DAM. UNIT 3. ACCESS USING OBJECT-RELATIONAL MAPPING (ORM). ACCESS TO RELATIONAL DATABASES USING DAO

DAM. Acceso a Datos (ADA) (a distancia en inglés)

Unit 3. ACCESS USING OBJECT-RELATIONAL MAPPING (ORM)

Access to relational databases using DAO

Abelardo Martínez

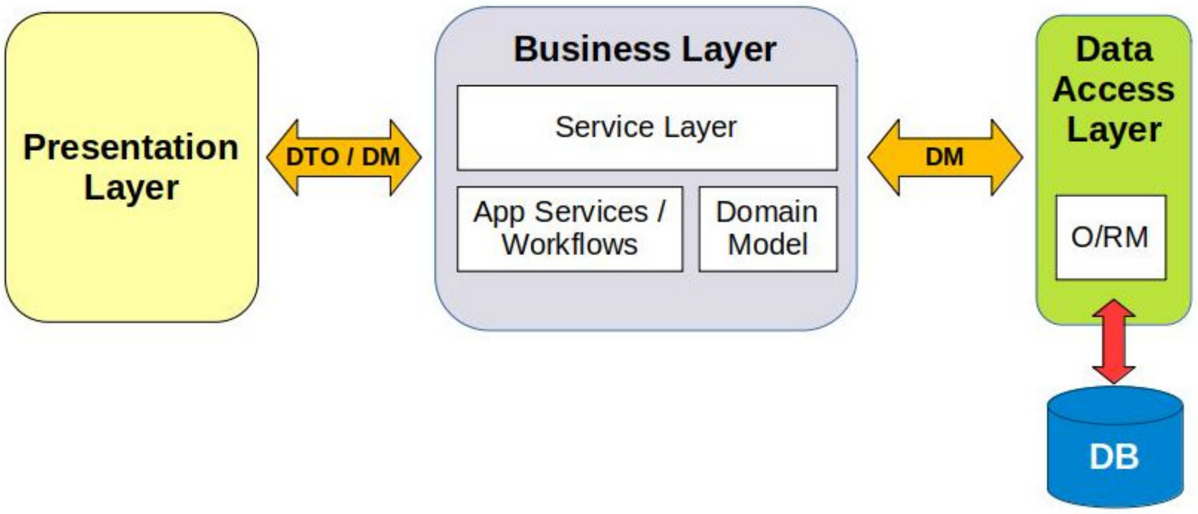
Based and modified from Sergio Badal (www.sergiobadal.com)

Year 2024-2025

1. What is DAO?

So far we have connected to the database using only one class (main) that contained both the connection and the selection statement. Now we will separate the code according to its functionality in order to make a differentiated access by layers with the goal of increasing the reusability of the code.

The **Data Access Object (DAO)** pattern is a structural pattern that allows us work with a database using an abstract API. The API hides from the application all the complexity of performing **CRUD operations** (**C**reate, **R**ead, **U**pdate, **D**elete) in the underlying storage mechanism. This permits both layers to evolve separately without knowing anything about each other.



1.1. Example

To understand how does DAO work, let’s have a look at the next piece of code.

- As you can see, we are working with a database using no SQL instruction by accessing classes and Java/objects elements instead of tables or any database element/object.
- We can either say we are using LAYERS to hide the complexity of the database or, in other words we could say:

“Data Access Object (DAO) pattern is a structural pattern that allows us to isolate the application/business layer from the persistence layer (usually a relational database but could be any other persistence mechanism) using an abstract API”.

Example from: https://www.baeldung.com/java-dao-pattern

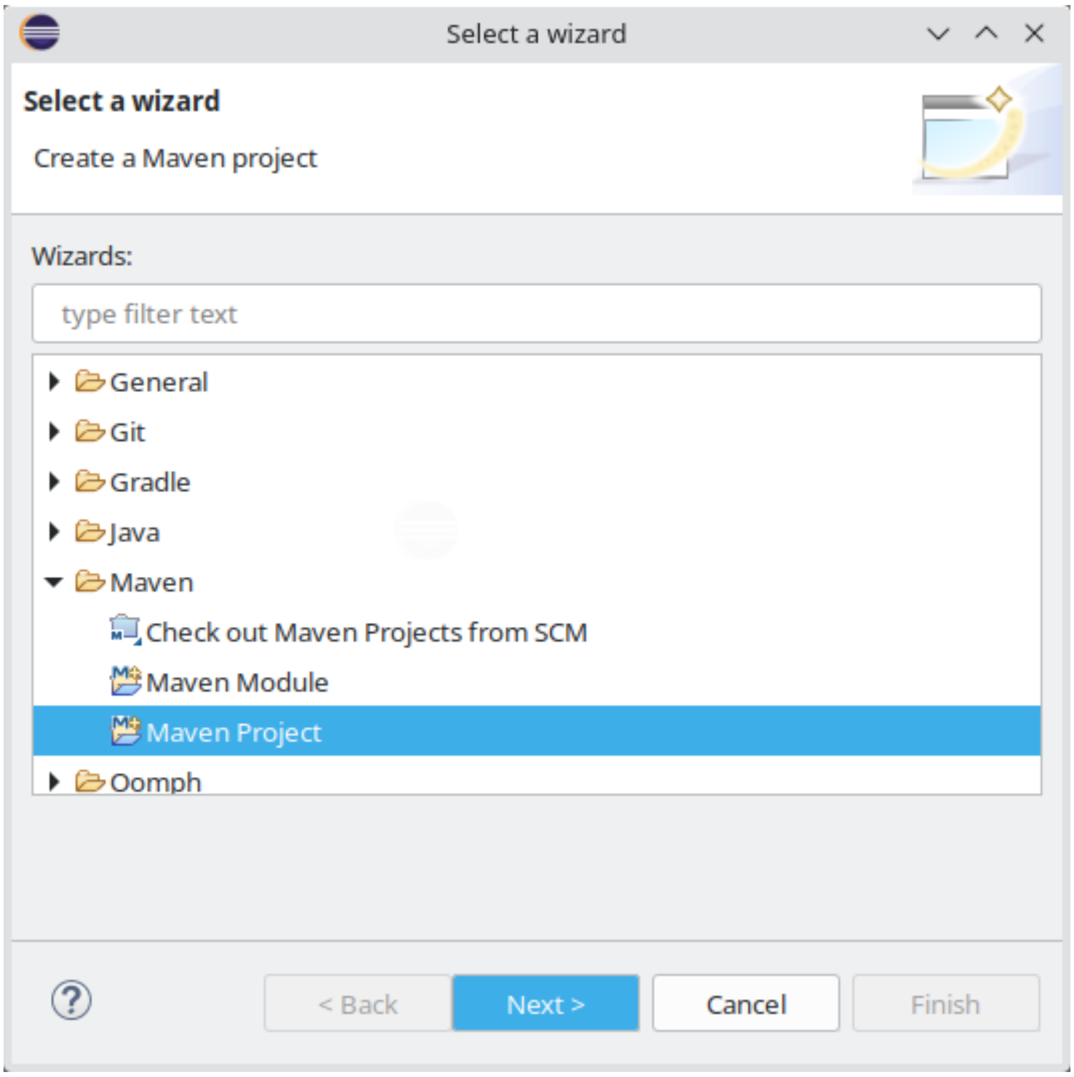
```
public class UserApplication {

    private static Dao<User> jpaUserDao;

    public static void main(String[] args) {
    // Read user with ID = 1
        User user1 = getUser(1);
        System.out.println(user1);
    // Update user with ID = 1
        updateUser(user1, new String[]{"Jake", "jake@domain.com"});
    // Insert new user
        saveUser(new User("Monica", "monica@domain.com"));
    // Delete that new user
        deleteUser(getUser(2));
    // Read all users (SELECT)
     getAllUsers().forEach(user -> System.out.println(user.getName()));
    }

    public static User getUser(long id) {
        Optional<User> user = jpaUserDao.get(id);
        return user.orElseGet(
          () -> new User("non-existing user", "no-email"));
    }

    public static List<User> getAllUsers() {
        return jpaUserDao.getAll();
    }

    public static void updateUser(User user, String[] params) {
        jpaUserDao.update(user, params);
    }

    public static void saveUser(User user) {
        jpaUserDao.save(user);
    }

    public static void deleteUser(User user) {
        jpaUserDao.delete(user);
    }
}
```

Now, we are going to create a Java project to access a relational database from scratch and run CRUD operations, step by step, using DAO. Our chosen IDE will be Eclipse, but you can use whatever IDE you feel comfortable with.

2. Using DAO with Java

2.1. Setting up the project

Create an empty Java Maven Project with these parameters as we saw at UNIT 2:


Select a wizard

Select a wizard

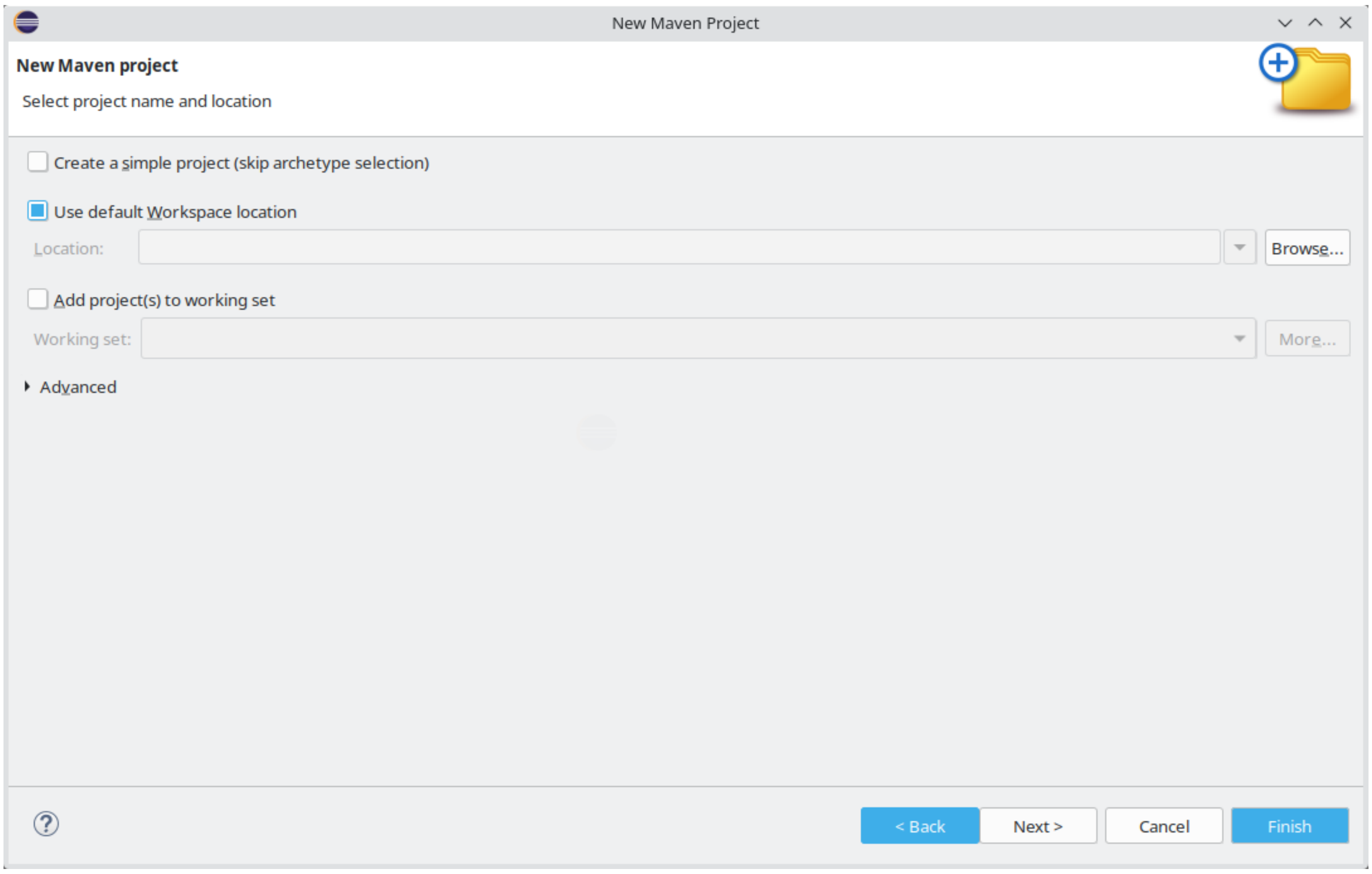
Create a Maven project

Wizards:

type filter text

- General
- Git
- Gradle
- Java
- Maven
  - Check out Maven Projects from SCM
  - Maven Module
  - Maven Project
- Oomph

< Back | Next > | Cancel | Finish



New Maven Project

New Maven project

Select project name and location

[ ] Create a simple project (skip archetype selection)

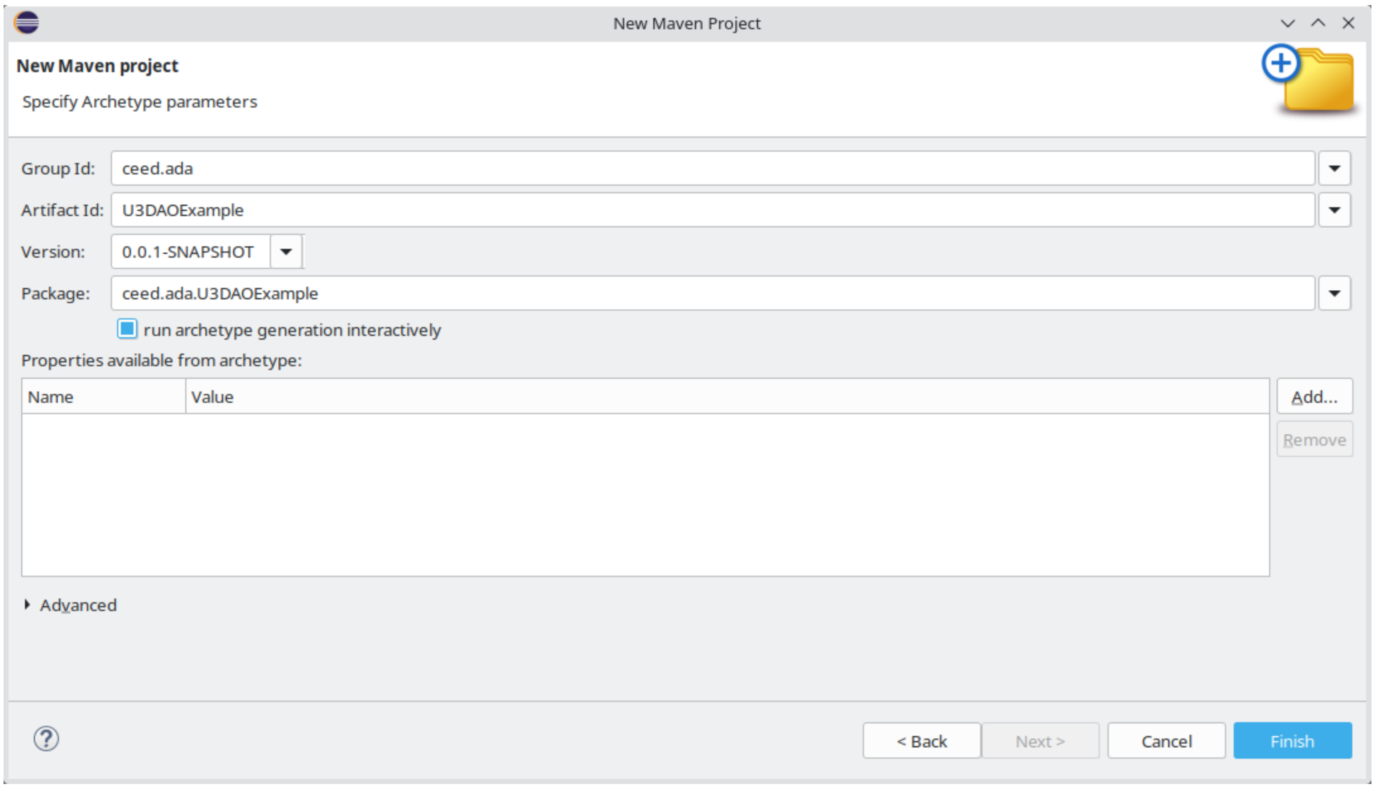
[✓] Use default Workspace location

Location:

[ ] Add project(s) to working set

Working set:

Advanced



New Maven Project

New Maven project

Specify Archetype parameters

Group Id: ceed.ada

Artifact Id: U3DAOExample

Version: 0.0.1-SNAPSHOT

Package: ceed.ada.U3DAOExample

[✓] run archetype generation interactively

Properties available from archetype:

Name	Value

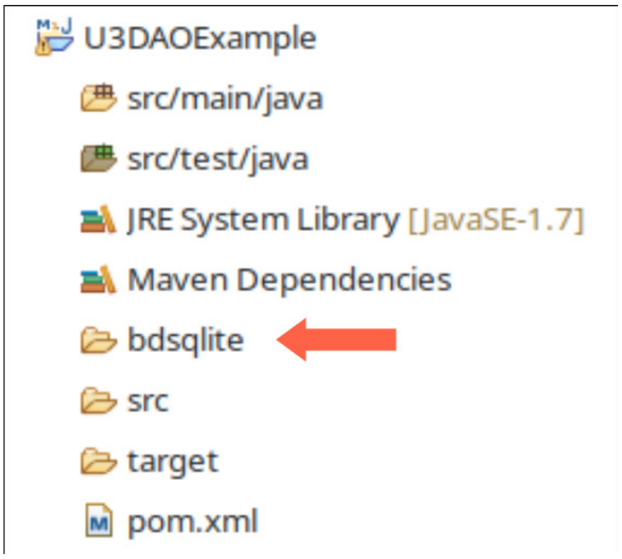
Advanced

2.2. Setting up the database

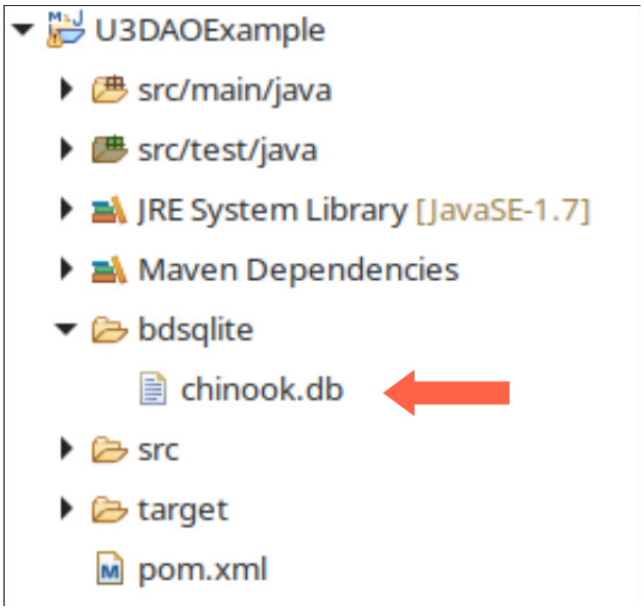
Since the main purpose of this unit is to explain how DAO lets you work with any relational database using layers focusing on Java, we will use one of the simplest database you can find: SQLite.

Follow these steps to create a table in an existing SQLite database:

1. Create a folder inside your project called “bdslqlite”.



2. Go to this URL and download chinook.db: https://www.sqlitetutorial.net/sqlite-sample-database/. Copy the DB to the folder.



3. Install SQLite. https://www.sqlite.org/download.html
4. Go to the console/terminal and execute

```
sqlite3 chinook.db
```

5. Go to this URL if you’re having problems: https://www.sqlitetutorial.net/download-install-sqlite/
6. Inside sqlite console, create PEOPLE table with this command:

```
sqlite> CREATE TABLE IF NOT EXISTS People (
             personID INTEGER PRIMARY KEY,
             name TEXT,
             age INTEGER);
```

7. Check the table is created successfully with .tables command (do not add “.” at the end)

```
sqlite> .tables
```

8. Insert four random registers inside the table, such us:

```
sqlite> INSERT INTO People VALUES (1, 'Sergio Fuentes', 20);
sqlite> INSERT INTO People VALUES (2, 'Juan Hernández', 23);
sqlite> INSERT INTO People VALUES (3, 'Ana Del Río', 34);
sqlite> INSERT INTO People VALUES (4, 'Laura Pérez', 31);
```

9. Check everything went as expected and close SQLite console:

```
sqlite> SELECT * FROM People;
1|Sergio Fuentes|20
2|Juan Hernández|23
```

```
3|Ana Del Río|34
4|Laura Pérez|31
sqlite> .exit
```

2.3. Connecting to the database

Follow the steps described at UNIT 2 to create a Java Maven Project to access a simple **SQLite** database with a table called **People** with the columns:

- **personID**
- **name**
- **age**

---

1. You need to set the dependencies at the **POM file** (be sure to set the right version number)

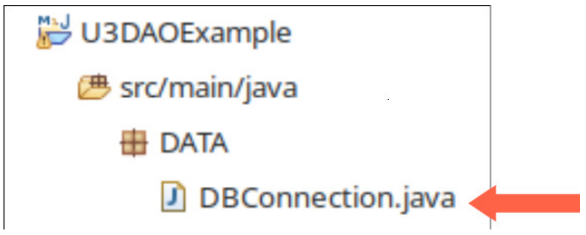
```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>ceed.ada</groupId>
  <artifactId>U3DAOExample</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <name>U3DAOExample</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
```

```
        <scope>test</scope>
      </dependency>
      <!-- https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc -->
      <dependency>
          <groupId>org.xerial</groupId>
          <artifactId>sqlite-jdbc</artifactId>
          <version>3.39.3.0</version>
      </dependency>
    </dependencies>
</project>
```

---

2. Create a new class called **ConnectToDB** inside a new package called DATA, containing:

- The connection string as a static final variable.
- A method to connect to the database.
- Several methods to overload close() predefined methods of every object we're going to use while working with databases such us ResultSet, Statement, PreparedStatement, and Connection.
- The code could be this one:

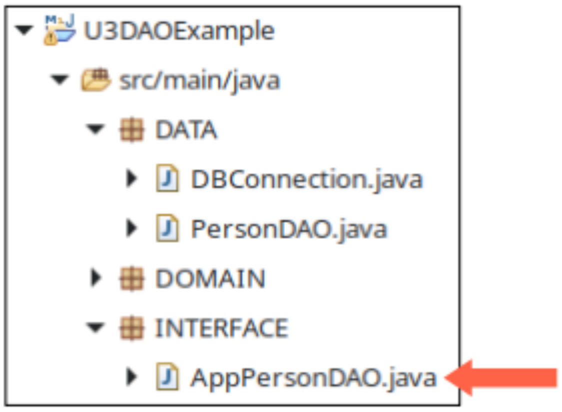


```java
package DATA;

import java.sql.Statement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;


/**
 * ======================================================================
 * Program to manage database connection and closing operations
 * @author Abelardo Martínez. Based and modified from Sergio Badal
 * ======================================================================
 */

public class DBConnection {
    /**
     * ----------------------------------------
     * GLOBAL CONSTANTS AND VARIABLES
     * ----------------------------------------
     */
    //URL connection
    private static final String URL = "jdbc:sqlite:bdsqlite/chinook.db"; //linux version
    //private static final String URL = "jdbc:sqlite:bdsqlite\\chinook.db"; //windows version

    /*
     * ------------------------
     * CONNECTION MANAGEMENT
     * ------------------------
     */
    /*
     * Static method: Just try to connect to the database
     */
    public static Connection ConnectToDB() {
        try {
            Connection cnDB = DriverManager.getConnection(URL);
            return cnDB;
        } catch (SQLException sqle) {
            System.out.println("Something was wrong while trying to connect to the database!");
            sqle.printStackTrace(System.out);
        }
        return null;
    }

    /*
     * Static method: Just try to disconnect to the database
     */
    public static void close(Connection cnDB) throws SQLException {
        cnDB.close();
    }

    /*
     * -------------------
     * CLOSING RESOURCES
     * -------------------
     */
    //ResultSet
    public static void close(ResultSet rsCursor) throws SQLException {
        rsCursor.close();
    }

    //Statement
    public static void close(Statement staSQL) throws SQLException {
        staSQL.close();
    }

    //PreparedStatement
    public static void close(PreparedStatement pstaSQL) throws SQLException {
        pstaSQL.close();
    }

}
```

---

3. Finally, we just need a **main method** to check that the connection is working properly.

- You can create a new package called INTERFACE.



```
U3DAOExample
  src/main/java
    DATA
      DBConnection.java
      PersonDAO.java
    DOMAIN
    INTERFACE
      AppPersonDAO.java
```

- The code could be this one:

> **You MUST understand every line of this code before continuing with this UNIT 3. Go to UNIT 2 extended documentation to review all the required knowledge.**

```
import java.sql.*;
import DATA.DBConnection;

public class TestSQLiteDAO {
    public static void main(String[] stArgs) {
        try {
            Connection cnDB = DBConnection.ConnectToDB();
            Statement staSQL = cnDB.createStatement();
            String stSQLSelect = "SELECT * FROM People";
            System.out.println("Executing: " + stSQLSelect);
            ResultSet rsPerson = staSQL.executeQuery(stSQLSelect);
            int iNumItems = 0;
            while (rsPerson.next()) {
                iNumItems++;
                System.out.println("ID: " + rs.getInt("personID"));
                System.out.println("Name: " + rs.getString("name"));
                System.out.println("Age: " + rs.getInt("age"));
            }
            if (iNumItems == 0)
                System.out.println("No items found on table People");
            ConnectToDB.close(rsPerson); //close resultset
            ConnectToDB.close(cnDB); //close connection to the DB
        } catch (SQLException sqle) {
            System.out.println("Something was wrong while reading!");
            sqle.printStackTrace(System.out);
        }

    }
}
```

2.4. Creating classes and reading data

Now we need to create two classes for every entity/table of the DB:

A) Person.java to store the data of every row of the database.

- Create this class inside a new package called **DOMAIN**.
- Define the attributes, setters, getters and 4 constructors:
  - Empty constructor
  - Constructor that provides only the ID (e.g. to delete we only need to have the personId, which is the primary key -PK-)
  - Constructor to insert new record (we don't need to specify the personId)
  - Constructor to modify a record (needs all attributes)

```
package DOMAIN;

/**
 * =====================================================================
 * Object Person. Class
 * @author Abelardo Martínez. Based and modified from Sergio Badal
 * =====================================================================
 */

public class Person {

    /*
     * ------------------------------------------
     * ATTRIBUTES
     * ------------------------------------------
     */
    private int ipersonID;
    private String stName;
    private int iAge;

    /*
     * ------------------------------------------
     * METHODS
     * ------------------------------------------
     */
    /*
     * Empty constructor
     */
    public Person() {
    }
    /*
     * ID constructor. Only primary key
     */
    public Person(int ipersonID) {
        this.ipersonID = ipersonID;
    }
    /*
     * Constructor without ID. All fields, except primary key
     */
    public Person(String stName, int iAge) {
        this.stName = stName;
        this.iAge = iAge;
    }
    /*
     * Full constructor with all fields
     */
    public Person(int ipersonID, String stName, int iAge) {
        this.ipersonID = ipersonID;
        this.stName = stName;
        this.iAge = iAge;
    }

    /*
     * ------------------------------------------
     * GETTERS
     * ------------------------------------------
     */
    public int getPersonID() {
        return ipersonID;
    }

    public int getAge() {
        return iAge;
    }

    public String getName() {
        return stName;
    }

    /*
```

```
     * ------------------------------------------
     * SETTERS
     * ------------------------------------------
     */
    public void setPersonID(int ipersonID) {
        this.ipersonID = ipersonID;
    }

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

    public void setAge(int iAge) {
        this.iAge = iAge;
    }

    /*
     * --------------------
     * FORMAT DATA METHODS
     * --------------------
     */

    @Override
    public String toString() {
        return "Person [ID = " + ipersonID + ", Name = " + stName + ", Age = " + iAge + "]";
    }
}
```

B) PersonDAO.java to manage the operations between the database and the above class

- Create this class inside a new package called data
- Define the DB access statements at the beginning of the class as static variables (good practice)
- Create a method to read all the items from this entity

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import DOMAIN.Person;

/**
 * ======================================================================
 * Program to manage CRUD operations to the DB
 * @author Abelardo Martínez. Based and modified from Sergio Badal
 * ======================================================================
 */

public class PersonDAO {

    /**
     * ----------------------------------------
     * GLOBAL CONSTANTS AND VARIABLES
     * ----------------------------------------
     */
    /*
     * ------------------------
     * SQL QUERIES
     * ------------------------
     */
    private static final String SELECT = "SELECT * FROM People";

    /*
     * ------------------
     * SELECT
     * ------------------
     */
    public ArrayList<Person> selectAllPeople() throws SQLException {

        Connection cnDB = null;
        PreparedStatement pstaSQLSelect = null;
        ResultSet rsPerson = null;
        Person objPerson = null;
        ArrayList<Person> arlPerson = new ArrayList<Person>();

        try {
            cnDB = DBConnection.ConnectToDB(); //connect to DB
            pstaSQLSelect = cnDB.prepareStatement(SELECT); //select statement
            rsPerson = pstaSQLSelect.executeQuery(); //execute select
            while (rsPerson.next()) {
                int ipersonID = rsPerson.getInt("personID");
                String stName = rsPerson.getString("name");
                int iAge = rsPerson.getInt("age");
                objPerson = new Person(ipersonID, stName, iAge);
                arlPerson.add(objPerson);
            }
        } catch (SQLException sqle) {
            System.out.println("Something was wrong while reading from people table");
            sqle.printStackTrace(System.out);
        } finally {
            DBConnection.close(rsPerson); //close resultset
            DBConnection.close(pstaSQLSelect); //close preparedstatement
            DBConnection.close(cnDB); //close connection to the DB
        }
        return arlPerson;
    }
}
```

Finally, we need to use the class **TestSQLiteDAO** to create an instance of **PersonDAO** and just list the records of the table **People**. **Notice no exception handling is required since they are handled inside DAO classes**.

```
import java.sql.*;
import java.util.ArrayList;

import DATA.*;
import DOMAIN.*;

public class TestSQLiteDAO {
    /*
     * -----------------
     * MAIN PROGRAMME
     * -----------------
     */
    public static void main(String[] stArgs) throws SQLException {

        PersonDAO objPersonDAO = new PersonDAO();
        ArrayList<Person> arlPerson;

        // SELECT
        System.out.println("**** LIST INITIAL RECORDS");
        arlPerson = objPersonDAO.selectAllPeople();
        for (Person objPerson : arlPerson) {
            System.out.println("Person: " + objPerson);
        }
    }
}
```

2.5. Inserting data

To allow inserting data we just need to create another method inside DAO class and another SQL sentence. The big difference now is we need to set some “markers“ to add values later on:

```java
private static final String INSERT = "INSERT INTO People (name, age) VALUES (?,?)";

/*
 * ------------------
 * INSERT
 * ------------------
 */
public int insertPerson(Person objPerson) throws SQLException {

    Connection cnDB = null;
    PreparedStatement pstaSQLInsert = null;
    int iNumreg = 0;

    try {
        cnDB = DBConnection.ConnectToDB(); //connect to DB
        pstaSQLInsert = cnDB.prepareStatement(INSERT); //insert statement
        pstaSQLInsert.setString(1, objPerson.getName());
        pstaSQLInsert.setInt(2, objPerson.getAge());
        iNumreg = pstaSQLInsert.executeUpdate(); //execute insert
    } catch (SQLException sqle) {
        System.out.println("Something wrong while inserting!");
        sqle.printStackTrace(System.out);

    } finally {
        DBConnection.close(pstaSQLInsert); //close preparedstatement
        DBConnection.close(cnDB); //close connection to the DB
    }
    return iNumreg;
}
```

2.6. Deleting data

To allow deleting data we just need to create another method inside DAO class and another SQL sentence. We also need to set some “markers“ to add values later on:

```
private static final String DELETE = "DELETE FROM People WHERE personID = ?";

/*
 * ------------------
 * DELETE
 * ------------------
 */
public int deletePerson(Person objPerson) throws SQLException {

    Connection cnDB = null;
    PreparedStatement pstaSQLDelete = null;
    int iNumreg = 0;

    try {
        cnDB = DBConnection.ConnectToDB(); //connect to DB
        pstaSQLDelete = cnDB.prepareStatement(DELETE); //delete statement
        pstaSQLDelete.setInt(1, objPerson.getPersonID());
        iNumreg = pstaSQLDelete.executeUpdate(); //execute delete
    } catch (SQLException sqle) {
        System.out.println("Something wrong while deleting!");
        sqle.printStackTrace(System.out);
    } finally {
        DBConnection.close(pstaSQLDelete); //close preparedstatement
        DBConnection.close(cnDB); //close connection to the DB
    }
    return iNumreg;
}
```

2.7. Updating data

To allow updating data we just need to create another method inside DAO class and another SQL sentence. We also need to set some “markers“ to add values later on:

```java
private static final String UPDATE = "UPDATE People SET name = ?, age = ? WHERE personID= ?";

/*
 * -----------------
 * UPDATE
 * -----------------
 */
public int updatePerson(Person objPerson) throws SQLException {

    Connection cnDB = null;
    PreparedStatement pstaSQLUpdate = null;
    int iNumreg = 0;

    try {
        cnDB = DBConnection.ConnectToDB(); //connect to DB
        pstaSQLUpdate = cnDB.prepareStatement(UPDATE); //update statement
        pstaSQLUpdate.setString(1, objPerson.getName());
        pstaSQLUpdate.setInt(2, objPerson.getAge());
        pstaSQLUpdate.setInt(3, objPerson.getPersonID());
        iNumreg = pstaSQLUpdate.executeUpdate(); //execute delete
    } catch (SQLException sqle) {
        System.out.println("Something wrong while updating!");
        sqle.printStackTrace(System.out);
    } finally {
        DBConnection.close(pstaSQLUpdate); //close preparedstatement
        DBConnection.close(cnDB); //close connection to the DB
    }
    return iNumreg;
}
```

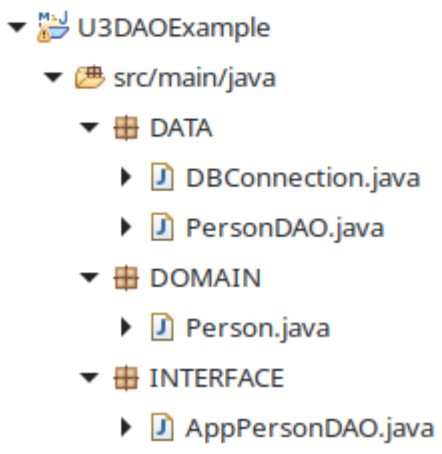
2.8. CRUD operations (final DAO class)

If we put all the methods together we can get something like this:

```
package DATA;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

import DOMAIN.Person;

/**
 * ======================================================================
 * Program to manage CRUD operations to the DB
 * @author Abelardo Martínez. Based and modified from Sergio Badal
 * ======================================================================
 */

public class PersonDAO {

    /**
     * ----------------------------------------
     * GLOBAL CONSTANTS AND VARIABLES
     * ----------------------------------------
     */
    /*
     * ------------------------
     * SQL QUERIES
     * ------------------------
     */
    private static final String SELECT = "SELECT * FROM People";
    private static final String INSERT = "INSERT INTO People (name, age) VALUES (?,?)";
    private static final String DELETE = "DELETE FROM People WHERE personID = ?";
    private static final String UPDATE = "UPDATE People SET name = ?, age = ? WHERE personID= ?";

    /**
     * ----------------------------------------
     * CRUD OPERATIONS
     * ----------------------------------------
     */
    /*
     * ------------------
     * SELECT
     * ------------------
     */
    public ArrayList<Person> selectAllPeople() throws SQLException {

        Connection cnDB = null;
        PreparedStatement pstaSQLSelect = null;
        ResultSet rsPerson = null;
        Person objPerson = null;
        ArrayList<Person> arlPerson = new ArrayList<Person>();

        try {
            cnDB = DBConnection.connectToDB(); //connect to DB
            pstaSQLSelect = cnDB.prepareStatement(SELECT); //select statement
            rsPerson = pstaSQLSelect.executeQuery(); //execute select
            while (rsPerson.next()) {
                int ipersonID = rsPerson.getInt("personID");
                String stName = rsPerson.getString("name");
                int iAge = rsPerson.getInt("age");
                objPerson = new Person(ipersonID, stName, iAge);
                arlPerson.add(objPerson);
            }
        } catch (SQLException sqle) {
            System.out.println("Something was wrong while reading from people table");
            sqle.printStackTrace(System.out);
        } finally {
            DBConnection.close(rsPerson); //close resultset
            DBConnection.close(pstaSQLSelect); //close preparedstatement
            DBConnection.close(cnDB); //close connection to the DB
        }
        return arlPerson;
    }

    /*
     * ------------------
     * INSERT
     * ------------------
     */
    public int insertPerson(Person objPerson) throws SQLException {

        Connection cnDB = null;
        PreparedStatement pstaSQLInsert = null;
        int iNumreg = 0;

        try {
            cnDB = DBConnection.connectToDB(); //connect to DB
            pstaSQLInsert = cnDB.prepareStatement(INSERT); //insert statement
            pstaSQLInsert.setString(1, objPerson.getName());
            pstaSQLInsert.setInt(2, objPerson.getAge());
            iNumreg = pstaSQLInsert.executeUpdate(); //execute insert
        } catch (SQLException sqle) {
            System.out.println("Something wrong while inserting!");
            sqle.printStackTrace(System.out);

        } finally {
            DBConnection.close(pstaSQLInsert); //close preparedstatement
            DBConnection.close(cnDB); //close connection to the DB
        }
        return iNumreg;
    }

    /*
     * ------------------
     * DELETE
     * ------------------
     */
    public int deletePerson(Person objPerson) throws SQLException {

        Connection cnDB = null;
        PreparedStatement pstaSQLDelete = null;
        int iNumreg = 0;

        try {
            cnDB = DBConnection.connectToDB(); //connect to DB
            pstaSQLDelete = cnDB.prepareStatement(DELETE); //delete statement
            pstaSQLDelete.setInt(1, objPerson.getPersonID());
            iNumreg = pstaSQLDelete.executeUpdate(); //execute delete
        } catch (SQLException sqle) {
            System.out.println("Something wrong while deleting!");
            sqle.printStackTrace(System.out);
        } finally {
            DBConnection.close(pstaSQLDelete); //close preparedstatement
            DBConnection.close(cnDB); //close connection to the DB
        }
        return iNumreg;
    }

    /*
```

```
     * -----------------
     * UPDATE
     * -----------------
     */
    public int updatePerson(Person objPerson) throws SQLException {

        Connection cnDB = null;
        PreparedStatement pstaSQLUpdate = null;
        int iNumreg = 0;

        try {
            cnDB = DBConnection.connectToDB(); //connect to DB
            pstaSQLUpdate = cnDB.prepareStatement(UPDATE); //update statement
            pstaSQLUpdate.setString(1, objPerson.getName());
            pstaSQLUpdate.setInt(2, objPerson.getAge());
            pstaSQLUpdate.setInt(3, objPerson.getPersonID());
            iNumreg = pstaSQLUpdate.executeUpdate(); //execute delete
        } catch (SQLException sqle) {
            System.out.println("Something wrong while updating!");
            sqle.printStackTrace(System.out);
        } finally {
            DBConnection.close(pstaSQLUpdate); //close preparedstatement
            DBConnection.close(cnDB); //close connection to the DB
        }
        return iNumreg;
    }

}
```

Your project should look like this:



3. Full CRUD example using DAO

Now that we have all methods defined at DAO class we can operate with the database. Have a look at this simple CRUD example:

```java
package INTERFACE;

import java.sql.*;
import java.util.ArrayList;

import DATA.*;
import DOMAIN.*;

/**
 * ======================================================================
 * Program to manage PERSONS
 * @author Abelardo Martínez. Based and modified from Sergio Badal
 * ======================================================================
 */

public class AppPersonDAO {
    /*
     * -----------------
     * MAIN PROGRAMME
     * -----------------
     */
    public static void main(String[] stArgs) throws SQLException {

        PersonDAO objPersonDAO = new PersonDAO();
        ArrayList<Person> arlPerson;

        // SELECT
        System.out.println("**** LIST INITIAL RECORDS");
        arlPerson = objPersonDAO.selectAllPeople();
        for (Person objPerson : arlPerson) {
            System.out.println(objPerson);
        }

        // INSERT
        System.out.println("**** INSERT NEW RECORDS");
        Person objPerson1 = new Person("Herminia Fuster", 55);
        objPersonDAO.insertPerson(objPerson1);
        Person objPerson2 = new Person("Mario Caballero", 25);
        objPersonDAO.insertPerson(objPerson2);
        // DELETE
        System.out.println("**** DELETE FIRST RECORD");
        Person objPersonToDelete = new Person(1);
        objPersonDAO.deletePerson(objPersonToDelete);

        // UPDATE
        System.out.println("**** UPDATE SECOND RECORD");
        Person objPersonToUpdate = new Person(2, "Juan Manuel Hernández", 22);
        objPersonDAO.updatePerson(objPersonToUpdate);

        //FINAL DATA RESULT
        System.out.println("**** LIST FINAL RECORDS");
        arlPerson = objPersonDAO.selectAllPeople();
        for (Person objPerson : arlPerson) {
            System.out.println(objPerson);
        }
    }
}
```

4. Bibliography

Sources

- How to Create a Maven Project in Eclipse. https://www.simplilearn.com/tutorials/maven-tutorial/maven-project-in-eclipse
- Josep Cañellas Bornas, Isidre Guixà Miranda. Accés a dades. Desenvolupament d'aplicacions multiplataforma. Creative Commons. Departament d'Ensenyament, Institut Obert de Catalunya. Dipòsit legal: B. 29430-2013. https://ioc.xtec.cat/educacio/recursos
- Alberto Oliva Molina. Acceso a datos. UD 3. Herramientas de mapeo objeto relacional (ORM). IES Tubalcaín. Tarazona (Zaragoza, España).
- JavaTPoint. POJO. https://www.javatpoint.com/pojo-in-java
- SQLite Tutorial. SQLite Sample Database. https://www.sqlitetutorial.net/sqlite-sample-database/