GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

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

## Access to relational databases using DAO

**Acceso a Datos (ADA)** **(a distancia en inglés)**
**CFGS Desarrollo de Aplicaciones Multiplataforma (DAM)**

**Abelardo Martínez**
**Year 2024-2025**

# Credits

- Notes made by Abelardo Martínez.
- Based and modified from Sergio Badal (www.sergiobadal.com).
- The images and icons used are protected by the LGPL licence and have been obtained from:
  - https://commons.wikimedia.org/wiki/Crystal_Clear
  - https://www.openclipart.org

# Contents



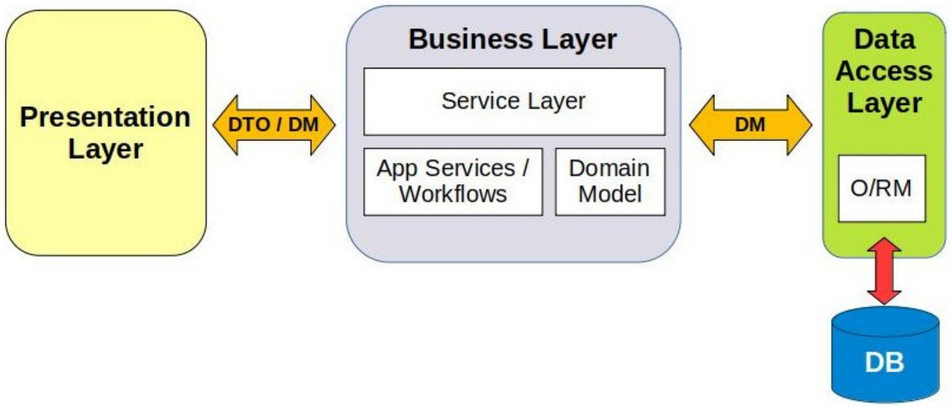
# 1. WHAT IS DAO?

# 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 to 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.

**Data Access Object (DAO)** pattern is a structural pattern that allows us to isolate the application/business layer from the persistence layer using an abstract API.



# DAO. Simple example

Have a look to this piece of code and try to understand what it's doing.

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 are using layers: using DAO!

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);
    }
}
```

# DAO. Full example

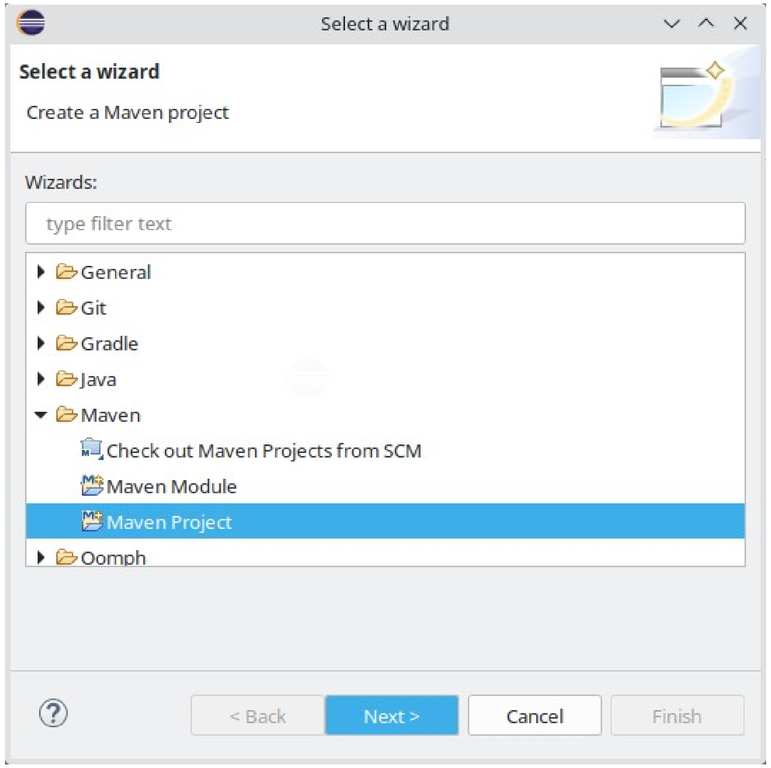
We’re now building a Java project from scratch to create a CRUD application to work with a table called “People” within a simple SQLite database using DAO.

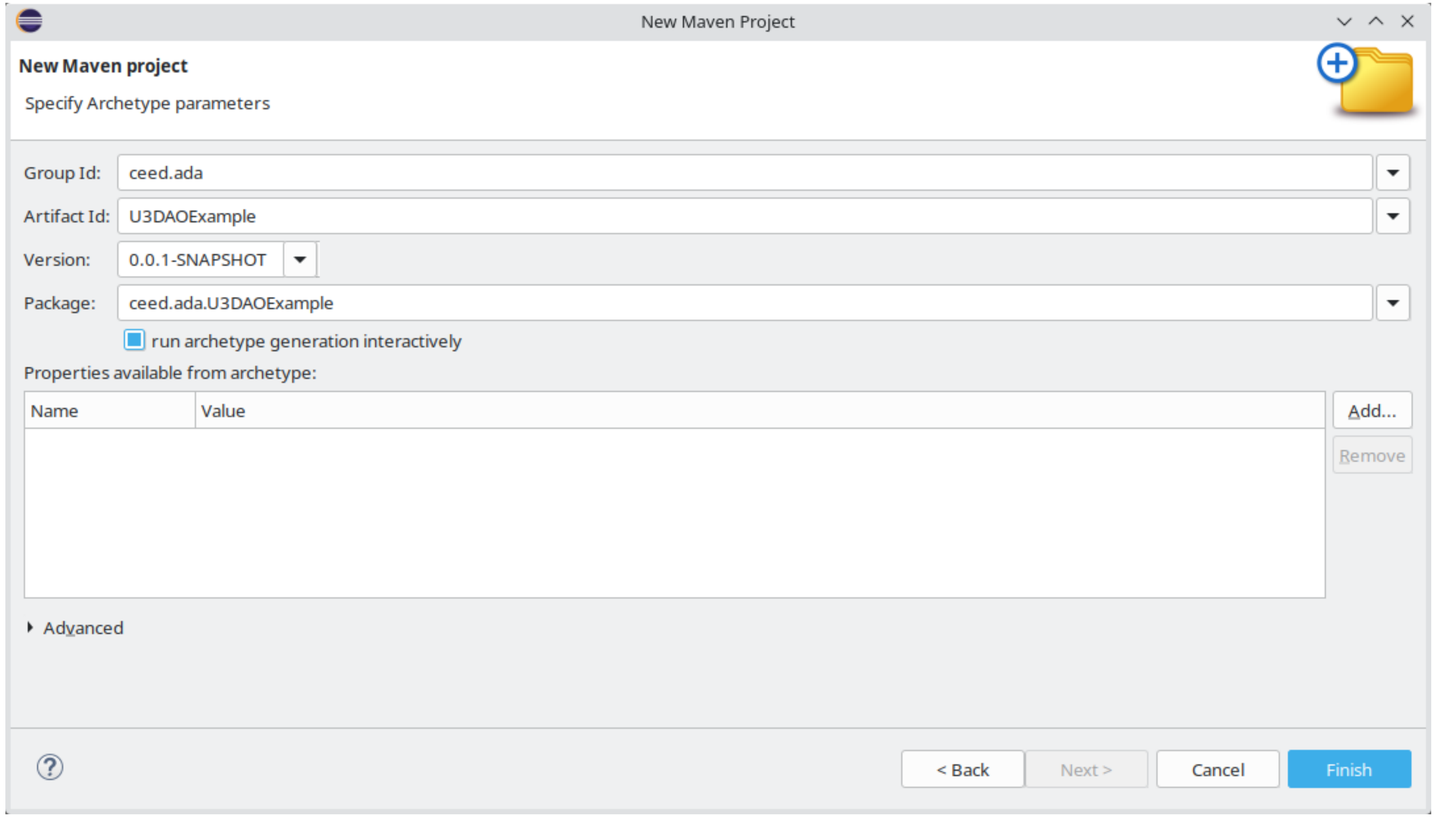
**Get in!**

# 2. DAO: SETTING UP THE PROJECT

# The (Java-Maven) project

- In ECLIPSE, create an empty Java Maven Project with these parameters as we saw at UNIT 2:



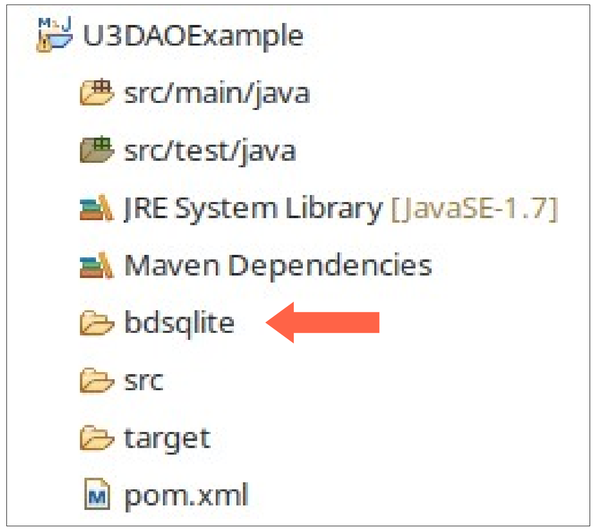



# 3. DAO: SETTING UP THE DATABASE

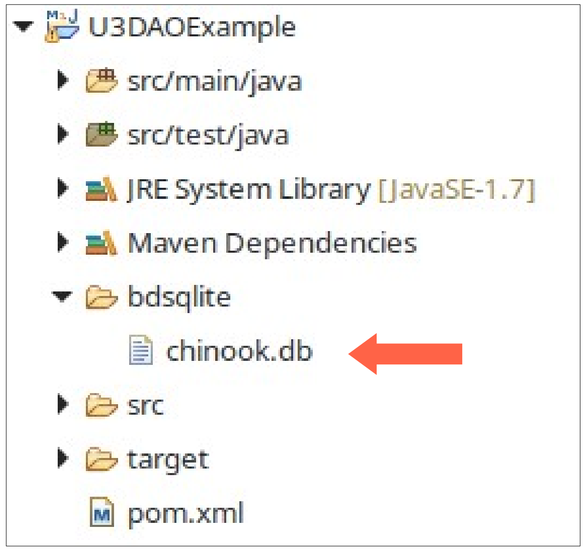
# The (SQLite) database

- 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 “bdsqlite”




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



# The (SQLite) database

Follow the steps you can find at the extended documentation to achieve this:

1) Install SQLite and access to its console
2) Create a new table called People with personID, name, age as columns
3) Insert four random registers

```
sqlite> CREATE TABLE People (
personID    INTEGER PRIMARY KEY,
name        TEXT,
age         INTEGER
);
```

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

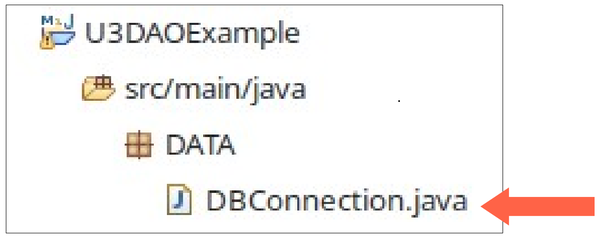
# 4. DAO: CONNECTING TO THE DATABASE

# The (Database) connection

**(1 of 3)** Add the dependencies to the Maven Project (pom.xml) for SQLite (check your version)

```xml
<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>
```

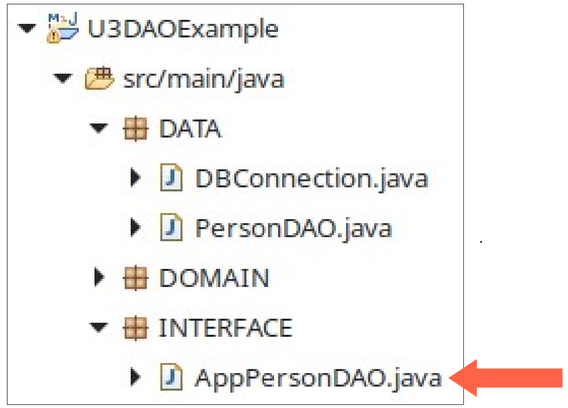
**(2 of 3)** Create a class inside a new **package** called **data** to manage the connection and the “closing” operations.




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


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();
    }

}
```

# The (Database) connection

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




Output:

```
ID: 1
Name: Sergio Fuentes
Age: 20
ID: 2
Name: Juan Hernández
Age: 23
ID: 3
Name: Ana Del Río
Age: 34
ID: 4
Name: Laura Pérez
Age: 31
```

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

public class AppPersonDAO {
   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: " + rsPerson.getInt("personID"));
            System.out.println("Name: " + rsPerson.getString("name"));
            System.out.println("Age: " + rsPerson.getInt("age"));
         }
         if (iNumItems == 0)
            System.out.println("No items found on table People");
         DBConnection.close(rsPerson); //close resultset
         DBConnection.close(cnDB); //close connection to the DB
      } catch (SQLException sqle) {
         System.out.println("Something was wrong while reading!");
         sqle.printStackTrace(System.out);
      }

   }
}
```

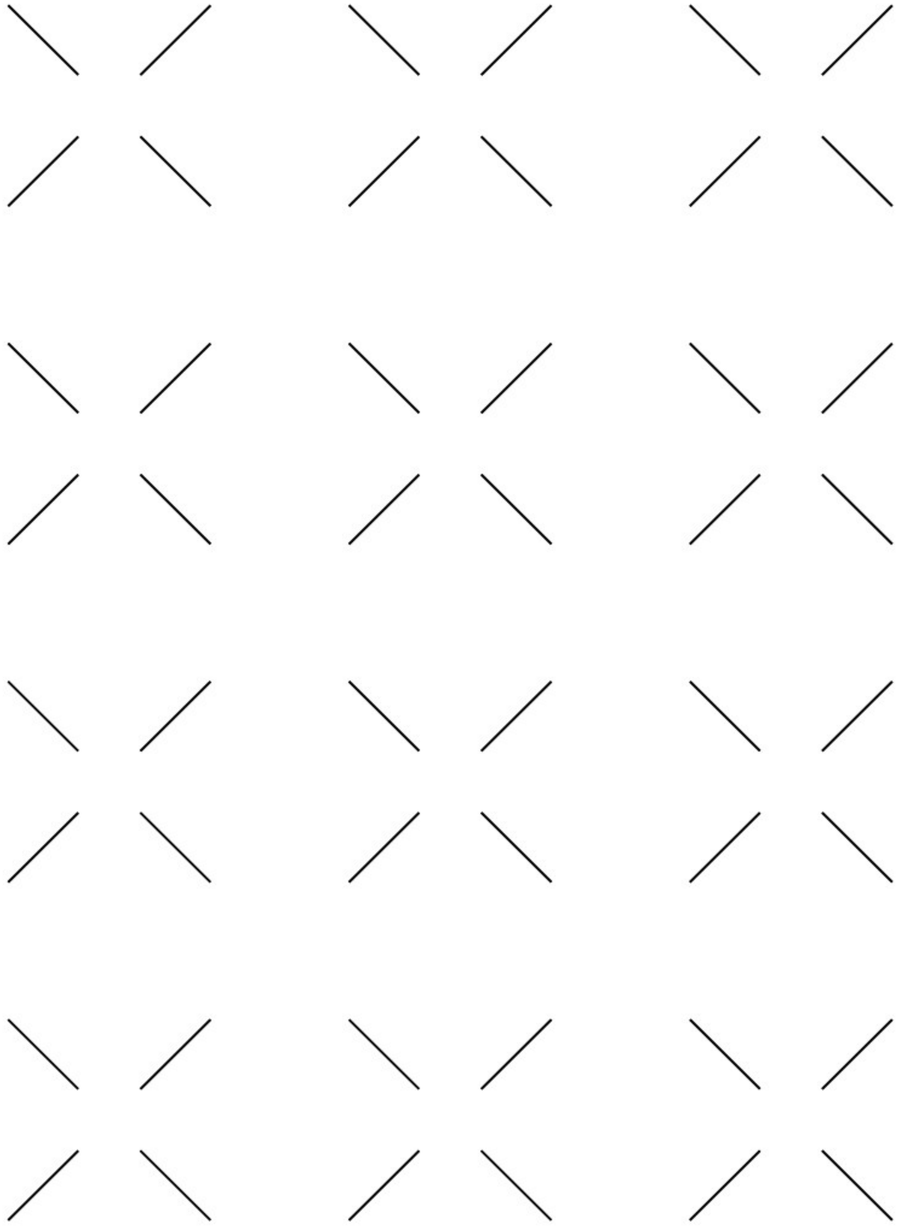
# The (Database) connection

You MUST understand every line of the code provided so far before going further.

Go to UNIT 2 extended documentation to review all the required knowledge.

# 5. CREATING MAIN CLASSES

# 5.1 Main classes to store the data

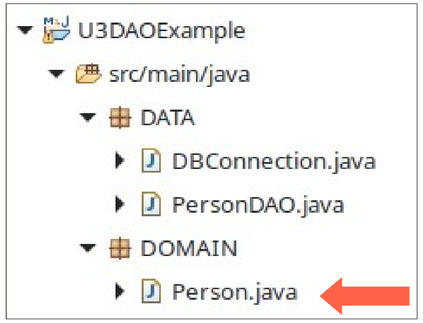


# Creating main classes

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

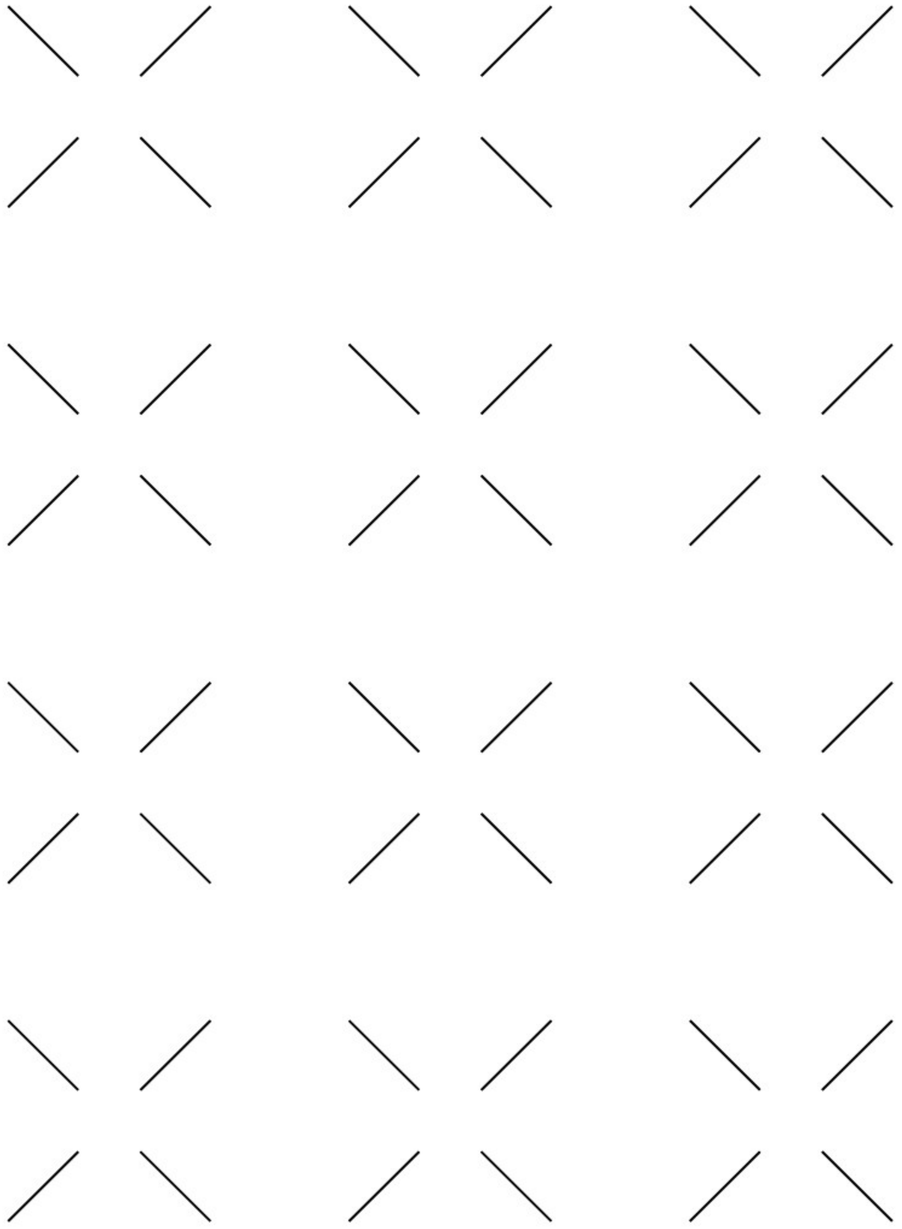
**(1 of 2) Person.java** to store the data of every row of the database.

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




```java
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 + "]";
	}
}
```

## 5.2 Main classes to manage the data

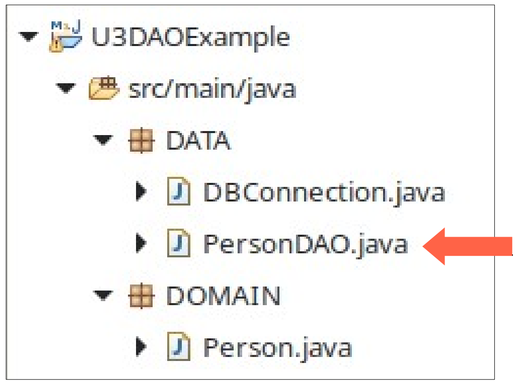


# Creating main classes

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

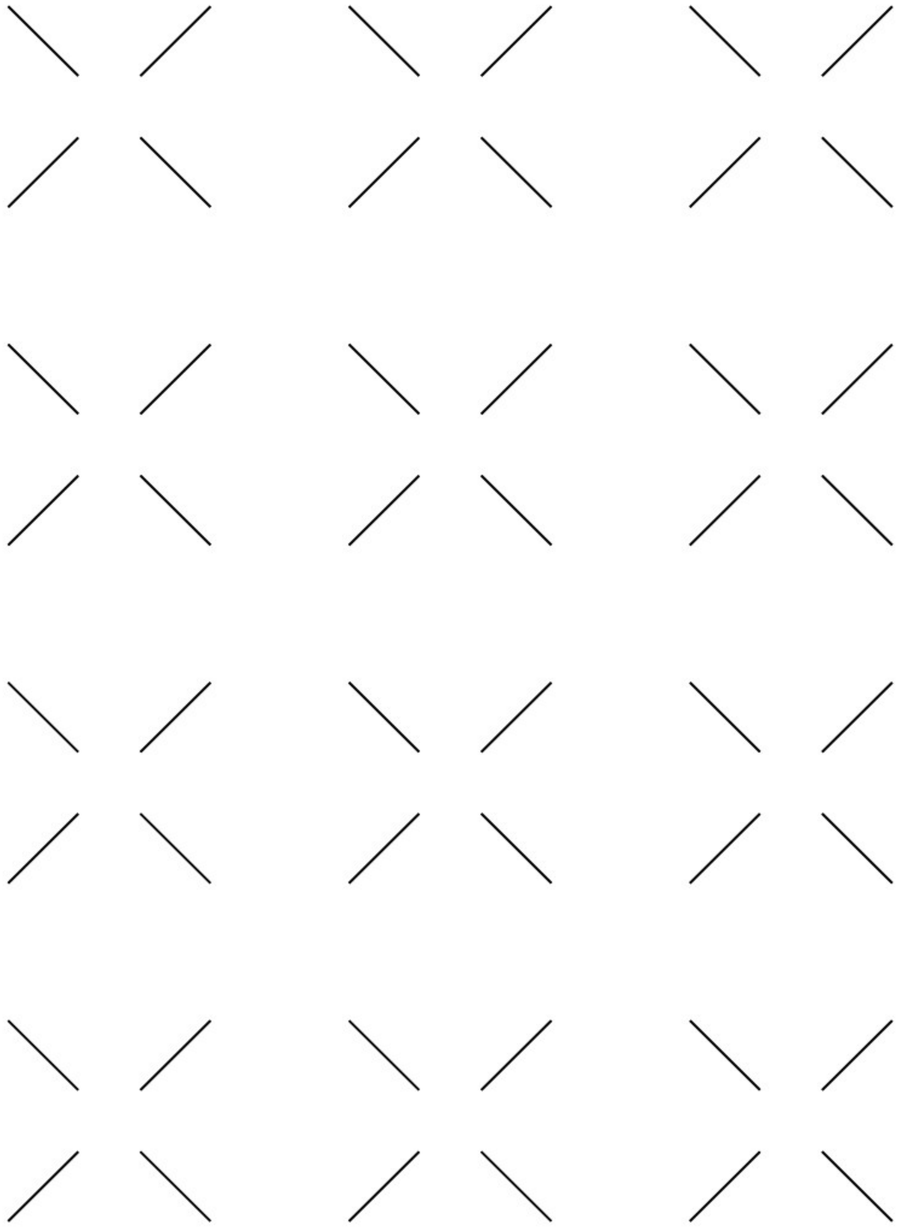
**(2 of 2) PersonDAO.java** to manage the operations between the database and the above class

- 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




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

import DOMAIN.Person;


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;
    }
}
```

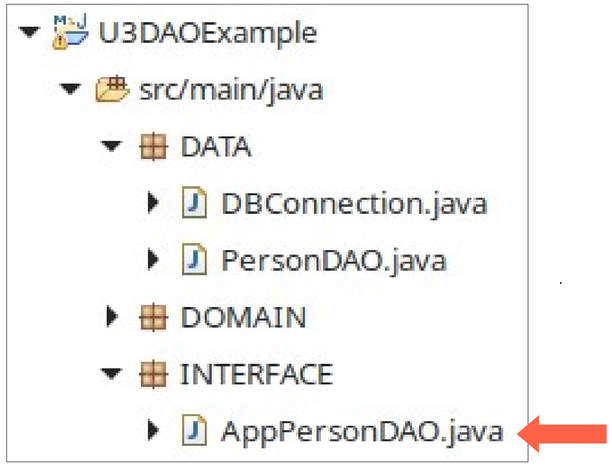
## 5.3 Main class to apply DAO



# Interface class

Finally, we need to use the class **TestSQLiteDAO** to create an instance of **PersonDAO** and just list the records of the table “People” (via **Person**).

Notice no exception handling is required since they are handled inside DAO classes.




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

import DATA.*;
import DOMAIN.*;

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("Person: " + objPerson);
      }
   }
}
```

Output:

```
**** LIST INITIAL RECORDS
Person [ID = 1, Name = Sergio Fuentes, Age = 20]
Person [ID = 2, Name = Juan Hernández, Age = 23]
Person [ID = 3, Name = Ana Del Río, Age = 34]
Person [ID = 4, Name = Laura Pérez, Age = 31]
```

# 6. DAO: INSERTING, UPDATING & DELETING DATA

# Inserting data

We just need to add more methods to **PersonDAO**.

```
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;
}
```

# Deleting data

We just need to add more methods to **PersonDAO**.

```
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;
}
```

# Updating data

We just need to add more methods to **PersonDAO**.

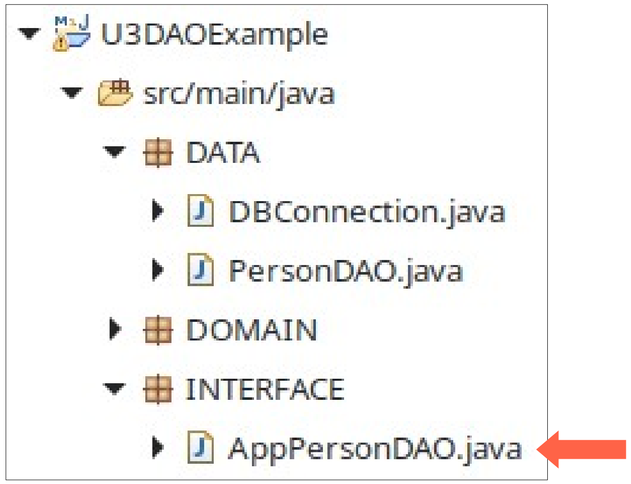
```
private static final String UPDATE = "UPDATE People SET name = ?, age = ? WHERE personID= ?";

/*
 * ------------------
 * 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;
}
```

# 7. CRUD USING DAO

# CRUD using DAO

If we put all the methods together and we add some CRUD operations, we can get this:

Notice no exception handling is required since they are handled inside DAO classes.




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

import DATA.*;
import DOMAIN.*;

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("Person: " + 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("Person: " + objPerson);
        }
    }
}
```

# 8. ACTIVITIES FOR NEXT WEEK

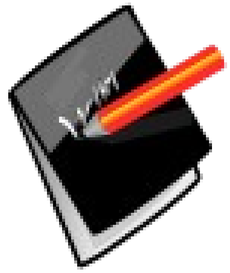
# Proposed activities

Check the suggested exercises you will find at the “Aula Virtual”. **These activities are optional and non-assessable but** understanding these non-assessable activities is essential to solve the assessable task ahead.

Shortly you will find the proposed solutions.

# 9. BIBLIOGRAPHY

# Resources




- 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