	UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA **FACULTAD DE INGENIERÍA (UNIDAD MEXICALI)** **DOCUMENTO DEL SISTEMA DE CALIDAD**

Formatos para prácticas de laboratorio

CARRERA	PLAN DE ESTUDIO	CLAVE ASIGNATURA	NOMBRE DE LA ASIGNATURA
Ing. En Comp. y L.S.C.	2003-1	5038	Programación Orientada a Objetos II

PRÁCTICA No.	LABORATORIO DE	Ingeniero en Computación y Licenciado en Sistemas Computacionales	DURACIÓN (HORA)
6	NOMBRE DE LA PRÁCTICA	Conexión con base de datos	2

1. INTRODUCCIÓN

Con java es posible comunicarse con diversos gestores de bases de datos por medio del API Java DataBase Connectivity (JDBC por sus siglas en ingles), que es una especificación formada por una colección de interfaces y clases abstractas. Con base a esta especificación se fabrican los drivers para cada base de datos que desee tener compatibilidad con el API JDBC. Aunque no es la única forma de conectarse con una base de datos, elegir este método permite mantener una independencia de plataforma. Como se conoce, java no depende del sistema operativo o hardware, y con el uso de drivers para la conexión con bases de datos escritos totalmente en java es posible mantener una independencia con respecto a la base de datos también.

Otro aspecto importante durante el desarrollo de esta práctica es el uso de patrones de diseño, los cuales proporcionan un camino que se ha probado en varias ocasiones y que asegura la estabilidad y funcionamiento de la solución implementada bajo este esquema.

2. OBJETIVO (COMPETENCIA)

El alumno será capaz de establecer una conexión con una base de datos por medio del API JDBC de java y ejecutar operaciones básicas de bases de datos (Altas, Bajas, Cambios, Consulta o también conocido como CRUD por sus siglas en ingles “Create, Update, Read and Delete”), basándose en el uso de los patrones de diseño como DTO, DAO, BaseDAO, DELEGATE y FACADE, patrones que están relacionados con el acceso a datos.

3. FUNDAMENTO

¿Qué es y que hace JDBC?

JDBC cuenta con las siguientes características:

- Provee acceso a bases de datos
- Es parte integral de la plataforma java desde la versión 1.1.X
- Independiza a las aplicaciones del motor de bases de datos usado.

- JDBC Define un conjunto de interfaces que el proveedor de bases de datos implementa en un código llamado driver. El driver es usado por la MVJ para traducir las invocaciones genéricas en invocaciones de la base de datos propietaria véase figura 1.

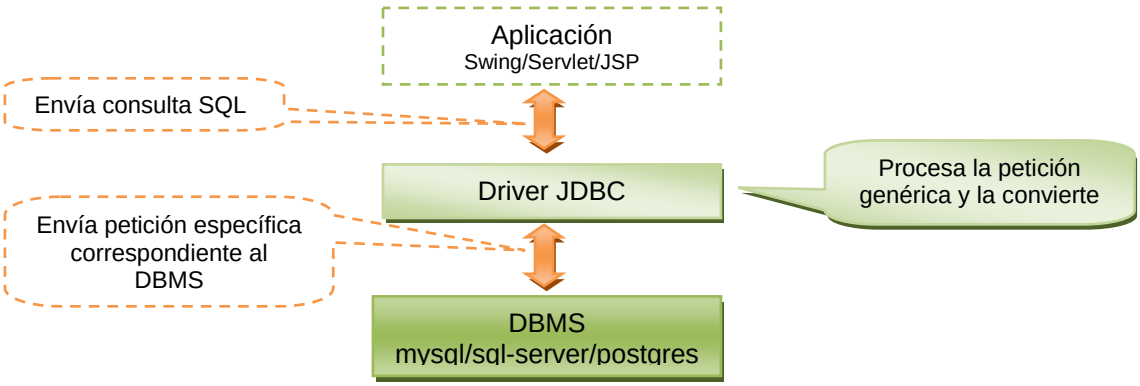

Figura 1 Muestra una aplicación que hace uso de JDBC

Como se menciono antes el método de conexión aquí mostrado no es el único, existen cuatro tipos distintos de drivers para conectar java con las diferentes bases de datos, la presente práctica solo se contempla utilizar el API JDBC a partir de un driver escrito en java puro, que corresponde al tipo cuatro. Este tipo de drivers:

- Son escritos íntegramente en **java puro** que habla directamente con la base de datos, mediante protocolos de red estándares
- Son altamente **portables**. Una vez compilados pueden usarse en cualquier sistema
- Es el método más **eficiente** de acceso a datos
- No requiere de ninguna librería adicional ni de la instalación de un middleware, con lo que es más **simple** su puesta en producción.
- La mayoría de los fabricantes de bases de datos proveen drivers JDBC de tipo cuatro para sus bases de datos.

Otro concepto importante que es necesario entender, es que la presente práctica utiliza patrones de diseño, pero ¿qué es un patrón de diseño?

Según Christopher Alexander, Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, entonces es necesario describir la mejor solución al problema, de tal forma que se pueda utilizar dicha solución un millón de veces más. Aunque Alexander se refería a patrones en la construcción de edificios dicha concepción es aplicable al paradigma de programación orientada a objetos.

En general un patrón cuenta según GoF (Group of Four) con cuatro elementos básicos:

1. El nombre del patrón
2. El problema
3. La solución
4. Las consecuencias

Por tanto si se parte de que ***Un patrón de diseño es un conjunto de reglas que describen como afrontar tareas y solucionar problemas que surgen durante el desarrollo de software***, programar aplicaciones con acceso a bases de datos actualmente es un problema recurrente, para lo cual existen patrones creados para dicho problema, estos patrones son DTO, DAO, BaseDAO, FACADE, y DELEGATE.

Tabla 1. Muestra la lista de patrones a utilizar durante el desarrollo de la practica

Nombre	Intención	También conocido como
Data Transfer Object (DTO)	✓ Agrupar uno o un conjunto de valores provenientes del dominio	Trasfer Object Value Object Replicate Object
Base DAO	✓ Crear una sola conexión con la base de datos para todos los objetos DAO.	No aplica
Data Access Object	✓ Desacoplar la lógica de negocio de la lógica de datos de manera que se pueda cambiar la fuente de datos fácilmente	Data Access Component
FACADE	✓ Encapsula la complejidad de las interacciones entre los objetos de negocio y participantes en un flujo de trabajo ✓ Maneja los objetos de negocio y proporciona un servicio de acceso uniforme a los clientes	Session Facade Session Entity Facede Distribuited Facade
DELEGATE	✓ Un objeto que reside en la capa de presentación y en beneficio de los otros componentes de la capa de presentación llama a métodos remotos en los objetos de la capa de negocio ✓ Oculta tecnología usada en el modelo ✓ Representa un conjunto de casos de uso lógicamente relacionados	Bussines Delegate

Como se puede observar en la tabla 1, cada patrón cuenta con un nombre y un objetivo pero **¿cómo trabajan juntos?.** Contamos con cinco patrones descritos en la tabla uno, todos tiene un papel importante durante el proceso de transacción con la base de datos véase figura 2.

Figura 2. Muestra cómo trabajan los patrones de diseño durante una transacción con la base de datos

Como se puede observar en la figura dos el patrón BaseDAO es el que se encuentra en la capa más baja, en este se creara la conexión con la base de datos, y todas aquellas clases que requieran de acceso a la capa de datos deberán heredar de esta clase. El siguiente en la lista es el DAO, en el se encuentran las sentencias SQL que se deseen ejecutaran en la base de datos. Posteriormente se encuentra el patrón FACADE, en se colocara código referente a la lógica de negocio, y el ultimo el DELEGATE, este debe corresponder a un caso de uso en especial.

4. PROCEDIMIENTO (DESCRIPCIÓN)

A) EQUIPO NECESARIO	MATERIAL DE APOYO
Equipo de computo con SDK de Java Editor de Java – NetBeans Opcion a: Mysql Driver de mysql Scrip de la base de datos Opcion b: Dervy para net beans Screen cast para la creación de la BD.	Practica impresa

B) DESARROLLO DE LA PRÁCTICA

La práctica se divide de la siguiente forma:

a. Creación de la base de datos (Como se mencionó en el material, la práctica considera dos caminos).
b. Crear las clases que correspondan a los patrones de diseño dentro de la tabla uno.
c. Crear una interfaz gráfica que permita ejecutar las cuatro operaciones básicas.

a. La base de datos propuesta es la siguiente.

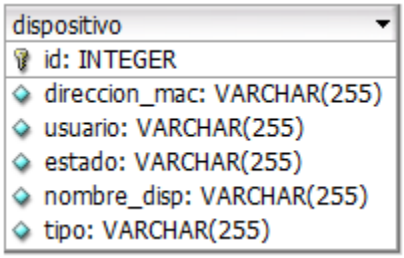

Figura 3 Tabla de la base de datos propuesta

Para la creación de la base de datos:

1) Ejecutar el siguiente script (figura 4) en la base de datos

```
1. create database vecindario;
2. use vecindario;
3. CREATE TABLE dispositivo ( id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
4. direccion_mac VARCHAR(255) NULL, usuario VARCHAR(255) NULL,
5. estado VARCHAR(255) NULL, nombre_disp VARCHAR(255) NULL,
6. tipo VARCHAR(255) NULL, PRIMARY KEY(id));
```

Figura 4 Script de la base de datos (Dominio del sistema)

2) Ver el video tutorial para crear la base en Netbeans

b. Crear las clases correspondientes para cada patrón

```java
package bt.modelo.dto;
import java.io.Serializable;
public class DispositivoDTO implements Serializable{
    private String usuario;
    private String direccionMac;
    private String estado;
    private String nombreDisp;
    private String tipo;

    public String getUsuario() {
        return usuario;
    }

    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }

    public String getDireccionMac() {
        return direccionMac;
    }

    public void setDireccionMac(String direccionMac) {
        this.direccionMac = direccionMac;
    }

    public String getEstado() {
        return estado;
    }

    public void setEstado(String estado) {
        this.estado = estado;
    }

    public String getNombreDisp() {
        return nombreDisp;
    }

    public void setNombreDisp(String nombreDisp) {
        this.nombreDisp = nombreDisp;
    }

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

}
```

Figura 5. Implementación del Patrón DTO

La figura 5 muestra la implementación del patrón DTO para el dominio propuesto anteriormente, existen ciertas consideraciones que son necesarias tomar en cuenta véase tabla 2.

Tabla 2. Características y consideraciones del patrón DTO

Ventajas del uso dentro del sistema	Mantener un control del dominio del sistema a nivel de objetos dentro de la aplicación
Consideraciones importantes	✓ Debe existir una clase dto por cada tabla en la base de datos ✓ El nombre de la clase deberá llevar el nombre de la tabla más el sufijo DTO, por ejemplo DispositivoDTO. ✓ El constructor no debe recibir ningún parámetro ✓ La clase deberá contar con un método toString el cual imprimirá el estado del objeto ✓ La clase deberá implementar la interfaz Serializable, ya que este objeto será el encargado de llevar los datos a través de toda la aplicación ✓ Entre java y las distintas bases de datos existe una correspondencia con respecto al tipo de dato que se maneja. Es decir, si en la base de datos un dato es de tipo

	varchar, en java deberá ser declarado como String, para lo cual existe una tabla de correspondencia de tipos de datos que se proporcionará como material extra en la presente práctica.
Nota	No se muestra el código completo de la práctica

```
1  package bt.modelo.dao;
2  import java.sql.Connection;
3  import java.sql.DriverManager;
4  import java.sql.PreparedStatement;
5  import java.sql.ResultSet;
6  public class ConexionBD {
7      protected Connection conexion;
8      public ConexionBD() {
9          //Parametros para configurar la conexion de la base de datos
10         String usr = "root";
11         String pwd = "root";
12         String url = "jdbc:mysql://localhost:3306/movil";
13         String driver = "com.mysql.jdbc.Driver";
14         try {
15             //Carga el driver
16             Class.forName(driver);
17             //Crea la conexion con la base de datos por medio del driver
18             conexion = DriverManager.getConnection(url, usr, pwd);
19         } catch (Exception ex) {
20             ex.printStackTrace();
21         }
22     }
23     protected void cerrar(PreparedStatement ps) throws Exception {
24         if (ps != null) {
25             ps.close();
26         }
27     }
28     protected void cerrar(ResultSet rs) throws Exception {
29           if (rs != null) {
30             rs.close();
31         }
32     }
33 }
```

Figura 6. Implementación del patrón BaseDAO o ConexionDB

```
10         String usr = "root";
11         String pwd = "root";
12         String url = "jdbc:derby://localhost:1527/movil";
13         String driver = "org.apache.derby.jdbc.ClientDriver";
```

Figura 7. Muestra el código en la clase ConexionDB en caso de cambio de Bd y MBD

La figura 6 muestra el código correspondiente al patrón conexionDB o BaseDAO, el cual establece la conexión con la base de datos, la estructura del código permite realizar un cambio de base de datos y/o manejador de base de datos con solo cambiar los valores de las variables usr, pwd, url y driver, como se muestra en la figura 7.

Tabla 2. Características y consideraciones del patrón ConexionDB

Ventajas del uso dentro del sistema	Mantener un control del acceso a la base de datos desde una sola clase
Consideraciones importantes	✓ Debe ser la única clase que se conecte con el dominio ✓ Las clases que deseen conectarse al dominio deberán heredar de esta ✓ Los Resultset y PreparedStatement serán cerrados desde esta clase
Nota	Se debe contar con el driver asociado a la base de datos con la que se desea conectar.

```
1   package bt.modelo.dao;
2   import bt.modelo.dto.DispositivoDTO;
3   import java.sql.*;
4   import java.util.*;
5   public class DispositivoDAO extends ConexionBD {
6       private static final String DIRECCION_MAC = "direccion_mac";
7       private static final String USUARIO = "usuario";
8       private static final String ESTADO = "estado";
9       private static final String NOMBRE_DISP = "nombre_disp";
10      private static final String TIPO = "tipo";
11      private static final String SQL_SELECT_ALL = "SELECT * FROM DISPOSITIVO";
12      private static final String SQL_INSERT = "INSERT INTO DISPOSITIVO" +
13              "(" + DIRECCION_MAC +
14              "," + USUARIO +
15              "," + ESTADO +
16              "," + NOMBRE_DISP +
17              "," + TIPO +
18              ")VALUES (?,?,?,?,?)";
19      private static final String SQL_READ = "SELECT * FROM DISPOSITIVO WHERE " +
20              DIRECCION_MAC + " = ?";
21      private static final String SQL_DELETE = "DELETE FROM DISPOSITIVO WHERE " +
22              DIRECCION_MAC + " = ?";
23      private static final String SQL_UPDATE = "UPDATE DISPOSITIVO SET " +
24              USUARIO + " = ?," +
25              ESTADO + " = ?," +
26              NOMBRE_DISP + " = ?," +
27              TIPO + " =? " +
28              "WHERE " + DIRECCION_MAC + " = ?";
29      public DispositivoDAO() {
30          super();
31      }
32      public List readAll() throws Exception {
33          //Objeto sobre el cual se almacena la consulta SQL previamente creada
34          PreparedStatement ps = null;
```

```
35          //Encargado de almacenar el resultado de la consulta a la base de datos
36          ResultSet rs = null;
37          List result = new ArrayList();
38          //Coloca la sentencia sql en el PreparedStatment
39          ps = conexion.prepareStatement(SQL_SELECT_ALL);
40          //Ejecuta la consulta en la base de datos, y almacena el resultado en el ResuktSet
41          rs = ps.executeQuery();
42          while (rs.next()) {
43              //Obtiene uno a uno los objetos del rs para almacenarlos en la lista que sera regresada
44              result.add(getObject(rs));
45          }
46          //Cierra los flujos de datos
47          cerrar(ps);
48          cerrar(rs);
49          return result;
50      }
51      public void append(DispositivoDTO dto) throws Exception {
52          PreparedStatement ps = null;
53          //Objeto sobre el cual se almacena la consulta SQL previamente creada
54          ps = conexion.prepareStatement(SQL_INSERT);
55          //ps.setString sustituye cada uno de los simbolos de interrogacion que se encuentren en la sentencia SQL, por los
56          //valores deseados. A cada simbolo de interrogacion le corresponde una posicion.
57          ps.setString(1, dto.getDireccionMac());
58          ps.setString(2, dto.getUsuario());
59          ps.setString(3, dto.getEstado());
60          ps.setString(4, dto.getNombreDisp());
61          ps.setString(5, dto.getTipo());
62          //Ejecuta la acualizacion
63          ps.executeUpdate();
64          cerrar(ps);
65      }
66      public void update(DispositivoDTO dto) throws Exception {
67          PreparedStatement ps = null;
68          ps = conexion.prepareStatement(SQL_UPDATE);
69          ps.setString(1, dto.getUsuario());
70          ps.setString(2, dto.getEstado());
71          ps.setString(3, dto.getNombreDisp());
72          ps.setString(4, dto.getTipo());
73          ps.setString(5, dto.getDireccionMac());
74          ps.executeUpdate();
75          cerrar(ps);
76      }
77      public void delete(DispositivoDTO dto) throws Exception {
78          PreparedStatement ps = null;
79          ps = conexion.prepareStatement(SQL_DELETE);
80          ps.setString(1, dto.getDireccionMac());
81          ps.executeUpdate();
82          cerrar(ps);
83      }
84      public DispositivoDTO read(DispositivoDTO dto) throws Exception {
85          PreparedStatement ps = null;
86          ResultSet rs = null;
87          DispositivoDTO result = null;
88          ps = conexion.prepareStatement(SQL_READ);
89          ps.setString(1, dto.getDireccionMac());
90          rs = ps.executeQuery();
91          if (rs.next()) {
92              result = getObject(rs);
93          }
94          cerrar(ps);
95          cerrar(rs);
96
97          return result;
98      }
99      private DispositivoDTO getObject(ResultSet rs) throws Exception {
100         DispositivoDTO dtoDispositivo = new DispositivoDTO();
101         dtoDispositivo.setUsuario(rs.getString(USUARIO));
102         dtoDispositivo.setNombreDisp(rs.getString(NOMBRE_DISP));
```

```
103             dtoDispositivo.setTipo(rs.getString(TIPO));
104             dtoDispositivo.setEstado(rs.getString(ESTADO));
105             dtoDispositivo.setDireccionMac(rs.getString(DIRECCION_MAC));
106             return dtoDispositivo;
107         }
108     }
```

Figura 8. Implementación del patrón DAO

La figura 8 muestra el código necesario para implementar el patrón DAO y en la tabla 3 se muestran sus características y consideraciones.

Tabla 3 Consideraciones y características

Ventajas del uso dentro del sistema	Las tracciones con una tabla en la base de datos se encapsulan en una clase
Consideraciones importantes	✓ Debe existir una clase dao por cada tabla en la base de datos ✓ El nombre de la clase deberá llevar el nombre de la tabla más el sufijo DAO, por ejemplo DispositivoDAO. ✓ La clase deberá contar con un método para (Altas, Bajas, Cambios, Consultas) ✓ La clase deberá heredera de la conexión de la base de datos extendiendo a la clase ConexionDB y llamando al constructor de la super clase.
Nota	Se agregó un método getObject(ResultSet rs) que permite obtener los objetos resultantes a partir de las transacciones realizadas con la base de datos

c) Crear una interfaz grafica para hacer uso de los métodos expuestos por el FACADE.

Es este punto de la práctica es posible crear una instancia del patrón FACADE y probar los métodos ofertados por este, a continuación se coloca un fragmento de código que ejemplifica como agregar un registro a la BD.

```
3   package uni.test;
4
5   import bt.modelo.dto.DispositivoDTO;
6   import bt.modelo.facade.DispositivoFACADE;
7   import java.util.logging.Level;
8   import java.util.logging.Logger;
9
10
11  public class Test {
12      public static void main(String[] args) {
13          DispositivoDTO dto = new DispositivoDTO();
14          dto.setDireccionMac("llllll");
15          dto.setEstado("conectado");
16          dto.setTipo("cell");
17          dto.setUsuario("Yo");
18          DispositivoFACADE facade = new DispositivoFACADE();
19          try {
20              facade.append(dto);
21          } catch (Exception ex) {
22              Logger.getLogger(Test.class.getName()).log(Level.SEVERE, null, ex);
23          }
```

Figura 9 Inserción de un dato desde un objeto FACADE

En esta última parte de la práctica el alumno ya cuenta con los conocimientos necesarios para crear una interfaz grafica que permita realizar las transacciones básicas con base de datos.

C) CÁLCULOS Y REPORTE

Se realizarán preguntas al alumno para verificar la comprensión del tema.

5. RESULTADOS Y CONCLUSIONES

El alumno podrá realizar una conexión a una Base de datos utilizando los patrones asociados a esta tarea.

6. ANEXOS

VideoTutorial (Creación de la base de datos directamente desde NetBeans)
Script de la base de datos en caso de usar un manejador de base de datos externo
Lista de correspondencia de los tipos de datos entre una base de datos y java.

7. REFERENCIAS