GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

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

Unidad 3. ACCESO UTILIZANDO MAPEO RELACIONAL DE OBJETOS (ORM)

Acceso a bases de datos relacionales usando DAO

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

CFGS Desarrollo de Aplicaciones Multiplataforma (DAM)

Abelardo Martinez

Año 2023-2024

Créditos

•Apuntes realizados por Abelardo Martínez. •Basado y modificado de Sergio Badal (www.sergiobadal.com). •Las imágenes e iconos utilizados están protegidos por la LGPL . licencia y haber sid de:

- •https://commons.wikimedia.org/wiki/Crystal_Clear
- •https://www.openclipart.org

Progreso de la unidad

Contenido

1. ¿QUÉ ES DAO?

¿Qué es DAO?

Hasta ahora nos hemos conectado a la base de datos usando solo una clase (principal) que contenía tanto la conexión como la declaración de selección.

Ahora separaremos el código según su funcionalidad para poder realizar un acceso diferenciado por capas.

con el objetivo de aumentar la reutilización del código.

- El patrón Data Access Object (DAO) es un patrón estructural que nos permite trabajar con una base de datos utilizando un API abstracta.
- La API oculta a la aplicación toda la complejidad de realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en el mecanismo de almacenamiento subyacente. Esto permite que ambas capas evolucionen por separado sin saber nada entre sí.

El patrón de objeto de acceso a datos (DAO) es un patrón estructural que nos permite aislar la capa de aplicación/ negocio de la capa de persistencia utilizando una API abstracta.

DAO. Ejemplo sencillo

Eche un vistazo a este fragmento de código e intente comprender qué está haciendo.

Como puede ver, estamos trabajando con una base de datos sin usar instrucciones SQL al acceder a clases y elementos Java/objetos en lugar de tablas o cualquier elemento/objeto de la base de datos.

Estamos usando capas: ¡usando DAO!

DAO. Ejemplo completo

Ahora estamos construyendo un proyecto Java desde cero para crear una aplicación CRUD para trabajar con una tabla llamada "Personas" dentro de una base de datos SQLite simple usando DAO.

¡Entra!

2. DAO: CREACIÓN DEL PROYECTO

El proyecto (Java-Maven)

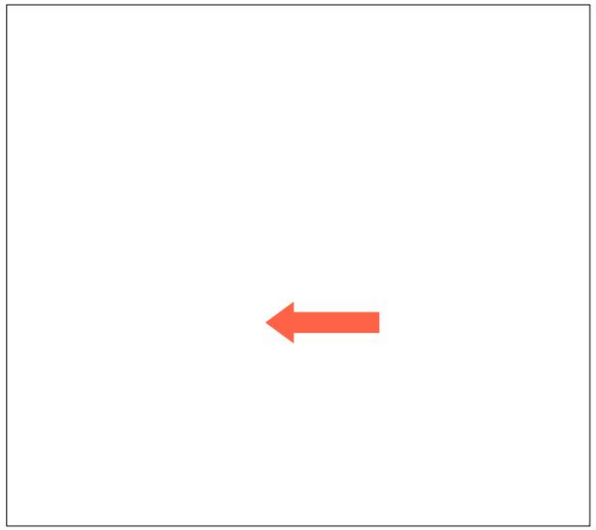
- En ECLIPSE, cree un proyecto Java Maven vacío con estos parámetros como vimos en la UNIDAD 2:

3. DAO: CONFIGURACIÓN DE LA BASE DE DATOS

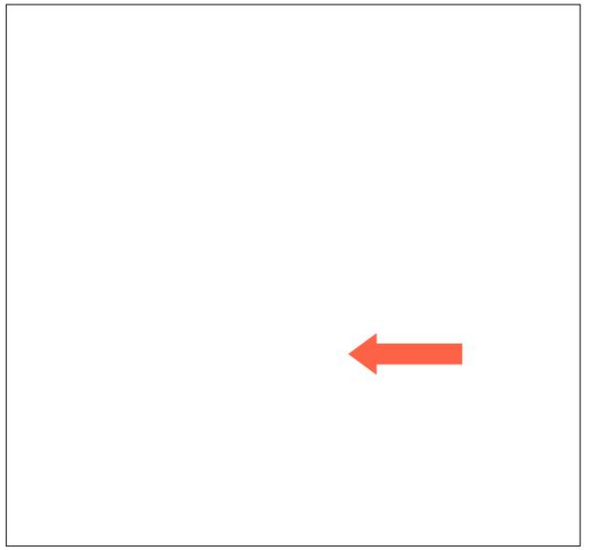
La base de datos (SQLite)

- Dado que el objetivo principal de esta unidad es explicar cómo DAO te permite trabajar con cualquier base de datos relacional usando capas centrándose en Java, usaremos una de las bases de datos más simples que puedas encontrar: SQLite.
- Siga estos pasos para crear una tabla en una base de datos SQLite existente:

1. Crea una carpeta dentro de tu proyecto llamada "bdsqlite"



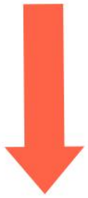
2. Vaya a esta URL y descargue chinook.db:
https://www.sqlitetutorial.net/sqlite-sample-database/



La base de datos (SQLite)

Siga los pasos que puede encontrar en la documentación ampliada para lograrlo:

1) Instale SQLite y acceda a su consola.
2) Cree una nueva tabla llamada Personas con ID de persona, nombre y edad como columnas.
3) Inserte cuatro registros aleatorios



4. DAO: CONECTANDO A LA BASE DE DATOS

La conexión (base de datos)

(1 de 3) Agregue las dependencias al Proyecto Maven (pom.xml) para SQLite (verifique su versión)

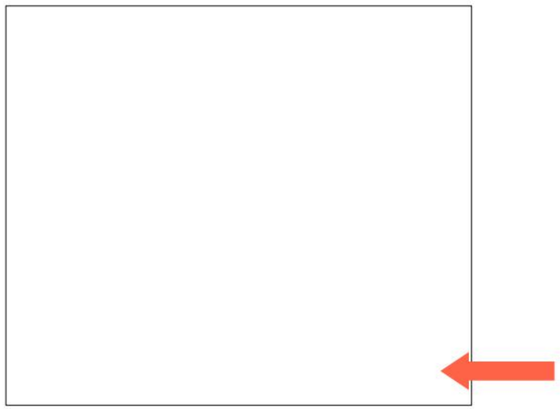
```
<dependencias>
   <dependencia>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <versión>4.11</versión>
      <scope>test</scope> </
   dependency>
   <!-- https:// mvnrepository.com/artifact/org.xerial/sqlite-jdbc --> <dependencia>

   <groupId>org.xerial</groupId>
         <artifactId>sqlite-jdbc</artifactId>
         <versión>3.39.3.0</versión> < /
   dependencia> </
dependencias>
```

(2 de 3) Cree una clase dentro de un nuevo paquete llamado datos para gestionar la conexión y las operaciones de "cierre".

```
importar java.sql.Statement; importar
java.sql.Conexión; importar
java.sql.DriverManager; importar
java.sql.PreparedStatement; importar
java.sql.ResultSet; importar
java.sql.SQLException;


Conexión DB de clase pública {

        /**
         * ---------------------------------
         * CONSTANTES Y VARIABLES GLOBALES
         * ---------------------------------
         */
        //Conexión URL privada
        estática final String URL = "jdbc:sqlite:bdsqlite/chinook.db"; // versión de Linux // URL de cadena final estática privada =
        "jdbc:sqlite:bdsqlite\\chinook.db"; //versión de Windows

        /*
         * --------------------
         * GESTIÓN DE CONEXIÓN
         * --------------------
         */ /
        *
         * Método estático: simplemente intenta conectarte a la base de datos
         */
        Conexión estática pública ConnectToDB() { intentar

                { Conexión cnDB = DriverManager.getConnection(URL); devolver cnDB; }
                        captura
                (SQLException sqle) {
                        System.out.println("¡Algo salió mal al intentar conectarse a la base de datos!"); sqle.printStackTrace(System.out);


                } devolver nulo;
        }

        /*
         * Método estático: simplemente intenta desconectarte de la base de datos
         */
        cierre público estático vacío (Conexión cnDB) lanza SQLException {
                cnDB.close();
        }

        /*
         *---------------------
         * RECURSOS DE CIERRE
         *---------------------
         */

        //ResultSet
        public static void close(ResultSet rsCursor) lanza SQLException { rsCursor.close();

        }

        //Declaración
        public static void close(Declaración staSQL) lanza SQLException { staSQL.close();

        }

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

}
```

La conexión (base de datos)

(3 de 3) Finalmente, solo necesitamos un método principal para comprobar que la conexión funciona correctamente



Producción:

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

clase pública TestSQLiteDAO { public
    static void main(String[] stArgs) {
        intente
            { Conexión cnDB = DBConnection.ConnectToDB(); Declaración
            staSQL = cnDB.createStatement(); String stSQLSelect =
            "SELECCIONAR * DE Personas"; System.out.println("Ejecutando:
            " + stSQLSelect); ResultSet rsPerson =
            staSQL.executeQuery(stSQLSelect); int iNumItems = 0; mientras
            (rsPerson.next())
            { iNumItems++; System.out.println("ID:
                " +
                rsPerson.getInt("IDpersona")); System.out.println("Nombre: " +
                rsPerson.getString("nombre")); System.out.println("Edad: " +
                rsPerson.getInt("edad"));

            } si (iNumItems == 0)
                System.out.println("No se encontraron elementos en la tabla Personas");
            DBConnection.close(rsPersona); //cerrar conjunto de resultados
            DBConnection.close(cnDB); // cerrar conexión a la base de datos } catch
        (SQLException sqle) {
            System.out.println("¡Algo salió mal mientras leía!"); sqle.printStackTrace(System.out);

        }

    }
}
```

La conexión (base de datos)

DEBE comprender cada línea del código proporcionado hasta ahora antes de continuar.

Accede a la documentación ampliada de la UNIDAD 2 para revisar todos los conocimientos requeridos.

5. CREACIÓN DE CLASES PRINCIPALES

5.1 Clases principales para almacenar los datos.

Creando clases principales

Ahora necesitamos crear dos clases para cada entidad/tabla de la base de datos:

(1 de 2) Person.java para almacenar los datos de cada fila de la base de datos.

• Cree esta clase dentro de un nuevo paquete llamado dominio. • Defina los atributos, definidores, captadores y 4 constructores:

1)Constructor vacío 2)

Constructor que proporciona solo el ID (por ejemplo, para eliminar solo necesitamos tener el ID de la persona, que es la PK)

3) Constructor para insertar un nuevo registro (no necesitamos especificar el ID de la persona)

4) Constructor para modificar un registro (necesita todos los atributos)

```
Persona de clase pública {

        /*
          *----------------------------------------
          * ATRIBUTOS
          *----------------------------------------
          */
        identificación interna
        privada ; cadena privada
        stName; edad privada ;

        /*
          *----------------------------------------
          * MÉTODOS
          *----------------------------------------
          */ /
        *
          * Constructor vacío */

        Persona pública () { } /*

        *
          Constructor de ID. Sólo clave primaria */

        Persona pública (int iID) { this.iID
                = iID;

        } /
          * * Constructor sin ID. Todos los campos, excepto la clave principal */

        Persona pública (String stName, int iAge) { this.stName =
                stName; this.iEdad = iEdad;


        } /*
          * Constructor completo con todos los campos */

        Persona pública (int iID, String stName, int iAge) { this.iID = iID;
                this.stName =
                stName; this.iEdad = iEdad;

        }

        /*
          *----------------------------------------
          * RECOGEDORES
          *----------------------------------------
          */
        public int getPersonID() { return iID;

        }

        public int getAge() { return
                iAge;
        }

        public String getName() { return
                stName;
        }

        /*
          *----------------------------------------
          * CONFIGURADORES
          *----------------------------------------
          */
        public void setPersonID(int iID) { this.iID = iID;

        }

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

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

        /*
          *------------------
          * MÉTODOS DE FORMATO DE DATOS
          *------------------
          */

        @Override
        public String toString() { return
                "Persona [ID =                "     + iID + ", Nombre =          "     + stNombre + ", Edad =     "     + iEdad + "]";
        }
}
```

5.2 Clases principales para gestionar los datos

Creando clases principales

Ahora necesitamos crear otras dos clases para cada entidad/tabla de la base de datos:

(2 de 2) PersonDAO.java para gestionar las operaciones entre la base de datos y la clase anterior

• Crear esta clase dentro de un nuevo paquete llamado datos •

Definir las declaraciones de acceso a la base de datos al comienzo de la clase como variables estáticas (buena práctica)

• Crear un método para leer todos los elementos de esta entidad

```
importar java.sql.Conexión; importar
java.sql.PreparedStatement; importar
java.sql.ResultSet; importar
java.sql.SQLException; importar
java.util.ArrayList;

importar DOMINIO.Persona;

/**
 * ================================================== ====================
 *Programa para gestionar operaciones CRUD a la BD* @autor
 Abelardo Martínez. Basado y modificado de Sergio Badal.
 * ================================================== ====================
 */

clase pública PersonaDAO {

        /**
         * ---------------------------------
         * CONSTANTES Y VARIABLES GLOBALES
         * ---------------------------------
         */ /
        *
         * --------------------
         * CONSULTAS SQL
         * --------------------
         */
        Cadena final estática privada SELECT = "SELECT * FROM People";

        /*
         * ----------------
         * SELECCIONAR
         * ----------------
         */
        public ArrayList<Persona> SelectAllPeople() lanza SQLException {

                Conexión cnDB = nula;
                PreparedStatement pstaSQLSelect = nulo;
                Conjunto de resultados rsPerson = nulo;
                Persona objPerson = nulo;
                ArrayList<Persona> arlPerson = new ArrayList<Persona>();

                intente
                        { cnDB = DBConnection.ConnectToDB(); // conectarse a la base de
                        datos pstaSQLSelect = cnDB.prepareStatement(SELECT); // instrucción de selección
                        rsPerson = pstaSQLSelect.executeQuery(); //ejecutar select while
                        (rsPerson.next()) { int iID =
                                rsPerson.getInt("personID"); String stName =
                                rsPerson.getString("nombre"); int iEdad =
                                rsPerson.getInt("edad"); objPerson = nueva
                                Persona(iID, stName, iEdad); arlPerson.add(objPerson);

                        }
                } captura (SQLException sqle) {
                        System.out.println("Algo salió mal al leer de la tabla de personas"); sqle.printStackTrace(System.out); }
                        finalmente { DBConnection.close(rsPerson); //
                cerrar conjunto
                        de resultados DBConnection.close(pstaSQLSelect); //cerrar
                        declaración preparada DBConnection.close(cnDB); //cerrar conexión a la base de
                        datos

                } return arlPersona;
        }
}
```

5.3 Clase principal para aplicar DAO

Clase de interfaz

Finalmente, necesitamos usar la clase TestSQLiteDAO para crear una instancia de PersonDAO y simplemente enumerar los registros de la tabla "Personas" (a través de Persona).

Tenga en cuenta que no se requiere manejo de excepciones ya que se manejan dentro de las clases DAO.

Producción:

```
importar java.sql.*;
importar java.util.ArrayList; datos de

importacion .*;
importar DOMINIO.*;

clase pública TestSQLiteDAO {
    /*
     * ----------------
     * PROGRAMA PRINCIPAL
     * ----------------
     */
    public static void main(String[] stArgs) lanza
SQLExcepción {

        PersonaDAO objPersonDAO = nueva PersonaDAO();
        ArrayList<Persona> arlPersona;

        //
        SELECCIONAR System.out.println("**** LISTA DE REGISTROS
        INICIALES"); arlPerson = objPersonDAO.SelectAllPeople();
        for (Persona objPerson : arlPerson)
            { System.out.println("Persona: " + objPerson);
        }
    }
}
```

6. DAO: INSERTAR, ACTUALIZAR Y ELIMINAR DATOS

Insertando datos

Solo necesitamos agregar más métodos a PersonDAO.

```
    cadena final estática privada INSERT = "INSERT INTO Personas (nombre, edad) VALORES (?,?)";
    /*
      * ------------------
      * INSERTAR
      * ------------------
      */
    public int InsertPerson(Persona objPerson) lanza SQLException {

        Conexión cnDB = nula;
        PreparedStatement pstaSQLInsert = nulo; int iNumreg
        = 0;

        intente
            { cnDB = DBConnection.ConnectToDB(); // conectarse a la base de
            datos pstaSQLInsert = cnDB.prepareStatement(INSERT); //insertar declaración
            pstaSQLInsert.setString(1, objPerson.getName());
            pstaSQLInsert.setInt(2, objPerson.getAge()); iNumreg =
            pstaSQLInsert.executeUpdate(); //ejecutar inserción
        } catch (SQLException sqle)
            { System.out.println("¡Algo anda mal al insertar!");
            sqle.printStackTrace(System.out); } finalmente


        { DBConnection.close(pstaSQLInsert); //cerrar declaración preparada
            DBConnection.close(cnDB); //cerrar conexión a la base de datos

        } devolver iNumreg;
    }
```

Eliminar datos

Solo necesitamos agregar más métodos a PersonDAO.

```
Cadena final estática privada ELIMINAR = "ELIMINAR DE Personas DONDE personID =?";

/*
  * ----------------
  * BORRAR
  * ----------------
  */
public int DeletePerson(Person objPerson) lanza SQLException {

    Conexión cnDB = nula;
    PreparedStatement pstaSQLDelete = nulo; int
    iNumreg = 0;

    intente
        { cnDB = DBConnection.ConnectToDB(); // conectarse a la base de
        datos pstaSQLDelete = cnDB.prepareStatement(DELETE); //eliminar declaración
        pstaSQLDelete.setInt(1, objPerson.getPersonID()); iNumreg =
        pstaSQLDelete.executeUpdate(); //ejecutar eliminar
    } catch (SQLException sqle)
        { System.out.println("¡Algo anda mal al eliminar!");
        sqle.printStackTrace(System.out); } finalmente

    { DBConnection.close(pstaSQLDelete); //cerrar declaración preparada
        DBConnection.close(cnDB); //cerrar conexión a la base de datos

    } devolver iNumreg;
}
```

Actualizando datos

Solo necesitamos agregar más métodos a PersonDAO.

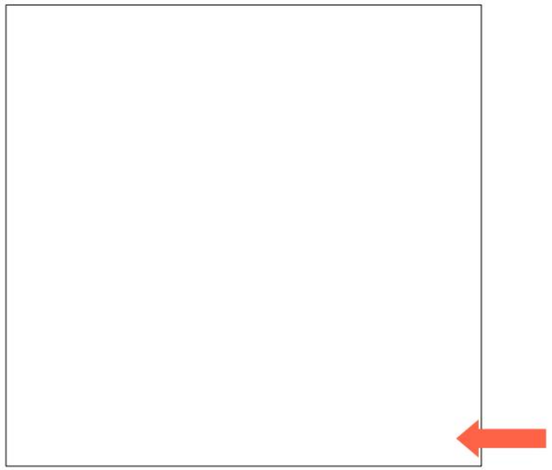
```
Cadena final estática privada ACTUALIZAR = "ACTUALIZAR Personas SET nombre =?, edad =? ¿DÓNDE personaID =?";

/*
  * ------------------
  * ACTUALIZAR
  * ------------------
  */
public int UpdatePerson (Persona objPerson) lanza SQLException {

      Conexión cnDB = nula;
      PreparedStatement pstaSQLUpdate = nulo; int
      iNumreg = 0;

      intente
            { cnDB = DBConnection.ConnectToDB(); // conectarse a la base de
            datos pstaSQLUpdate = cnDB.prepareStatement(UPDATE); // declaración de actualización
            pstaSQLUpdate.setString(1, objPerson.getName());
            pstaSQLUpdate.setInt(2, objPerson.getAge());
            pstaSQLUpdate.setInt(3, objPerson.getPersonID()); iNumreg =
            pstaSQLUpdate.executeUpdate(); //ejecutar eliminar
      } catch (SQLException sqle)
            { System.out.println("¡Algo anda mal al actualizar!");
            sqle.printStackTrace(System.out); } finalmente

      { DBConnection.close(pstaSQLUpdate); //cerrar declaración preparada
            DBConnection.close(cnDB); //cerrar conexión a la base de datos

      } devolver iNumreg;
}
```

7. CRUD USANDO DAO

CRUD usando DAO

Si juntamos todos los métodos y agregamos algunas operaciones CRUD, podemos obtener esto:

Tenga en cuenta que no se requiere manejo de excepciones ya que se manejan dentro de las clases DAO.



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

datos de
importacion .*; importar DOMINIO.*;


clase pública TestSQLiteDAO { /*

        * ----------------
        * PROGRAMA PRINCIPAL
        * ----------------
        */
       public static void main(String[] stArgs) lanza SQLException {

            PersonaDAO objPersonDAO = nueva PersonaDAO();
            ArrayList<Persona> arlPersona;

            //
            SELECCIONAR System.out.println("**** LISTA DE REGISTROS
            INICIALES"); arlPerson = objPersonDAO.SelectAllPeople();
            for (Persona objPerson : arlPerson)
                  { System.out.println("Persona: " + objPerson);
            }

            // INSERTAR
            System.out.println("**** INSERTAR NUEVOS REGISTROS");
            Persona objPerson1 = nueva Persona("Herminia Fuster", 55);
            objPersonDAO.InsertPerson(objPerson1); Persona
            objPerson2 = nueva Persona("Mario Caballero", 25);
            objPersonDAO.InsertPerson(objPerson2);

            // BORRAR
            System.out.println("**** BORRAR EL PRIMER REGISTRO");
            Persona objPersonToDelete = nueva Persona(1);
            objPersonDAO.DeletePerson(objPersonToDelete);

            //
            ACTUALIZAR System.out.println("**** ACTUALIZAR EL SEGUNDO
            REGISTRO"); Persona objPersonToUpdate = nueva Persona(2, "JOAN LEUIN",
            22); objPersonDAO.UpdatePerson(objPersonToUpdate);

            //RESULTADO DE LOS
            DATOS FINALES System.out.println("**** LISTA DE
            REGISTROS FINALES"); arlPerson =
            objPersonDAO.SelectAllPeople(); for (Persona
                  objPerson : arlPerson) { System.out.println("Persona: " + objPerson);
            }
       }
}
```

8. ACTIVIDADES PARA LA PRÓXIMA SEMANA

Actividades propuestas

Consulta las sugerencias de ejercicios que encontrarás en el “Aula Virtual”. Estas actividades son opcionales y no evaluables, pero comprenderlas es esencial para resolver la tarea evaluable que tenemos por delante.

En breve encontrará las soluciones propuestas.

9. BIBLIOGRAFÍA

Recursos

- Josep Cañellas Bornas, Isidre Guixà Miranda. Accés a dades. Desarrollo de aplicaciones multiplataforma. Comunes creativos. Departamento de Enseñanza, Institut Obert de Catalunya. Dipósito legal: B. 29430-2013. https://ioc.xtec.cat/educacio/recursos
- Alberto Oliva Molina. Acceso a datos. UD 3. Herramientas de mapeo de objetos relacionales (ORM). IES Tubalcaín. Tarazona (Zaragoza, España).

GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

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