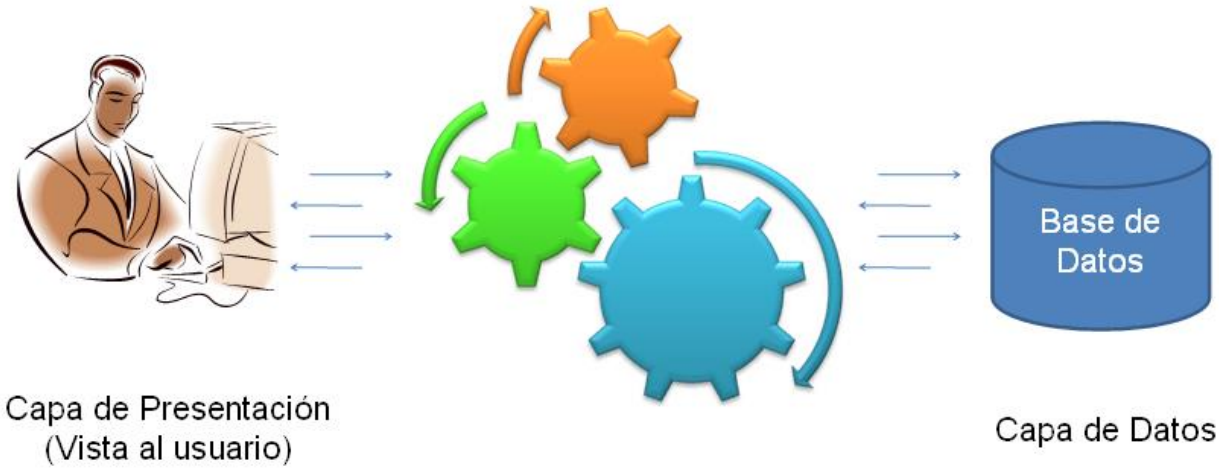
MODULO II
Desarrollo de aplicaciones avanzadas

Instructora:
Issela Mejía
issela_mejia@ricaldone.edu.sv

Arquitectura de Software - Modelo de 3 capas

La arquitectura se refiere a la forma en la que se diseñada una aplicación, tanto física como lógicamente. **Diseño físico** se refiere al lugar donde estarán las piezas de la aplicación y **Diseño lógico** es donde se especifica la estructura de la aplicación y sus componentes, sin tener en cuenta donde se localizará el software ni el hardware ni la infraestructura.

Existen una serie arquitecturas para la fabricación de software, por lo cual unicamente nos enfocaremos en una: **Modelo de 3 capas**.



El modelo de 3 capas está destinado a ayudar a construir componentes físicos a partir de los niveles lógicos. Así que se puede empezar tomando decisiones sobre qué parte lógica de la aplicación se va a encapsular en cada uno de los componentes, de igual modo que se encapsula los componentes en varios niveles.

Un nivel está conformado por varios componentes, por lo tanto, suple varios servicios a la vez. La relación entre los niveles puede ser bilateral, es decir, demandan y ofrecen servicios.

Capa de presentación

Los componentes del nivel de usuario, proporcionan la **interfaz visual** que los clientes utilizarán para ver la información y los datos. En este nivel, los componentes son responsables de solicitar y recibir servicios de otros componentes del mismo nivel o del nivel de servicios de negocio. Es muy importante destacar que, a pesar de que las funciones del negocio residen en otro nivel, **para el usuario es transparente la forma de operar**.

Capa de negocios

Como los servicios de usuario no pueden contactar directamente con el nivel de servicios de datos, es responsabilidad de los servicios de negocio hacer de puente entre estos. Los **objetos** de negocio (**clases**) proporcionan servicios que completan las tareas de negocio tales como **verificar los datos enviados por el usuario**, antes de llevar a cabo una transacción en la base de datos.

Capa de Datos

El nivel de datos se encarga de las típicas tareas que realizamos con los datos: **inserción**, **modificación**, **consulta** y **eliminación**. La clave del nivel de datos es que las tareas de negocio no son implementadas aquí, aunque un componente de servicio de datos es responsable de la gestión de las peticiones realizadas por un objeto de negocio. Un nivel de servicios de datos apropiadamente implementado, **debería permitir cambiar su localización sin afectar a los servicios proporcionados por los componentes de negocio.**

Conexión entre Java y SQL Server

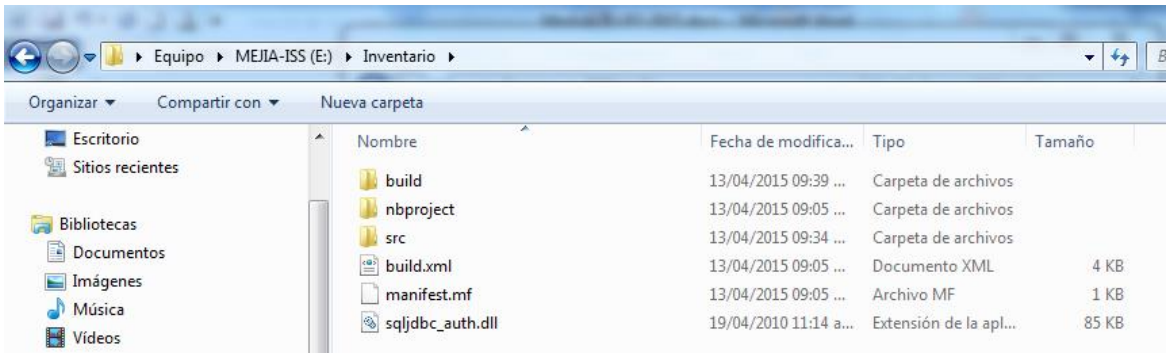
Para el desarrollo de ésta y demás practicas del módulo, se utilizaran los siguientes componentes:

- SQL Server
- SQL Server Management Studio
- JDK
- NetBeans IDE

- sqljdbc.jar (driver para establecer la conexión con la base de datos)
- sqljdbc_auth.dll (librería para la autenticación con Windows – x32 o x64 según sea el caso)

Agregar la librería para controlar la autenticación

Copiar la librería sqljdbc_auth.dll en el directorio raíz donde se encuentra el projecto, como se muestra en la imagen:



Agregar el driver para la conexión

En Netbeans, dentro del proyecto ubicarse en Libraries (Librerias), dar clic derecho, seleccionar agregar Archivo (Add File) JAR/carpeta, seleccionar la librería y darle abrir o agregar.

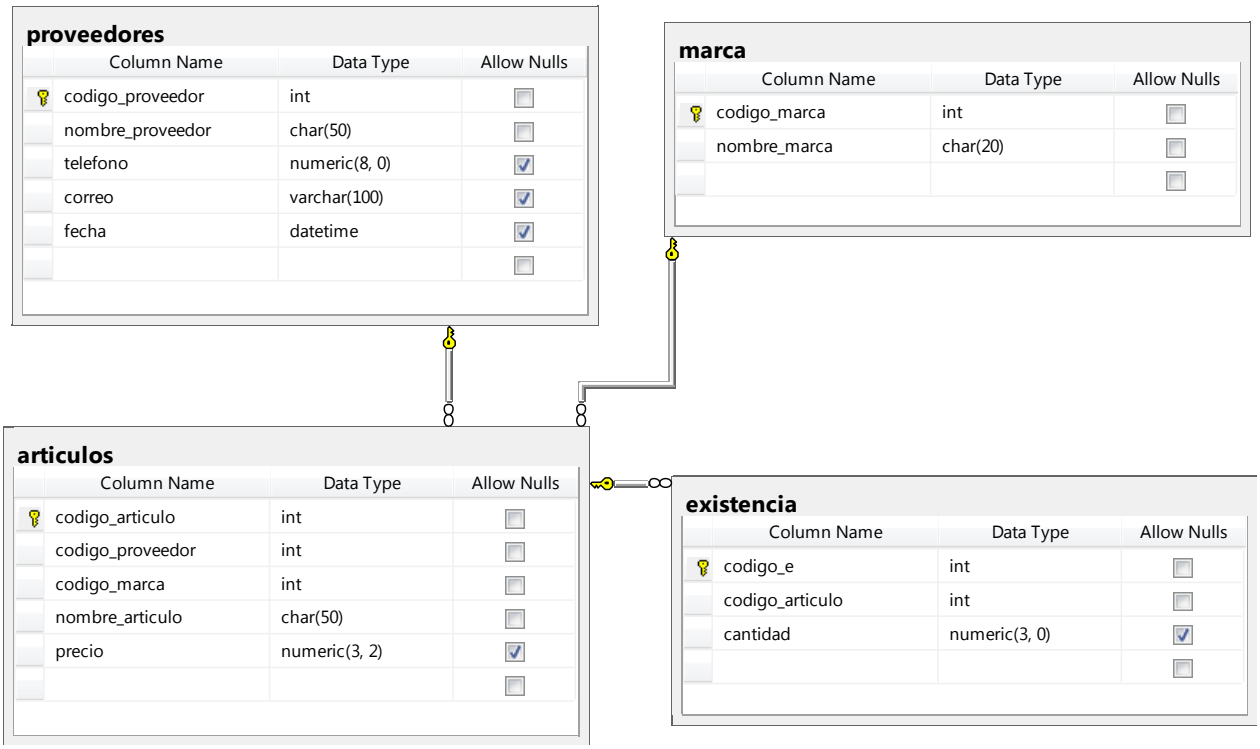
Planteamiento del ejercicio:

Desarrollar un sistema informático que gestione el modelo de inventario de una empresa, aplicando el modelo de 3 capas.

Creando la capa de datos:

Base de datos: **inventario**

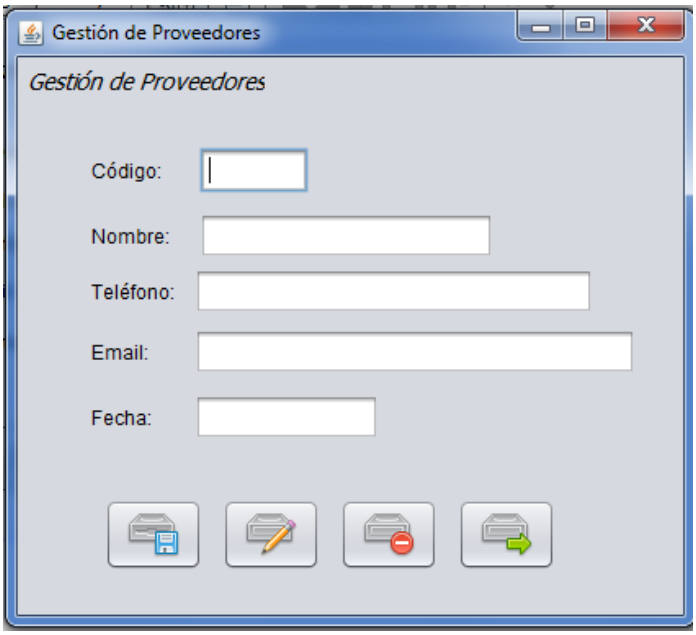
Utilizando SQL Server Management Studio, se construirá la base de datos en base al siguiente modelo:



Tener presente el nombre del servidor y el nombre de la instancia de SQL Server donde se crea la base de datos, ya que serán utilizados al momento de establecer la conexión entre la capa de negocios y la capa de datos.

Diseñando la capa de presentación:

Se crea un JFrame llamado frmProveedores, que llevara como título Gestión de Proveedores, desarrollando el diseño que se muestra a continuación:



Pasos complementarios antes de pasar a la siguiente capa:

- Agregar el driver (.jar) a las librerias del proyecto.
- Copiar la librería (.dll) al directorio raíz del proyecto.
- Verificar que el protocolo TCP/IP de SQL Server este habilitado. Esto se hace en el Sql Server Configuration Manager.
- Verificar que el servicio SQL Server Browser este en ejecución.

Creando la capa de negocios:

En esta parte se va a colocar la mayor parte de la programacion, esto deberá hacerse dentro de clases independientes, ya que la interfaz de usuario unicamente servirá para pedir y mostrar datos.

Tomar en cuenta lo siguiente:

Se crearán 2 clases: una llamada **Conexion** que contendrá la conexión a la base de datos y la otra **MtoProveedores** que contendrá el mantenimiento a la tabla **proveedores** (agregar, modificar, eliminar y consultar).

A continuación se presenta el código para la clase conexión:

Clase conexión

```
package inventario;

import java.sql.Connection;
import java.sql.*;

/**...4 lines */
public class conexion {
   public Connection conectar()
    {
       Connection cn = null;
       try
       {
           Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
           cn = DriverManager.getConnection("jdbc:sqlserver://ISSELA_PC\\SQLEXPRESS;databaseName=inventario;integratedSecurity=true;");
       }
       catch(Exception ex)
       {
           System.out.println(ex.getMessage());
       }
       return cn;
    }
```

Clase Mantenimiento de Proveedores

```
package inventario;

import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;

/**
 *
 * @author Issela
 */
public class MtoProveedores {
//Declarando los atributos de la clase
   private Connection cn;
   private Integer codigo;
   private String nombre;
   private Double telefono;
   private String email;
   private Date fecha;
```

/*metodos set y get para los atributos, puede generarlos dando clic derecho, seleccionar reestructurar y luego encapsulate field, deja el set y get para los campos necesarios y luego seleccionar refactor.
La otra forma es seleccionar los atributos, luego clic derecho, insertar código, luego Getter y Setter, seleccionar los atributos y luego generar. También los puede escribir si asilo prefiere.*/

```
 /**
    * @return the codigo
    */
   public Integer getCodigo() {
      return codigo;
```

```
}
/**
 * @param codigo the codigo to set
 */
public void setCodigo(Integer codigo) {
   this.codigo = codigo;
}
/**
 * @return the nombre
 */
public String getNombre() {
   return nombre;
}
/**
 * @param nombre the nombre to set
 */
public void setNombre(String nombre) {
   this.nombre = nombre;
}
/**
 * @return the telefono
 */
public Double getTelefono() {
   return telefono;
}
/**
 * @param telefono the telefono to set
 */
public void setTelefono(Double telefono) {
   this.telefono = telefono;
}
/**
 * @return the email
 */
public String getEmail() {
   return email;
}

/**
 * @param email the email to set
 */
public void setEmail(String email) {
   this.email = email;
}

/**
 * @return the fecha
```

```
     */
    public Date getFecha() {
        return fecha;
    }

    /**
     * @param fecha the fecha to set
     */
    public void setFecha(Date fecha) {
        this.fecha = fecha;
    }
```

Se utiliza el constructor para establecer la conexión

```
    public MtoProveedores()
    {
        //Establecemos la conexión
        conexion con = new conexion();
        cn = con.conectar();
    }
```

Creando los metodos para guarder, modificar, eliminar y consultar

```
    public boolean guardarProveedor()
    {
        boolean resp = false;
        try
        {
            //Realizar consulta INSERT
            String sql = "INSERT INTO proveedores (codigo, nombre, telefono, email, fecha) "
                    + "VALUES(?, ?, ?, ?, ?)";
            PreparedStatement cmd = cn.prepareStatement(sql);
            //Llenar los parámetros de la clase
            cmd.setInt(1, codigo);
            cmd.setString(2, nombre);
            cmd.setDouble(3, telefono);
            cmd.setString(4, email);
            cmd.setDate(5, fecha);
            //Si da error devuelve 1, caso contrario 0
            //Tomar en cuenta el "!" de negación
            if(!cmd.execute())
            {
                resp = true;
            }
            cmd.close();
            cn.close();
        }
        catch (Exception e) {
            System.out.println(e.toString());
```

```
        }
        return resp;
    }

public boolean consultarProveedor()
    {
        boolean resp = false;
        try
        {
            //Realizar consulta SELECT
            String sql = "SELECT * FROM proveedores WHERE codigo_proveedor = ?;";
            PreparedStatement cmd = cn.prepareStatement(sql);
            //Llenar los parámetros
            cmd.setInt(1, codigo);
            //Ejecutar la consulta
            ResultSet rs = cmd.executeQuery();
            //Recorrer la lista de registros
            if(rs.next()) {
                resp = true;
//asignándole a los atributos de la clase
                codigo = rs.getInt(1);
                nombre = rs.getString(2);
                telefono = rs.getDouble(3);
                email = rs.getString(4);
                fecha = rs.getDate(5);
            }
            cmd.close();
            cn.close();
        }
        catch (Exception e) {
            System.out.println(e.toString());
        }
        return resp;
    }

    public boolean modificarProveedor()
    {
        boolean resp = false;
        try
        {
            //Realizar consulta UPDATE
            String sql = "UPDATE proveedores SET nombre_proveedor = ?, telefono = ?, "
                    + "correo = ?, fecha = ? WHERE codigo_proveedor = ?;";
            PreparedStatement cmd = cn.prepareStatement(sql);
            //Llenar los parámetros como está en la clase
            cmd.setString(1, nombre);
            cmd.setDouble(2, telefono);
```

```
            cmd.setString(3, email);
            cmd.setDate(4, fecha);
            cmd.setInt(5, codigo);
            //Si da error devuelve 1, caso contrario 0
            //Tomar en cuenta el "!" de negación
            if(!cmd.execute())
            {
               resp = true;
            }
            cmd.close();
            cn.close();
         }
         catch (Exception e) {
            System.out.println(e.toString());
         }
         return resp;
      }

      public boolean eliminarProveedor()
      {
         boolean resp = false;
         try
         {
            //Realizar consulta DELETE
            String sql = "DELETE FROM proveedores WHERE codigo_proveedor = ?;";
            PreparedStatement cmd = cn.prepareStatement(sql);
            //Llenar los parámetros
            cmd.setInt(1, codigo);
            //Si da error devuelve 1, caso contrario 0
            //Tomar en cuenta el "!" de negación
            if(!cmd.execute())
            {
               resp = true;
            }
            cmd.close();
            cn.close();
         }
         catch (Exception e) {
            System.out.println(e.toString());
         }
         return resp;
      }
}
```

Se destacan algunos puntos importantes:

1. Las variables globales corresponden a cada campo de la capa de datos (codigo, nombre, telefono, email, fecha).

2. Los métodos: **get** y **set** se utilizan para obtener y colocar los valores de las variables globales, que posteriormente se utilizan en las consultas DML.
3. En el método constructor `public MtoProveedores()` se realiza la conexión con la base de datos.
4. Los métodos independientes donde se realizan las 4 consultas DML (guardar, consultar, modificar, eliminar), contienen sentencias parametrizadas para evitar problemas como SQL Injection.

Ahora establecemos el enlace entre la capa de presentación y la capa de negocio a traves de los botones:

BOTON LIMPIAR

```
txtCodigo.setText("");
txtNombre.setText("");
txtTelefono.setText("");
txtEmail.setText("");
txtFecha.setText("");
```

BOTÓN GUARDAR

```
//Tomando los datos del formulario y pasándolos a los atributos de la clase
MtoProveedores obj = new MtoProveedores();
 obj.setCodigo(Integer.toString(txtCodigo.getText()));
 obj.setNombre(txtNombre.getText());
 obj.setTelefono(Double.parseDouble(txtTelefono.getText()));
 obj.setEmail(txtEmail.getText());
 obj.setFecha(Date.valueOf(txtFecha.getText()));
 if(obj.guardarProveedor())
 {
    JOptionPane.showMessageDialog(this, "Datos guardados");
 }
 else
 {
    JOptionPane.showMessageDialog(this, "Error al guardar datos");
 }
```

BOTON MODIFICAR

```
MtoProveedores obj = new MtoProveedores();
 obj.setCodigo(Integer.parseInt(txtCodigo.getText()));
 obj.setNombre(txtNombre.getText());
 obj.setTelefono(Double.parseDouble(txtTelefono.getText()));
 obj.setEmail(txtEmail.getText());
```

```
obj.setFecha(Date.valueOf(txtFecha.getText()));
if(obj.modificarProveedor())
{
   JOptionPane.showMessageDialog(this, "Datos modificados");
}
else
{
   JOptionPane.showMessageDialog(this, "Error al modificar");
}
```

BOTÓN ELIMINAR

```
MtoProveedores obj = new MtoProveedores();
   obj.setCodigo(Integer.parseInt(txtCodigo.getText()));
   int eliminar = JOptionPane.showConfirmDialog(this, "¿Está seguro que desea eliminar?",
         "Atención", JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
   if(eliminar == 0)
   {
      if(obj.eliminarProveedor())
      {
         JOptionPane.showMessageDialog(this, "Datos eliminados");
      }
      else
      {
         JOptionPane.showMessageDialog(this, "Error al eliminar");
      }
   }
```

BOTÓN CONSULTAR

```
MtoProveedores obj = new MtoProveedores();
   obj.setCodigo(Integer.parseInt(txtCodigo.getText()));
   if(obj.consultarProveedor())
   {
      txtNombre.setText(obj.getNombre());
      txtTelefono.setText(obj.getTelefono().toString());
      txtEmail.setText(obj.getEmail());
      txtFecha.setText(obj.getFecha().toString());
   }
   else
   {
      JOptionPane.showMessageDialog(this, "Datos no encontrados");
   }
```

PRÁCTICA:

- Crear el mantenimiento para las tablas restantes.