
SELECT TOP (1000) [id]
      ,[fistName]
      ,[lastName]
      ,[marks]
  FROM [tutorial].[dbo].[STUDENTS]
```

|  | id | fistName | lastName | marks |
|---|---|---|---|---|
| 1 | 1 | Lam | Nguyen | 9 |

Query executed successfully.

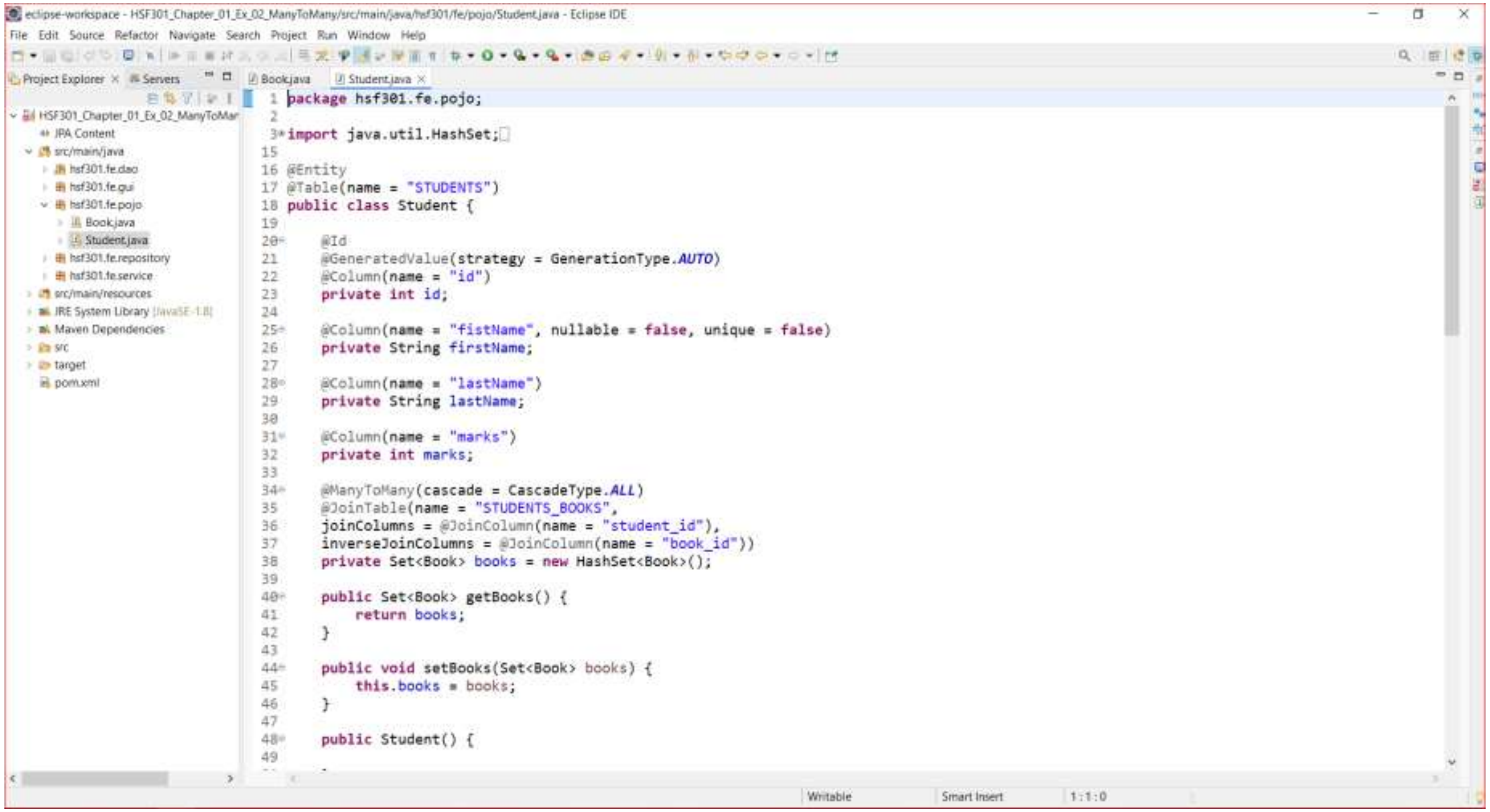
## Demo JPA (Many To Many)

# 1. Create Books in Pojo's Package


eclipse-workspace - HSF301_Chapter_01_Ex_02_ManyToMany/src/main/java/hsf301/fe/pojo/Book.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Servers
Book.java
HSF301_Chapter_01_Ex_02_ManyToMan
JPA Content
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
Writable
Smart Insert
1 : 1 : 0

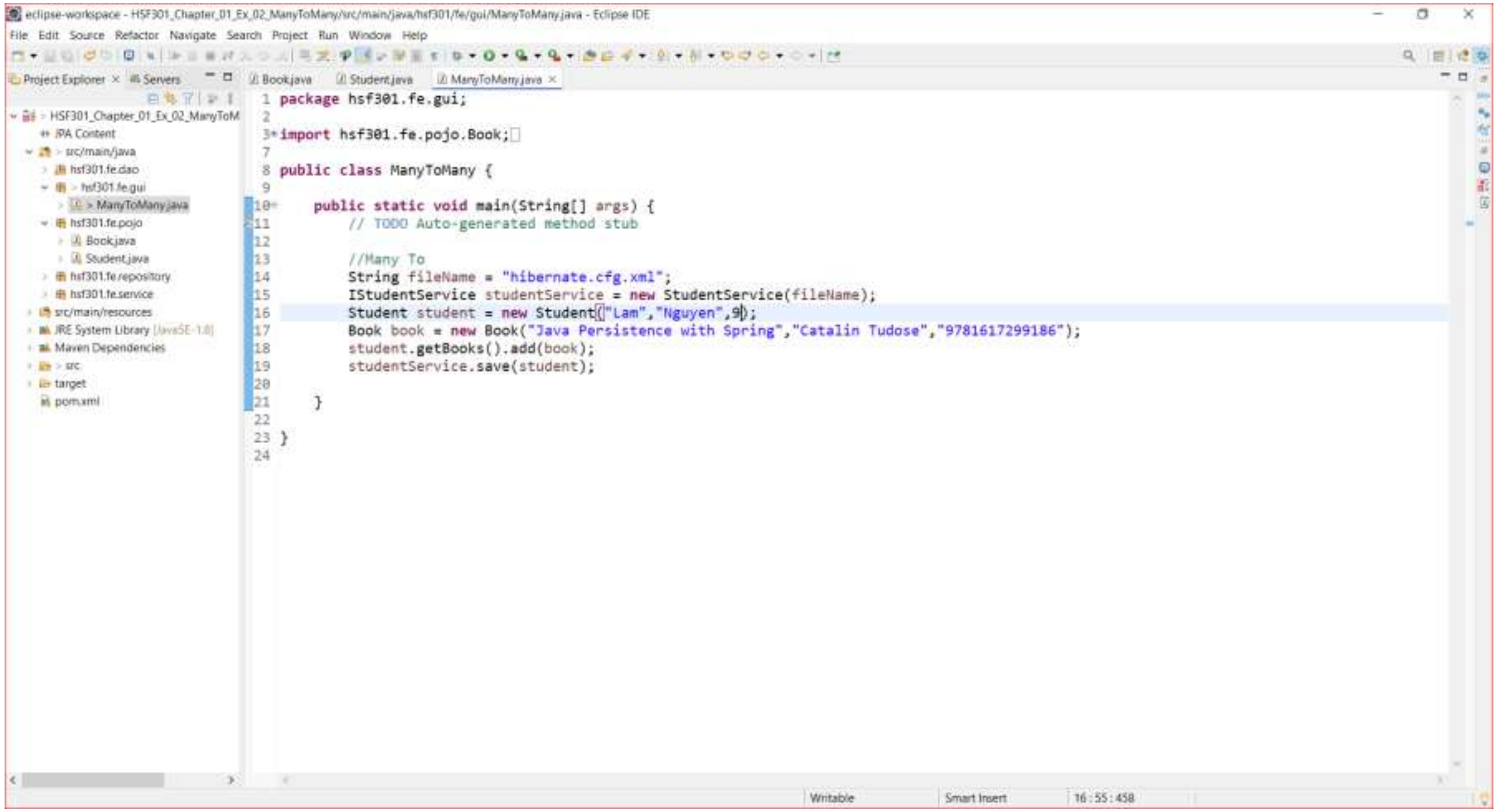
```java
package hsf301.fe.pojo;

import java.util.HashSet;

@Entity
@Table(name = "BOOKS")

public class Book {

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

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

    private String author;

    private String isbn;

    @ManyToMany(mappedBy = "books")
    private Set<Student> student = new HashSet<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;
```

## 2. Create Students in Pojo’s Package



```java
package hsf301.fe.pojo;

import java.util.HashSet;

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

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

    @Column(name = "fistName", nullable = false, unique = false)
    private String firstName;

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

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

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(name = "STUDENTS_BOOKS",
    joinColumns = @JoinColumn(name = "student_id"),
    inverseJoinColumns = @JoinColumn(name = "book_id"))
    private Set<Book> books = new HashSet<Book>();

    public Set<Book> getBooks() {
        return books;
    }

    public void setBooks(Set<Book> books) {
        this.books = books;
    }

    public Student() {

```

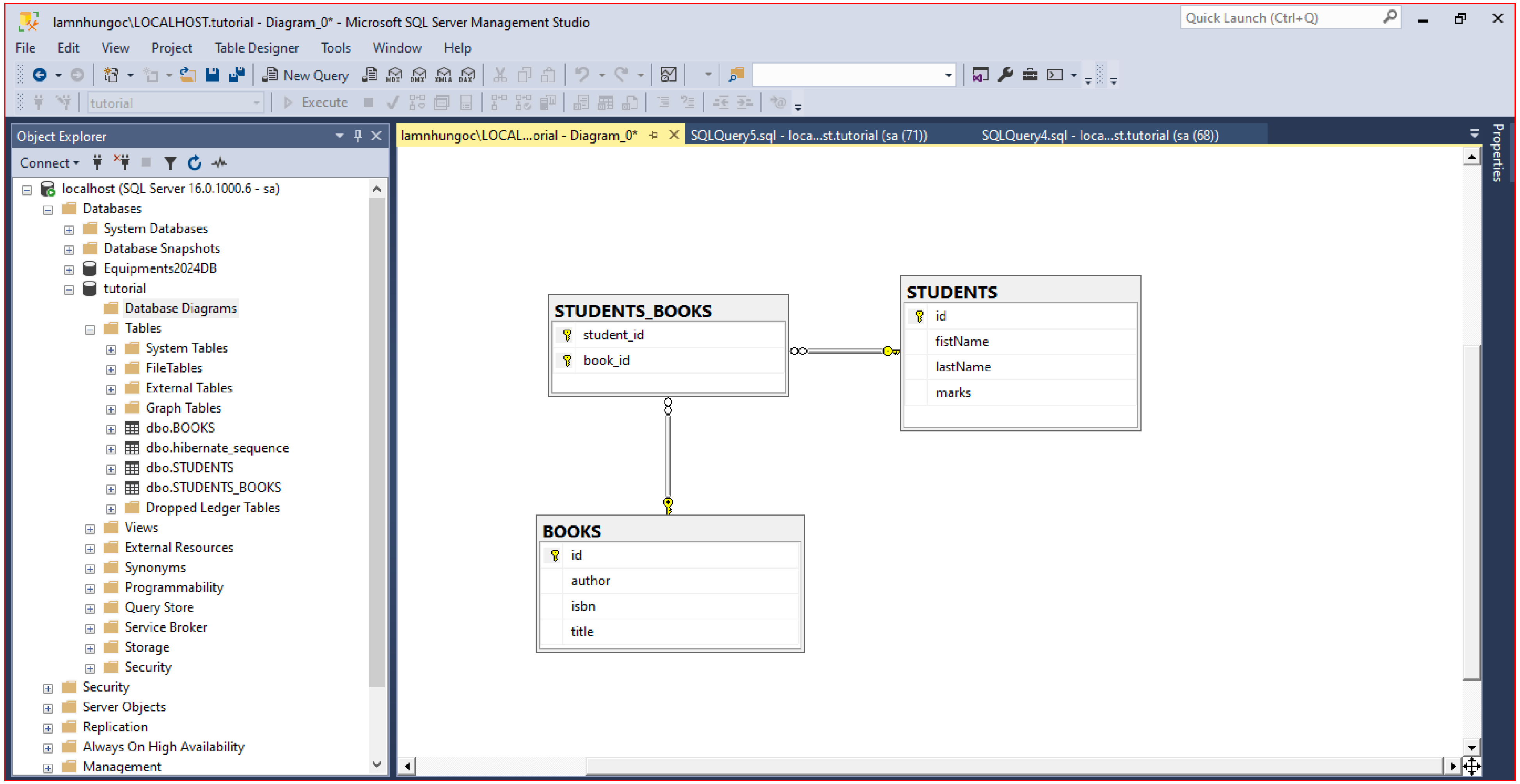
## 3. Run Program



eclipse-workspace - HSF301_Chapter_01_Ex_02_ManyToMany/src/main/java/hsf301/fe/gui/ManyToMany.java - Eclipse IDE

```
package hsf301.fe.gui;

import hsf301.fe.pojo.Book;

public class ManyToMany {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        //Many To
        String fileName = "hibernate.cfg.xml";
        IStudentService studentService = new StudentService(fileName);
        Student student = new Student("Lam","Nguyen",9);
        Book book = new Book("Java Persistence with Spring","Catalin Tudose","9781617299186");
        student.getBooks().add(book);
        studentService.save(student);

    }

}
```

## 4. Result


lamnhungoc\LOCALHOST.tutorial - Diagram_0* - Microsoft SQL Server Management Studio
File
Edit
View
Project
Table Designer
Tools
Window
Help
New Query
Execute
tutorial
Object Explorer
Connect
lamnhungoc\LOCAL...orial - Diagram_0*
SQLQuery5.sql - loca...st.tutorial (sa (71))
SQLQuery4.sql - loca...st.tutorial (sa (68))
Properties

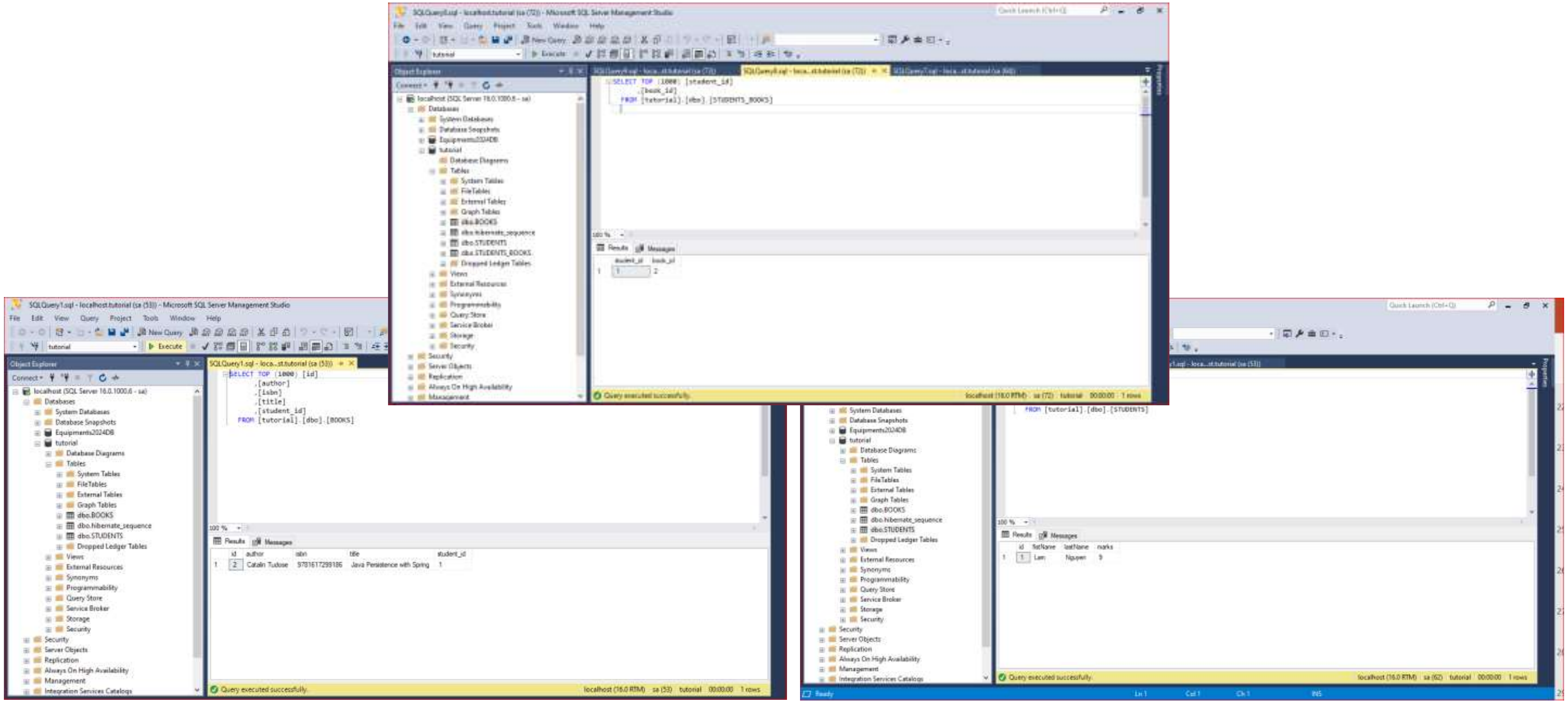
- localhost (SQL Server 16.0.1000.6 - sa)
  - Databases
    - System Databases
    - Database Snapshots
    - Equipments2024DB
    - tutorial
      - Database Diagrams
      - Tables
        - System Tables
        - FileTables
        - External Tables
        - Graph Tables
        - dbo.BOOKS
        - dbo.hibernate_sequence
        - dbo.STUDENTS
        - dbo.STUDENTS_BOOKS
        - Dropped Ledger Tables
      - Views
      - External Resources
      - Synonyms
      - Programmability
      - Query Store
      - Service Broker
      - Storage
      - Security
  - Security
  - Server Objects
  - Replication
  - Always On High Availability
  - Management

STUDENTS_BOOKS: student_id, book_id

STUDENTS: id, fistName, lastName, marks

BOOKS: id, author, isbn, title

## 5. Result



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

```
   student_id  book_id
1  1           2
```

Query executed successfully.    localhost (16.0 RTM)    sa (72)    tutorial    00:00:00    1 rows

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

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

Query executed successfully.    localhost (16.0 RTM)    sa (53)    tutorial    00:00:00    1 rows

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

```
   id  firstName  lastName  marks
1  1   Lam        Nguyen    9
```

Query executed successfully.    localhost (16.0 RTM)    sa (62)    tutorial    00:00:00    1 rows

# Summary

- Concepts were introduced:
  - Overview about Hibernate
  - Architecture Overview new features of Hibernate
  - Explain and demo using Eclipse IDE to create Hibernate Console
  - Create and Run cross-platform Console application with Java connect to MSSQL with Repository Pattern