

Conexión a Bases de Datos.

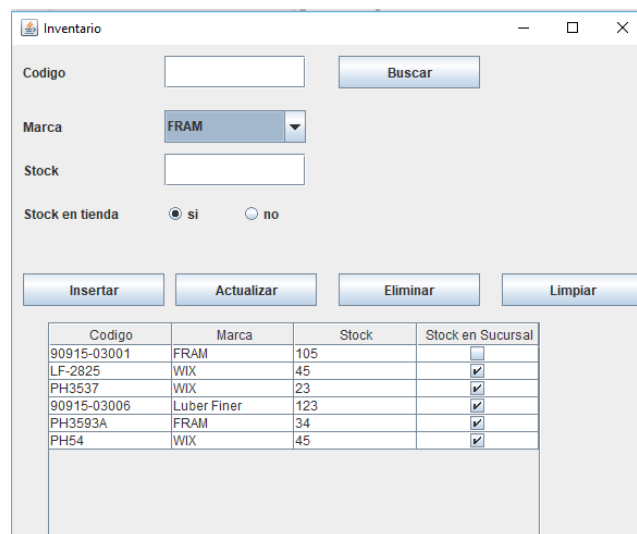
JDBC significa Java Data Base Connectivity, que es una API Java estándar para la conectividad independiente de la base de datos entre el lenguaje de programación Java y una amplia gama de bases de datos.

La biblioteca JDBC incluye API para cada una de las tareas comúnmente asociadas con el uso de la base de datos:

- Haciendo una conexión a una base de datos
- Crear instrucciones SQL.
- Ejecutando consultas SQL.
- Visualización y modificación de los registros resultantes

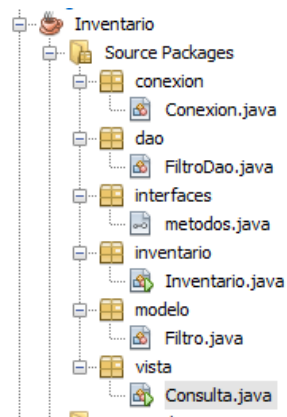
Practica Guiada

El objetivo de la practica será realizar el siguiente formulario que nos permitirá administrar un pequeño inventario de filtros de aceite para vehículos.



Codigo	Marca	Stock	Stock en Sucursal
90915-03001	FRAM	105	<input type="checkbox"/>
LF-2825	WIX	45	<input checked="" type="checkbox"/>
PH3537	WIX	23	<input checked="" type="checkbox"/>
90915-03006	Luber Finer	123	<input checked="" type="checkbox"/>
PH3593A	FRAM	34	<input checked="" type="checkbox"/>
PH54	WIX	45	<input checked="" type="checkbox"/>

1. Crearemos la siguiente estructura de paquetes.



2. Clase Filtro.java

```
public class Filtro {  
    private int id;  
    private String codigo;  
    private String marca;  
    private int stock;  
    private boolean existencia;  
  
    public Filtro() {  
    }  
  
    public Filtro(int id, String codigo, String marca, int stock, boolean existencia) {  
        this.id = id;  
        this.marca = marca;  
        this.stock = stock;  
        this.existencia = existencia;  
        this.codigo = codigo;  
    }  
  
    public Filtro(String codigo, String marca, int stock, boolean existencia) {  
        this.codigo = codigo;  
        this.marca = marca;  
        this.stock = stock;  
        this.existencia = existencia;  
    }  
  
    public Filtro(String marca, int stock, boolean existencia) {  
        this.marca = marca;  
        this.stock = stock;  
        this.existencia = existencia;  
    }  
}
```

3. Clase Conexión.java.

Class.forName("Driver"): Nos permite abrir un canal para poder establecer la comunicación con la base de datos.

DriverManager.getConnection (): Devuelve un objeto Connection, el cual representa la conexión física a la base de datos. Recibe como parámetros:

- ✓ Url: Es el lugar donde está alojado nuestra base de datos, esta se escribe de la siguiente forma: jdbc:mysql://server/nombre_bd, en donde server es la dirección ip o hostname del servidor.
- ✓ User: Usuario de la bd.
- ✓ Pass: Es la contraseña del usuario de User.

```
public class Conexion {
    private String user;
    private String pass;
    private String driver;
    private String url;

    private Connection cnx;

    public static Conexion instance;

    public synchronized static Conexion conectar()
    {
        if(instance == null)
        {
            return new Conexion();
        }
        return instance;
    }
}
```

```
private Conexion() {
    cargarCredenciales();

    try {
        //le enviamos el driver que usara para la conexion a la base de datos
        Class.forName(this.driver);
        cnx = (Connection) DriverManager.getConnection(this.url, this.user, this.pass);
    } catch (ClassNotFoundException | SQLException ex) {
        Logger.getLogger(Conexion.class.getName()).log(Level.SEVERE, null, ex);
    }
}

//credenciales de mi servidor de bases de datos
private void cargarCredenciales() {
    user = "root";
    pass="";
    driver = "com.mysql.jdbc.Driver";
    url = "jdbc:mysql://localhost/filtros";
}

public Connection getCnx() {
    return cnx;
}

public void cerrarConexion()
{
    instance = null;
}

} //fin de clase Conexion
```

4. Clase Interfaz metodos.java.

El objetivo de la interfaz será para establecer los métodos para insertar, eliminar, buscar y actualizar.

```
public interface metodos <Generic>{  
    public boolean create(Generic g);  
    public boolean delete (Object key);  
    public boolean update (Generic c);  
  
    public Generic read(Object key);  
    public ArrayList<Generic> readAll();  
}
```

5. Clase FiltroDao.java.

Objeto	Descripción
PreparedStatement	<p>Se usa para ejecutar consultas parametrizadas. Con lo cual podemos evitar ataques de tipo SQLInjection. Retorna un objeto de tipo ResultSet.</p> <p>Métodos</p> <p>.prepareStatement("SQL") -> Enviamos una consulta como la siguiente: DELETE FROM filtros_aceite WHERE codFiltro=?, donde "?" es el valor que parametrizaremos o en otra palabras preparamos la consulta.</p> <p>.setTipoDato(posición,dato) -> Nos permite cambiar el "?" por el dato que enviaremos a la base datos. La posición comienza a partir del número 1. Para la consulta del ejemplo anterior nos quedaría: .setString(1, "dato").</p> <p>.executeUpdate() -> Ejecutamos la consulta, si fue exitosa nos devolverá un número mayor a 0.</p>
Satement	<p>proporciona métodos para ejecutar consultas con la base de datos. Retorna un objeto de tipo ResultSet.</p> <p>Métodos</p> <p>.executeQuery("SQL") -> Ejecuta una consulya sin parámetros decir no lleva "?".</p>

ResultSet	<p>Mantiene un cursor apuntando a una fila de una tabla. Inicialmente, el cursor apunta a antes de la primera fila.</p> <p>Métodos.</p> <p>.Next() -> Mueve el cursor a la siguiente fila de la tabla.</p> <p>Rs.getTipoDato() -> Extrae el dato de una columna especifica. Puede recibir como parámetro el número de la columna (ojo aca inicia apartir del numero 1) o el nombre de la columna.</p> <p>.close() -> Cierra el cursor.</p>
------------------	--

```

public class FiltroDao implements metodos<Filtro> {
    //creando nuestra queries

    private static final String SQL_INSERT = "INSERT INTO filtros_aceite (codFiltro,marca,stock,existencia) VALUES (?, ?, ?, ?)";
    private static final String SQL_UPDATE = "UPDATE filtros_aceite SET marca = ? ,stock = ?,existencia = ? WHERE codFiltro=?";
    private static final String SQL_DELETE = "DELETE FROM filtros_aceite WHERE codFiltro=?";
    private static final String SQL_READ = "SELECT * FROM filtros_aceite WHERE codFiltro=?";
    private static final String SQL_READALL = "SELECT * FROM filtros_aceite";

    @Override
    public boolean create(Filtro g) {
        //nos servira para preparar la consulta de insert
        PreparedStatement ps;
        try {
            ps = con.getCnx().prepareStatement(SQL_INSERT);
            ps.setString(1, g.getCodigo());
            ps.setString(2, g.getMarca());
            ps.setInt(3, g.getStock());
            ps.setBoolean(4, true);
            if (ps.executeUpdate() > 0) {
                return true;
            }
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
            Logger.getLogger(FiltroDao.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            con.cerrarConexion();
        }
        return false;
    }

    @Override
    public boolean delete(Object key) {

        PreparedStatement ps;
        try {
            ps = con.getCnx().prepareStatement(SQL_DELETE);
            ps.setString(1, key.toString());

            if (ps.executeUpdate() > 0) {
                return true;
            }
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
            Logger.getLogger(FiltroDao.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            con.cerrarConexion();
        }
        return false;
    }
}

```

```

@Override
public boolean update(Filtro c) {
    PreparedStatement ps;
    try {
        System.out.println(c.getCodigo());
        ps = con.getCnx().prepareStatement(SQL_UPDATE);
        ps.setString(1, c.getMarca());
        ps.setInt(2, c.getStock());
        ps.setBoolean(3, c.getExistencia());
        ps.setString(4, c.getCodigo());
        if (ps.executeUpdate() > 0) {
            return true;
        }
    } catch (SQLException ex) {
        System.out.println(ex.getMessage());
        Logger.getLogger(FiltroDao.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        con.cerrarConexion();
    }
    return false;
}

@Override
public Filtro read(Object key) {
    Filtro f = null;
    PreparedStatement ps;
    ResultSet rs;
    try {
        ps = con.getCnx().prepareStatement(SQL_READ);
        ps.setString(1, key.toString());

        rs = ps.executeQuery();

        while (rs.next()) {
            f = new Filtro(rs.getInt(1), rs.getString(2), rs.getString(3), rs.getInt(4), rs.getBoolean(5));
        }
        rs.close();
    } catch (SQLException ex) {
        System.out.println(ex.getMessage());
        Logger.getLogger(FiltroDao.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        con.cerrarConexion();
    }
    return f;
}

@Override
public ArrayList<Filtro> readAll() {
    ArrayList<Filtro> all = new ArrayList();
    Statement s;
    ResultSet rs;
    try {
        s = con.getCnx().prepareStatement(SQL_READALL);
        rs = s.executeQuery(SQL_READALL);
        while (rs.next()) {
            all.add(new Filtro(rs.getInt(1), rs.getString(2), rs.getString(3), rs.getInt(4), rs.getBoolean(5)));
        }
        rs.close();
    } catch (SQLException ex) {
        Logger.getLogger(FiltroDao.class.getName()).log(Level.SEVERE, null, ex);
    }
    return all;
}

```

6. Clase Consulta.java

```
public class Consulta extends JFrame {

    public JLabel lblCodigo, lblMarca, lblStock, lblExistencia;

    public JTextField codigo, descripcion, stock;
    public JComboBox marca;

    ButtonGroup existencia = new ButtonGroup();
    public JRadioButton no;
    public JRadioButton si;
    public JTable resultados;

    public JPanel table;

    public JButton buscar, eliminar, insertar, limpiar, actualizar;

    private static final int ANCHOC = 130, ALTOC = 30;

    DefaultTableModel tm;

    public Consulta() {
        super("Inventario");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(null);
        agregarLabels();
        formulario();
        llenarTabla();
        Container container = getContentPane();
        container.add(lblCodigo);
        container.add(lblMarca);
        container.add(lblStock);
        container.add(lblExistencia);
        container.add(codigo);
        container.add(marca);
        container.add(stock);
        container.add(si);
        container.add(no);
        container.add(buscar);
        container.add(insertar);
        container.add(actualizar);
        container.add(eliminar);
        container.add(limpiar);
        container.add(table);
        setSize(600, 600);
        eventos();
    }

    public final void agregarLabels() {
        lblCodigo = new JLabel("Codigo");
        lblMarca = new JLabel("Marca");
        lblStock = new JLabel("Stock");
        lblExistencia = new JLabel("Stock en tienda");
        lblCodigo.setBounds(10, 10, ANCHOC, ALTOC);
        lblMarca.setBounds(10, 60, ANCHOC, ALTOC);
        lblStock.setBounds(10, 100, ANCHOC, ALTOC);
        lblExistencia.setBounds(10, 140, ANCHOC, ALTOC);
        //////////////////////////////////////
    }
}
```

```

public final void formulario() {
    //elementos
    codigo = new JTextField();
    marca = new JComboBox();
    stock = new JTextField();
    si = new JRadioButton("si", true);
    no = new JRadioButton("no");
    resultados = new JTable();
    buscar = new JButton("Buscar");
    insertar = new JButton("Insertar");
    eliminar = new JButton("Eliminar");
    actualizar = new JButton("Actualizar");
    limpiar = new JButton("Limpiar");

    table = new JPanel();
    //agregar elementos al combobox marca
    marca.addItem("FRAM");
    marca.addItem("WIX");
    marca.addItem("Luber Finer");
    marca.addItem("OSK");
    //agregando los radio a un grupo
    existencia = new ButtonGroup();
    existencia.add(si);
    existencia.add(no);
    //////////////////////////////////////

    //////////////////////////////////////
    codigo.setBounds(140, 10, ANCHOC, ALTOC);
    marca.setBounds(140, 60, ANCHOC, ALTOC);
    stock.setBounds(140, 100, ANCHOC, ALTOC);
    si.setBounds(140, 140, 50, ALTOC);
    no.setBounds(210, 140, 50, ALTOC);

    buscar.setBounds(300, 10, ANCHOC, ALTOC);
    insertar.setBounds(10, 210, ANCHOC, ALTOC);
    actualizar.setBounds(150, 210, ANCHOC, ALTOC);
    eliminar.setBounds(300, 210, ANCHOC, ALTOC);
    limpiar.setBounds(450, 210, ANCHOC, ALTOC);
    resultados = new JTable();
    table.setBounds(10, 250, 500, 200);
    table.add(new JScrollPane(resultados));
}

```

Agregamos datos al
combo box

```

public void llenarTabla() {
    // Aca le colocamos el tipo de dato que tendra nuestras columnas
    // si el un dato booleano apareciera un checkbox en el JTABLE
    tm = new DefaultTableModel() {
        public Class<?> getColumnClass(int column) {
            switch (column) {
                case 0:
                    return String.class;
                case 1:
                    return String.class;
                case 2:
                    return String.class;
                default:
                    return Boolean.class;
            }
        }
    };
    //Agregamos las columnas que se mostraran con su respectivo nombre
    tm.addColumn("Codigo");
    tm.addColumn("Marca");
    tm.addColumn("Stock");
    tm.addColumn("Stock en Sucursal");

    ///Realizamos la consulta a nuestra base de datos por medio del metodo readALL
    FiltroDao fd = new FiltroDao();
    ArrayList<Filtro> filtros = fd.readAll();
}

```

Indicamos el tipo de
datos de las columnas

Agregamos las
columnas a nuestro
modelo


```

        //Agregamos el resultado a nuestro JTable
        // Se agragan de tipo Object
        for (Filtro fi : filtros) {
            tm.addRow(new Object[]{fi.getCodigo(), fi.getMarca(), fi.getStock(), fi.getExistencia()});
        }

        //le agramos el modelo a nuestra tabla
        resultados.setModel(tm);
    }

    public void eventos() {
        //insertar
        insertar.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                FiltroDao fd = new FiltroDao();
                Filtro f = new Filtro(codigo.getText(), marca.getSelectedItem().toString(),
                    Integer.parseInt(stock.getText()), true);

                if (no.isSelected()) {
                    f.setExistencia(false);
                }

                if (fd.create(f)) {
                    JOptionPane.showMessageDialog(null, "Filtro registrado con exito");
                    limpiarCampos();
                    llenarTabla();
                } else {
                    JOptionPane.showMessageDialog(null, "Ocurrio un problema al momento de crear el filtro");
                }
            }
        });

        actualizar.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                FiltroDao fd = new FiltroDao();
                Filtro f = new Filtro(codigo.getText(), marca.getSelectedItem().toString(),
                    Integer.parseInt(stock.getText()), true);

                if (no.isSelected()) {
                    f.setExistencia(false);
                }

                if (fd.update(f)) {
                    JOptionPane.showMessageDialog(null, "Filtro Modificado con exito");
                    limpiarCampos();
                    llenarTabla();
                } else {
                    JOptionPane.showMessageDialog(null, "Ocurrio un problema al momento de modificar el filtro");
                }
            }
        });

        eliminar.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                FiltroDao fd = new FiltroDao();
                if (fd.delete(codigo.getText())) {
                    JOptionPane.showMessageDialog(null, "Filtro Eliminado con exito");
                    limpiarCampos();
                    llenarTabla();
                } else {
                    JOptionPane.showMessageDialog(null, "Ocurrio un problema al momento de eliminar el filtro");
                }
            }
        });
    }

```

```

eliminar.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        FiltroDao fd = new FiltroDao();
        if (fd.delete(codigo.getText())) {
            JOptionPane.showMessageDialog(null, "Filtro Eliminado con exito");
            limpiarCampos();
            llenarTabla();
        } else {
            JOptionPane.showMessageDialog(null, "Ocurrio un problema al momento de eliminar el filtro");
        }
    }
});

buscar.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        FiltroDao fd = new FiltroDao();
        Filtro f = fd.read(codigo.getText());
        if (f == null) {
            JOptionPane.showMessageDialog(null, "EL filtro buscado no se ha encontrado");
        } else {
            codigo.setText(f.getCodigo());
            marca.setSelectedItem(f.getMarca());
            stock.setText(Integer.toString(f.getStock()));

            if (f.getExistencia()) {
                si.setSelected(true);
            } else {
                no.setSelected(true);
            }
        }
    }
});

limpiar.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        limpiarCampos();
    }
});

public void limpiarCampos() {
    codigo.setText("");
    marca.setSelectedItem("FRAM");
    stock.setText("");
}

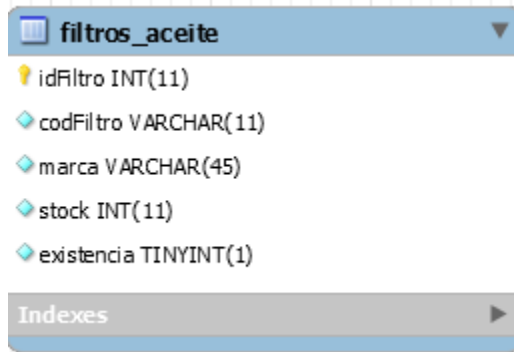
public static void main(String[] args) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            new Consulta().setVisible(true);
        }
    });
}
}

```

Base de datos.

Nombre de la base: filtros.

Gestor de Bases de Datos: MariaDB.



filtros_aceite	
idFiltro	INT(11)
codFiltro	VARCHAR(11)
marca	VARCHAR(45)
stock	INT(11)
existencia	TINYINT(1)
Indexes	