UNIREMINGTON®
CORPORACIÓN UNIVERSITARIA REMINGTON
RES. 2661 MEN JUNIO 21 DE 1996

DESARROLLO DE SOFTWARE II

TRANSVERSAL

FACULTAD DE CIENCIAS BÁSICAS E INGENIERÍA

Vicerrectoría de Educación a Distancia y virtual

2016

AUTOR

Cesar Augusto Jaramillo Henao
Ingeniero de Sistemas
Cesar.jaramillo@uniremington.edu.co

Nota: el autor certificó (de manera verbal o escrita) No haber incurrido en fraude científico, plagio o vicios de autoría; en caso contrario eximió de toda responsabilidad a la Corporación Universitaria Remington, y se declaró como el único responsable.

RESPONSABLES

Jorge Mauricio Sepúlveda Castaño
Decano de la Facultad de Ciencias Básicas e Ingeniería
jsepulveda@uniremington.edu.co

Eduardo Alfredo Castillo Builes
Vicerrector modalidad distancia y virtual
ecastillo@uniremington.edu.co

Francisco Javier Álvarez Gómez
Coordinador CUR-Virtual
falvarez@uniremington.edu.co

GRUPO DE APOYO

Personal de la Unidad CUR-Virtual
EDICIÓN Y MONTAJE

Primera versión. Febrero de 2011.
Segunda versión. Marzo de 2012
Tercera versión. noviembre de 2015
Cuarta versión 2016

TABLA DE CONTENIDO

Pág.

1 MAPA DE LA ASIGNATURA

DESARROLLO DE SOFTWARE II

PROPÓSITO GENERAL DEL MÓDULO

Los avances permanentes en el desarrollo de software nos obligan a estar a la vanguardia, esos temas que se convierten en estándares empresariales son los que tocaremos, buscando siempre las mejores prácticas en la elaboración de aplicativos competitivos, realizando almacenamiento permanente con BD, conociendo los conceptos de la multitarea, el trabajo en red, las herramientas complementarias como los frameworks y una introducción a la programación web.

OBJETIVO GENERAL

Desarrollar habilidades que permitan la creación de software, abarcando un sin número de opciones que permitan dar solución a las necesidades del mercado según su experticia, cubriendo el almacenamiento permanente de la información en motores de bases de datos, el uso de herramientas para le expansión y los recursos para un desempeño ilimitado de posibilidades, además de herramientas de fácil y rápida manipulación y dando la alternativa para la migración o complementándolo a la web según el perfil.

OBJETIVOS ESPECÍFICOS

UNIDAD 1

Complementar habilidades de almacenamiento de la información en repositorios adicionales a los vistos en semestres previos, permitiendo un panorama de opciones de gran utilidad y de gran expansión.

UNIDAD 2

Aplicar herramientas que permitan el aprovechamiento de los recursos del sistema independientemente del sistema operativo sobre el que esté desarrollando, buscando en todos los casos el mejoramiento en rendimiento y haciendo más eficiente la respuesta solicitada

UNIDAD 3

Ampliar el alcance de los desarrollos no solo a nivel unipersonal sino a un sistema de gran envergadura con múltiples usuarios con un alcance transaccional, seguro y de gran escalabilidad.

UNIDAD 4

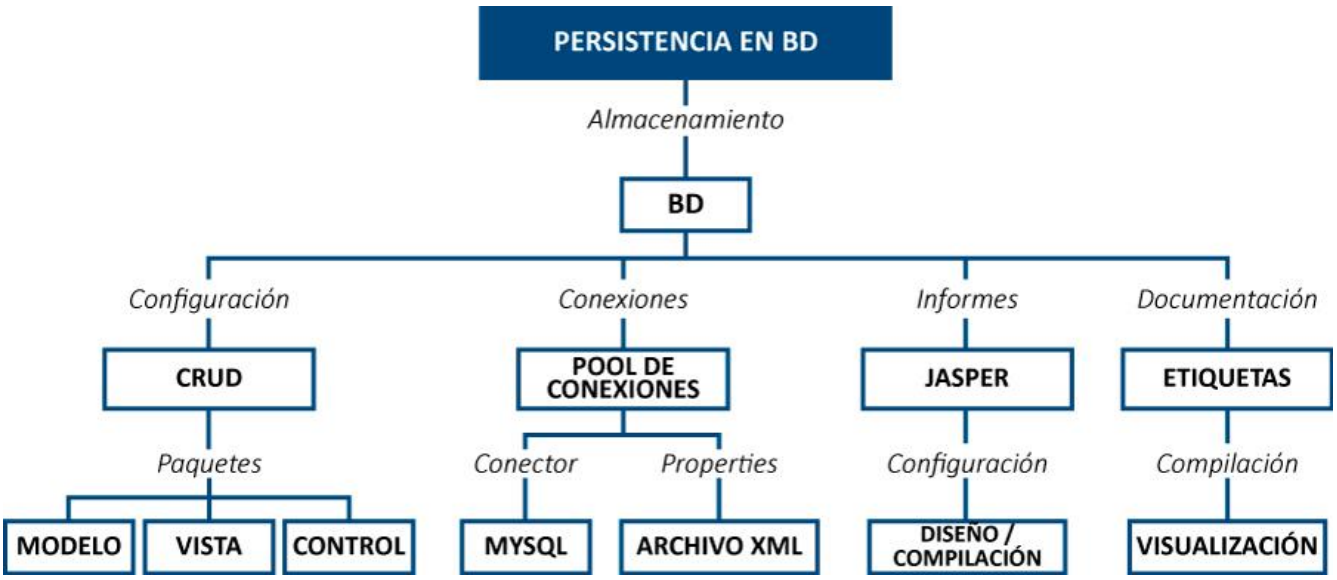
aplicar herramientas que facilitan la creación de tareas típicas, dando espacio para entrar en la capa de negocio de una manera rápida y aplicando mayor tiempo a las pruebas y soporte del desarrollo.

UNIDAD 5

identificar herramientas de actualidad con un enfoque hacia la web, utilizando distintas alternativas y su implementación con un sistema de almacenamiento, pasando por el diseño y los controles de la información.

2 UNIDAD 1 PERSISTENCIA EN BASES DE DATOS

2.1.1 RELACIÓN DE CONCEPTOS



BD	Estructura que permite el almacenamiento de información de forma organizada y relacionada
CRUD	Acrónimo de Create, Read, Update y Delete (Insertar, Consultar, Modificar y Eliminar)
Paquete	espacio creado para organizar la información y / o conjunto de clases del sistema
Modelo	paquete o capa donde se administran los datos (clase principal)
Vista	todo lo relacionado con las tareas que ve el usuario
Control	paquete que representa la lógica del negocio
Conexión	archivo o clase que estable las características que permiten la comunicación entre una forma y un espacio de almacenamiento
Pool de Conexión	herramienta de conexión con múltiples posibilidades, no limita los archivos ni depende de la compilación del proyecto
MySQL	Administrador de BD
XML	Lenguaje muy común con etiquetas “personalizadas” que permiten la comunicación o administración de datos
Jasper	aplicativo que permite la creación de reportes personalizados

Etiquetas	conjunto de elementos preestablecido para la creación de la documentación de un aplicativo
Compilación	creación de la documentación en un proyecto con formato HTML, ideal para el seguimiento paso a paso de los componentes lógicos de un aplicativo

2.1.2 OBJETIVO GENERAL

Complementar habilidades de almacenamiento de la información en repositorios adicionales a los vistos en semestres previos, permitiendo un panorama de opciones de gran utilidad y de gran expansión.

2.1.3 OBJETIVOS ESPECÍFICOS

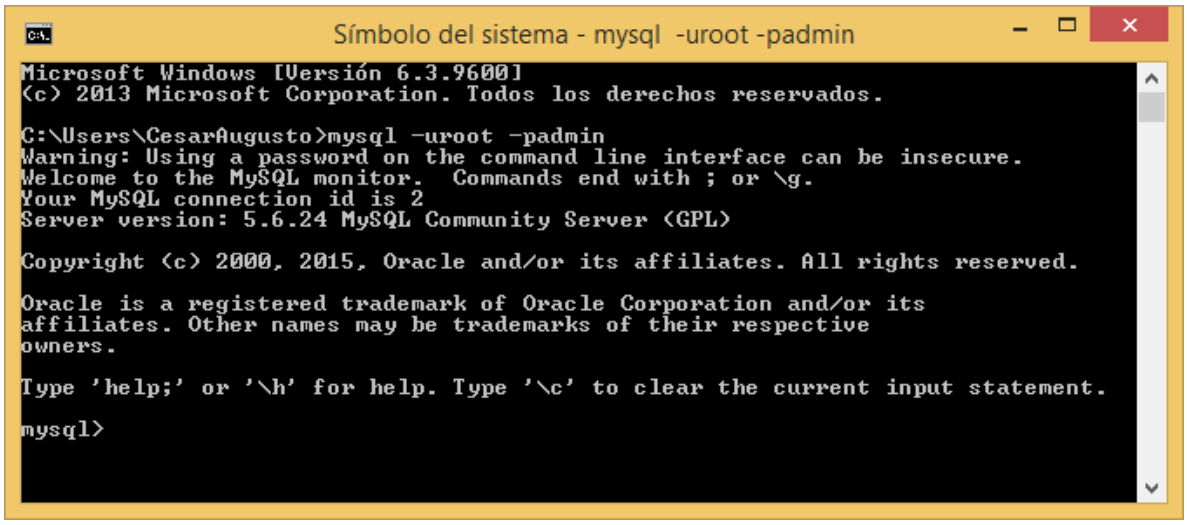
- Desarrollar adecuadamente los procesos del CRUD mediante Bases de Datos, con todos los requerimientos necesarios.
- Complementar el desarrollo de un aplicativo típico con herramientas como informes para el usuario final y documentación para el desarrollador.

2.2 TEMA 1 APLICACIÓN CRUD

Durante mucho tiempo se han realizado aplicaciones simples con manejo de la información en la memoria (**RAM**), el cual es un aspecto fundamental para el inicio de las primeras aplicaciones permitiendo hacer pruebas, pero todo de manera temporal. Dentro de la evolución de este tema se encontrara el manejo de archivos, opción muy útil cuando se desea almacenar la información de forma permanente, este proceso toma el nombre de CRUD por sus siglas en ingles (Create, Read, Update and Delete) Crear, Obtener, Actualizar y Borrar, pero para este nivel se tomara el camino de las Bases de Datos (BD), que funciona de una forma similar a los archivos aunque mucho mas estructurado.

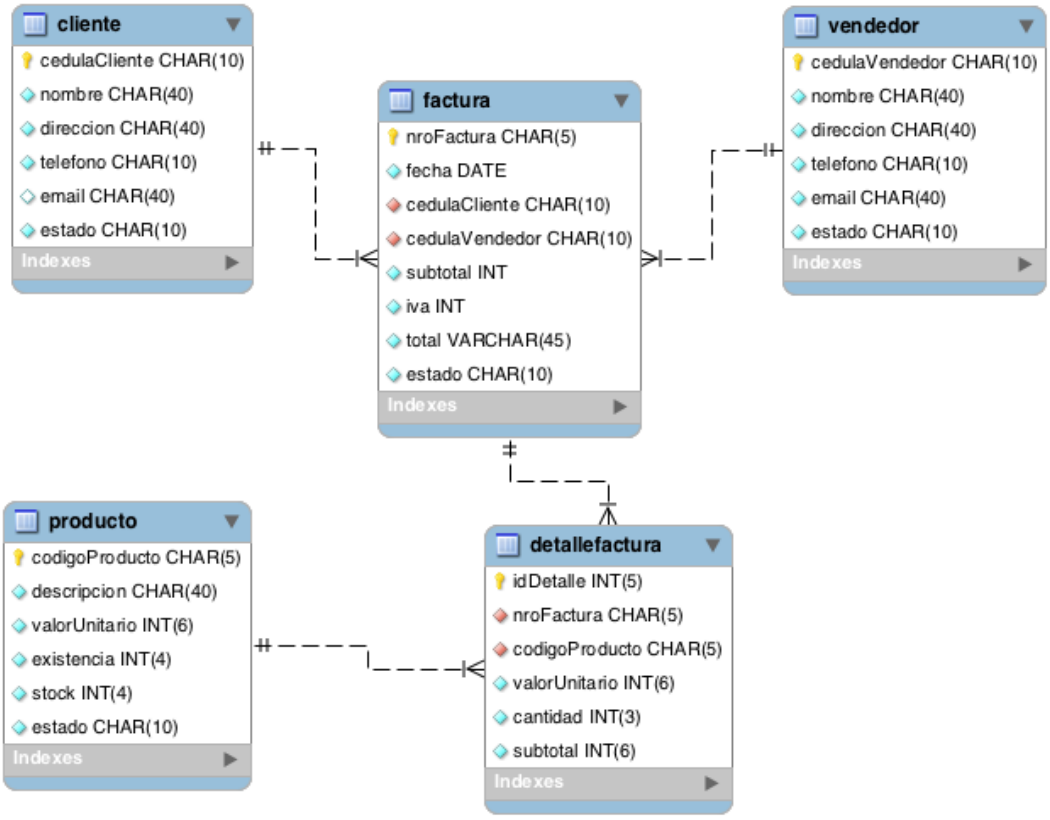
Nuestra herramienta de trabajo para las BD es MySQL, aunque existe una gran variedad de herramientas que realizan tareas similares, MySQL es una herramienta muy sencilla de manejar, con gran alcance, esta utilidad se podrán descargar desde el sitio www.mysql.com.

Después de la descarga e instalación y su posterior ingreso encontramos una consola como la siguiente.



Este es el ambiente típico de trabajo, podrían igualmente utilizar la herramienta phpmyadmin o herramientas comunes como mysql-workbench o mysql query Browser entre muchas otras, y nos permitirán realizar las operaciones esenciales desde un ambiente grafico.

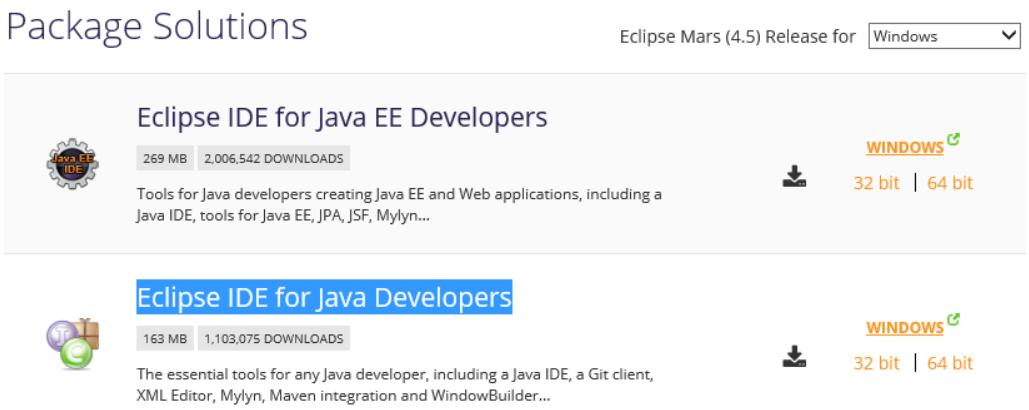
Lo primero que se creara para este propósito es una BD y sus respectivas tablas, generando un MER (Modelo Entidad Relación) como el siguiente.



Con este MER podremos trabajar esta primera unidad relacionada con el CRUD, para tal efecto aplicaremos un concepto adiciona de las BD como son los procedimientos almacenados, esta herramienta nos hará el proceso un poco más simplificado en la codificación a utilizar, los procedimientos que se van a utilizar son listar, consultar, modificar, eliminar e insertar

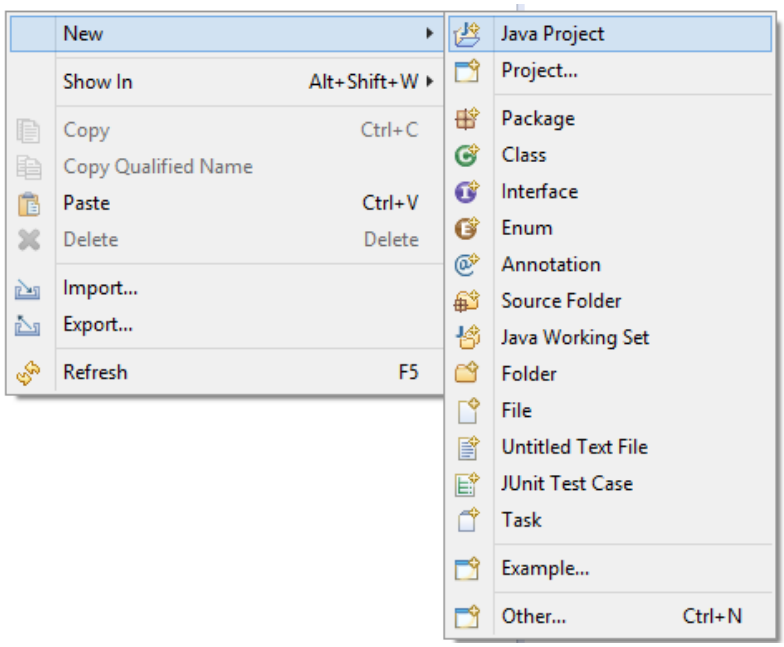
Proyecto

Utilizando el IDE (Ambiente de Desarrollo Integrado) de su preferencia, sea este Eclipse, NetBeans, JDevelopert entre muchos otros, este proyecto particularmente se desarrollará mediante Eclipse SE, este se podrá descargar del sitio www.eclipse.org.

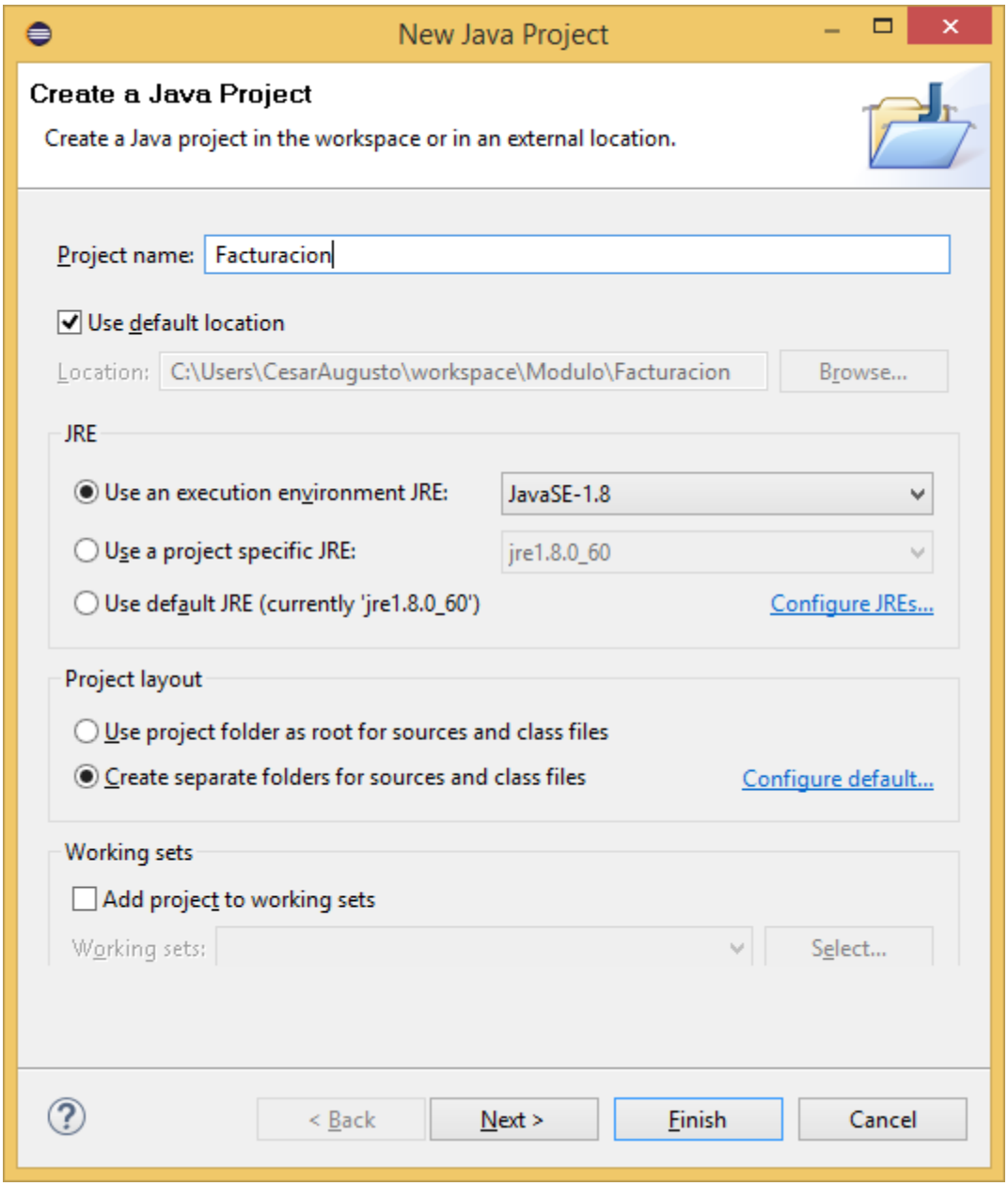


Creación del Proyecto Facturación

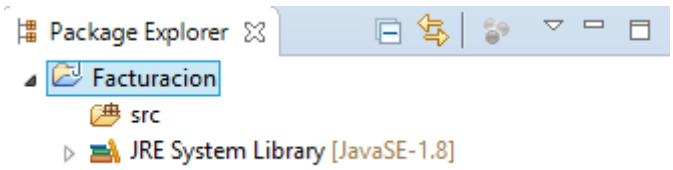
Luego de ingresar al IDE, con el botón emergente en el área izquierda de la pantalla (Package Explorer), se selecciona new / Java Project



Nombre del Proyecto Facturación

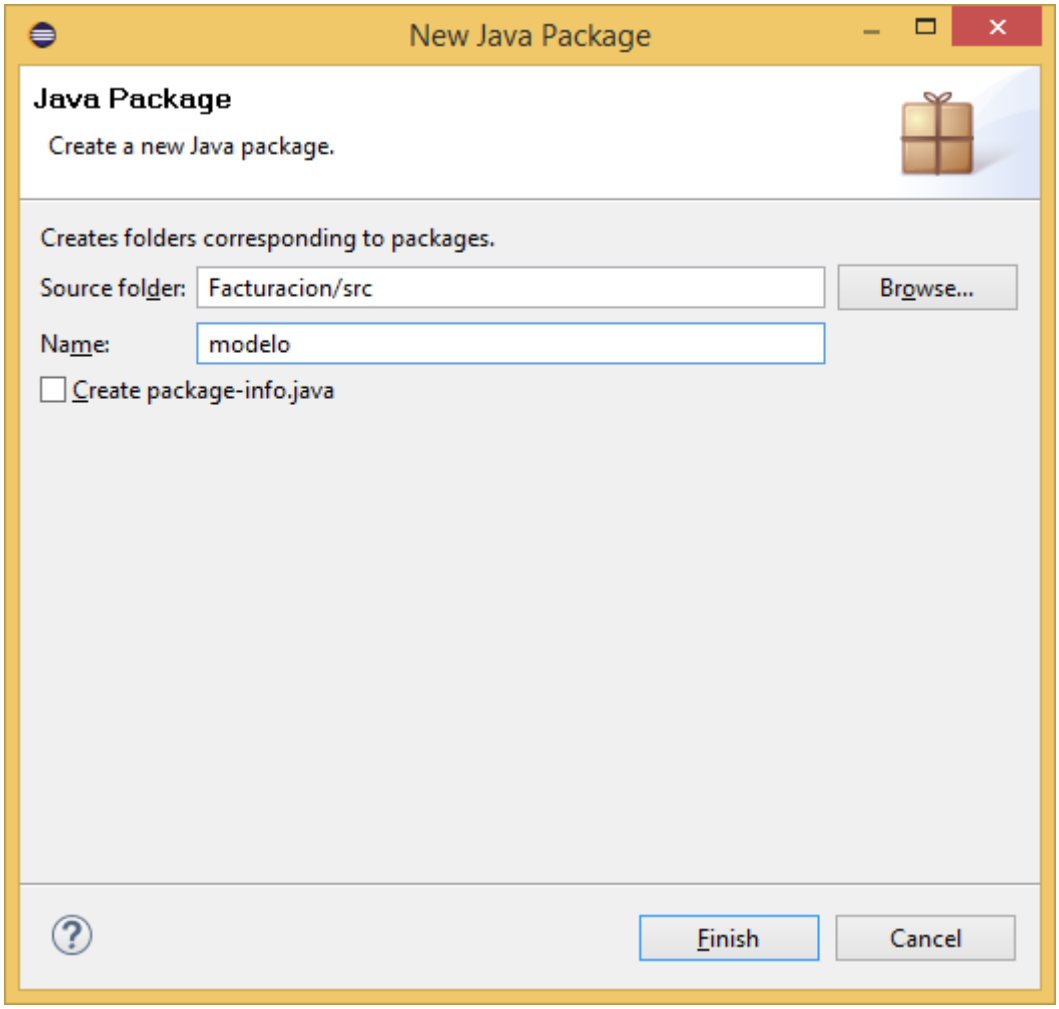


Se dejan las opciones de configuración por defecto y se selecciona el botón Finish y se obtiene el siguiente resultado



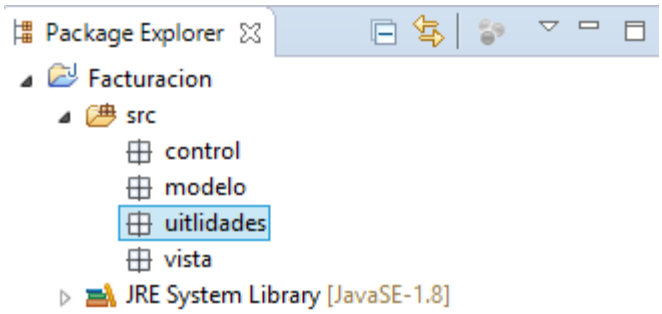
En la opción src irán los archivos y / o paquetes del proyecto, para una mejor clasificación de los archivos, el propósito de estos es crearan 4 paquetes con los nombres modelo, vista, control y utilidades.

Con el botón emergente sobre el nombre del proyecto se selecciona New / Package



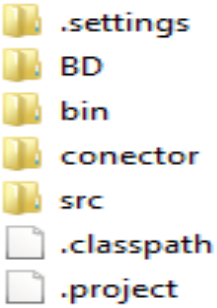
Luego el botón Finish para terminar la creación del paquete, este proceso se realiza para cada paquete especifico.

El resultado es el siguiente



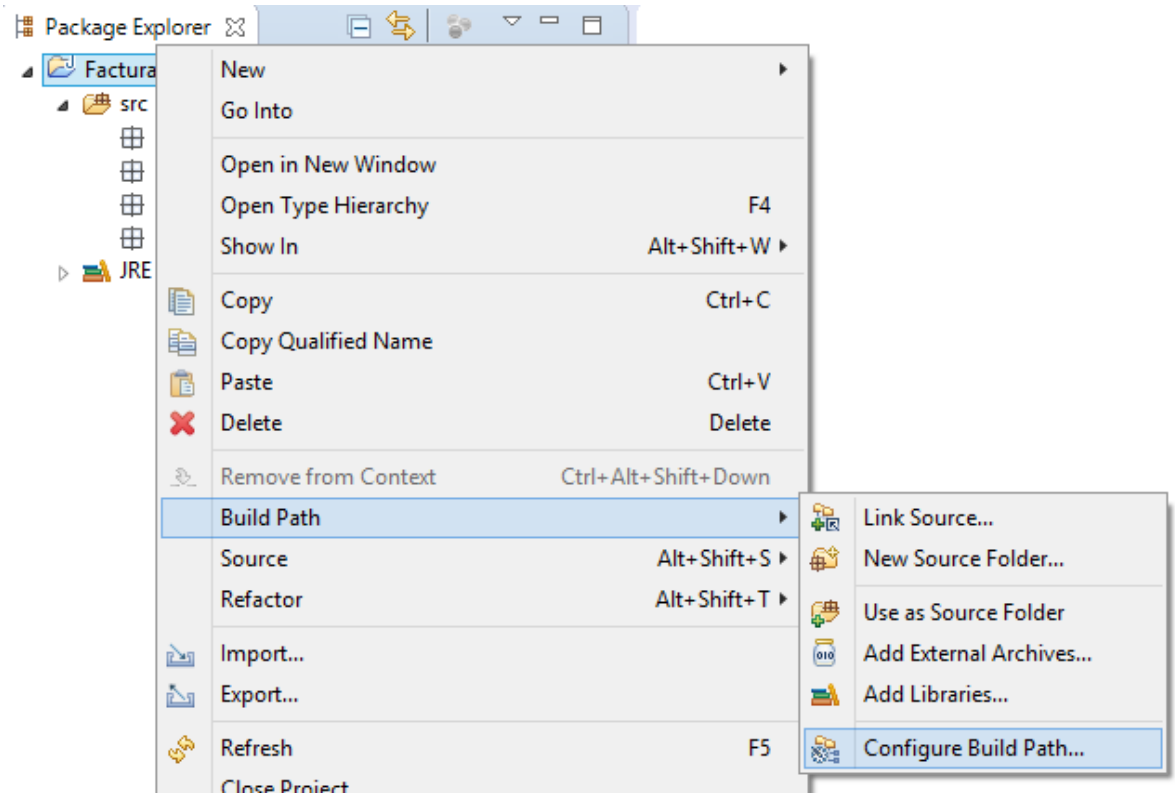
Conector de la BD

Para establecer un proyecto con BD requerimos de un archivo (librería) externo que podemos descargar del sitio www.mysql.com, este archivo tiene por extensión .jar y su nombre es mysql-connector-java-5.1.36 (cambia la ultima parte según la versión), este archivo se puede descargar compreso o de instalación, después de haberlo realizado, descomprima dicho archivo y copie el archivo mysql-connector-java-5.1.36-bin.jar en su proyecto para un trasporte mas simplificado (no tiene que ser siempre esta ruta), cree en el proyecto una carpeta con el nombre conector o librería y una carpeta con el nombre de BD para mayor control de los elementos:

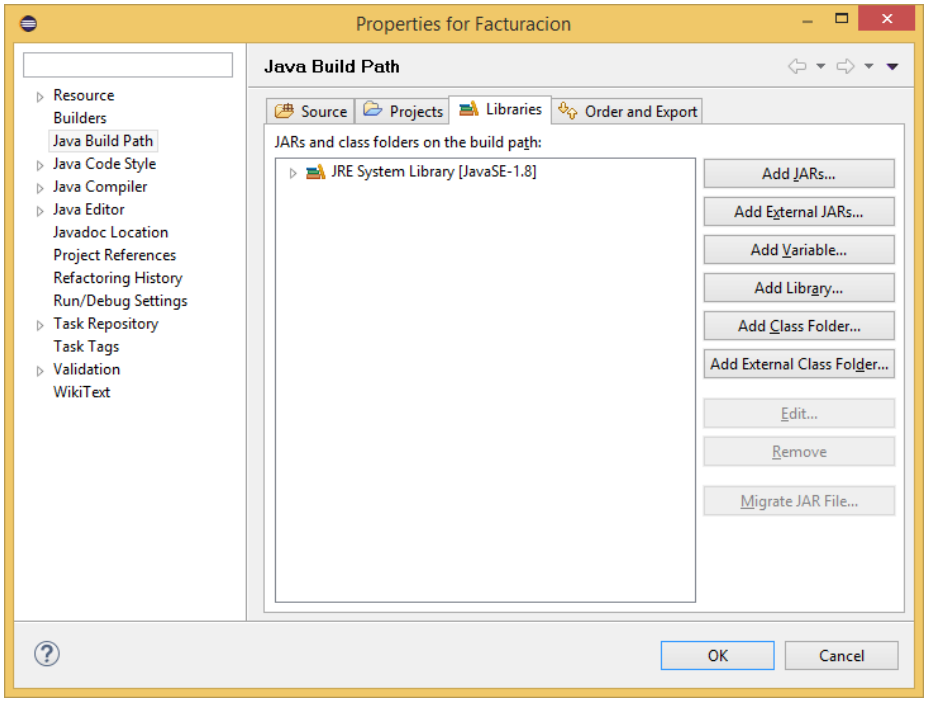


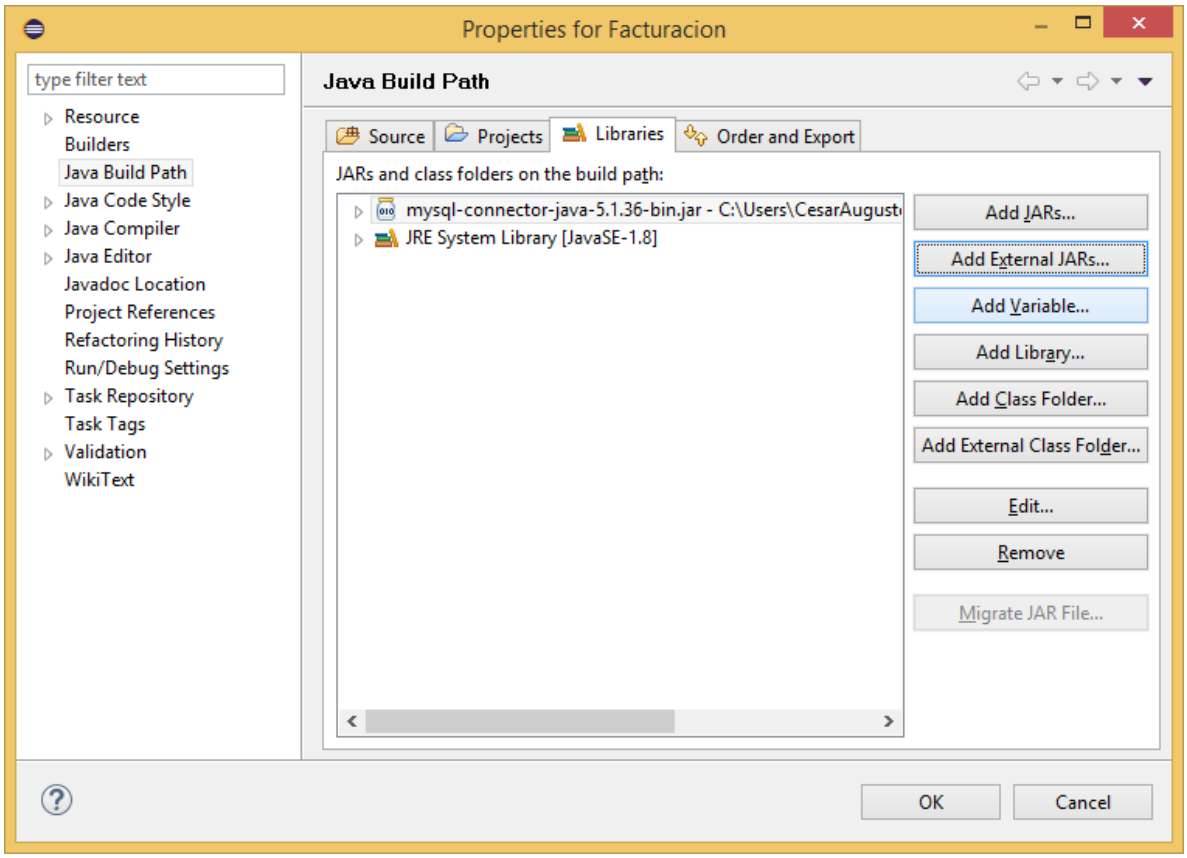
Regrese al proyecto, para la vinculación de este archivo .jar con el proyecto de facturación se aplica lo siguiente

Botón emergente sobre el proyecto



En la ventana resultante seleccione el botón Add External JARs… y busque el archivo mysql-connector-java-5.1.36-bin.jar dentro de la carpeta creada recientemente





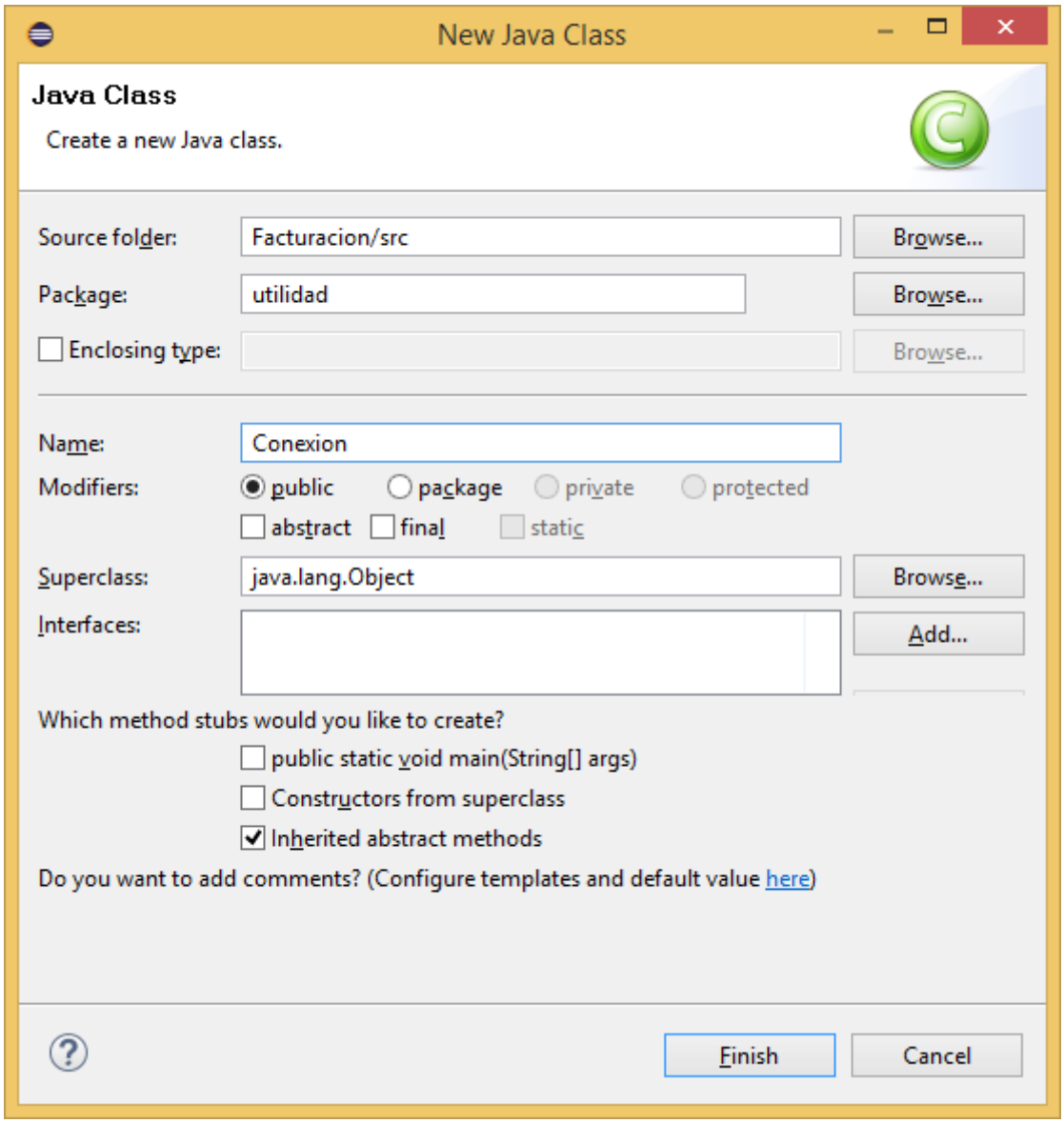
Seleccione el botón OK, con este proceso el sistema ya tiene el conector vinculado.

Archivo de Conexión a la BD

Uno de los procesos fundamentales dentro del manejo de un CRUD con BD es el archivo o proceso de conexión, este nos establece un vínculo directo entre la BD y el proyecto, el proceso inicial ira cambiando con el paso del tiempo, será un proceso muy elemental, básico y con muchas limitantes pero es funcional, a medida que se avance en el tema se trabajara en la solución de estos posibles problemas mediante el Pool de Conexiones.

Creación de un archivo Java para Conexión

Ubicados en el paquete Utilidades y mediante el botón emergente, selecciona New / Class



Este primer archivo no es definitivo, servirá como prueba de trabajo y verificación de funcionamiento

```
package utilidad;

import java.sql.Connection;

public class Conexion {

    static Connection con = null;

    public static Connection getConnection () {

        try {

            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection("jdbc:mysql://localhost/facturacion", "root", "admin");
        }
        catch (SQLException se) {

            se.printStackTrace();
            throw new RuntimeException ("Error al crear la Conexion");
        }
        catch (ClassNotFoundException ce) {

            ce.printStackTrace();
            throw new RuntimeException ("Error al crear la Conexion");
        }
        return con;
    }
}
```

El archivo inicial para pruebas se conformaría como el caso anterior, se tiene una clase típica Conexión, existe una clase Connection, esta clase contiene los estándares y las formas apropiadas para establecer una conexión a la BD, un DriverManager, contiene el protocolo que especifica el servidor local, se conforma por los siguientes parámetros

jdbc: conectividad de BD de Java

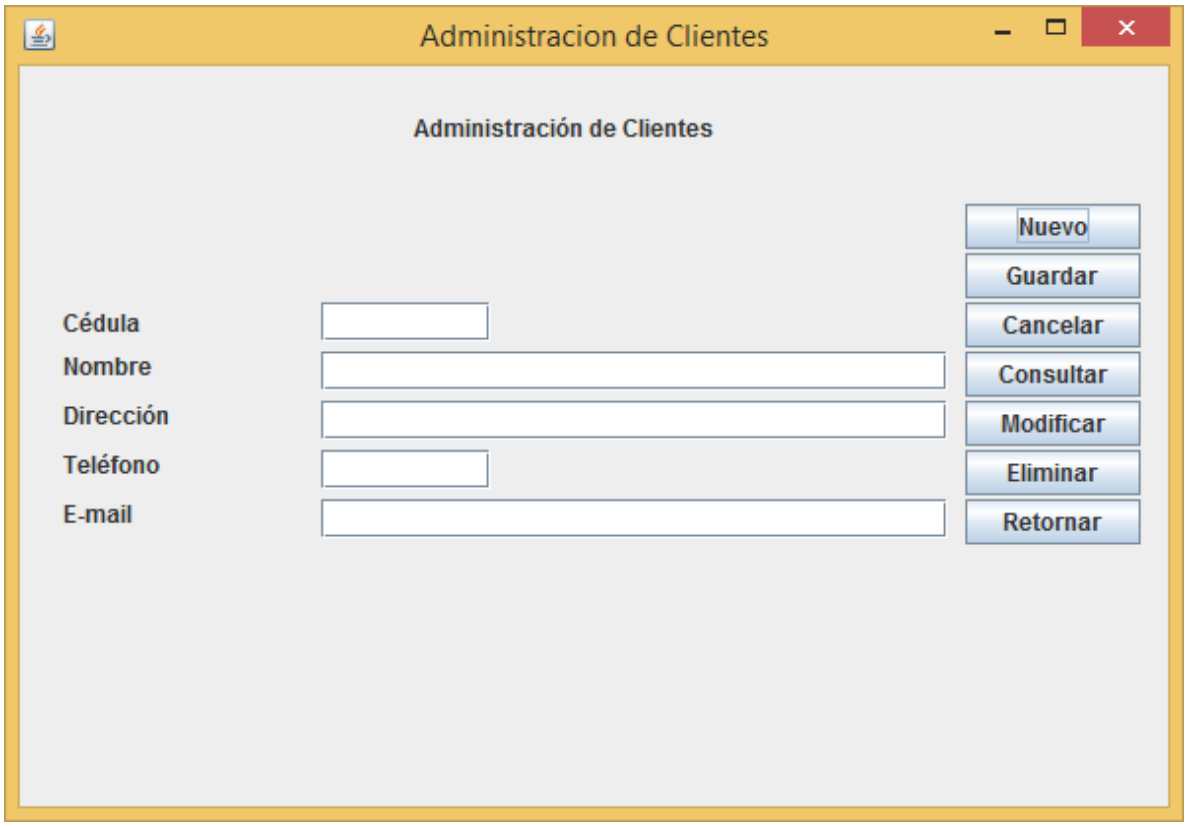
mysql://localhost: especifica el servidor local sobre el que se va a realizar la operación de la BD

Facturación: representa la BD de trabajo

root: usuario por defecto de mysql, este puede ser cambiado o creado

admin: contraseña del sistema de BD de MySQL, puede ser cambiada o creada

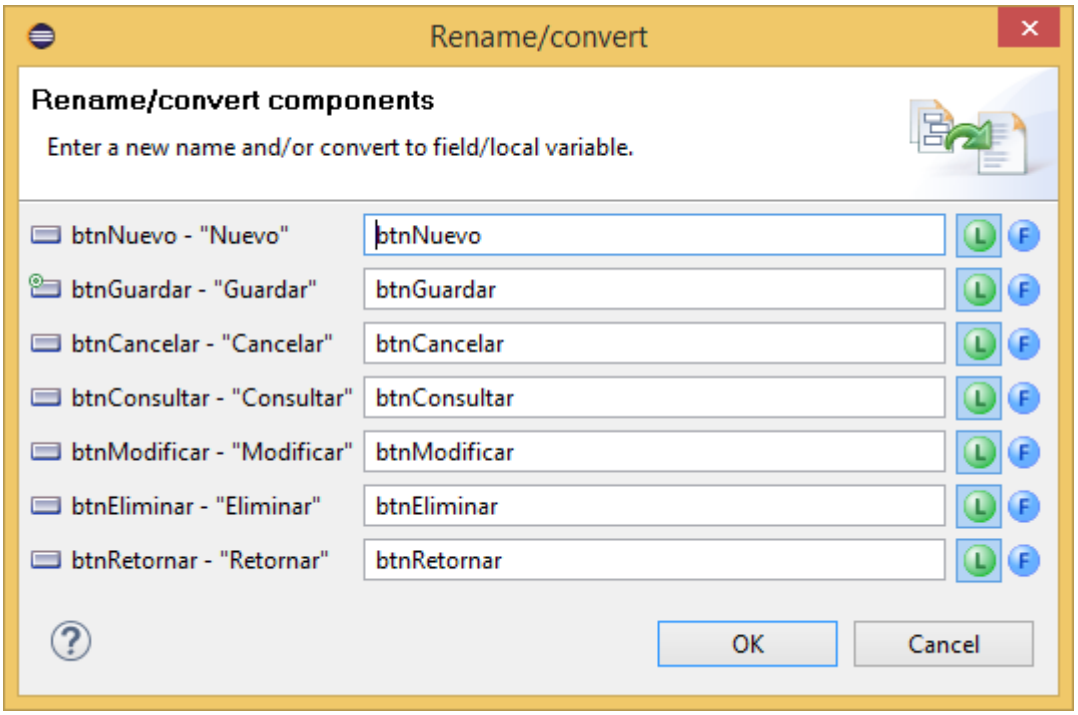
Con este proceso terminado, se procede a la construcción de la primera interfaz grafica, que contendrá las opciones necesarias para representar la tabla de cliente según el MER, este proceso se creara dentro del paquete vista y tendrá el nombre FrmCliente, se trabajara con un JFrame.



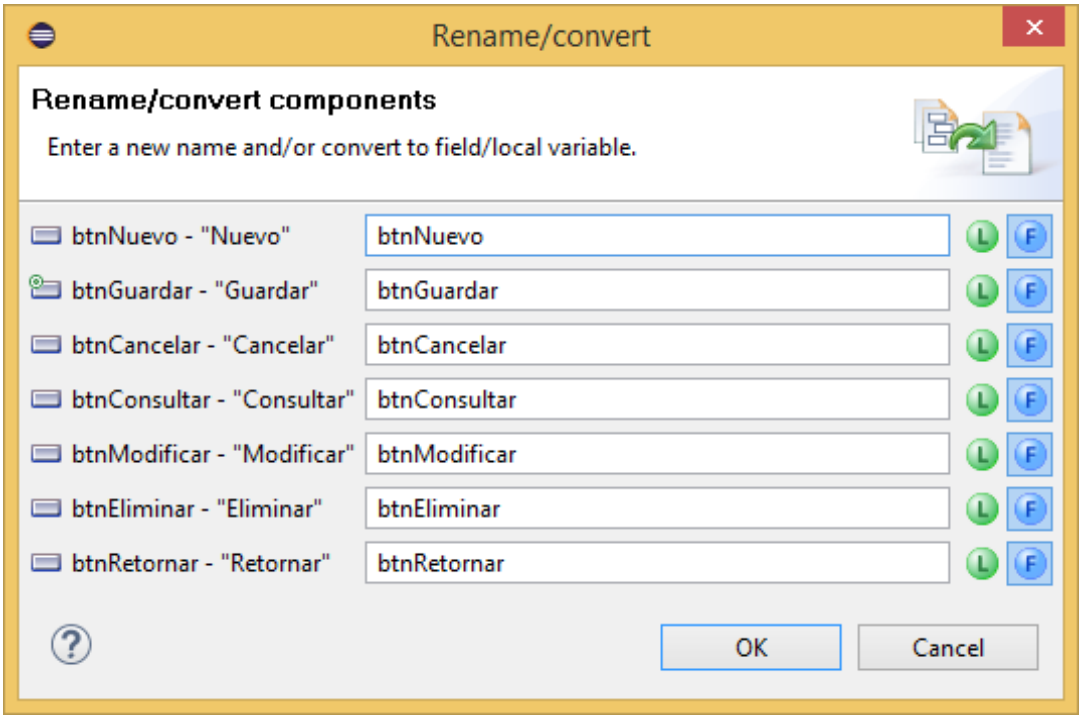
Este formulario inicial de trabajo deberá estar completamente validado en sus cajas de texto permitiendo el control de los datos, cedula debe de ser un valor numérico, de máximo 10 cifras, no deberá estar vacío, de cumplirse estas condiciones y presionar la tecla enter llevará el foco al campo de nombre, cada campo debe cumplir condiciones similares según los requerimientos que se establezcan al inicio del aplicativo, el ultimo campo llevará el foco al botón de Guardar.

Cambios en los Botones de comando

Los botones de comando están definidos como variables y para el propósito de mayor alcance se requiere que estén como campos, el cambio de este alcance es muy simple, solo seleccionamos los botones, con el menú emergente selecciona Rename.



Cambie L (local) por F (fields)



Creación de métodos funcionales

Existe un sin numero de opciones para el buen funcionamiento de los procesos, uno de ellos es la construcción de los métodos que permitan la adecuada administración de las tareas, dentro de ellos tendremos un método que limpie las cajas de texto, de habilite/deshabilite los botones de comando, etc., en el Source o código de este formulario, ubicados al final del archivo se crearan los siguientes.

- Método Limpiar

Diseñado para las cajas de texto

```
private static void limpiar () {

    txtCedula.setText("");
    txtNombre.setText("");
    txtDireccion.setText("");
    txtTelefono.setText("");
    txtEmail.setText("");
}
```

Método Deshabilitar

Bloquea las cajas de texto al inicio del aplicativo

```
private static void deshabilitar () {

    txtCedula.setEditable(false);
    txtNombre.setEditable(false);
    txtDireccion.setEditable(false);
    txtTelefono.setEditable(false);
    txtEmail.setEditable(false);
}
```

Métodos de activar y desactivar

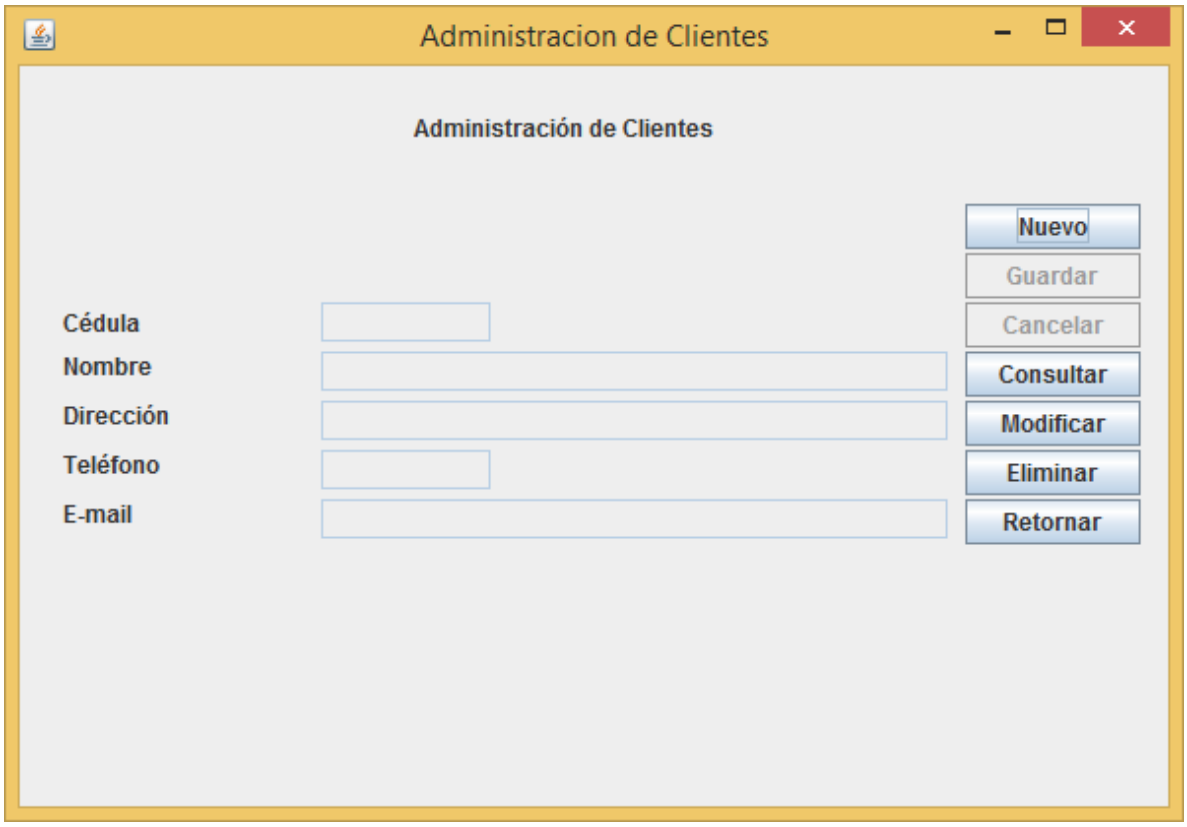
Controla los botones según la operación que se este realizando

```
private static void activar () {

    btnNuevo.setEnabled(true);
    btnGuardar.setEnabled(false);
    btnCancelar.setEnabled(false);
    btnConsultar.setEnabled(true);
    btnModificar.setEnabled(true);
    btnEliminar.setEnabled(true);
    btnRetornar.setEnabled(true);
}

private static void desactivar () {

    btnNuevo.setEnabled(false);
    btnGuardar.setEnabled(false);
    btnCancelar.setEnabled(true);
    btnConsultar.setEnabled(false);
    btnModificar.setEnabled(false);
    btnEliminar.setEnabled(false);
    btnRetornar.setEnabled(false);
}
```

Llamado de los métodos

```
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                FrmCliente frame = new FrmCliente();
                frame.setVisible(true);
                limpiar ();
                activar ();
                deshabilitar ();

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
```

Así se vera el proyecto con estos cambios



Creación de variables adicionales

```
public class FrmCliente extends JFrame {

    private JPanel contentPane;
    private static JTextField txtCedula;
    private static JTextField txtNombre;
    private static JTextField txtDireccion;
    private static JTextField txtTelefono;
    private static JTextField txtEmail;
    private static JButton btnNuevo;
    private static JButton btnGuardar;
    private static JButton btnCancelar;
    private static JButton btnConsultar;
    private static JButton btnModificar;
    private static JButton btnEliminar;
    private static JButton btnRetornar;
    private int sw = 0;
    private int opc = 0;
```

En el inicio del archivo, se encontrarán las definiciones de las cajas de texto, botones y demás elementos que comprendan el formulario de trabajo, al final se declararan 2 variables complementarias, sw será una bandera

de encendido y apagado, mediante esta se determinara si existe o no un elemento consultado, la variable opc se utilizara como clasificación según la operación a realizar, sea esta una inserción, consulta, modificación o eliminación.

Asignación de los botones de comando

La siguiente asignación de código a cada botón permitirá el buen desempeño de las operaciones, para realizar esta tarea basta con presionar doble clic sobre el botón mencionado.

Botón Nuevo

```
limpiar ();
desactivar ();
deshabilitar ();
opc = 1;
txtCedula.setEditable(true);
txtCedula.requestFocus();
```

Botón Cancelar

```
limpiar ();
activar();
deshabilitar ();
```

Botón Guardar

```
deshabilitar ();
activar ();
```

Botón Consultar

```
limpiar ();
deshabilitar ();
opc = 2;
txtCedula.setEditable(true);
txtCedula.requestFocus();
```

Botón Modificar

```
limpiar ();
desactivar ();
deshabilitar ();
opc = 3;
txtCedula.setEditable(true);
txtCedula.requestFocus();
```

Botón Eliminar

```
limpiar ();
deshabilitar ();
opc = 4;
txtCedula.setEditable(true);
txtCedula.requestFocus();
```

Botón Salir

```
dispose ();
```

En cada botón se aplican una serie de métodos que permiten desde limpiar las cajas de texto, hasta bloquearlas, esto con el fin de evitar posibles ingresos involuntario de información que pueden afectar las tareas principales del formulario, estos procesos acompañados de las validaciones se vuelven fundamentales para unas buenas practicas del control de los datos.

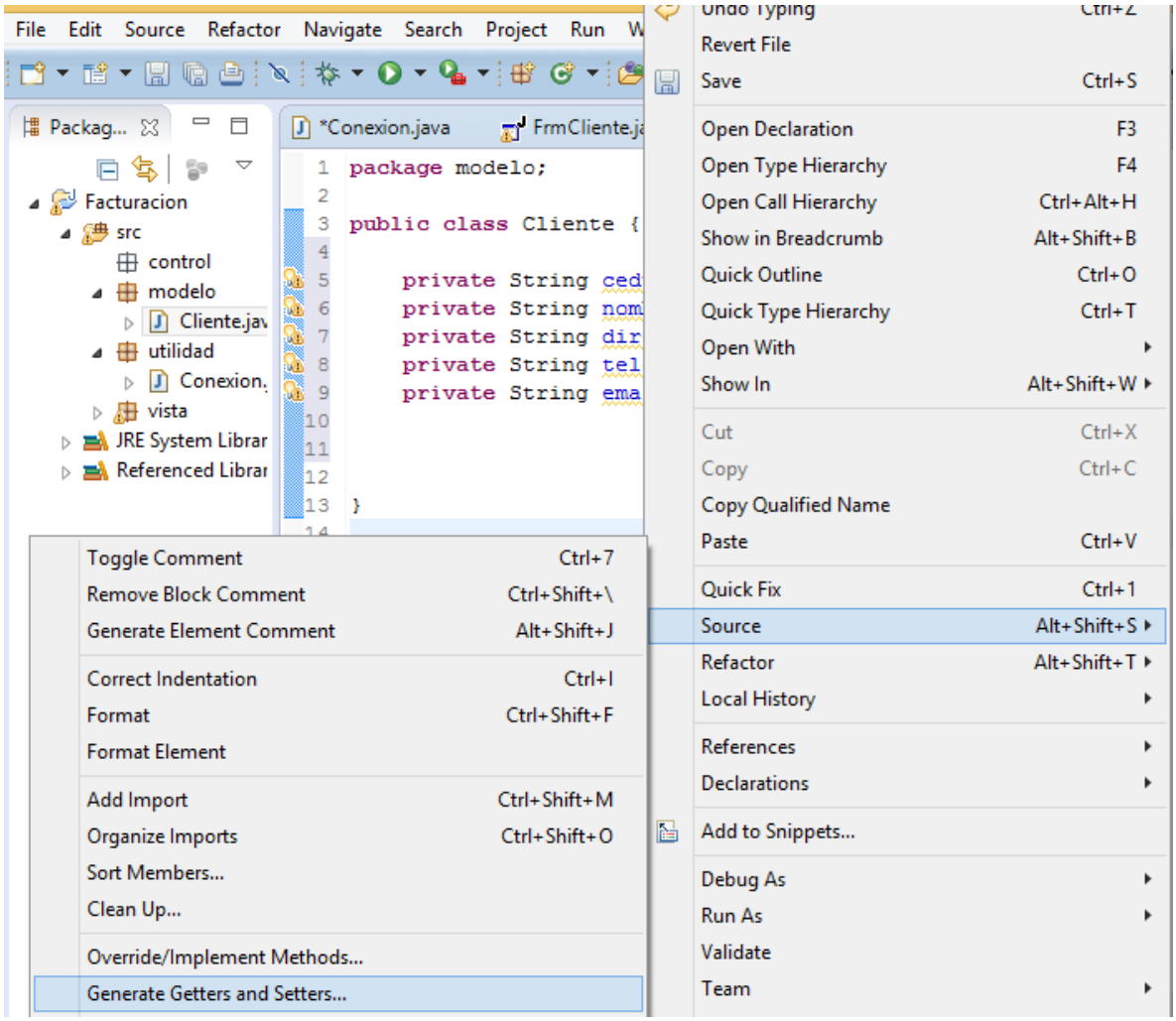
La variable opc es de vital importancia porque con ella se determinar la función a realizar, 1 para registros nuevos, 2 para consultar información, 3 para modificar y 4 para la eliminación, esta eliminación se realizará de manera lógica mas no física, lógica indica que el registro permanecerá en la tabla cliente, pero no será visible a el usuario, permitirá el control de históricos de la BD.

Creación de la Clase principal (Beans)

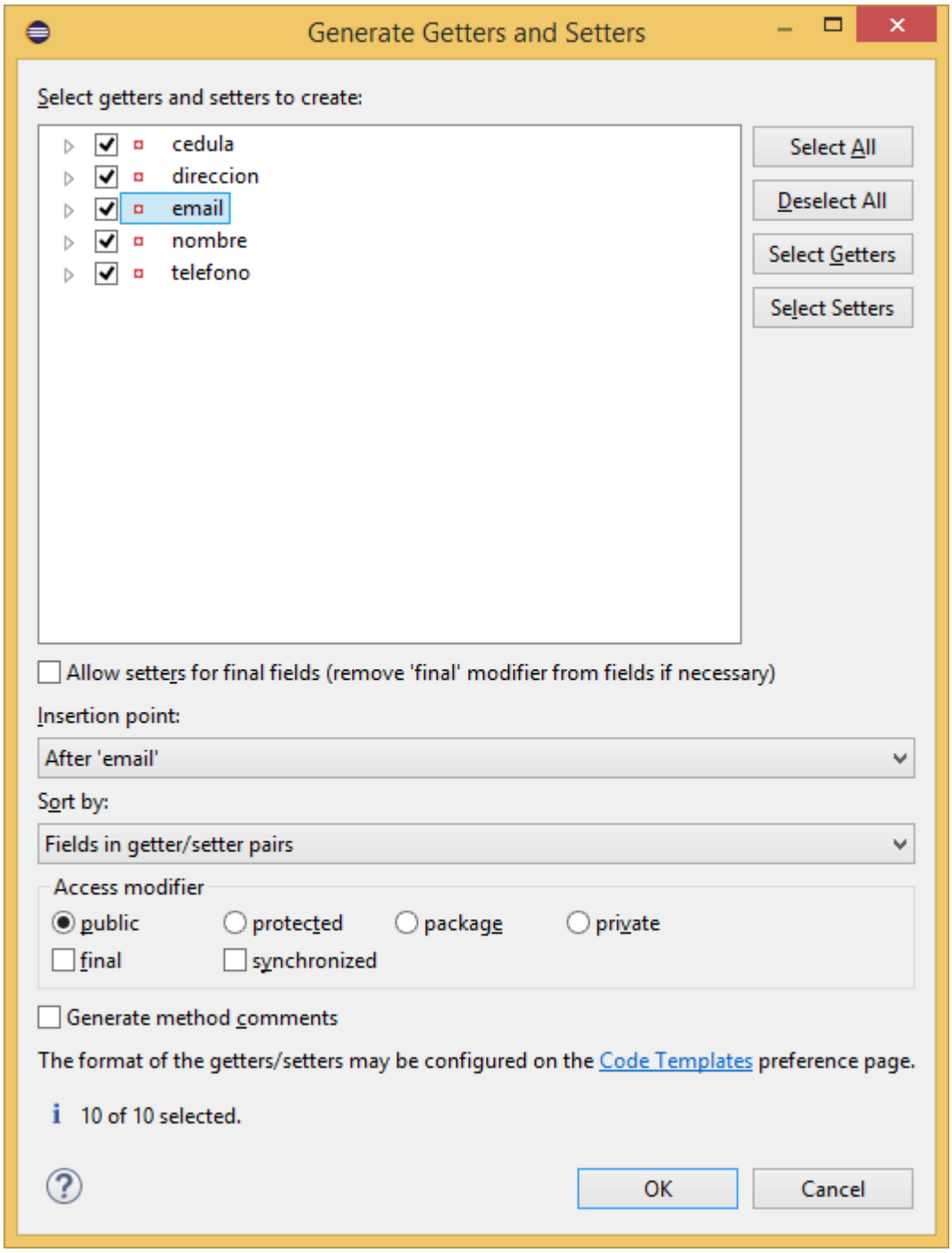
La clase principal del formulario inicial es muy simple pero de gran importancia, se definen las variables (campos) a trabajar y se generaran los getters y setters, estos serán los contenedores de la información en distintas etapas, los datos serán enviados y / o tomados de ellos, evitando así acceder a una clase de formulario o a otro proceso que pueda vulnerar la seguridad del aplicativo, para este caso se crea una clase en el paquete modelo, con el nombre Cliente, quedando de la siguiente manera

```
package modelo;

public class Cliente {

    private String cedula;
    private String nombre;
    private String direccion;
    private String telefono;
    private String email;



}
```

Posterior a este proceso se generan los getters y setters, para este proceso se elije el botón emergente sobre cualquiera de los campos creados



Se seleccionan los campos con los que se trabajara, no es obligatorio seleccionarlos todos, solo los que tengan a bien ser utilizados o actualizados permanentemente.



Y luego el botón OK

```
package modelo;

public class Cliente {


	private String cedula;

	private String nombre;

	private String direccion;

	private String telefono;

	private String email;


	public String getCedula() {

		return cedula;

	}

	public void setCedula(String cedula) {

		this.cedula = cedula;

	}

	public String getNombre() {

		return nombre;

	}

	public void setNombre(String nombre) {

		this.nombre = nombre;

	}

	public String getDireccion() {

		return direccion;

	}

	public void setDireccion(String direccion) {

		this.direccion = direccion;
```

```
        }

        public String getTelefono() {

                return telefono;

        }

        public void setTelefono(String telefono) {

                this.telefono = telefono;

        }

        public String getEmail() {

                return email;

        }

        public void setEmail(String email) {

                this.email = email;

        }

}
```

Quedando de este modo, los sets (setters) actualizaran la información, los get (getters) la retornaran a quien la solicite.

Creación del patrón de diseño DAO (Data Access Object), los patrones de diseño, un patrón de diseño sugiere una solución a un problema, con un formato generalmente aceptado como un estándar, en el se ubicarán la lógica del negocio, los métodos del proceso.

Procedimientos Almacenados

Los procedimientos son herramientas de las BD que permiten realizar tareas de forma más flexible, sin exponer tanto el código y con la seguridad que pueda brindar la BD, permite las instrucciones más comunes del SQL como son las inserciones, consultas, modificaciones y eliminaciones, a continuación los 5 procedimientos a utilizar en este primer formulario.

Verificación, se utiliza para comprobar que el registro se encuentre o no dentro de la tabla

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `verificarCliente` $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `verificarCliente`(in _cedulaCliente char(10))
BEGIN
  select cedulaCliente from cliente where cedulaCliente = _cedulaCliente and estado = 'activo';
END $$

DELIMITER ;
```

Inserción

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `insertarCliente` $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `insertarCliente`(in _cedulaCliente char (10), in _nombre char (40),
                                    in _direccion char (40), in _telefono char (10), in _email char(40), in _estado char(10))
BEGIN
  insert into cliente (cedulaCliente, nombre, direccion, telefono, email, estado) values
                      (_cedulaCliente, _nombre, _direccion, _telefono, _email, _estado);
END $$

DELIMITER ;
```

Consulta

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `consultarCliente` $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `consultarCliente`(in _cedulaCliente char (10))
BEGIN
  select cedulaCliente, nombre, direccion, telefono, email from cliente where cedulaCliente = _cedulaCliente and estado ='activo';
END $$

DELIMITER ;
```

Modificación

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `modificarCliente` $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `modificarCliente`(in _cedulaCliente char (10), in _nombre char (40),
                                                               in _direccion char (40), in _telefono char (10), in _email char(40))
BEGIN
  update cliente set nombre = _nombre, direccion =_direccion, telefono = _telefono, email = _email
      where cedulaCliente = _cedulaCliente;
END $$

DELIMITER ;
```

Eliminación

La eliminación no se aplica con la sentencia delete como es tradicional, sino con update, se esta aplicando una eliminación lógica, que permite conservar los datos para consultas históricas.

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `eliminarCliente` $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `eliminarCliente`(in _cedulaCliente char(10))
BEGIN
  update cliente set estado = 'inactivo' where cedulaCliente = _cedulaCliente;
END $$

DELIMITER ;
```

Existe un archivo adicional en este modelo llamado ClienteDAO (Data Access Object), el modelo (patrón de diseño) DAO brinda la posibilidad de almacenar toda la lógica del negocio, para este caso particular los métodos de insertar, consultar, modificar, eliminar y verificar, este archivo se ubica en el paquete control.

Se ubicaran en el mismo orden de los procedimientos almacenados para mayor claridad.

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

import javax.swing.JOptionPane;

import modelo.Cliente;
import utilidad.Conexion;

public class ClienteDAO {

    static int rta = 0;
    static Connection cnn = Conexion.getConnection();
    static Cliente cliente = new Cliente ();
```

en la parte inicial de la clase se encuentra una variable rta iniciada en cero (0), esta variable será una sw que retornara si existe o no un valor consultado, la clase Connection permite la asignación a una variable cnn de del método getConnection de la clase Conexión, por ultimo se instancia la clase Cliente que contiene los getters y setters.

Verificar

Este método verificar contiene la sentencia PreparedStatement, esta instrucción permite tener una sentencia SQL de fácil lectura, al final de la sentencia podríamos colocar una instrucción tradicional o como en este caso el llamado a un procedimiento almacenado, el signo de interrogación (?) indica el parámetro con el que va trabajar, en la siguiente línea se "traduce" ese signo a que valor corresponde.

En la tercera línea se encuentra ResultSet, esta instrucción permite el almacenamiento de una consulta SQL en memoria.

En la misma línea se encuentra la instrucción executeQuery, acá se este ejecutando la instrucción SQL select con los parámetros que se hayan especificado, otra instrucción muy común es executeUpdate que aplicara para las demás sentencias.

La sentencia rta = rs.next() ? 1 : 0; nos permitirá determinar si el registro consultado existe (1) o no en la tabla (0), y al final retornara este resultado al formulario donde se precederá con las siguientes tareas.

La referencia registroCliente.setString determina el tipo de campo que esta llegando (carácter), si fuera un tipo entero se representaría con setInt, contiene 2 parámetros, el primero es la posición o el orden de los parámetros y el segundo el valor o referencia.

```
public static int verificar (String cedula){

    try {

        PreparedStatement registroCliente = cnn.prepareStatement("call verificarCliente (?)");
        registroCliente.setString(1,  cedula);
        ResultSet rs = registroCliente.executeQuery();
        rta = rs.next() ? 1 : 0;
    }
    catch (SQLException sqle) {

        sqle.printStackTrace ();
    }

    return rta;

}
```

Insertar

La inserción no retorna ningún valor, pero recibe la referencia de la clase cliente donde se podrán actualizar y enviar los valores solicitados.

En la instrucción setString se encuentran los dos parámetros mencionados anteriormente, el orden de los parámetros y en cliente.getCedulaCliente(), esta opción getCedulaCliente() esta tomando el valor de la clase Cliente.

Al final del try aparece el executeUpdate que procede a ejecutar la sentencia SQL.

```
public static void insertar (Cliente cliente){

    try {

        PreparedStatement registroCliente = cnn.prepareStatement("call insertarCliente (?, ?, ?, ?, ?, ?)");
        registroCliente.setString(1, cliente.getCedulaCliente());
        registroCliente.setString(2, cliente.getNombre());
        registroCliente.setString(3, cliente.getDireccion());
        registroCliente.setString(4, cliente.getTelefono());
        registroCliente.setString(5, cliente.getEmail());
        registroCliente.setString(6, "activo");
        registroCliente.executeUpdate ();
        JOptionPane.showMessageDialog(null, "Registro Almacenado");
    }
    catch (SQLException sqle) {

        sqle.printStackTrace ();
    }
}
```

Consultar

Se aplica la instrucción setCedulaCliente, esta instrucción actualiza la información en la clase Cliente.

rs.next (), si el sistema ingresa en esta instrucción indica que existen datos para ser consultados.

```
public static Cliente consultar (String cedula) {

    try {

        PreparedStatement registroCliente = cnn.prepareStatement("call consultarCliente (?)");
        registroCliente.setString(1, cedula);
        ResultSet rs = registroCliente.executeQuery();
        if (rs.next()) {

            cliente.setCedulaCliente(rs.getString("cedulaCliente"));
            cliente.setNombre(rs.getString("nombre"));
            cliente.setDireccion(rs.getString("direccion"));
            cliente.setTelefono(rs.getString("telefono"));
            cliente.setEmail(rs.getString("email"));
        }
    }
    catch (SQLException sqle) {

        sqle.printStackTrace ();
    }

    return cliente;

}
```

Modificar

```
public static void modificar (Cliente cliente) {

    try {

        PreparedStatement registroCliente = cnn.prepareStatement("call modificarCliente (?, ?, ?, ?, ?)");
        registroCliente.setString(1, cliente.getCedulaCliente());
        registroCliente.setString(2, cliente.getNombre());
        registroCliente.setString(3, cliente.getDireccion());
        registroCliente.setString(4, cliente.getTelefono());
        registroCliente.setString(5, cliente.getEmail());
        registroCliente.executeUpdate ();
        JOptionPane.showMessageDialog(null, "Registro Actualizado");
    }
    catch (SQLException sqle) {

        sqle.printStackTrace ();
    }
}
```

Eliminar

```
public static void eliminar (String cedula) {

    try {

        PreparedStatement registroCliente = cnn.prepareStatement("call eliminarCliente (?)");
        registroCliente.setString(1, cedula);
        registroCliente.executeUpdate ();
        JOptionPane.showMessageDialog(null, "Registro Eliminado");
    }
    catch (SQLException sqle) {

        sqle.printStackTrace ();
    }
}
```

Archivo Facade o Fachada, el patrón de diseño, permite la coordinación, el control, la adecuada distribución de los procesos, además de ser un puente de solicitudes y no la solicitud directa a un formulario o a un DAO, este archivo contendrá todos los llamados que sean necesarios para el aplicativo en general, no para cada formulario.

En el paquete utilidad se ubicará este archivo Facade

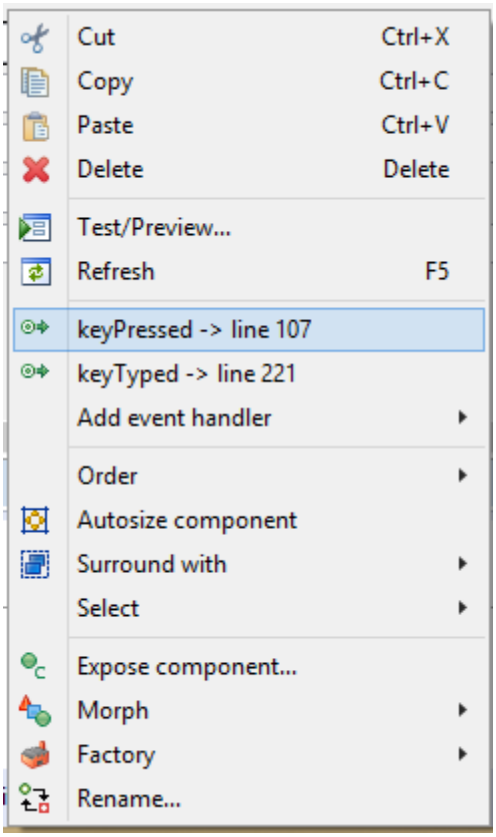
```
import control.ClienteDAO;
import modelo.Cliente;

public class Facade {

    static Cliente cliente;
    static ClienteDAO clienteDAO;

    public Facade (){

        cliente = new Cliente ();
        clienteDAO = new ClienteDAO ();
    }
```

El facade se ubicarán procesos muy cortos asociados a llamados de los métodos del DAO

```
public static int verificarCliente (String cedula) {

    return clienteDAO.verificar(cedula);
}

public static void insertarCliente (Cliente cliente) {

    clienteDAO.insertar (cliente);
}

public static Cliente consultarCliente (String cedula) {

    return clienteDAO.consultar(cedula);
}

public static void modificarCliente (Cliente cliente) {

    clienteDAO.modificar(cliente);
}

public static void eliminarCliente (String cedula) {

    clienteDAO.eliminar(cedula);
}
```

Acá se encuentran los 5 procesos según los 5 métodos creados en el DAO, tienen la misma estructura y se invoca en todos los casos el DAO, todas las solicitudes provienen del formulario de trabajo.

Para finalizar este procedo de CRUD queda pendiente la invocación de las tareas desde el formulario de trabajo, para este caso FrmCliente, presionando el botón emergente sobre la cedula (Clave primaria).



Se desplegará el código de validación de este campo (todas las cajas de texto deben de cumplir con un estándar de validación mínima.

```
public void keyPressed(KeyEvent arg0) {

    if (arg0.getKeyCode() == KeyEvent.VK_ENTER) {

        if (txtCedula.getText().equals("")) {

            JOptionPane.showMessageDialog(null, "Debe Digitar la Cedula");
            txtCedula.requestFocus();
        }
        else if (txtCedula.getText().length() < 6 || txtCedula.getText().length() > 10) {

            JOptionPane.showMessageDialog(null , "La Cedula es Incorrecta, el rango comprende entre 6 y 10 cifras");
            txtCedula.requestFocus();
        }
        else {
```

En este código se este validando que el campo no este vacío y que cumpla una longitud mínima de 6 cifras y una máxima de 10, en el else que queda abierto ira la codificación operacional

```
sw = facade.verificarCliente (txtCedula.getText());
switch (opc) {

    case 1:

        if (sw == 1) {

            JOptionPane.showMessageDialog (null, "la Cedula " + txtCedula.getText() + " ya Existe");
            activar ();
            limpiar ();
            deshabilitar ();
        }
        else {

            txtNombre.setEditable(true);
            txtNombre.requestFocus();
        }
        break;
```

La variable sw recibe un valor (0 o 1) que determinar si existe o no un registro, este proceso se realiza mediante el archivo facade en el método verificarCliente, este proceso a su vez invoca el método verificar de la clase ClienteDAO que realiza el recorrido dentro de la BD y arroja una respuesta.

En el switch se evalúa que operación se aplicara (1 nuevo, 2 consultar, 3 modificar y 4 eliminar), dentro de el se evalúa sw, para este ejemplo si el resultado es uno (1) indica que la cedula ya existe y no se podrá ingresar nuevamente si es cero (0) indica que el registro no existe y se podrá ingresar un nuevo registro, es por esto que se lleva el foco a la caja de texto de nombre.

```
case 2:

    if (sw == 1) {

        cliente = facade.consultarCliente(txtCedula.getText());
        mostrar (cliente);
        deshabilitar ();
    }
    else {

        JOptionPane.showMessageDialog(null, "NO Existe la Cedula Consultada");
        deshabilitar ();
        limpiar ();
    }
    break;
```

```
case 3:
    if (sw == 1) {
        cliente = facade.consultarCliente(txtCedula.getText());
        mostrar (cliente);
        deshabilitar ();
        int respuesta = JOptionPane.showConfirmDialog(null, "Seguro de Modificar?", null, JOptionPane.YES_NO_OPTION);
        if (respuesta == 0) {

            txtCedula.setEditable(false);
            txtNombre.setEditable(true);
            txtNombre.requestFocus();
        }
        else {

            deshabilitar ();
            activar ();
        }
    }
    else {

        JOptionPane.showMessageDialog(null, "No Existe la Cedula");
        activar ();
        limpiar ();
        deshabilitar ();
    }
    break;


    case 4:
        if (sw == 1) {
            cliente = facade.consultarCliente(txtCedula.getText());
            mostrar (cliente);
            deshabilitar ();
            int respuesta = JOptionPane.showConfirmDialog(null, "Seguro de Eliminar?", null, JOptionPane.YES_NO_OPTION);
            if (respuesta == 0) {

                facade.eliminarCliente(txtCedula.getText());
                limpiar ();
                deshabilitar ();
            }
            else {

                deshabilitar ();
                activar ();
            }
        }
        else {
            JOptionPane.showMessageDialog(null, "No Existe la Cedula");
            activar ();
            limpiar ();
            deshabilitar ();
        }
        break;
}
sw = 0;
```

Botón Guardar

El botón guardar ya tenia una codificación previa, esta codificación será ampliada con el fin de realizar el paso de los datos a modificar y a insertar.

```
cliente.setCedulaCliente(txtCedula.getText());
cliente.setNombre(txtNombre.getText());
cliente.setDireccion(txtDireccion.getText());
cliente.setTelefono(txtTelefono.getText());
cliente.setEmail(txtEmail.getText());

if (opc == 1)

    facade.insertarCliente (cliente);
else

    facade.modificarCliente(cliente);

deshabilitar ();
activar ();
```

En el FrmCliente, al final de la codificación falta la creación del método mostrar, este método recibe la información consultada y la lleva a las cajas de texto.

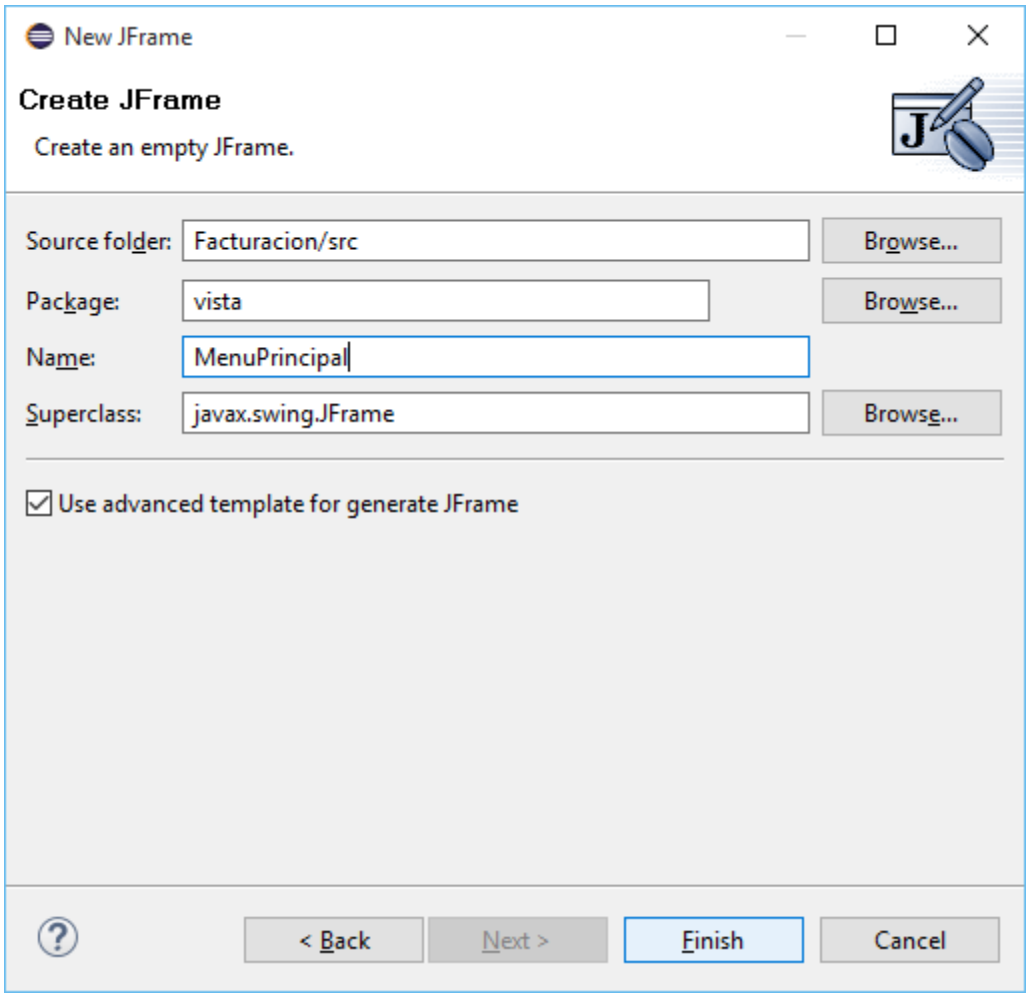
```
private void mostrar (Cliente cliente) {

    txtCedula.setText(cliente.getCedulaCliente());
    txtNombre.setText(cliente.getNombre());
    txtDireccion.setText(cliente.getDireccion());
    txtTelefono.setText(cliente.getTelefono());
    txtEmail.setText(cliente.getEmail());
}
```

Este es el proceso final, el aplicativo esta 100% terminado y se podría poner a prueba.

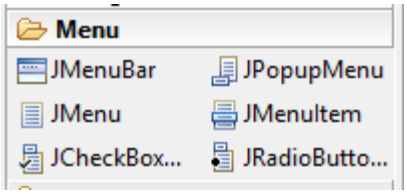
Creación de un Menú

Después de tener el primer CRUD completo de una tabla se presentan nuevas necesidades, dentro de ellas encontramos una de ¿qué hacemos cuando tengamos varios CRUDs y necesitemos llamarlo de forma dinámica?, la respuesta a esto se puede resolver de varias maneras, pero existe una que nos brinda Java, los menús!.

En el paquete vista se creará un nuevo formulario (JFrame) con el nombre de MenuPrincipal

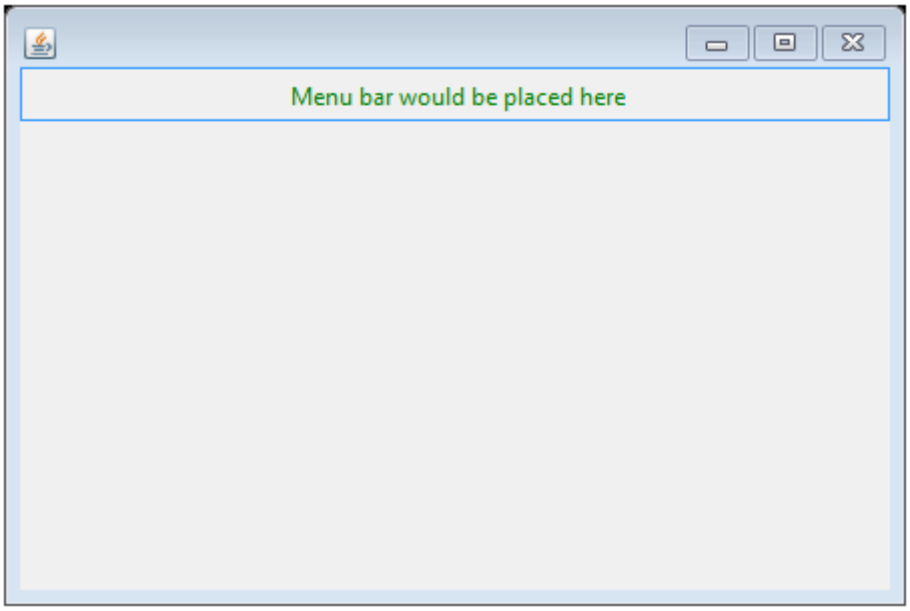


Se agrega un Layout como en el caso del primer formulario creado, para este caso se toma un Absolute Layout, en la paleta de controles se encuentra una pestaña llamada Menú

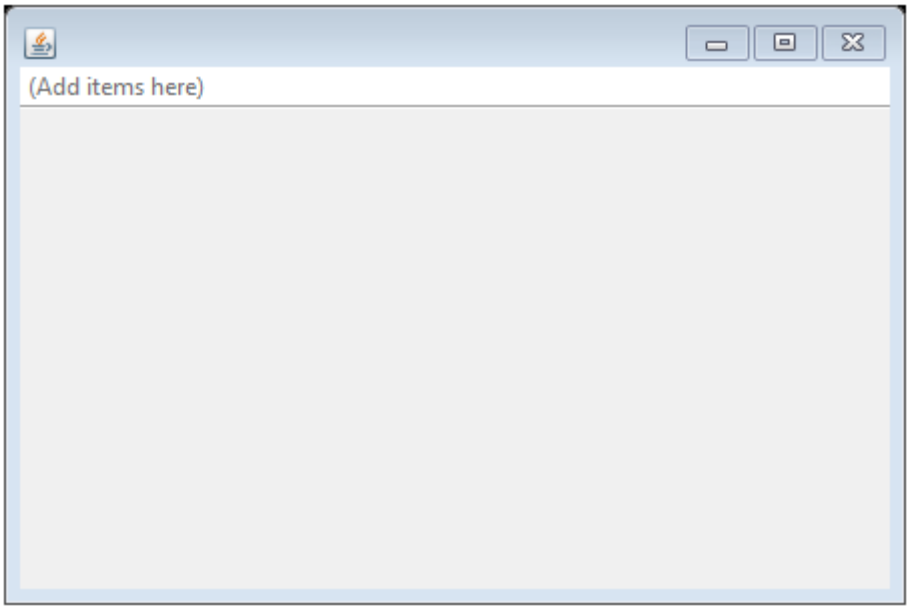


De estas opciones se requieren tres de ellas, JMenuBar (crea una barra de menú en la parte superior), JMenu (especifica cada opción principal del Menú horizontal), JMenuItem (especifica cada una de las opciones a ser invocadas).

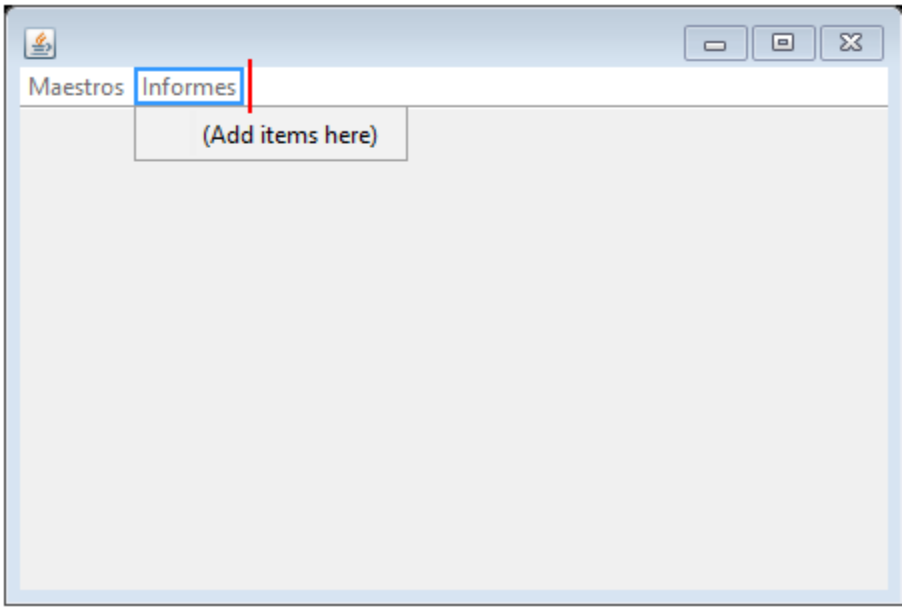
Creación de una JMenuBar



Seleccionando el control JMenuBar y ubicando el cursor levemente arriba el inicio de la Barra Menú aparecerá este contenedor quedando así:

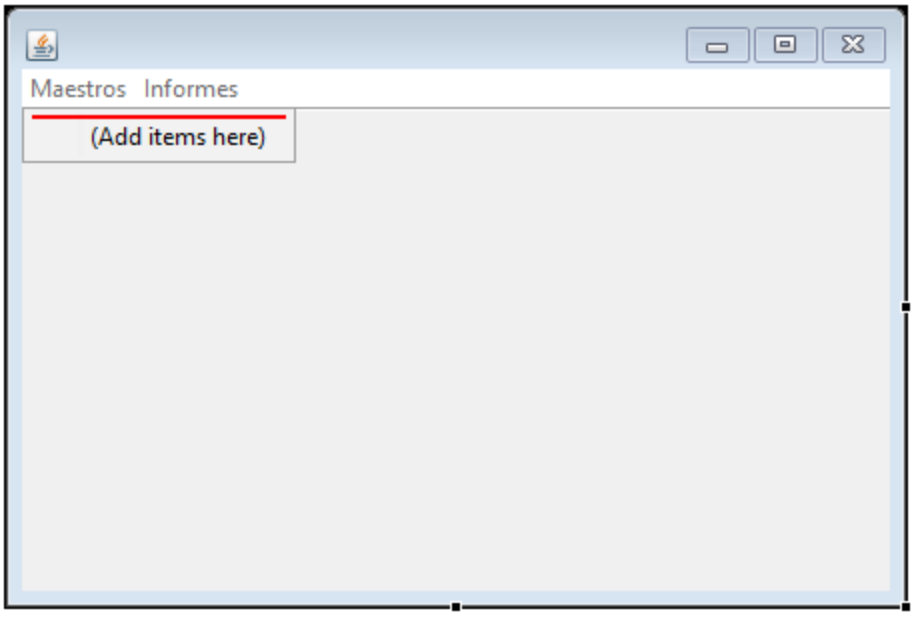


Sobre este contenedor se colocarán las opciones del menú principal, esta se realizará con la opción JMenu.

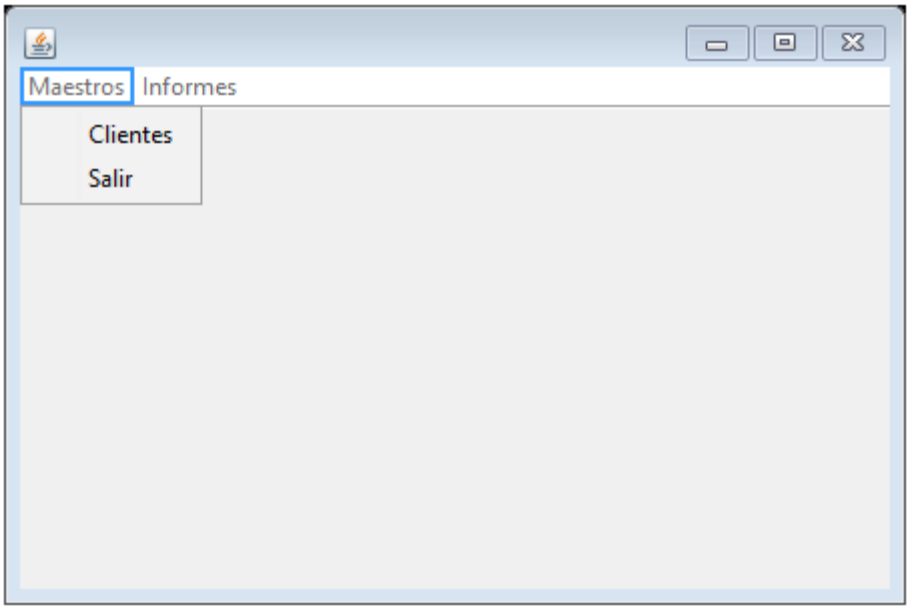


Se podrán agregar todos los que se consideren necesarios y el espacio permita

Por último, se procede a crear cada uno de los ítems que contendrá cada título Principal (JMenuItem)

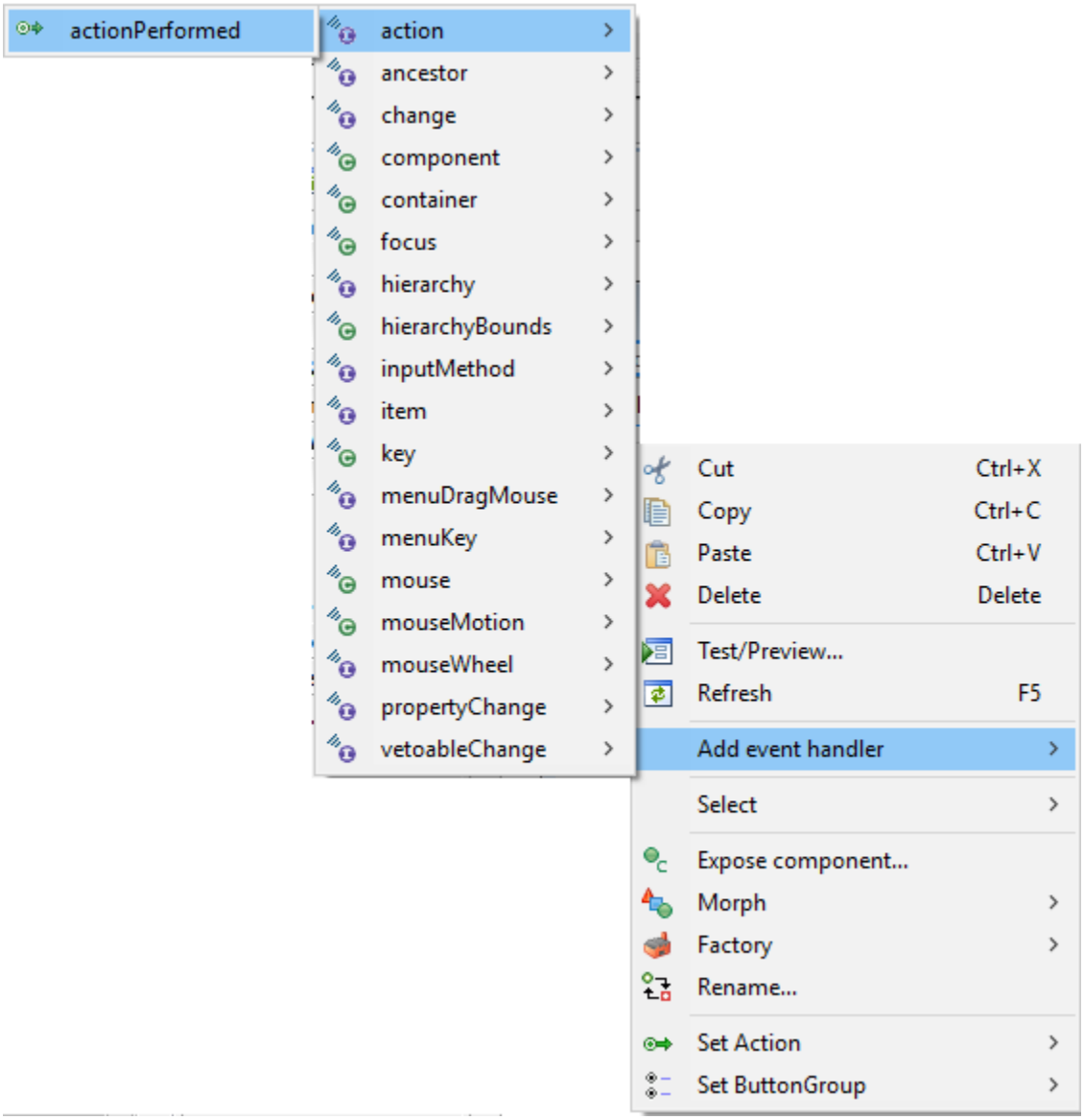


La línea roja es fundamental como guía de la ubicación del menú, cuando aparece vertical indica que es uno como en el JMenu, cuando parece horizontal indica que es desplegable, en este caso debajo de Maestros, para este caso se crearan 2, uno de Cliente y otro de Salir.



Programación del botón Salir

Con el botón emergente sobre la opción Salir, seleccione las opciones correspondientes.



Dentro de la opción que se genera se agrega la instrucción

```
JMenuItem mntmSalir = new JMenuItem("Salir");
mntmSalir.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

        System.exit(0);
    }
});
```

Esta instrucción permitirá el cerrado del proyecto

Para la invocación de cada opción del menú se aplicarán los mismos pasos con el botón emergente, para este caso sobre la opción de cliente.

```
JMenuItem mntmClientes = new JMenuItem("Clientes");
mntmClientes.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        FrmCliente frmCliente = new FrmCliente ();
        frmCliente.setVisible (true)
    }
});
```

Este se aplicará para todos los casos que conformen el menú y se tendría un proceso centralizado para el llamado y la administración de los componentes del proyecto.

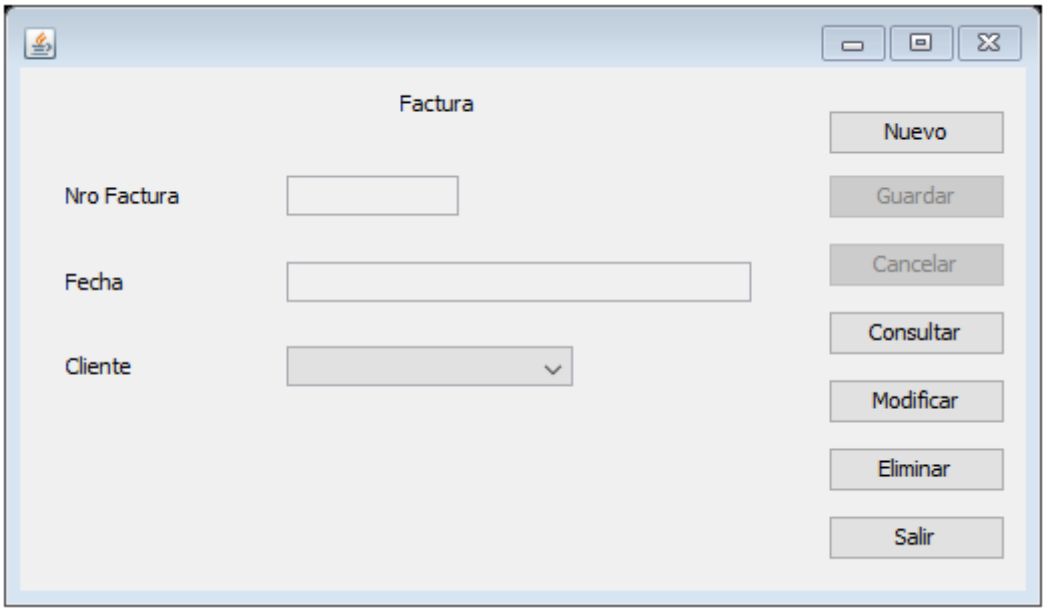
ComboBox

Los comboBox hacen parte de un gran número de controles disponibles en la plataforma de Java, nos permite colocar datos predeterminados o hacer un listado de opciones que provienen de una tabla

Como crear y llenar un ComboBox

Aprovechando la tabla de factura que tiene una referencia hacia la tabla cliente veamos el proceso

Diseño inicial



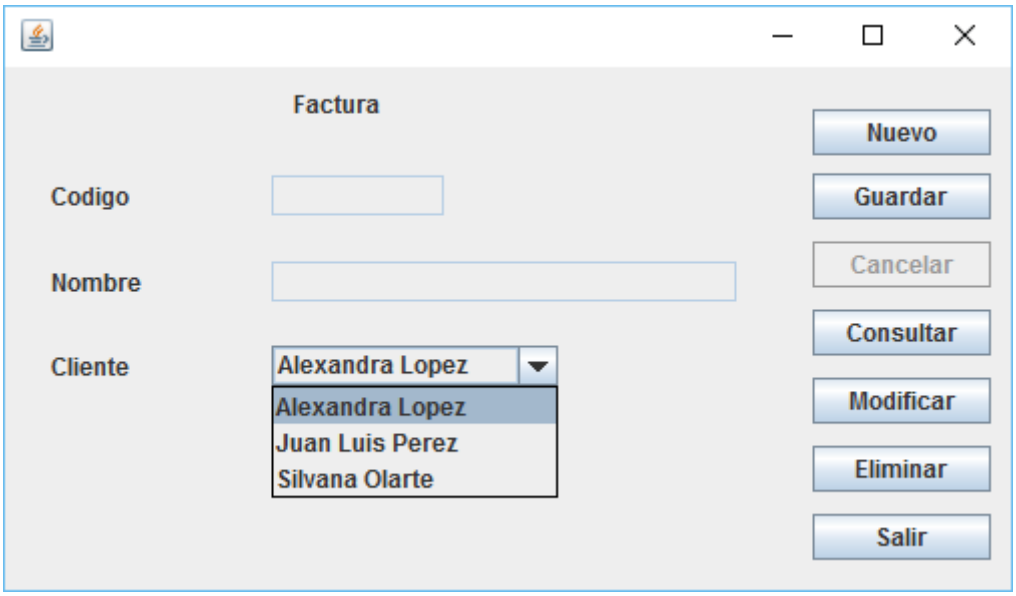
En este ejemplo se muestran los 3 primeros campos de la tabla, 7 botones, dos cajas de texto y un ComboBox, para este último se utilizará el nombre cboCliente y se aplicará como un control tipo campo, para esto seleccione el combo con el menú emergente, rename… y luego aplica cambios después de seleccionar la letra F.

Creación de un método de llenado del ComboBox

```
private static void llenarCboCliente () {

    Cliente cliente = new Cliente ();
    Connection cnn = Conexion.getConnection();
    try{

        PreparedStatement registroCliente = cnn.prepareStatement("Select nombre from cliente "
                + "where estado ='Activo' order by nombre");
        ResultSet rs = registroCliente.executeQuery();
        while (rs.next())

            cboCliente.addItem(rs.getString("nombre"));
    }
    catch (SQLException e) {

        System.out.print ("" + e);
    }
}
```

En este método se aplican todos los conceptos vistos previamente, para el llenado del combo aplica dentro del ciclo mientras donde se encuentra cboCliente.addItem y el campo con el que se desea llenar este combo.

Para la invocación del método se ubica al final del constructor de la clase del formulario y quedaría así:



Los ComboBox están compuestos por el valor a mostrar, en este caso los nombres de los clientes, cuando se requiere almacenar el identificador de cada cliente en la tabla referencial se aplicará otro método de la siguiente forma.

```
private static String recuperarCliente (int posicionRegistro) {

    Cliente cliente = new Cliente();
    Connection cnn = Conexion.getConnection();
    try{

        PreparedStatement registroCliente = cnn.prepareStatement("Select cedula from cliente "
                + "where estado ='Activo' order by nombre limit " + posicionRegistro + ",1");
        ResultSet rs = registroCliente.executeQuery();
        if (rs.next())

            cedula = rs.getString("cedula");
    }
    catch (SQLException e) {

        System.out.print ("" + e);
    }

    return cedula;
}
```

En el botón de guardar se aplicaría este valor.

```
int posicionRegistro = cboCliente.getSelectedIndex();
String cedula = recuperarCliente(posicionRegistro);

cliente.setCedula(txtCedula.getText());
cliente.setNombre(txtNombre.getText());
cliente.setCedula(cedula);

if (opc == 1)

    fachada.insertarCliente(cliente);
```

La variable posicionRegistro es un index de la ubicación del nombre del cliente seleccionado, según esto se busca en el método de cboCliente, a la variable cedula se le asignará el valor que identifica ese nombre y se podrá guardar

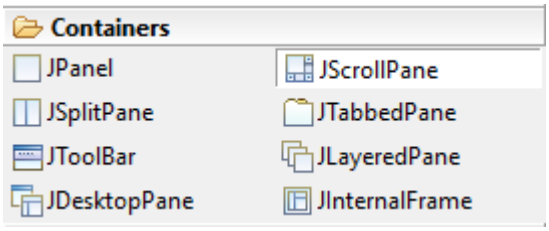
JTable

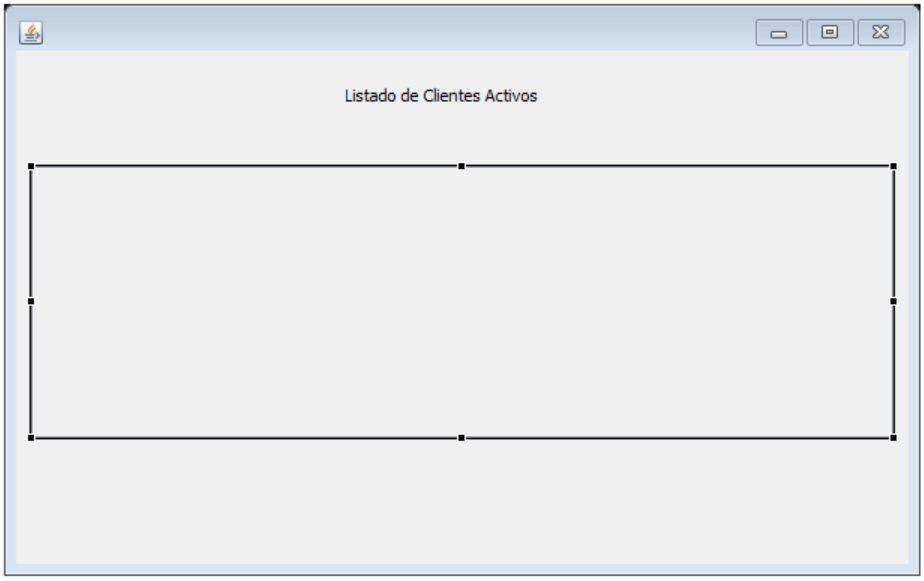
Un control adicional para este proyecto son los JTable, son controles que nos mostraran los datos que deseamos según una consulta o un listado general de datos, con esto podríamos realizar múltiples operaciones, dentro de ellas eliminar, facilitar la modificación entre otras.

Para este ejemplo se creará un nuevo formulario en el que se puedan visualizar los valores ingresados en una tabla, tenga presente que esto se puede hacer en un formulario como el de factura para adicionar los detalles de esta, o en el de cliente para listar todos los que se tengan en un momento dado.

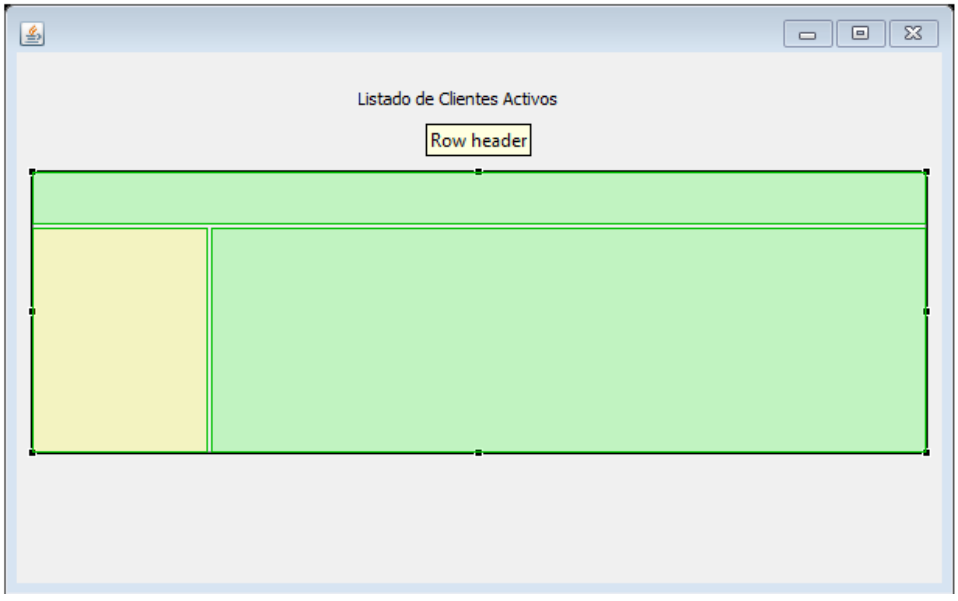
Cree un formulario llamado FrmClienteTotales, agregando el Layout utilizado anteriormente (Absolute Layout)

Agregue un control (componente) JScrollPane que está ubicado en la pestaña Containers, que ocupe el ancho del formulario y más o menos unos 5 centímetros del alto

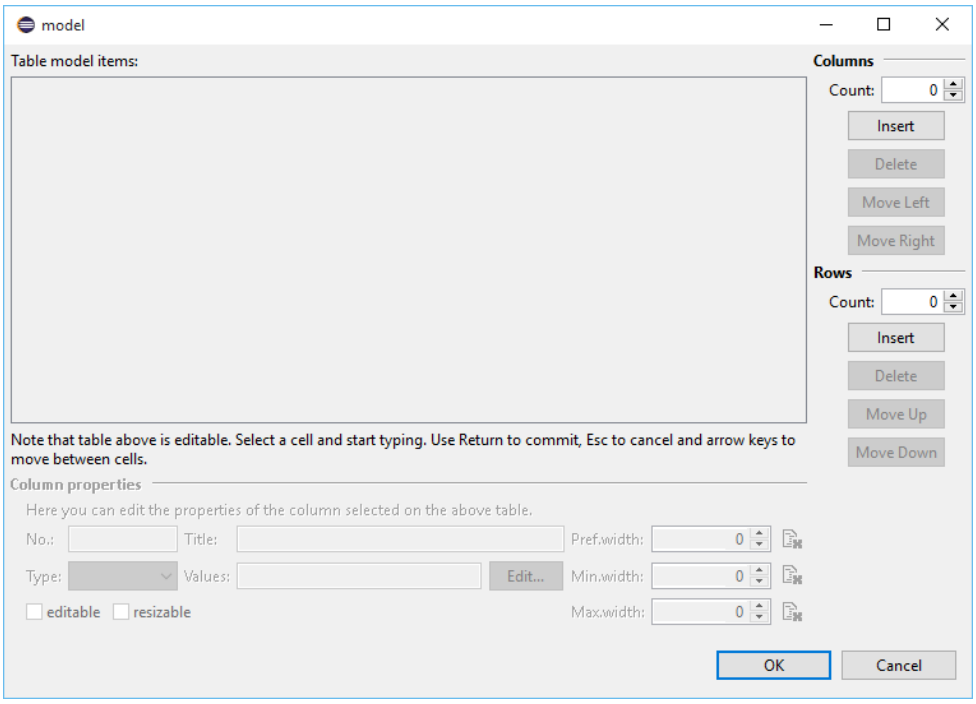




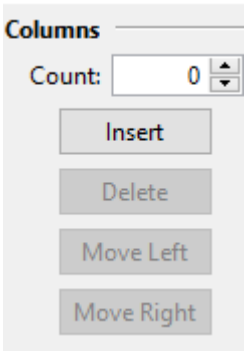
Luego seleccione el JTable, ubíquelo encima del control agregado recientemente



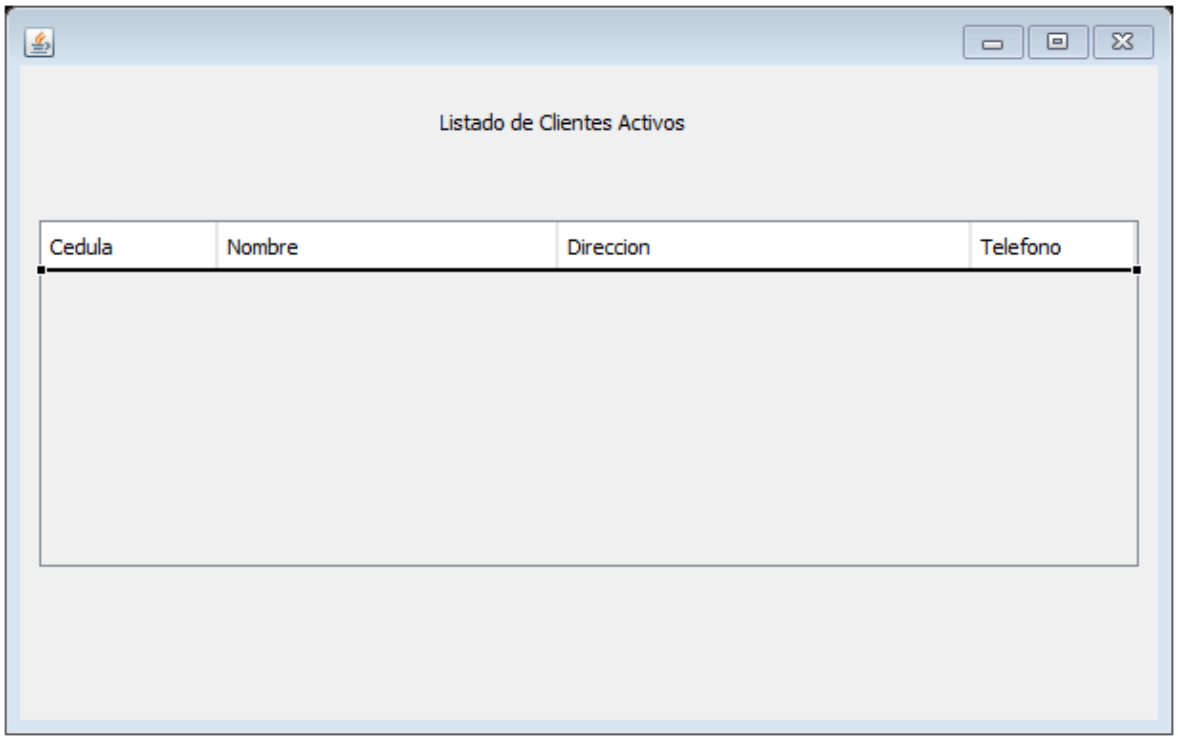
Seleccione el control JTable y en la ventana de propiedades elija la opción **model**



En la parte superior derecha selección insert para agregar los títulos de las columnas que llevara el JTabla



Para este caso se agregarán Cedula, Nombre, Dirección y Teléfono y luego seleccione el botón OK



Después de tener el diseño del formulario con el JTable, proceda a colocarle el nombre del control, para esto se ubica en las propiedades y en la primara opción variable, coloque tblCliente, este proceso trae un nuevo tema, se llama ArrayList, también conocido como un arreglo dinámico, maneja unas características similares a los arreglos tradicionales, pero con algunas ventajas en el manejo, en los "índices", la adición y eliminación de información.

El primer paso es ubicarse en el archivo DAO de cliente y proceder a crear un nuevo método.

```
public static ArrayList <Cliente> listarClientes (){

    ArrayList <Cliente> clientes = new ArrayList <Cliente> ();
    Cliente cliente = null;
    Connection cnn = Conexion.getConnection();

    try {

        PreparedStatement resultadoCliente = cnn.prepareStatement("select cedula, nombre, direccion, "
                + "telefono from Cliente where estado = 'Activo'");
        ResultSet rs = resultadoCliente.executeQuery();
        while(rs.next()){

            cliente = new Cliente();
            cliente.setCedula(rs.getString("cedula"));
            cliente.setNombre(rs.getString("nombre"));
            cliente.setDireccion(rs.getString("direccion"));
            cliente.setTelefono(rs.getString("telefono"));
            clientes.add (cliente);
            cliente = null;
        }
    } catch (SQLException sqle) {

        System.out.print("\n" + sqle);
    }

    return clientes;

}
```

Este método inicia con ArrayList <Cliente>, esto indica que retornara un arreglo dinámico, el nombre de este método es listarClientes y no tiene parámetros de entrada, en la segunda línea se encuentra nuevamente ArrayList <Cliente> clientes, este **clientes** es la referencia o el elemento que se encarga de recolectar todos los registros de la tabla que cumplan la condición de estar activos, las demás líneas ya son conocidas hasta la penúltima línea de try en la que aparece la referencia clientes nuevamente acompañada del método add, es acá donde llevamos un registro a el arrayList, por ultimo retornamos clientes, tengan presente que acá solo se está llenado un arreglo dinámico con la información, todavía no se muestra en pantalla.

Como segundo paso se ubicará en el archivo facade o fachada y se realizará el llamado el método anteriormente creado.

```
public static ArrayList listarClientes(){

    return clienteDAO.listar();
}
```

Se indica nuevamente que se retornara un ArrayList

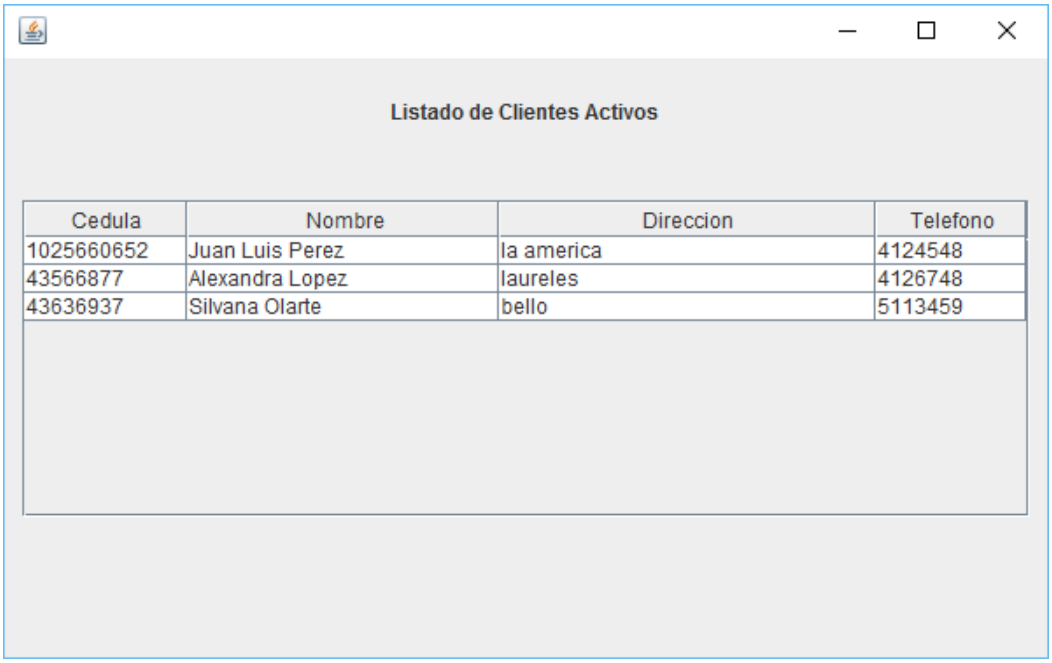
Y como tercer paso se creará un método para mostrar la información en pantalla, este se realiza en el formulario, en la parte inferior del código antes de la última llave.

```
private static void mostrarClientes (){

    ArrayList <Cliente> clientes = new ArrayList<Cliente> ();
    clientes = fachada.listarClientes();

    DefaultTableModel modelo = (DefaultTableModel) tblCliente.getModel();

    for(int i = 0; i < clientes.size(); i++){

        Vector <String> vector = new Vector<String>();
        vector.add(clientes.get(i).getCedula());
        vector.add(clientes.get(i).getNombre());
        vector.add(clientes.get(i).getDireccion());
        vector.add(clientes.get(i).getTelefono());
        modelo.addRow(vector);
    }
}
```

En este método encontramos nuevamente la referencia a un ArrayList con la aplicación del elemento de clientes, luego aparece el llamado de la fachada

En la tercera línea se encuentra la definición del JTabla donde se depositarán los datos, se crea otra referencia de la tblCliente llamado modelo.

En el ciclo para se realiza un recorrido de todos los datos a mostrar, para saber la totalidad de los datos se tiene el método size, adicional dentro del ciclo se trabaja con la clase Vector y se procede a "pintar" la información en el JTable, el resultado es el siguiente.



PISTAS DE APRENDIZAJE

Traer a la memoria:

CRUD	Acrónimo de Crear, Leer, Actualizar y Eliminar
BD	Base de Datos
Conector	Driver controlador entre el aplicativo y la BD
Paquete	Espacio que permite la clasificación de los procesos
Validación	Herramienta Fundamental para el control de la información
Facade	Patrón de Diseño
DAO	Data Access Object
Instancia	creación de una referencia de una clase
Clase Referencia	clase que contiene los Getters y los Setters
Gettes / Setters	Herramienta que permite tener mas control de los datos, así como un mayor alcance

2.3 TEMA 2 POOL DE CONEXIONES

Un pool de conexiones es **un conjunto limitado de conexiones** a una **BD**, un pool permite **centralizar** y **controlar** el acceso que se tenga por parte del sistema, con esto podemos concluir que la cantidad de conexiones abiertas es limitada teniendo en cuenta que estas consumen muchos recursos, memoria y tiempo de procesador, este proceso adema favorece **la escalabilidad de la aplicación.**

Dentro de las ventajas que se pueden observar en un Pool de Conexiones además de las ya descritas está el poder acceder o cambiar algunos parámetros sin necesidad de volver a compilar el aplicativo, cuando creamos una conexión tradicional como la vista en la primera unidad esta tiene los valores “quemados”, esto quiere decir que el conector, usuario, contraseña y la base de datos están predeterminados, si sucede algún cambio después de compilar y empaquetar el aplicativo habrá que volver a realizar el proceso, para evitar estos datos “quedamos” se procede a crear un archivo jdbc.properties en la raíz del src con la siguiente estructura.

jdbc.properties

```
1 usuario=root
2 clave=admin
3 controlador=com.mysql.jdbc.Driver
4 ruta=jdbc:mysql://localhost/facturacion
```

Posterior a este archivo se creará una clase llamada Conexion.java con la siguiente estructura.

```
package utilidad;

import java.sql.Connection;
import java.sql.DriverManager;
import java.util.ResourceBundle;
public class Conexion {

    private static Connection con = null;
    public static Connection getConnection () {

        try {

            if (con == null) {

                Runtime.getRuntime().addShutdownHook(new ShutdownHook());
                ResourceBundle res = ResourceBundle.getBundle("jdbc");
                String controlador = res.getString("controlador");
                String ruta = res.getString("ruta");
                String usuario = res.getString("usuario");
                String clave = res.getString("clave");
                Class.forName(controlador);
                con = DriverManager.getConnection(ruta, usuario, clave);
            }
            return con;
        }
        catch (Exception ex) {
            ex.printStackTrace();
            throw new RuntimeException("Error al crear la conexion!", ex);
        }
    }
```

Obsérvese que la clase recibe unos parámetros que están ubicados en el archivo properties luego de abrir el archivo jdbc, se leerán los datos, en este caso ruta contiene el jdbc y la Base de Datos, el usuario, la contraseña y el controlador, esto hará que se pueda manipular de una forma más simple todo el sistema.

Posterior a la última clase se agregará el siguiente fragmento de código complementario

```
    static class ShutdownHook extends Thread {

        public void run() {

            try {

                Connection con = Conexion.getConnection ();
                con.close();
            }
            catch( Exception ex ) {

                ex.printStackTrace();
                throw new RuntimeException(ex);
            }
        }
    }
```

PISTAS DE APRENDIZAJE

Traer a la memoria:

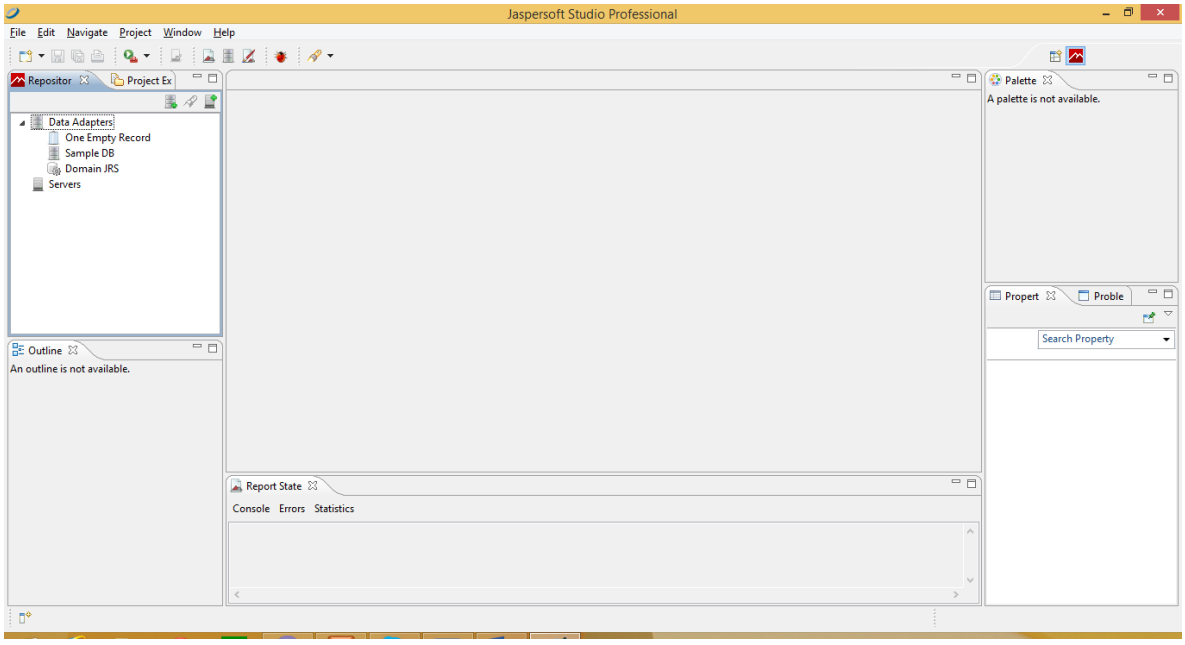
Conexión	todo el sistema de BD en una aplicación requiere el vínculo o la comunicación, esta tarea se realiza con un "puente" llamado conector.
Propertie	en un tipo de extensión aplicada a un archivo que puede contener valores no "quemados" y que se puedan cambiar sin necesidad de volver a compilar el aplicativo
Escalabilidad	diseño de un aplicativo a BD que permite que en un futuro pueda seguir creciendo sin necesidad de volver a iniciar o hacer cambios grandes

2.3.1 TEMA 3 REPORTES

La creación de reportes en un aplicativo se convierte en una necesidad fundamental, es la posibilidad de expresar los resultados de una sentencia SQL en papel o en un documento electrónico, mediante un archivo pdf, xlsx, html, docx entre otras extensiones.

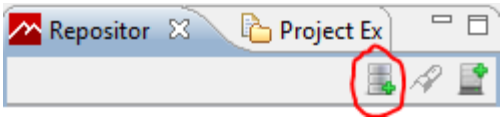
Para la generación de reportes se utilizará la herramienta jasper-studio, es un IDE muy simple de manejar y de gran alcance.

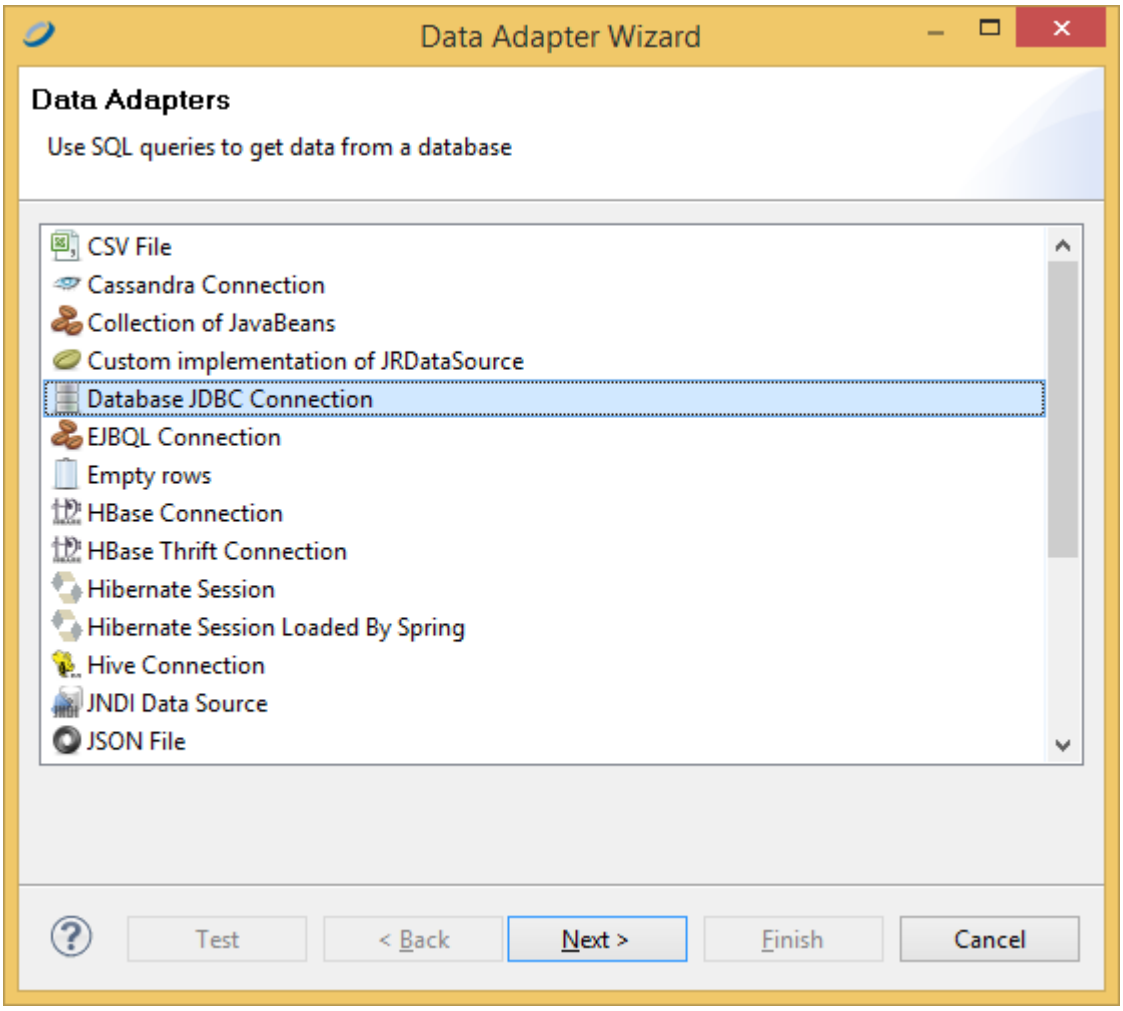
El Jasper Studio tiene un aspecto muy similar al eclipse, observémoslo



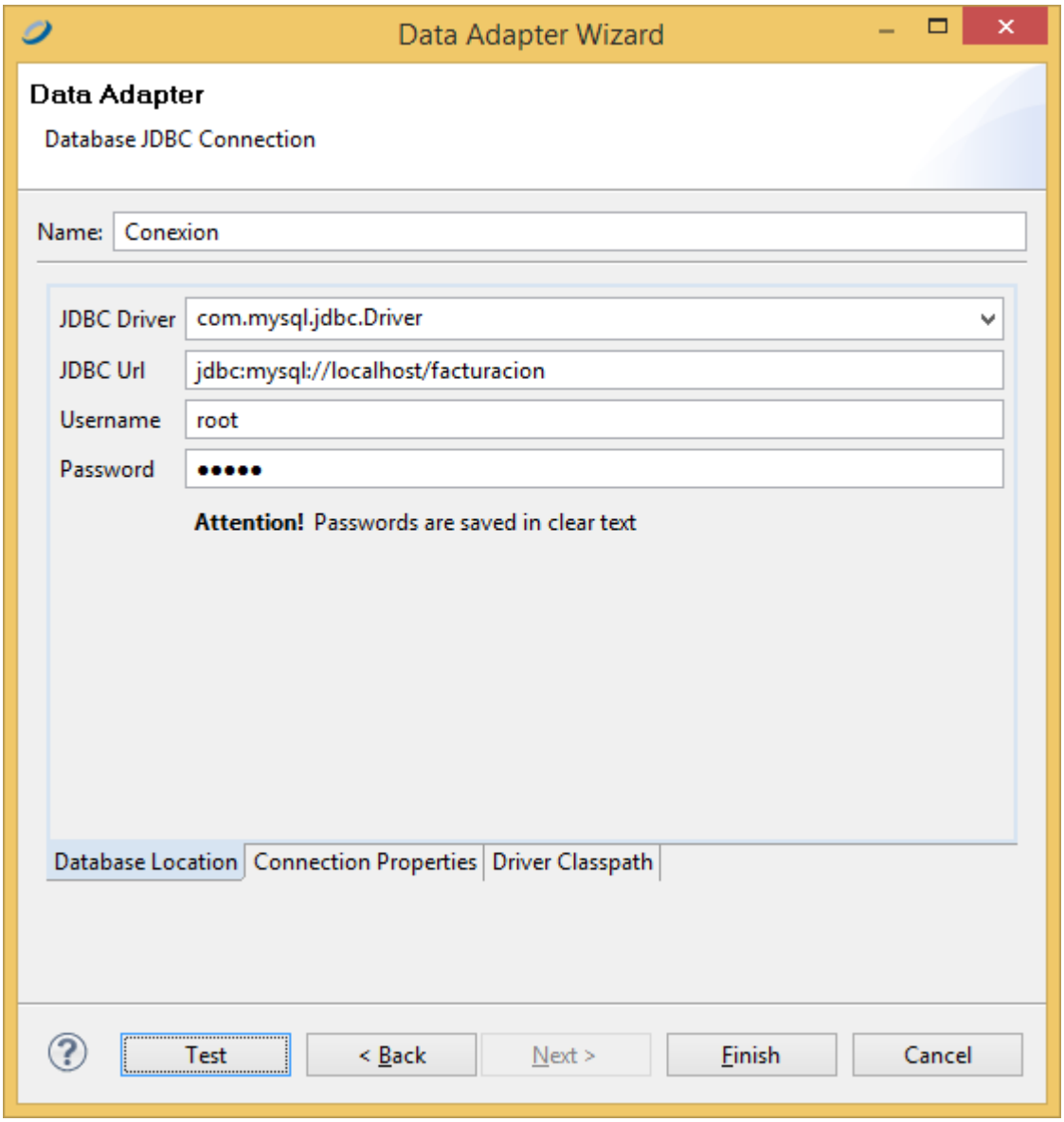
Configuración

Para la configuración del Jasper se requiere especificar el motor de bases de datos que se esta utilizando en el proyecto de Eclipse, para esto se selecciona el siguiente icono (Create Data Adapter)





Se selecciona Database JDBC Connection y se elige el botón Next



En esta ventana van las principales opciones de configuración

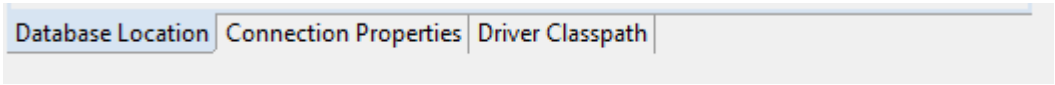
Name: especifica un nombre de conexión para cada proyecto o cada motor de base de datos

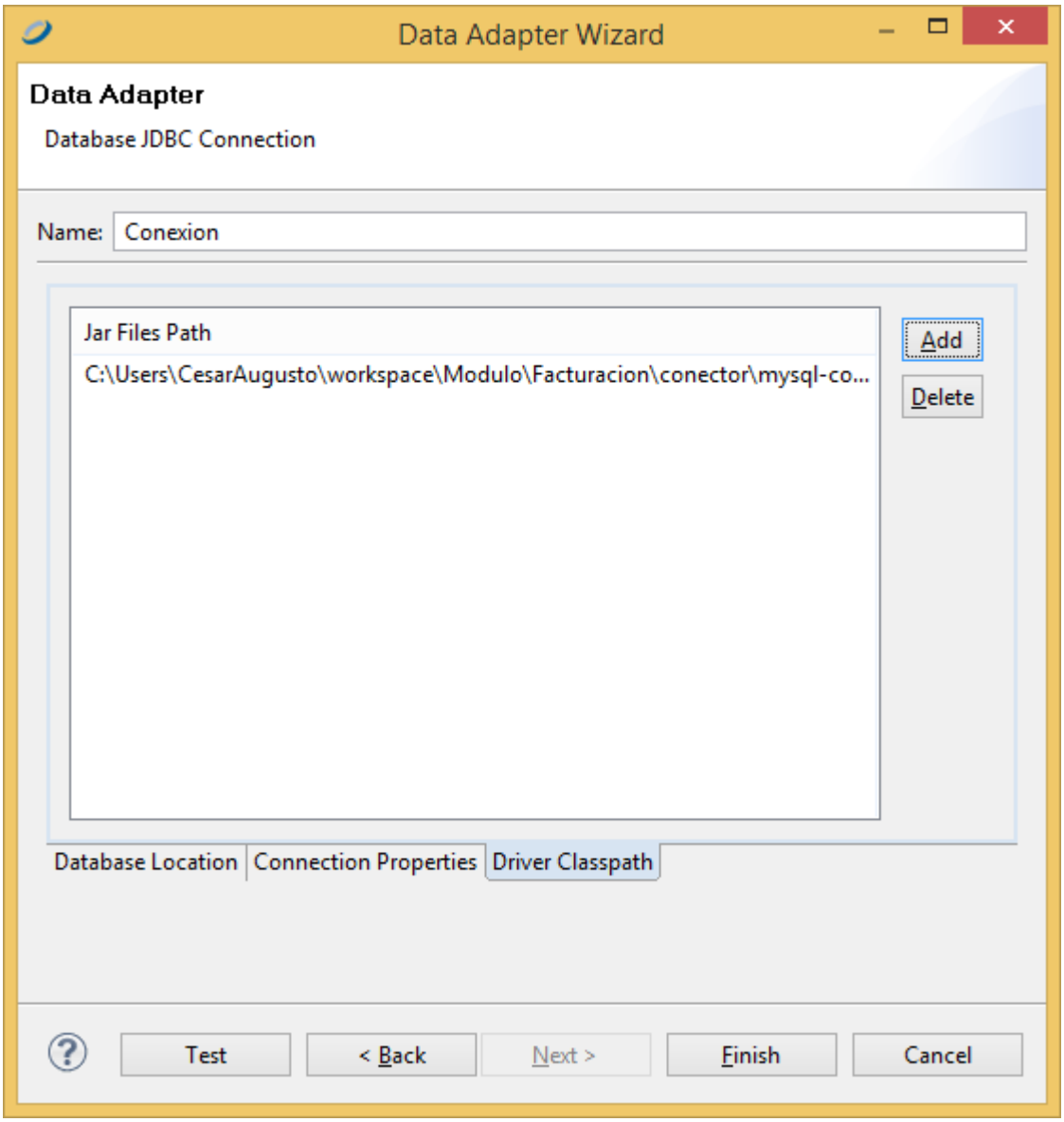
JDBC Url: solo se cambia después de la palabra localhost/ donde se ubica el nombre de la base de datos de trabajo

Username: nombre de usuario de MySQL

Password: contraseña de MySQL

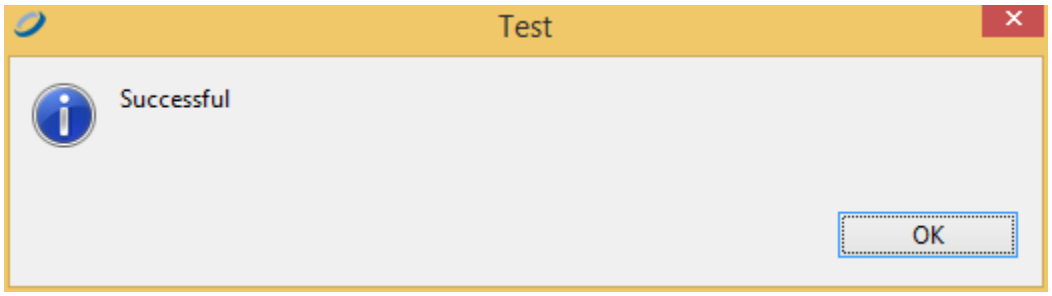
Luego se seleccionan en la parte inferior la pestana Driver Classpath





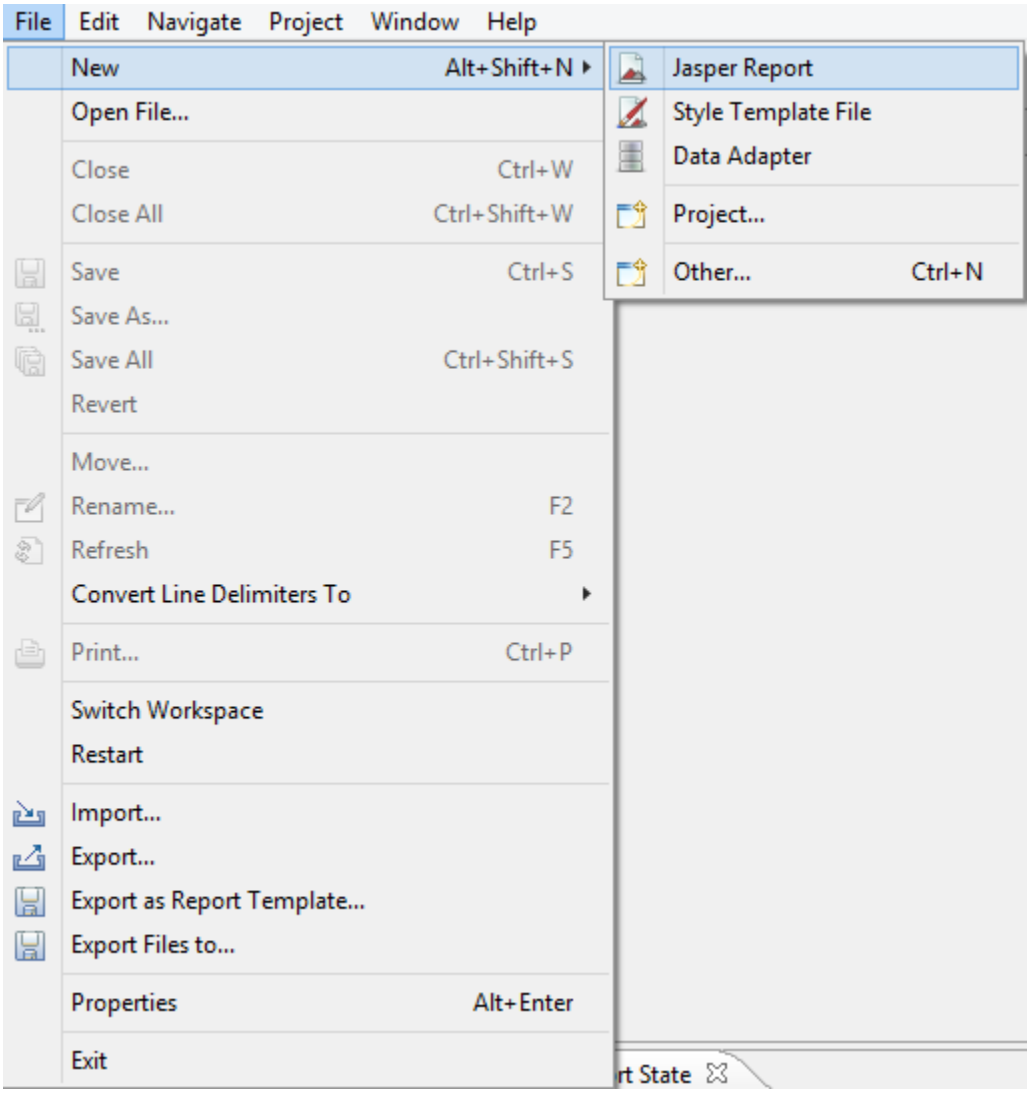
Mediante le botón Add se agrega el driver o conector que se esta utilizando para comunicar a el proyecto de java con MySQL

Cuando esto está establecido, se prueba mediante el botón Test

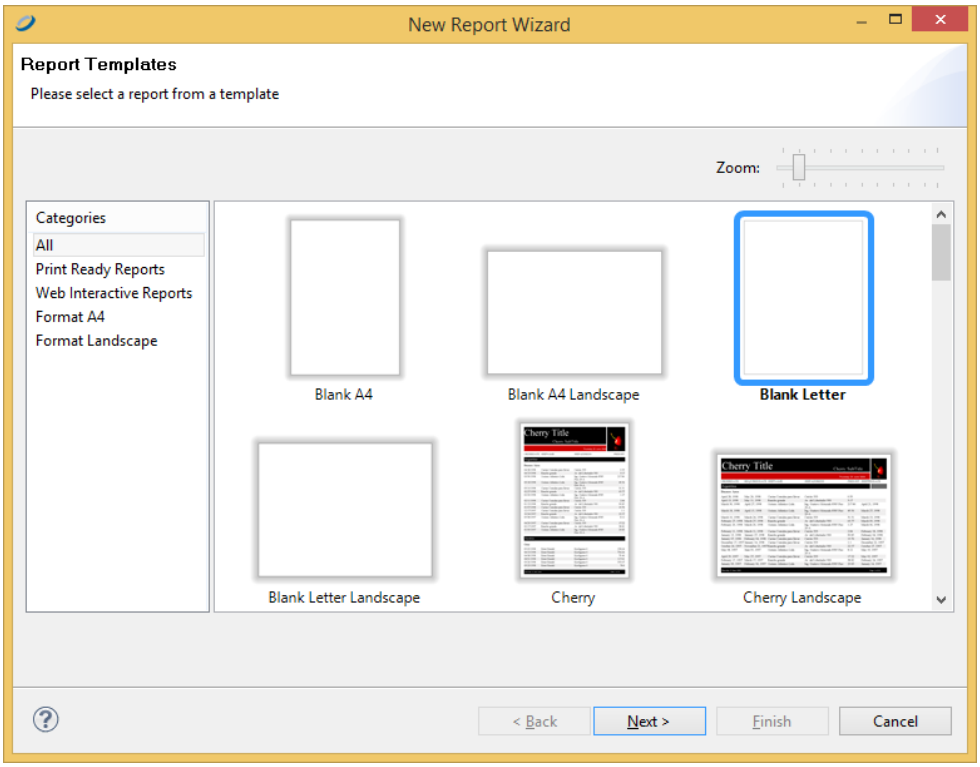


Y si es correcto el proceso deberá visualizarse el mensaje de satisfactorio, Ok, y Finish.

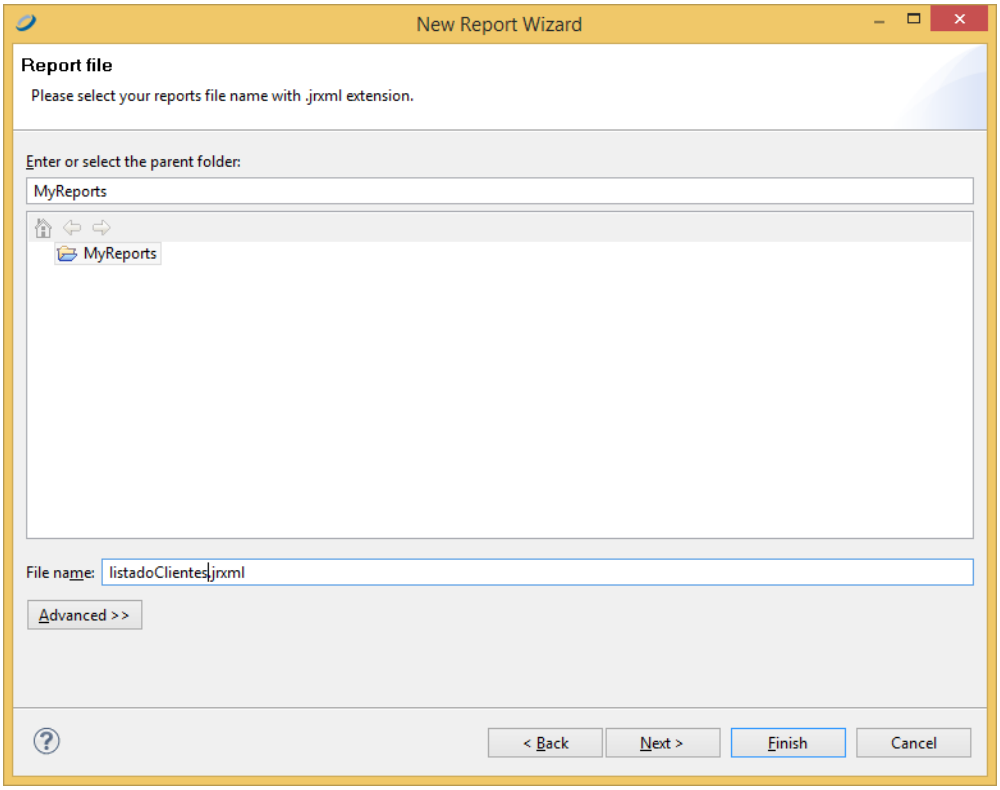
Creación de un Reporte en Blanco



Se encuentran muchos tipos de reportes, algunos de ellos pre configurados, con formatos establecidos, para este caso se utilizará uno en blanco tamaño carta

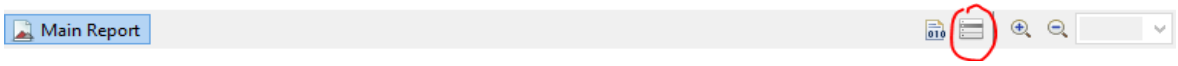


Se establecerá uno con el nombre de listadoClientes

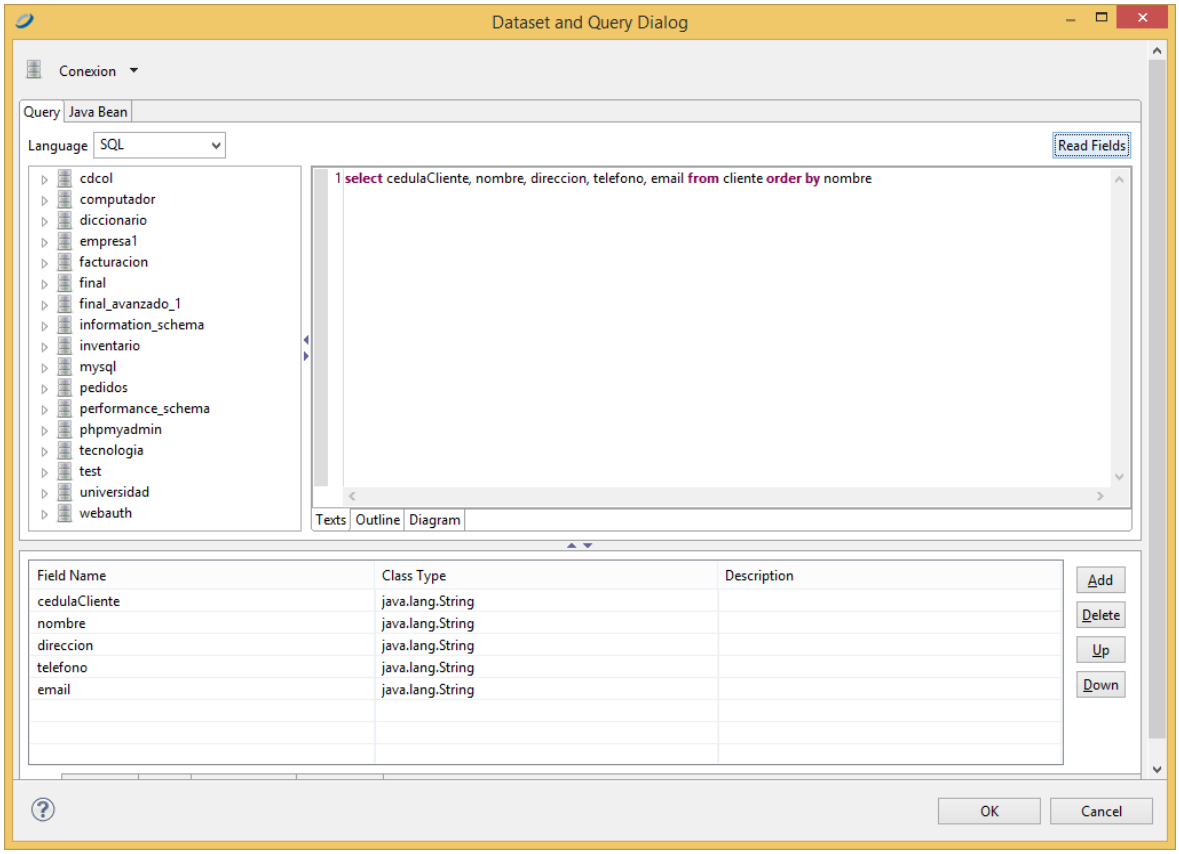


Se encuentra con la extensión jrxml, al ser un archivo xml se podrá editar manualmente o mediante asistente.

Establecer sentencia SQL

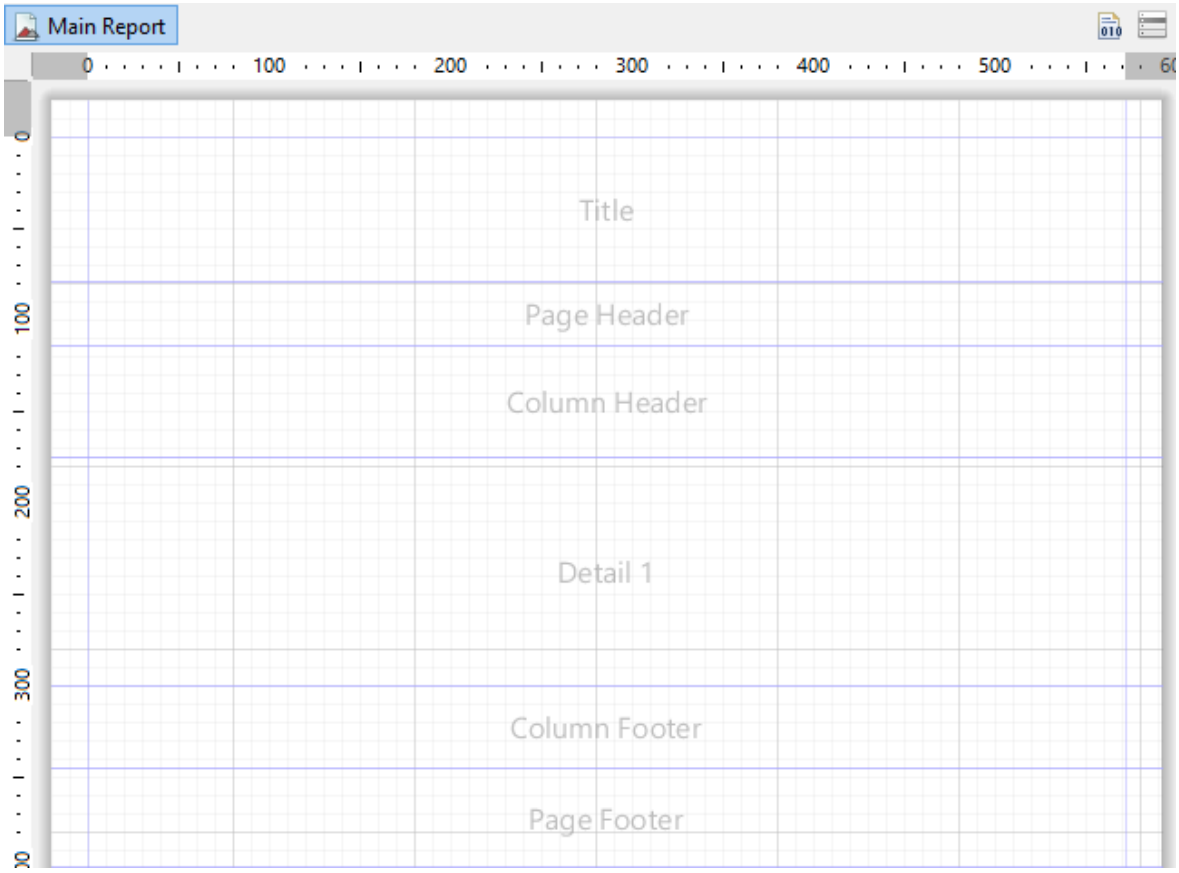


Mediante el icono Dataset and Query Editor dialog se creará la instrucción SQL de trabajo, inicialmente para una sentencia general.



En la esquina superior izquierda se selecciona el nombre de la conexión establecida para este proyecto (Conexion), en el dialogo principal se escribe la sentencia SQL sin punto y coma (;) al final para probarla y establecerla se selecciona el botón Read Fields, al pulsarlo aparecerán los campos en el dialogo de la parte inferior, OK para terminar

Área de Trabajo del Reporte



En el área de trabajo se encuentran varios items en un gris tenue, cada una de ellas para un tema puntual

Title titulo principal o cabecera de la empresa

Page Header títulos secundarios

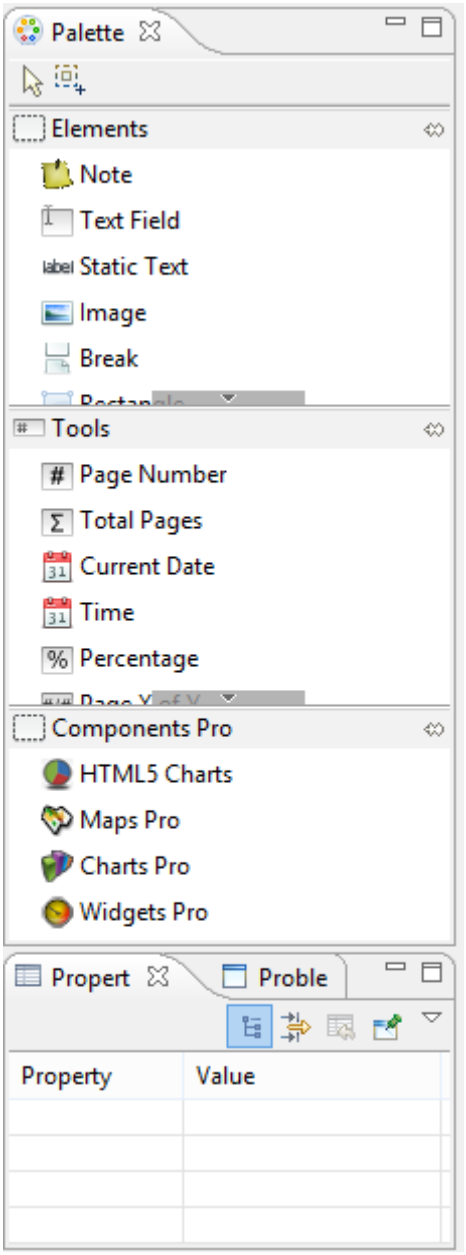
Column Header títulos de los campos o campos para información maestra

Detail 1 detallado de la información, esta área representa un ciclo con la información de la sentencia SQL

Column Footer información de pie de pagina

Page Footer Pie de Pagina

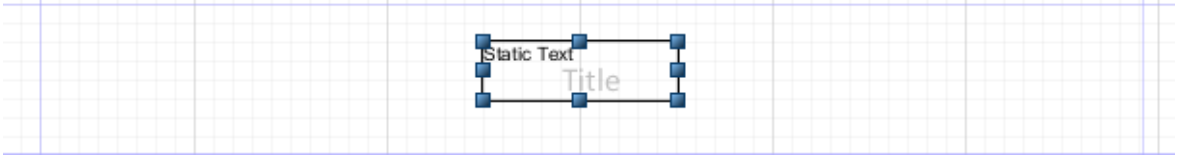
En el lado derecho se encuentran los controles con sus respectivas propiedades



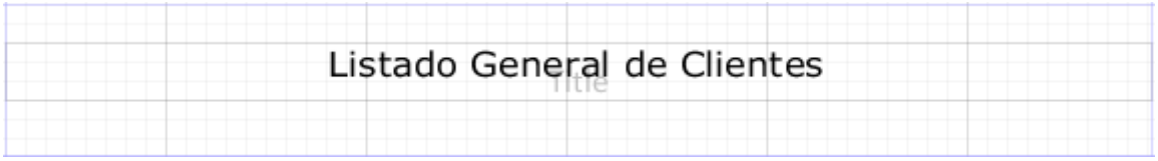
Y en el lado izquierdo otros componentes como campos y parámetros

Diseño Básico de un Reporte

Seleccione el elemento Static Text (label) del lado derecho de los componentes (Elements) y ubíquelo en el área tenue de Title

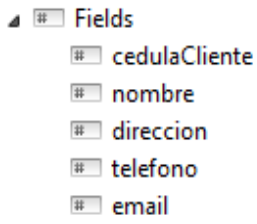


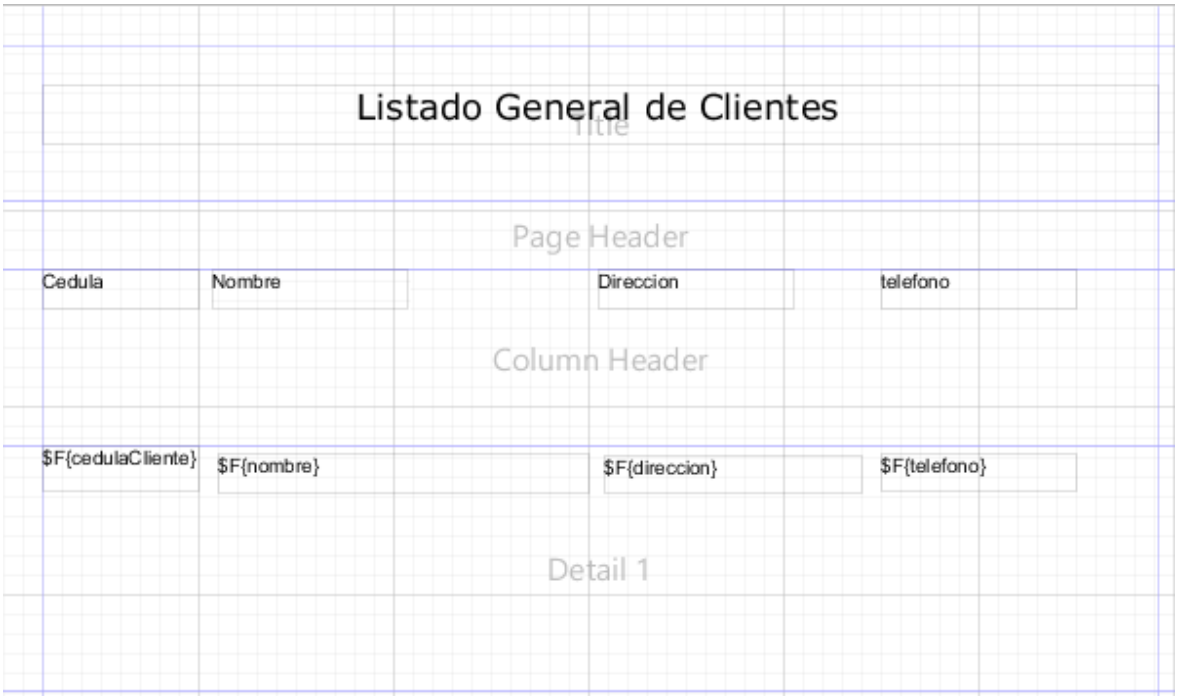
Edite el control y coloque el titulo o mensaje deseado para este informe, se pueden cambiar tipos de fuentes, tamaño y alineación



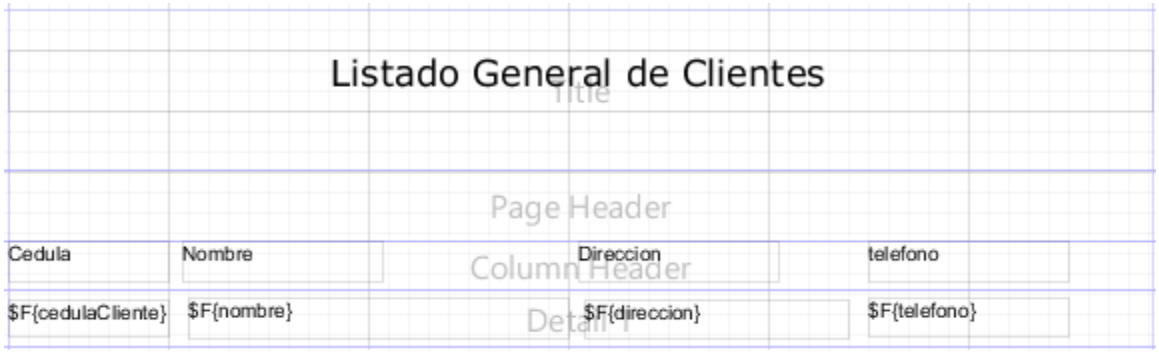
El resto del diseño depende exclusivamente de las necesidades visuales y creativas que se deseen aplicar, para este caso se prosigue con la ubicación de los campos.

Al lado izquierdo se encuentra una herramienta llamada Fields, al desplegar esta opción aparecen los capos de la sentencia SQL y se podrán arrastrar al área de Detail

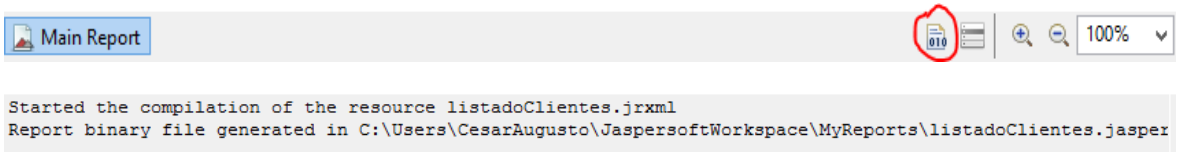




Disminuya los espacios de las áreas para una mejor distribución de los datos

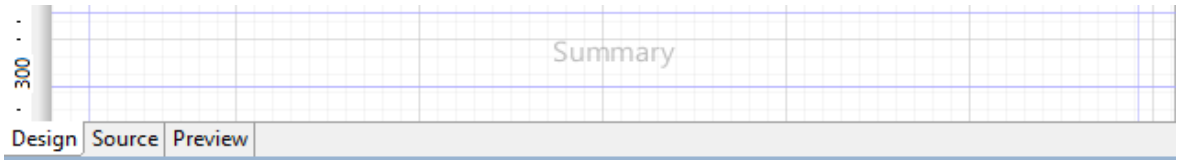


Guarde los cambios y compile el archivo (Compile Report)

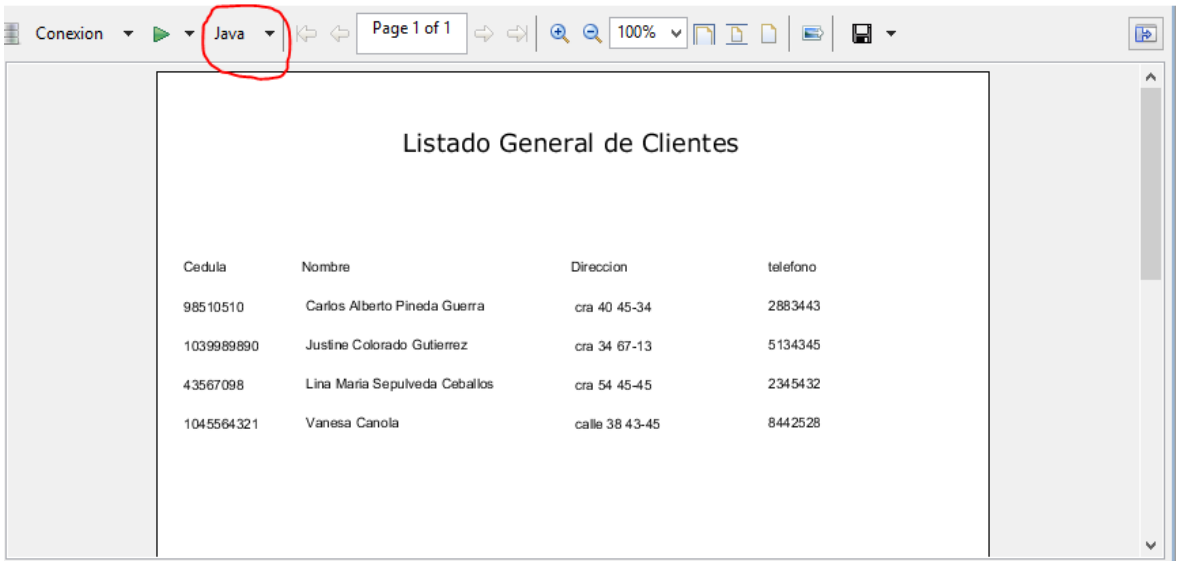


Este es el mensaje de compilación y la ruta de ubicación

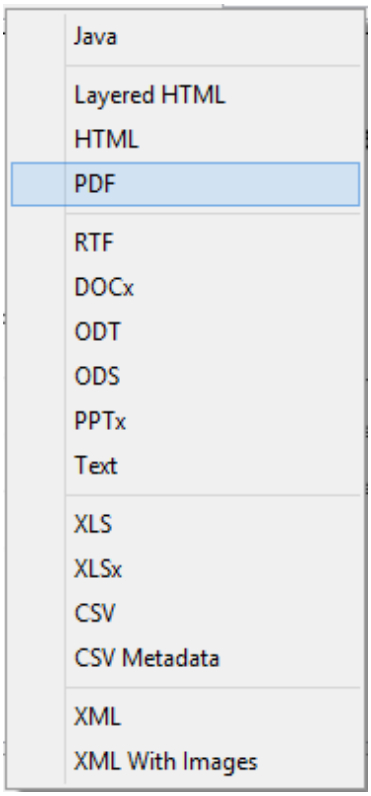
Visualización del reporte (Preview)



Elija la pestana Preview



Este es el informe generado por el Jasper, en el área marcada en la parte superior despliegue y elija el formato del archivo



Para este ejemplo se utilizará PDF

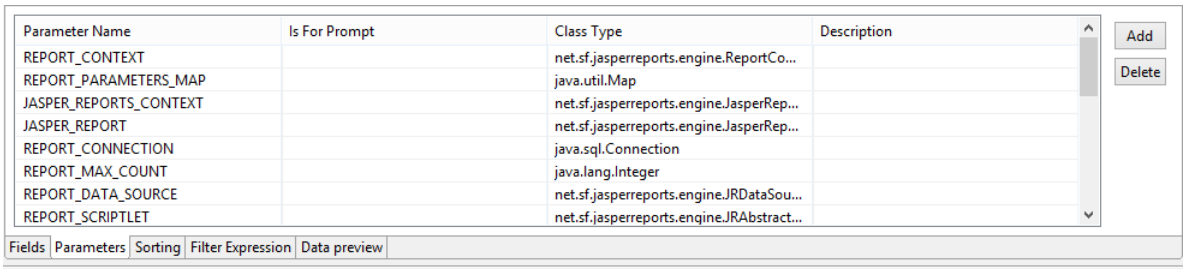
Recompile el proyecto para que tome los últimos cambios y el reporte está listo para ser vinculado al proyecto

El archivo creado es de extensión jasper, el otro archivo que el sistema genera jrxml es un archivo editable.

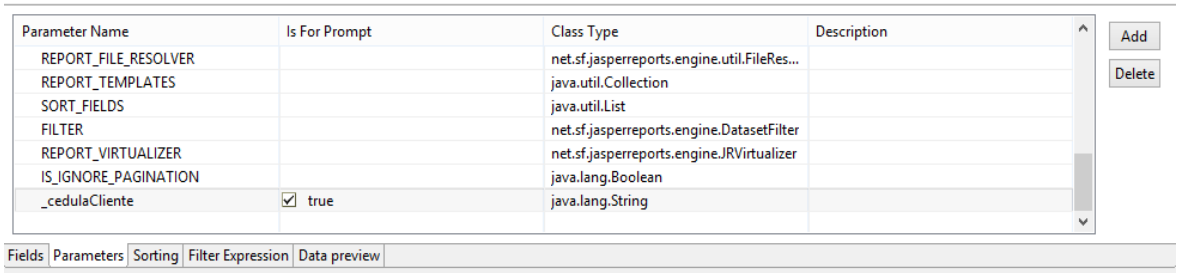
- **Creación de un reporte con parámetros**

La creación de un reporte con parámetros solo cambia en la sentencia SQL, todos los procesos de diseño son los mismos del reporte anterior.

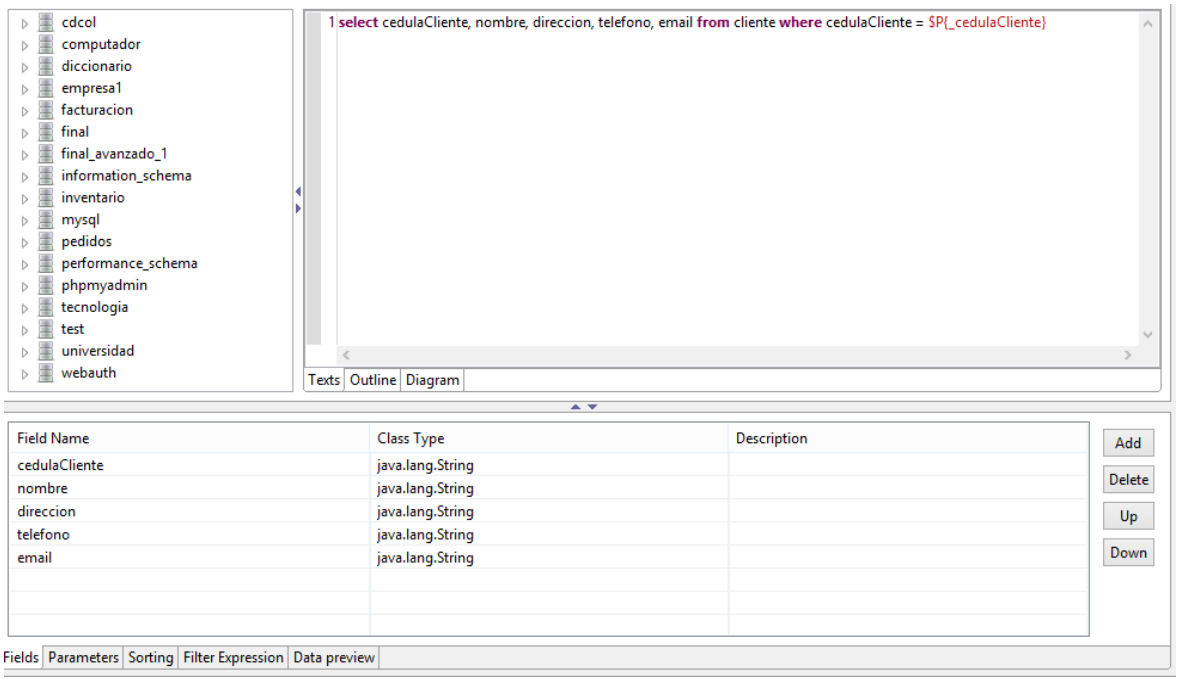
Creación de un parámetro



En la parte baja de la ventana DataSet and Query editor dialog se encuentra una pestaña Parameters, elija el botón add y escriba el nombre del parámetro

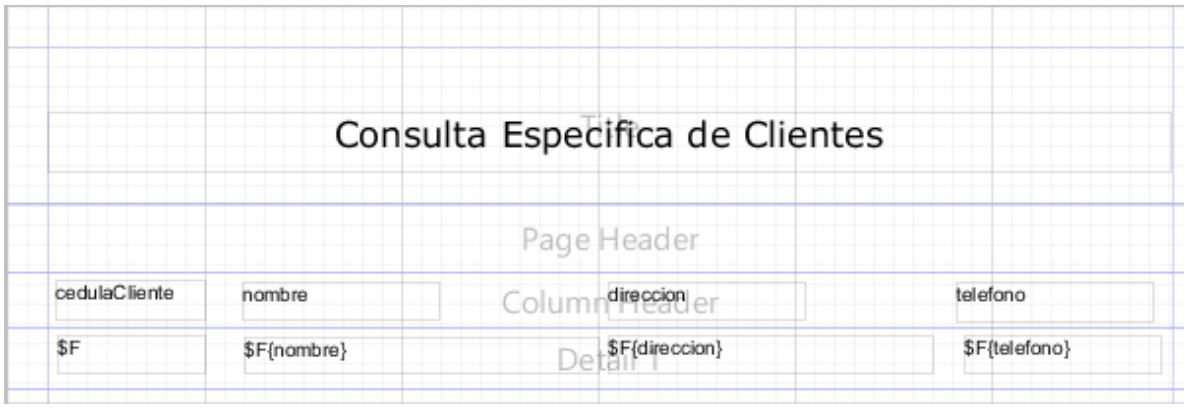


Vinculación de un parámetro a la sentencia SQL

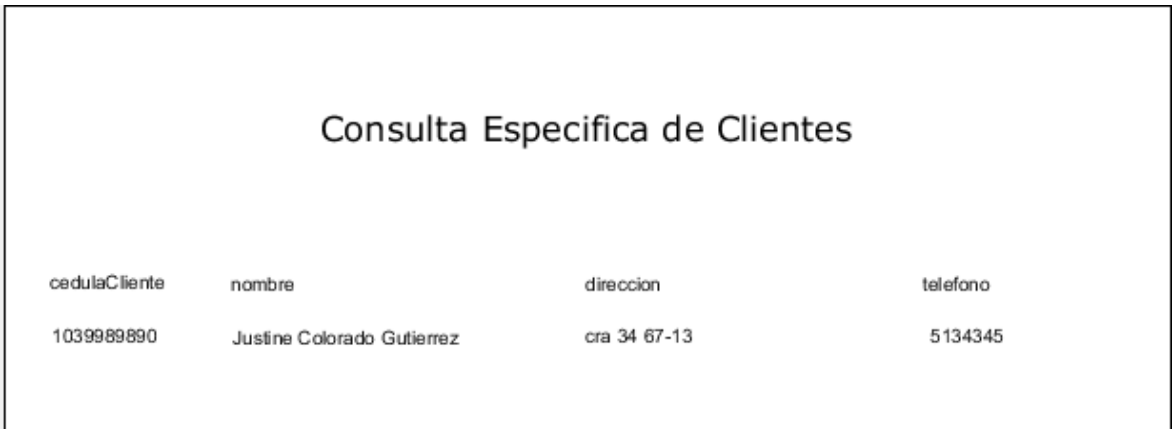


La sentencia se complementa con el parámetro, para este se utilizan los símbolos $P{parámetro}

Después de aplicar el botón Read Fields mostrara los campos en la parte inferior y esta listo el proceso para el diseño.



Dato consultado

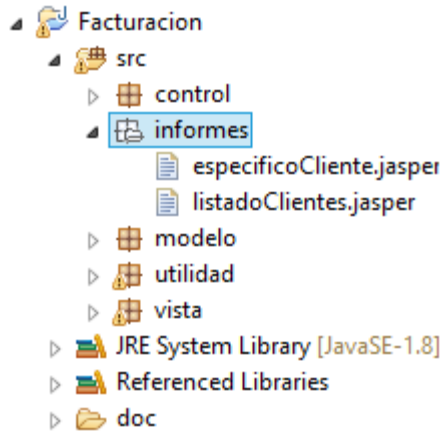


Antes de vincular los reportes al proyecto de Facturación se debe tener presenta un requerimiento importante, adicionar las librerías que acompañen este proceso, las librerías o archivos .jar que se requieren son

- commons-collections-2.1.1.jar
- commons-digester-2.1.jar
- commons-javaflow-20060411.jar
- commons-logging-1.1.1.jar
- groovy-all-2.0.1.jar
- iText-2.1.7.js2.jar
- jasperreports-5.2.0.jar
- jasperreports-applet-5.2.0.jar
- jasperreports-fonts-5.2.0.jar
- jasperreports-javaflow-5.2.0.jar
- png-encoder-1.5.jar
- poi-3.7-20101029.jar

Puede crear una carpeta dentro del proyecto donde los almacene y luego se vinculan mediante la opción Build Path

Cree un paquete informes y arrastre los archivos jasper a este paquete



Creación de una clase Informes en este mismo paquete

Configuración de la clase Informes

```
import java.net.URL;
import java.sql.Connection;
import java.util.HashMap;
import java.util.Map;

import javax.swing.JOptionPane;

import utilidad.Conexion;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.JasperPrint;
import net.sf.jasperreports.engine.JasperReport;
import net.sf.jasperreports.engine.util.JRLoader;
import net.sf.jasperreports.view.JasperViewer;
```

Método de llamado de un informe general

```
public void informeCliente  ()  {

    try {

        URL  ruta = this.getClass().getResource("listadoClientes.jasper");
        JasperReport archivo = (JasperReport) JRLoader.loadObject( ruta );
        JasperPrint imprimir = JasperFillManager.fillReport(archivo, null, Conexion.getConnection());
        JasperViewer verReporte = new JasperViewer(imprimir, false);
        verReporte.setVisible(true);
    }
    catch (Exception e) {

        JOptionPane.showMessageDialog(null,"Se produjo un error al leer el archivo .jasper" + e);
    }
}
```

Método de llamado de un informe con parámetros

```
public void especificoClientes (String cedula)  {

    try {

        URL  ruta = this.getClass().getResource("especificoCliente.jasper");
        JasperReport archivo = (JasperReport) JRLoader.loadObject( ruta );
        Map parametros = new HashMap();
        parametros.clear();
        parametros.put("_cedulaCliente", cedula);
        JasperPrint imprimir = JasperFillManager.fillReport(archivo, parametros, Conexion.getConnection());
        JasperViewer verReporte = new JasperViewer(imprimir, false);
        verReporte.setVisible(true);
     }
    catch(Exception e){

        JOptionPane.showMessageDialog(null,"Se produjo un error al leer el archivo .jasper" + e);
    }
}
```

El informe especifico podrá ser llamado desde el formulario de cliente y el general desde el menú principal, para el primero bastará con consultar una cedula y luego seleccionar el botón de informe, tenga presente instanciar la clase informe y la invocación del método que requiere.

2.4 TEMA 4 DOCUMENTACIÓN

Desarrollar aplicativos no comprende solamente el diseño de formularios, validaciones, y los procesos de funcionamiento del aplicativo, es mucho más que esto, hay que tener presente la ingeniería de requerimientos, el levantamiento de datos, el estudio pormenorizado de lo que se pretende hacer y cómo hacerlo, además de los manuales para el usuario final y la documentación o los manuales para el programador.

Java tiene una particularidad muy interesante que se denomina documentación, consiste en que a medida que se realiza el código del aplicativo se puede realizar dicha documentación y al final este se compila y tendríamos un formato de gran ayuda para el control del aplicativo.

Para generar este tipo de ayudas del programador, hay que tener muy presente a quien va dirigido este proceso, desarrollares y afines, son procesos muy técnicos que deberían de ubicarse en todos los procesos que comprenden el desarrollo

Identificación de un comentario y / o documentación

En los aplicativos de desarrollo cercanos al C++ como pueden ser JavaScript, entre otros coinciden en algunos símbolos como son

// que permite el comentario de una línea o anular una línea de código

/*

Este otro símbolo permite el comentario o anulación de múltiples líneas de código

*/

/**

Esta instrucción a pesar de que es similar al anterior se utiliza para representar comentarios

*/

Identificadores de los comentarios

@author

Especifica el autor (es) del aplicativo, modulo o fragmento de código

@version

Versión del aplicativo, módulo o fragmento de código

@see

Indica que referencias a otras clases o métodos existen

@param

Nombre de parámetro y descripción de uso y significado

@return

Describe lo que se devuelve

@exception

Nombre de la excepción que se está utilizando y excepción que puede lanzarse

@throws

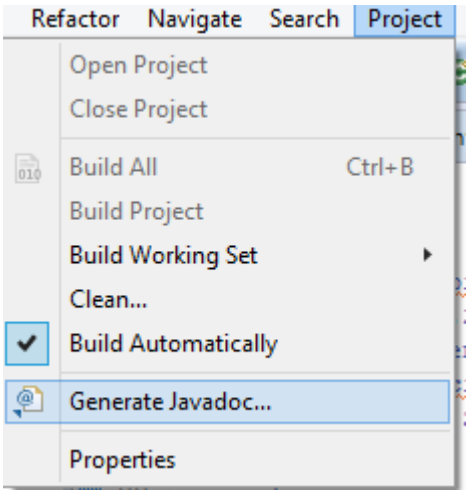
Nombre de la excepción que se está utilizando y excepción que puede lanzarse

Ejemplo

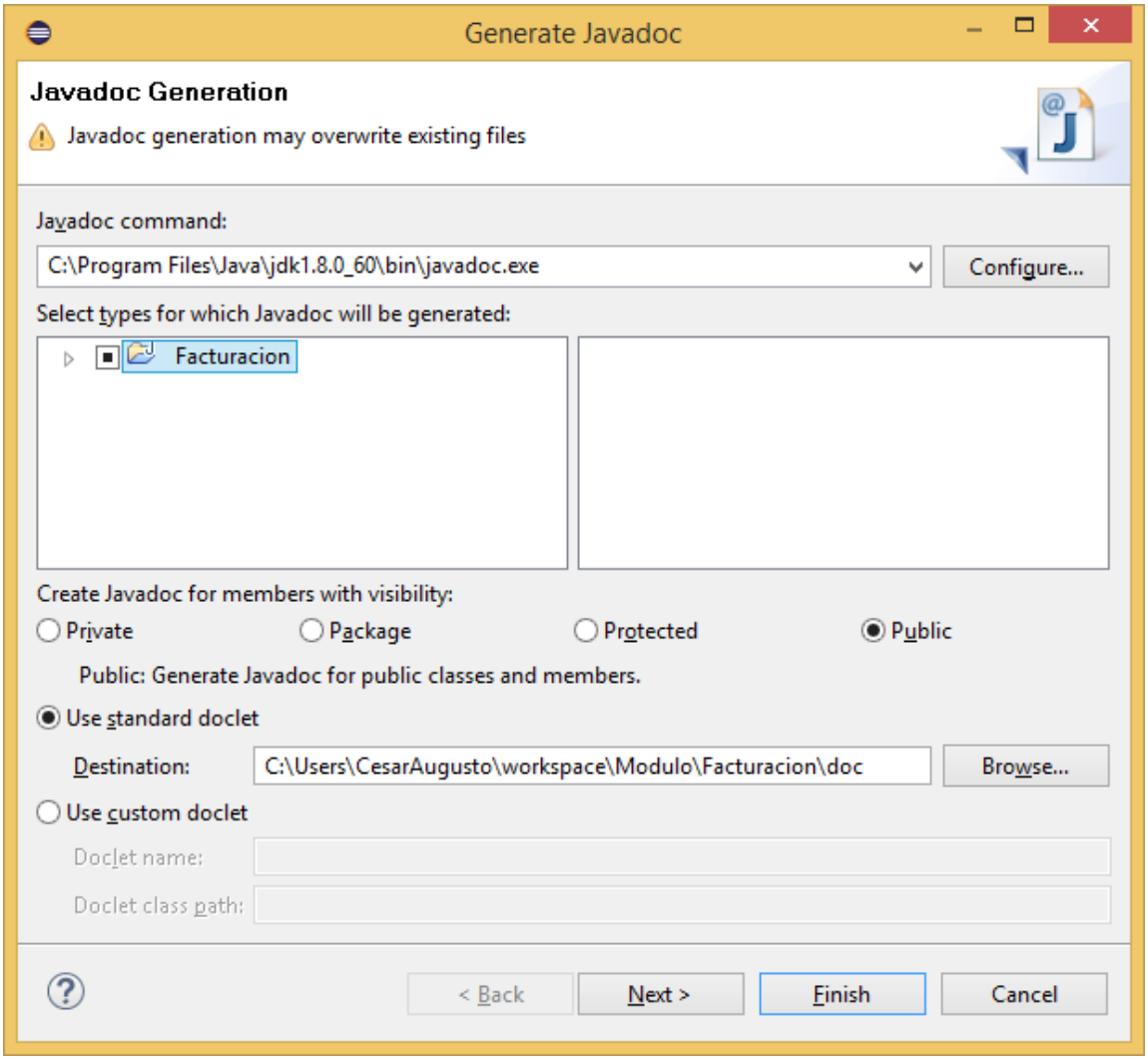
```
/**
 * @author Pepito Perez
 * @version 1.2
 * @param cliente
 * @return vacio
 * @exception SQLException si la sentencia SQL es incorrecta mostratra el mensaje de excepcion
 *
 */
public static void modificar (Cliente cliente) {

    try {

        PreparedStatement registroCliente = cnn.prepareStatement("call modificarCliente (?, ?, ?, ?, ?)");
        registroCliente.setString(1, cliente.getCedulaCliente());
        registroCliente.setString(2, cliente.getNombre());
        registroCliente.setString(3, cliente.getDireccion());
        registroCliente.setString(4, cliente.getTelefono());
        registroCliente.setString(5, cliente.getEmail());
        registroCliente.executeUpdate ();
        JOptionPane.showMessageDialog(null, "Registro Actualizado");
    }
    catch (SQLException sqle) {

        sqle.printStackTrace ();
    }
}
```

Se genera una documentación básica para uno de los métodos del archivo DAO, se especifica el autor, la versión, el parámetro, el retorno y una excepción, hay que tener presente que la documentación no debería de ser opcional, todos los procesos que se hagan deben de llevarlo con el fin de controlar y asegurar un funcionamiento optimo.

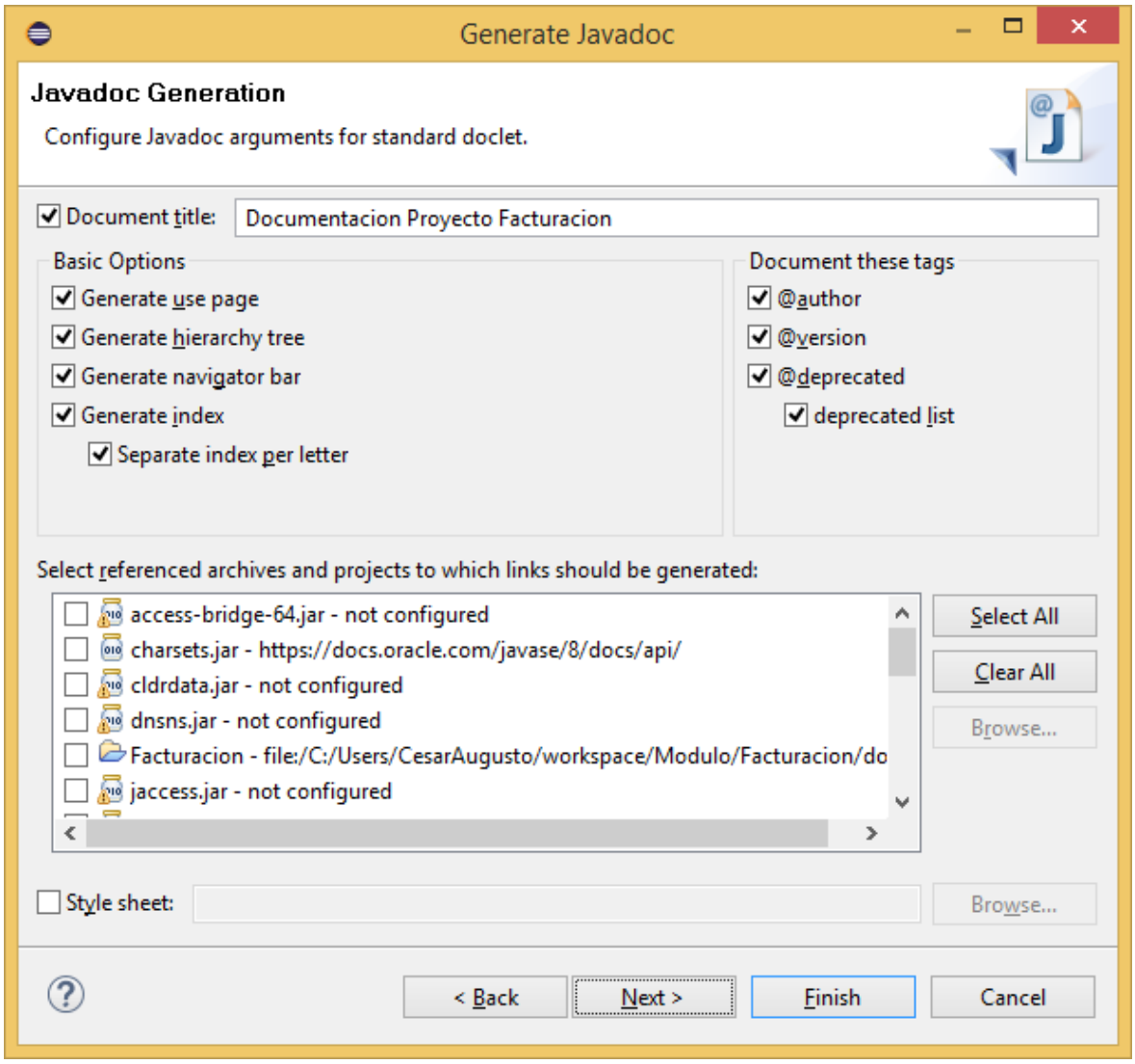
Cuando se realiza el proceso de documentación, se procede a la implementación de este.



En el menú principal se selecciona la opción Project y la opción Generate Javadoc...

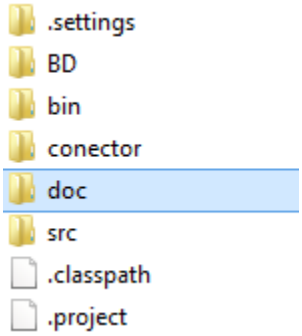


Se especifica el archivo que creara la documentación, javadoc.exe que se ubica en la carpeta bin del JDK

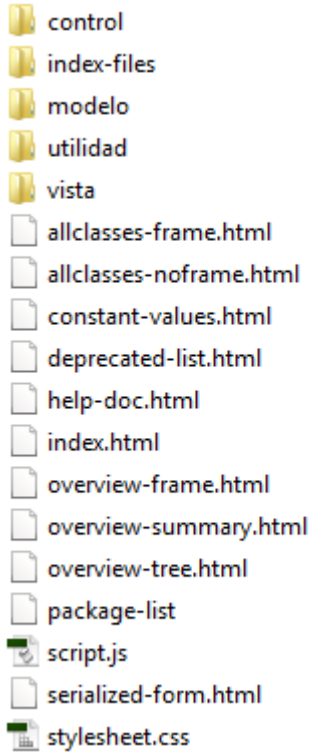


Se da un titulo de trabajo, y se podrá asignar un archivo de hoja de estilos si no se desea trabajar con el formato por defecto del java y luego el botón Finish

El sistema se creará con un formato HTML dentro del proyecto que se esta trabajando o en otra ruta si esta se especifico previamente.



Dentro de la carpeta doc encontramos



Y podemos ejecutar el archivo index.html

Encontramos algunos apartes como el siguiente

Method Summary

All Methods	Static Methods	Concrete Methods

Modifier and Type	Method and Description
static modelo.Cliente	consultar(java.lang.String cedula)
static void	eliminar(java.lang.String cedula)
static void	insertar(modelo.Cliente cliente)
static void	modificar(modelo.Cliente cliente)
static int	verificar(java.lang.String cedula)

En el aparecen los métodos utilizados, el nombre y el parámetro que recibe

Y algunos apartes de lo que se especifico en el método

modificar

```
public static void modificar(modelo.Cliente cliente)
```

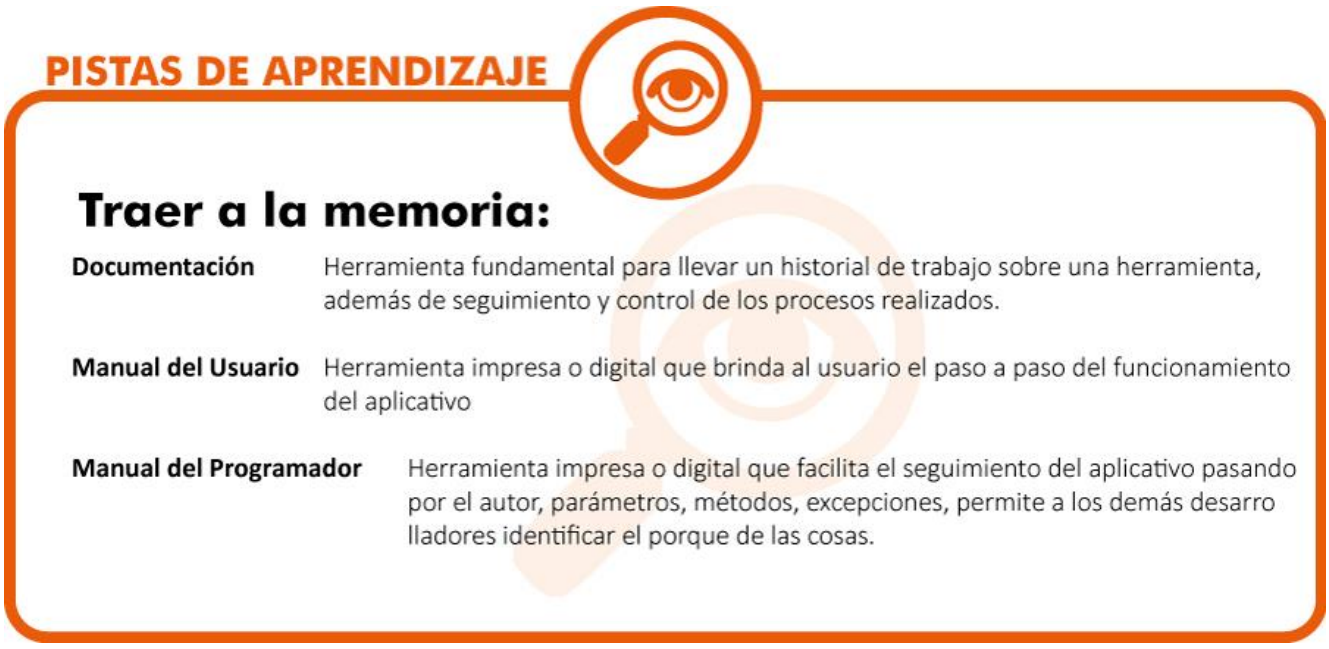
Parameters:

cliente –

Throws:

java.sql.SQLException – si la sentencia SQL es incorrecta mostratra el mensaje de excepcion

Es de vital importancia del manejo de esta herramienta en aplicaciones que involucran varios desarrolladores o que exista la posibilidad de que otras personas lleguen a manejarlo, el trabajo en equipo es de gran utilidad, y esta herramienta facilita esta tarea.



2.4.1 EJERCICIO DE APRENDIZAJE

Nombre del taller de aprendizaje: Sistema de Matricula	Datos del autor del taller: Cesar Augusto Jaramillo Henao
Escriba o plantee el caso, problema o pregunta: Elaborar un Modelo Entidad Relación que comprenda tablas para el control matriculas de estudiantes en una institución, representando facultad, carrera, docente, alumno, y las notas de estos aplicado mediante un CRUD en java	
Solución del taller: Mediante las herramientas vistas de la creación de un CRUD en java, documentación, reportes, pool de conexiones, aplique cada uno de los conceptos visto, utilice un modelo amplio de tablas que le den la experiencia que se requiere para proyectos más grandes	

2.4.2 TALLER DE ENTRENAMIENTO

Nombre del taller: Hospital	Modalidad de trabajo:
Actividad previa: Realice con detalle el ejercicio planteado en la primera unidad	
Describa la actividad: Cree un modelo relacional del funcionamiento básico de un hospital, en el que tome elementos esenciales de paciente, medico, enfermedades, medicamentos, incapacidades, cirugías, aplique este modelo en el lenguaje java con los conceptos de reportes, documentación, pool de conexiones	

3 UNIDAD 2 HILOS

3.1.1 RELACIÓN DE CONCEPTOS

Componentes son los elementos que permiten realizar la utilización de los hilos

Métodos espacio de trabajo en código para realizar distintas tareas lógicas

Run método principal que ejecuta un hilo

Start método que da inicio o llamado a un método principal

Sincronización organización del llamado de los procesos

3.1.2 OBJETIVO GENERAL

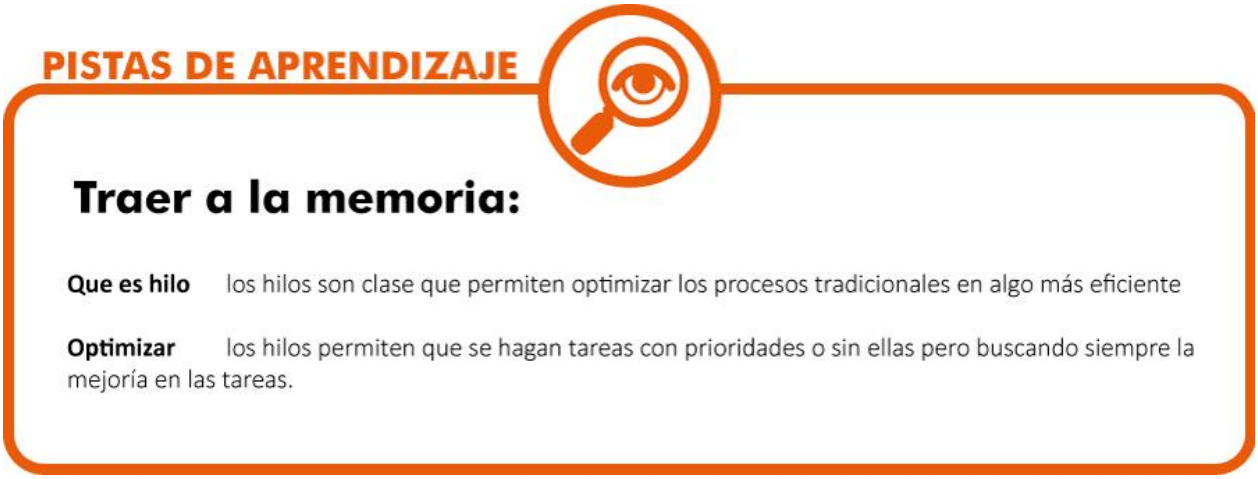
Aprender a utilizar los Hilos, herramienta fundamental en la POO, las características de aprovechamiento del reloj del sistema y las características del procesador.

3.1.3 OBJETIVOS ESPECÍFICOS

- Conocer los principales componentes del manejo de hilos
- Identificar que es un hilo
- Desarrollar habilidades de la ubicación de los hilos y que ventajas prestas en la etapa de programación.

3.2 TEMA 1 DEFINICIÓN Y OBJETIVOS

En los aplicativos que se han desarrollado hasta esa etapa están diseñados de forma secuencial (línea por línea), en muchas ocasiones se encontraran que este tipo de desarrollo no es el más útil o más aconsejable por la lentitud que puede generar y la poca efectividad que mostrara, para esta etapa se pasara a un tema de concurrencia o de procesos paralelos llamados hilos (threads).



3.3 TEMA 2 COMPONENTES

Existe una gran cantidad de componentes de los hilos, pero dentro de los más comunes para esta tarea son

Start() inicia la ejecución de un hilo, este se ubica en el run

Run() método principal del hilo

Runnable () implementación de la interfaz

Thread es una clase padre de la que dependerá nuestro hilo

3.4 TEMA 3 IMPLEMENTACIÓN DE LA INTERFAZ RUNNABLE

En java los hilos heredan de la clase Thread, de esta hereda el método run, en este método es donde se debe de programar el proceso que deseamos.

Dentro de este método se encontrará un método sleep que representa dormir o esperar para ejecutar otro proceso.

Para el ejemplo se mostrarán 5 nombres y mediante la función random se generará un tiempo de espera entre 0 y 9999 milisegundos

```
package contruccion;

public class EjemploHilo extends Thread{

    private String nombre;

    public EjemploHilo (String nombre) {

        this.nombre = nombre;
    }

    public void run () {

        try {

            int tiempoEspera = (int) (Math.random()*10000);
            Thread.sleep(tiempoEspera);
            System.out.print ("\nEl Nombre es: " + nombre +" el tiempo de espera es " + tiempoEspera + " milesegundos");
        }
        catch (Exception ex) {

            ex.printStackTrace();
        }
    }

     public static void main (String args []) {

         EjemploHilo ejemploHilo1 = new EjemploHilo ("Lina");
         EjemploHilo ejemploHilo2 = new EjemploHilo ("Miguel");
         EjemploHilo ejemploHilo3 = new EjemploHilo ("Johana");
         EjemploHilo ejemploHilo4 = new EjemploHilo ("Isabel");
         EjemploHilo ejemploHilo5 = new EjemploHilo ("Jose");

         ejemploHilo1.start();
         ejemploHilo2.start();
         ejemploHilo3.start();
         ejemploHilo4.start();
         ejemploHilo5.start();
     }
}
```

El proceso principal se ejecuta desde el método run que es el principal para esta tarea particular, en el void main se especifican 5 hilos con 5 nombres y la respectiva ejecución con el método start() que invoca el método run(), esta tomara un valor aleatorio y mostrara la información.

Runnable

Dentro de la clase Thread se implementa la interfaz Runnable que hereda del método run().

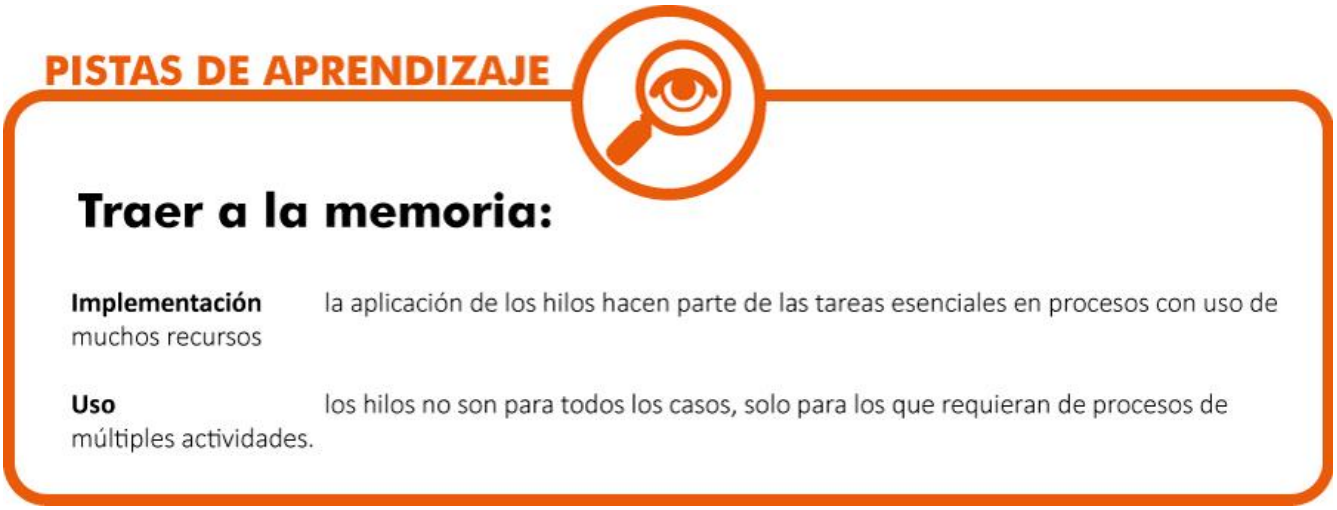
Acoplemos el ejemplo anterior a esta interfaz

```
package contruccion;

public class EjemploHilo implements Runnable{

    private String nombre;

    public EjemploHilo (String nombre) {

        this.nombre = nombre;
    }

    public void run () {

        try {

            int tiempoEspera = (int) (Math.random()*10000);
            Thread.sleep(tiempoEspera);
            System.out.print ("\nEl Nombre es: " + nombre +" el tiempo de espera es " + tiempoEspera + " milesegundos");
        }
        catch (Exception ex) {

            ex.printStackTrace();
        }
    }
```

```
    public static void main (String args []) {

        Thread ejemploHilo1 = new Thread (new EjemploHilo ("Lina"));
        Thread ejemploHilo2 = new Thread (new EjemploHilo ("Miguel"));
        Thread ejemploHilo3 = new Thread (new EjemploHilo ("Johana"));
        Thread ejemploHilo4 = new Thread (new EjemploHilo ("Isabel"));
        Thread ejemploHilo5 = new Thread (new EjemploHilo ("Jose"));

        ejemploHilo1.start();
        ejemploHilo2.start();
        ejemploHilo3.start();
        ejemploHilo4.start();
        ejemploHilo5.start();
    }
}
```

En este ejemplo la clase EjemploHilo no hereda de Thread pero si implementa la interfaz Runnable de la que aplica una sobreescritura del metodo run().

Esta nueva version del ejemplo es mas practica que la primera y mas flexible en su uso al no limitar la herencia de la clase.



3.5 TEMA 4 CICLO DE VIDA

Los hilos o thread tiene varias etapas, las más llamativas son la instancia y la ejecución, pero existen otras etapas hasta finalizar o morir, todo este recorrido se llama ciclo de vida.

Cuando se instancia un hilo se denomina que esta creado, cuando realizamos el llamado se aplica el método start(), y pasa al estado de lectura o ready, luego de este estado llega el running que esta administrado por el sistema operativo, existen otros como el scheduler o yield.

Cuando está en running está haciendo uso del procesador, el hilo puede ejecutar la última línea del método run y finalizaría la tarea programada, pero también se pueden ejecutar otros métodos como wait o sleep que se mencionó en el ejemplo visto anteriormente.

Si el hilo entra en el método wait saldrá de el cuándo se ejecute uno de los métodos notify o notifyall

Si está dormido (sleep) saldrá cuando finalice el tiempo de estar dormido

3.6 TEMA 5 PRIORIDADES

Dentro de las prioridades y la asignación de mayor o menor posibilidad de ejecución se determina mediante la instrucción scheduler, este proceso favorece al momento de asignar el tiempo del procesador, en este caso se encuentran elementos como Thread.MIN_PROPERTY o Thread.MAX_PROPERTY, estos procesos manejan unas constantes que van de 1 a 10

Ejemplo

```
package contruccion;

public class EjemploHilos2 {

    public static void main (String args []){

        Hilo hilo1 = new Hilo ("Miguel");
        Hilo hilo2 = new Hilo ("Luna");

        hilo1.setPriority(Thread.MAX_PRIORITY);

        System.out.print (Thread.MAX_PRIORITY);
        System.out.print (Thread.MIN_PRIORITY);

        hilo1.start();
        hilo2.start();
    }

    static class Hilo extends Thread {

        String nombre;

        public Hilo (String nombre) {

            this.nombre = nombre;
        }

    public void run () {

        for (int i = 0; i < 5; i++) {

            System.out.print ("\n " + nombre + "\t" + i);
            yield();
        }
    }
}
}
```

Obtendrá un resultado así

```
101
 Miguel 0
 Miguel 1
 Luna   0
 Miguel 2
 Luna   1
 Miguel 3
 Luna   2
 Miguel 4
 Luna   3
 Luna   4
```

Haga la prueba con un MIN_PROPERTY y vera los cambios en las prioridades

3.7 TEMA 6 SINCRONIZACIÓN

En muchas ocasiones se presentarán situaciones en las que un hilo intente acceder a los recursos de otro, esto puede ocasionar problemas porque podría afectar el resultado definitivo, si un hilo está haciendo una suma y otro una división los resultados probablemente no serán los esperados, es acá donde se procede a realizar un proceso de sincronización de tareas.

La sentencia utilizada para este tipo de caso es synchronized y permitirá que los métodos estén relacionados con el reloj del procesador.

Un ejemplo de un método sincronizado seria el siguiente.

```
public synchronized void poner (char c) throws Exception {

    while (lleno) {

        wait ();
    }

    buffer [tope++] = c;
    vacio = false;
    lleno = tope >= buffer.length;

    notifyAll ();
}
```

3.7.1 EJERICICIO DE APRENDIZAJE

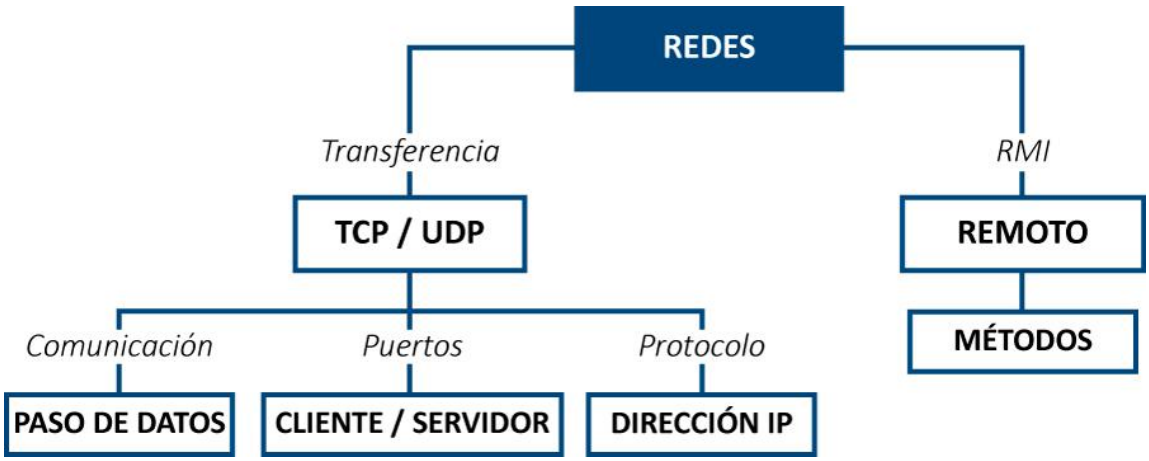
Nombre del taller de aprendizaje: Notas	Datos del autor del taller: Cesar Augusto Jaramillo
Escriba o plantee el caso, problema o pregunta: Cree un programa que calcule las notas definitivas con 300 estudiantes aplicando los conceptos básicos de programación y luego realice el mismo ejercicio con hilos	
Solución del taller: Aplique los conceptos vistos en la unidad	

3.7.2 TALLER DE ENTRENAMIENTO

Nombre del taller: Primos	Modalidad de trabajo: Individual
Actividad previa: Elaborar un aplicativo que calcule numero primos	
Describa la actividad: Imprimir los primeros 500 números primos, calcular con y sin hilos y compare la diferencia en tiempo	

4 UNIDAD 3 REDES

4.1.1 RELACIÓN DE CONCEPTOS



Servidor es un aplicativo gestor de la información, provee los recursos que se necesita por parte de un cliente

Cliente es un aplicativo que solicita información a un servidor

Dirección IP es la ubicación única dentro de una red

TCP protocolo de control de transmisión

UDP protocolo de nivel de transporte

4.1.2 OBJETIVO GENERAL

Conocer las características de la programación en red, los recursos de comunicación los protocolos y el manejo de los datos.

4.1.3 OBJETIVOS ESPECÍFICOS

- Identificar las características principales para programar en Red
- Conocer los componentes esenciales para la programación en red
- Identificar los comandos más comunes para la programación en red

4.2 TEMA 1 CONCEPTOS BÁSICOS

Muchos de los aplicativos que escribimos están diseñados para ser utilizados en una sola máquina, esto es muy limitante por el crecimiento constante de las empresas y de estar conectado a los distintos recursos que se pueden utilizar.

Para este propósito debemos de familiarizarnos con el manejo de los protocolos, esencialmente dos de ellos que nos permiten realizar esta tarea, TCP (Transmission Control Protocol) y UDP (User Datagram Protocol), estos protocolos implementan lo que conocemos como la capa de transporte.

4.3 TEMA 2 TCP / UDP

TCP: Es un protocolo orientado a conectar dos equipos de manera segura y confiable, cuando la comunicación se establece se crea un canal por medio del cual se puede enviar y recibir información, el protocolo TCP garantiza que los datos que se envían serán íntegros, de lo contrario reportara un error.

La comunicación TCP es análoga a una comunicación telefónica, en que un usuario llama y el otro determina o no atenderlo, cuando decide atenderlo establecen una “conversación” de forma bidireccional.

Dentro de los procesos más comunes de este tipo de protocolo están FTP, Telnet, HTTP, en estos procesos es fundamental respetar el orden de envío de las tareas.

UDP: La comunicación establecida mediante este protocolo no es confiable ni garantizada como en el caso de TCP, esto debido a que UDP no es un protocolo de conexión, en el UDP se envían paquetes de datos llamados datagramas, el envío de estos es comparable con el envió del correo o correspondencia tradicional, en este ejemplo nos encontramos que el envío de una carta no nos preocupa en qué orden llega a su destino.

Puerto: Los puertos son los mecanismos para hacer llegar la información al aplicativo que lo solicito, cada pc tiene una única conexión física por medio de la cual se recibe la información, los puertos constituyen una dirección interna que direcciona un proceso dentro del equipo de cómputo.

Dirección IP: Una dirección IP (Internet Protocol), es un numero de 32 bits que direcciona de manera única a un pc dentro de la red.

Aplicación Cliente / Servidor: Es un orden jerárquico de las aplicaciones de una red, una aplicación cliente solicita información a una aplicación servidor, este último proveerá los servicios a un cliente según las características del aplicativo gestor.

Sockets: Es conocido como uno de los extremos en una comunicación de programas, es la forma de comunicar un servidor con un cliente, este socket direcciona la información de forma única a la aplicación solicitante.

Servidor: Es un programa que permite la que se conecten los distintos programas clientes, esto se conoce como "escuchar" un cliente"

Clases comunes:

ServerSocket se utilizar para esperar y escuchar la llegada de los clientes

Socket se puede entablar la comunicación cliente/servidor

Ejemplo de servidor

```
public class Servidor {

    public static void main (String args []) throws Exception   {

        ObjectInputStream objectInputStream = null;
        ObjectOutputStream objectOutputStream = null;
        Socket socket = null;
        ServerSocket serverSocket = new ServerSocket(5432);

        while (true)    {

            try {

                socket = serverSocket.accept();
                System.out.print ("\nSe Conectaron desde la IP: " + socket.getInetAddress());
                objectInputStream = new ObjectInputStream (socket.getInputStream());
                objectOutputStream = new ObjectOutputStream (socket.getOutputStream());

                String nombre = (String) objectInputStream.readObject();
                String saludo = "Hola " + nombre + " " + System.currentTimeMillis();

                objectOutputStream.writeObject(saludo);
                System.out.print ("\nSaludo Viajando...");

            }
```

```
            catch (Exception ex){

                ex.printStackTrace();
            }
            finally {

                if (objectOutputStream != null)

                    objectOutputStream.close();
                if (objectInputStream != null)

                    objectInputStream.close();
                if (socket != null)

                    socket.close();

                System.out.print ("\nConexion cerrada");

            }
        }
    }
}
```

El servidor instancia un ServerSocket con un puerto aleatorio que como ejemplo se tendrá el 5432, la instrucción acept es la encarda de esperar la conexión de un cliente. La instrucción getInnetAddress tomara la IP del cliente, el manejo de los datos de la forma tradicional envia solo bytes pero con las clases ObjectInputStream y/o ObjectOutputStream se procesa como objetos, estas clases leen y escriben objetos por medio de la red.

Cliente

```
public class Cliente {

public static void main (String args []) throws Exception   {

        ObjectInputStream objectInputStream = null;
        ObjectOutputStream objectOutputStream = null;
        Socket socket = null;

        try {

            socket = new Socket ("127.0.0.1", 5432);
            objectOutputStream = new ObjectOutputStream (socket.getOutputStream());
            objectInputStream = new ObjectInputStream (socket.getInputStream());

            objectOutputStream.writeObject("Jose");

            String ret = (String) objectInputStream.readObject();
            System.out.print ("\n" + ret);
        }
        catch (Exception ex){

            ex.printStackTrace();
        }



            finally {

                if (objectInputStream != null)

                    objectInputStream.close();
                if (objectOutputStream != null)

                    objectOutputStream.close();

                if (socket != null)

                    socket.close();
            }
        }
    }
```

Para el cliente comunicarse con el servidor, se tiene un puerto y una dirección IP, para este caso se aplica un servidor local.

4.4 TEMA 2 RMI

RMI (Remote Method Invocation), es una tecnología de invocación remota de métodos, como su nombre lo indica invoca métodos, cuando estos se encuentran en una máquina virtual y los llama de otra máquina virtual, esto se conoce como objeto remoto.

El servidor se encarga de instanciar los objetos remotos y los hace disponibles al cliente, esto se ubica en una colección o repositorio de objetos.

Los objetos remotos son los publicados por el servidor a los que se podrán acceder por el cliente remotamente, ambas maquinas utilizan para esta tarea la máquina virtual, a la hora de considerar que un objeto sea remoto deberá heredar la clase java.rmi.UnicastRemoteObject.

4.5 TEMA 3 APLICACIÓN

A continuacion se vera un pequeño ejemplo de RMI

Para esto se crea un proyecto llamado RMI, un archivo ObjetoRomoto.java, este archivo es una interfaz y dara caracteristicas generales del proyecto

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ObjetoRemoto extends Remote {

    public String obtenerSaludo (String nombre) throws RemoteException;
}
```

El seguinte archivo es una clas tradicional llamada ObjetoRemotoImplementacion.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class ObjetoRemotoImplementacion extends UnicastRemoteObject

implements ObjetoRemoto {

    protected ObjetoRemotoImplementacion() throws RemoteException {

        super();
    }

    public String obtenerSaludo(String nombre) throws RemoteException {

        return "Hola RMI" + nombre;
    }
}
```

ServidorRMI

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class ServidorRMI {

    public static void main (String [] args) throws Exception {

        ObjetoRemotoImplementacion objetoRemotoImplementacion = new ObjetoRemotoImplementacion ();
        Registry registry = LocateRegistry.getRegistry(1099);
        registry.rebind("OBJRemoto", objetoRemotoImplementacion);
    }

}
```

ClienteRMI

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class ClienteRMI {

    public static void main (String args []) throws Exception {

        Registry registry = LocateRegistry.getRegistry("127.0.0.1", 1099);
        ObjetoRemoto objetoRemoto;
        objetoRemoto = (ObjetoRemoto) registry.lookup("OBJRemoto");
        String saludo = objetoRemoto.obtenerSaludo("Alexandra");
        System.out.print ("\n" + saludo);

    }
}
```

PISTAS DE APRENDIZAJE

Traer a la memoria:

Remoto proceso que se encuentra en un lugar distinto a la ubicación inicial

4.5.1 EJERCICIO DE APRENDIZAJE

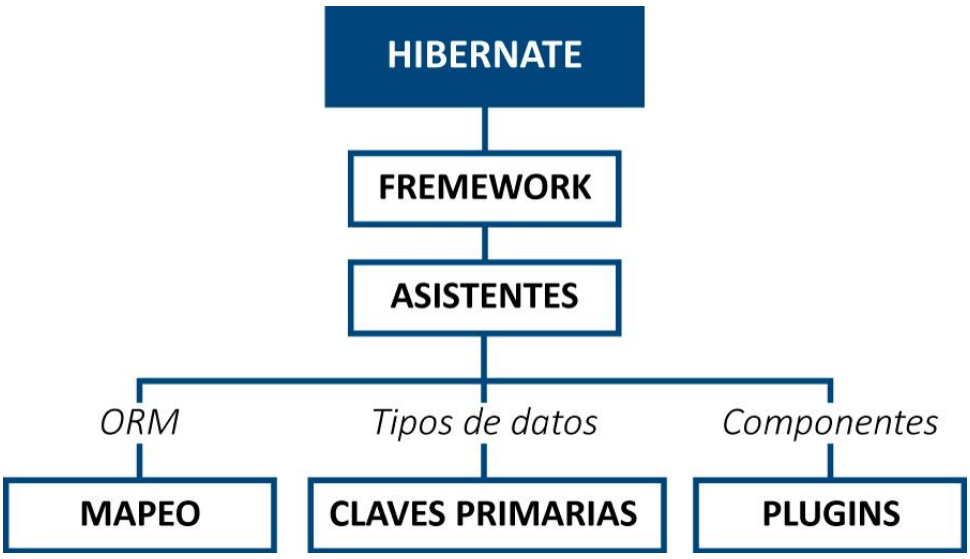
Nombre del taller de aprendizaje: Enviar/Recibir	Datos del autor del taller: Cesar augusto Jaramillo Henao
Escriba o plantee el caso, problema o pregunta: Elabore un aplicativo que permita el envío un saludo a un cliente y este le dé respuesta	
Solución del taller: Aplique todos los conceptos vistos en al unidad	

4.5.2 TALLER DE ENTRENAMIENTO

Nombre del taller: Control de Notas	Modalidad de trabajo: Individual
Actividad previa: Repase los métodos y procesos de envio y recepción de información	
Describa la actividad: Realice un ingreso de notas de un alumno y el cliente debe de tener la opción de consultar y hacer un reclamo sobre la nota optenida.	

5 UNIDAD 5 INTEGRACION CON HIBERNATE

5.1.1 RELACIÓN DE CONCEPTOS



ORM	es un mapeo de objetos relacionales
Claves Primarias	elemento principal de una tabla que no permite que se repita información de identificación
Tipos de datos	elementos que permiten la clasificación de la información.
Asistente	componente que permite realizar procesos complejos de una forma simple
Framework	herramienta que permite que la elaboración de un aplicativo se realice de una manera más simple y controlada

5.1.2 OBJETIVO GENERAL

Introducir al estudiante en nuevas herramientas de desarrollo como el Hibernate, conociendo las bondades de este tipo complemento que posibilita la construcción más rápido de un aplicativo tradicional.

5.1.3 OBJETIVOS ESPECÍFICOS

Conocer las principales características de un framework

Identificar los componentes, sentencias y formas de trabajo con Hibernate

5.2 TEMA 1 CONCEPTOS DE ORM

ORM (Object Relational Mapping), es una técnica de programación para convertir datos entre el lenguaje de programación orientado a objetos y el sistema de base de datos relacional.

El utilizar el ORM, puede proporcionar ciertas ventajas, teniendo en cuenta que es un framework que facilitará el trabajo de los procesos más repetitivos y se podrá invertir un poco más de tiempo a otras áreas del desarrollo.

Ventajas

- Rapidez en el desarrollo.
- Abstracción de la base de datos
- Reutilización
- Seguridad
- Mantenimiento del código
- Lenguaje propio para realizar las consultas.

5.3 TEMA 2 RELACIONES

El mapeo relacional

La ventaja de estos sistemas es la reducción considerable de código necesario para lograr lo que se conoce como persistencia de objetos, esto permite lograr una integración con otros patrones como el Modelo-Vista-Controlador.

En general los sistemas de información guardan datos en BD relacionales como Oracle, mysql, sqlServer, etc, dentro de los procesos más comunes tenemos que un departamento de una empresa tiene varios empleados, pero un empleado pertenece solo a un departamento.

Hibernate resuelve algunos inconvenientes con la representación de un modelo relacional mediante un conjunto de objetos, en este caso los modelos representan tablas y los atributos de las clases son los campos de las tablas.

Para mapear un modelo relacional se pueden utilizar formatos XML o con anotaciones.

5.4 TEMA 3 CLAVES PRIMARIAS Y TIPOS DE DATOS

Dentro de las características del hibernate están sus tipos de datos

integer

long

short

float

double

character

byte

boolean

yes_no

true_false

string

date

time

timestamp

text

binary

big_decimal

big_integer

Muchos de ellos muy conocidos por el trabajo de java otros no tanto y más comunes en este tipo de framework.

Estos datos tienen una clasificación como

Fecha y hora

Date, time y timestamp

Boolean

Yes_no, true_false, Boolean

Texto

String y text

Generacion de claves primarias

Hibernate tiene múltiples formas de tratar las claves primarias, la más simple es cuando el desarrollador indica la clave que tendrá el objeto, este proceso se conoce como “assigned”.

5.5 TEMA 3 HIBERNATE QUERY LANGUAGE

El HQL es el lenguaje de consultas del Hibérnate, este tipo de sentencias tienen algunas características que facilitan el uso de la herramienta, aunque hay que tener presente casos como la sensibilidad de las mayúsculas y minúsculas que en las sentencias como tal no influyen, teniendo presente que puede ser Select, seLect, selecT y no presentaría ningún inconveniente en el trabajo.

Es muy común ver en Hibernate la instrucción from sin procesos previos como se está acostumbrado a otras herramientas lo mismo que las uniones con la instrucción join.

Un ejemplo de este tipo de sentencia es

Query = “from empleado order by nombre”

Dentro de las sentencias Join se encuentran

inner join

left outer join

right outer join

5.6 TEMA 4 OBJETOS Y VALIDACIONES

Las validaciones en cualquier tipo de lenguaje se convierten en elementos fundamentales para un trabajo organizado, en hibernate es común encontrar que las validaciones están asociadas a anotaciones

@NotNull

Esta propiedad indica que no puede estar nulo

@Size(min=n,max=m):

Esta propiedad controla que la información no sea nula y que contenga un mínimo de caracteres y un máximo.

Otras validaciones son

@AssertFalse

@AssertTrue

@Digits(integer=n, fraction=m)

@Future

@Past

@Max(n)

@Min(n)

@NotNull

@Null

@Pattern(regexp="r")

@Size(min=n, max=m).

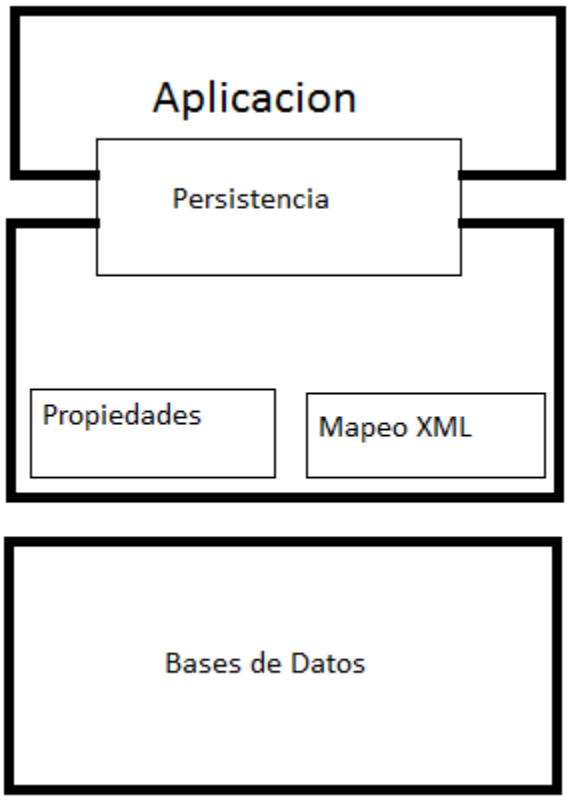
@Email

@NotBlank

@Valid

5.7 TEMA 6 ARQUITECTURA

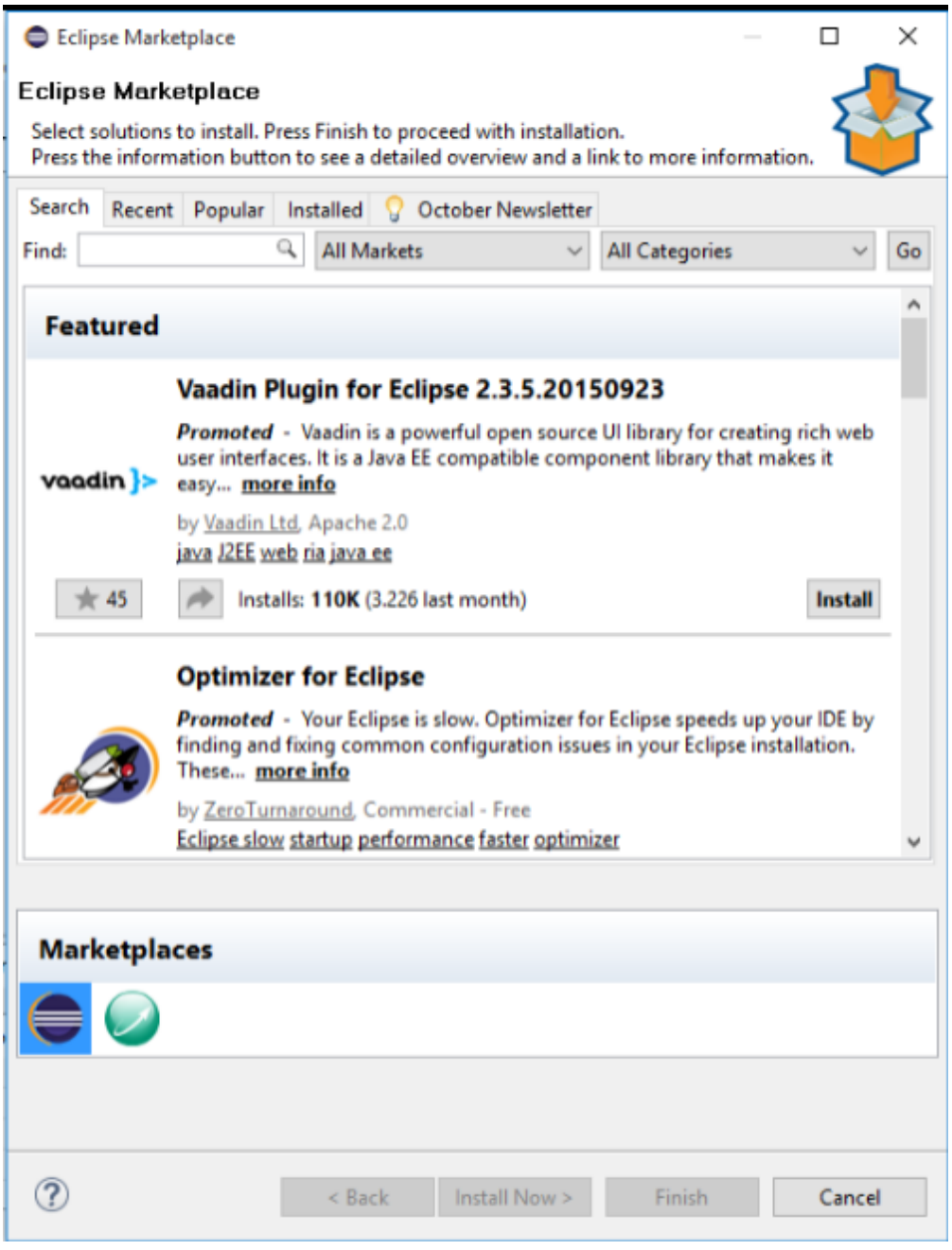
La arquitectura en términos generales del Hibenate es la siguiente



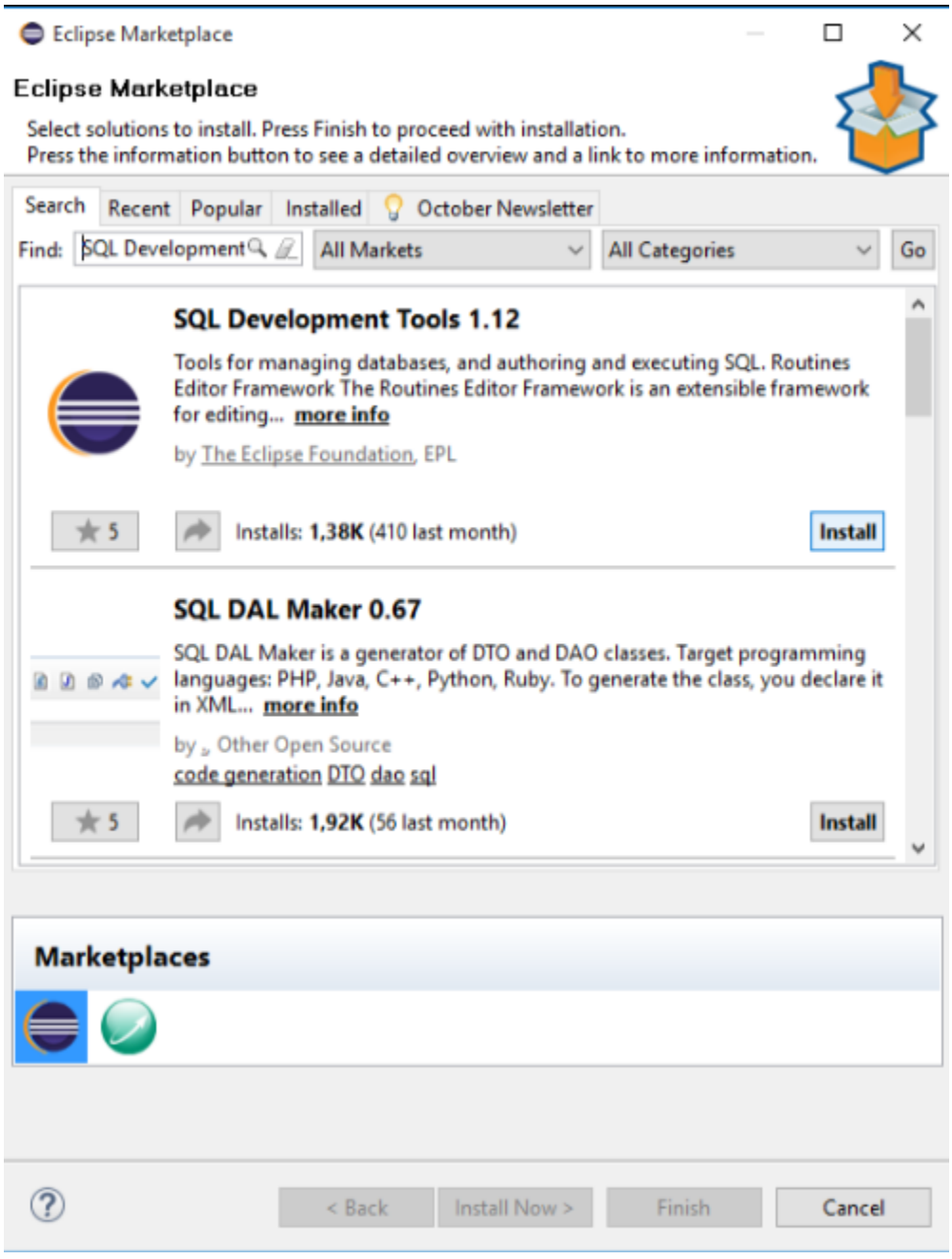
Luego de tener una BD organizada procedemos con la configuración inicial

Después de haber ingresado al eclipse y haber creado un proyecto de la forma tradicional se realiza la siguiente configuración, el proyecto tendrá como nombre biblioteca.

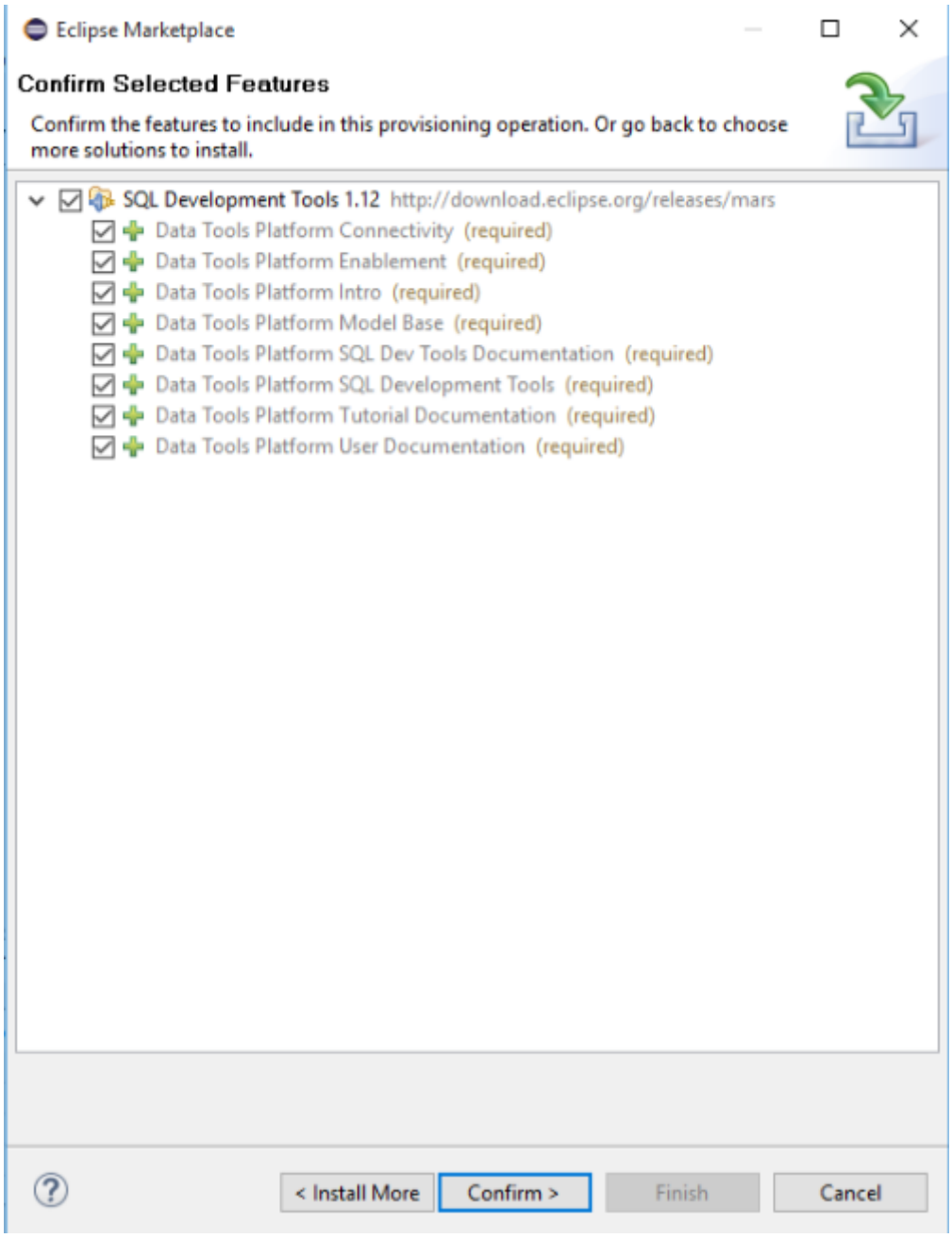
En el menú Help / Eclipse MarketPlace …



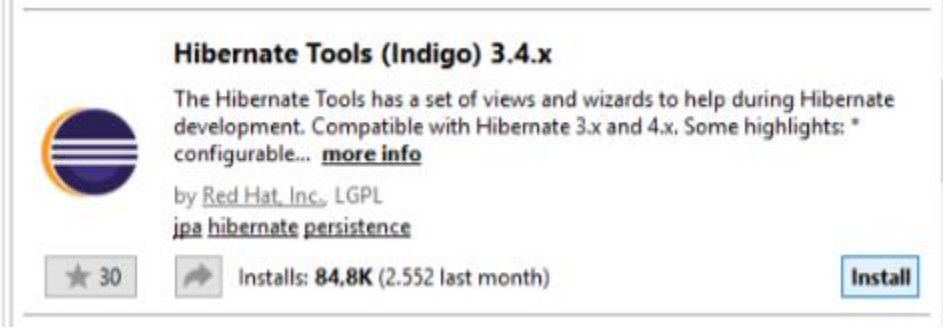
Se procede a buscar la instruccion SQL Development tools



Apereceran una serie de opciones, las cuales confirmaremos



Luego se procede con el proceso de agregar Hibernate y buscamos la informacion de la misma manera en el Help / eclipse marketPlace.



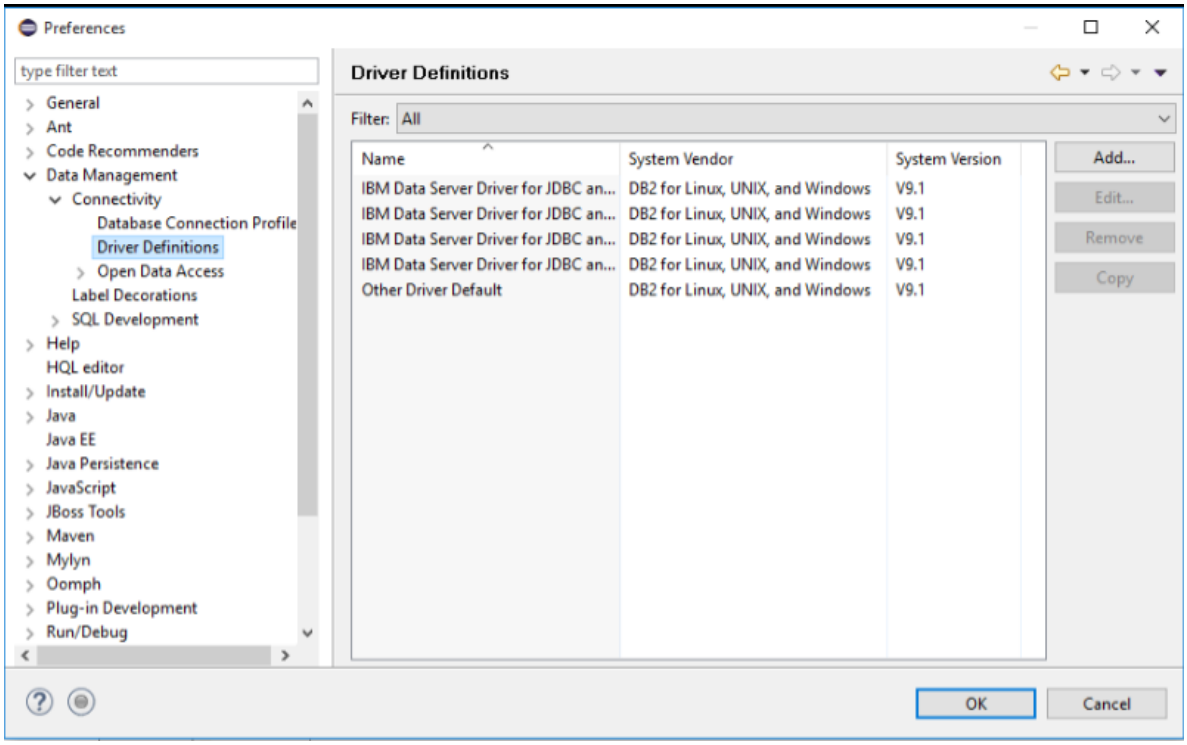
Se aceptan los terminos y se finaliza.

Se buscan las librerías

antlr-2.7.7	4/11/2015 12:02 p....	Executable Jar File	435 KB
c3p0-0.9.2.1	4/11/2015 12:02 p....	Executable Jar File	414 KB
dom4j-1.6.1	4/11/2015 12:02 p....	Executable Jar File	307 KB
ehcache-core-2.4.3	4/11/2015 12:02 p....	Executable Jar File	983 KB
geronimo-jta_1.1_spec-1.1.1	4/11/2015 12:02 p....	Executable Jar File	16 KB
h2-1.4.190	5/11/2015 6:34 a. m.	Executable Jar File	1.669 KB
hibernate	4/11/2015 12:02 p....	Executable Jar File	29 KB
hibernate-c3p0-5.0.2.Final	4/11/2015 12:02 p....	Executable Jar File	41 KB
hibernate-commons-annotations-5.0.0.F...	4/11/2015 12:02 p....	Executable Jar File	74 KB
hibernate-core-5.0.2.Final	4/11/2015 12:02 p....	Executable Jar File	5.421 KB
hibernate-ehcache-5.0.2.Final	4/11/2015 12:02 p....	Executable Jar File	138 KB
hibernate-entitymanager	4/11/2015 12:02 p....	Executable Jar File	577 KB
hibernate-envers	4/11/2015 12:02 p....	Executable Jar File	398 KB
hibernate-infinispan-5.0.2.Final	4/11/2015 12:02 p....	Executable Jar File	144 KB
hibernate-infinispan-5.0.2.Final-tests	4/11/2015 12:02 p....	Executable Jar File	394 KB
hibernate-jpa-2.1-api-1.0.0.Final	4/11/2015 12:02 p....	Executable Jar File	111 KB
hibernate-jpamodelgen	4/11/2015 12:02 p....	Executable Jar File	176 KB
hibernate-osgi	4/11/2015 12:02 p....	Executable Jar File	22 KB
hibernate-proxool-5.0.2.Final	4/11/2015 12:02 p....	Executable Jar File	41 KB
infinispan-commons-7.2.1.Final	4/11/2015 12:02 p....	Executable Jar File	663 KB
infinispan-core-7.2.1.Final	4/11/2015 12:02 p....	Executable Jar File	2.693 KB
jandex-1.2.2.Final	4/11/2015 12:02 p....	Executable Jar File	77 KB
javassist-3.18.1-GA	4/11/2015 12:02 p....	Executable Jar File	698 KB
jboss-logging-3.3.0.Final	4/11/2015 12:02 p....	Executable Jar File	66 KB
jboss-marshalling-osgi-1.4.10.Final	4/11/2015 12:02 p....	Executable Jar File	368 KB
jboss-transaction-api_1.1_spec-1.0.1.Final	4/11/2015 12:02 p....	Executable Jar File	25 KB
jgroups-3.6.2.Final	4/11/2015 12:02 p....	Executable Jar File	2.256 KB
mchange-commons-java-0.2.3.4	4/11/2015 12:02 p....	Executable Jar File	568 KB

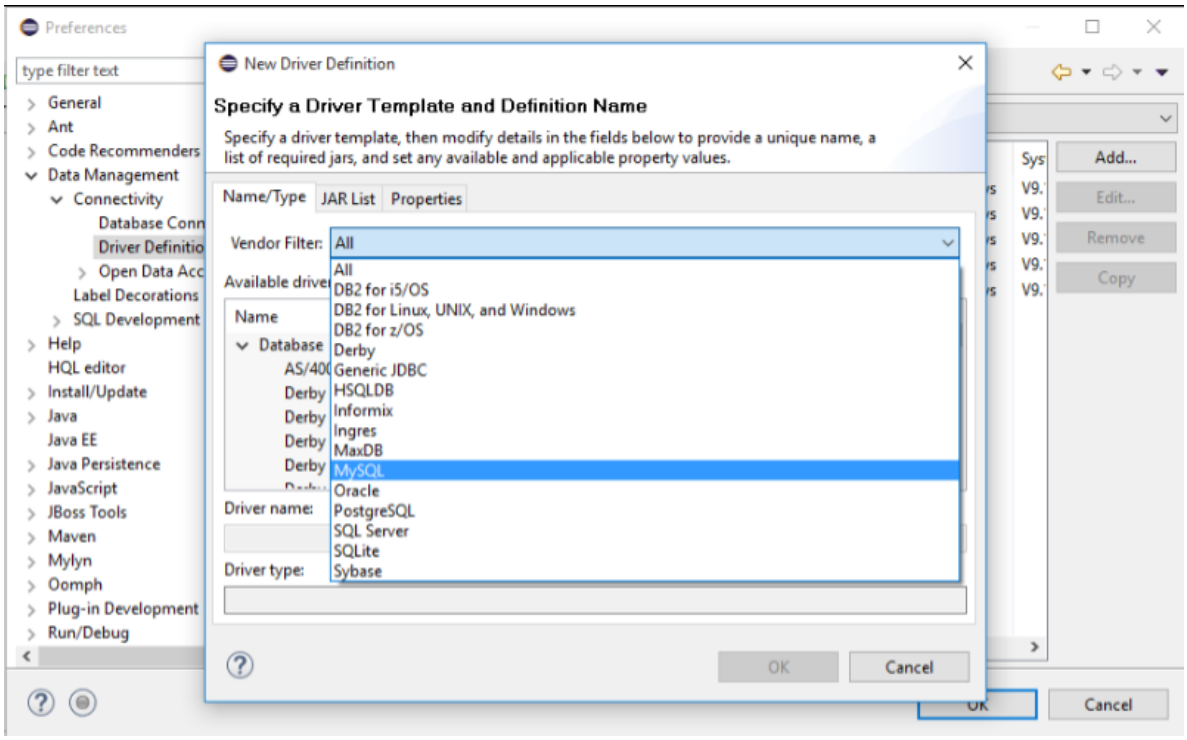
Se copia y se pegan de la siguiente manera, se crea en el proyecto un folder con el nombre de Lib y dentro de esta se pegan las librerias seleccionadas.

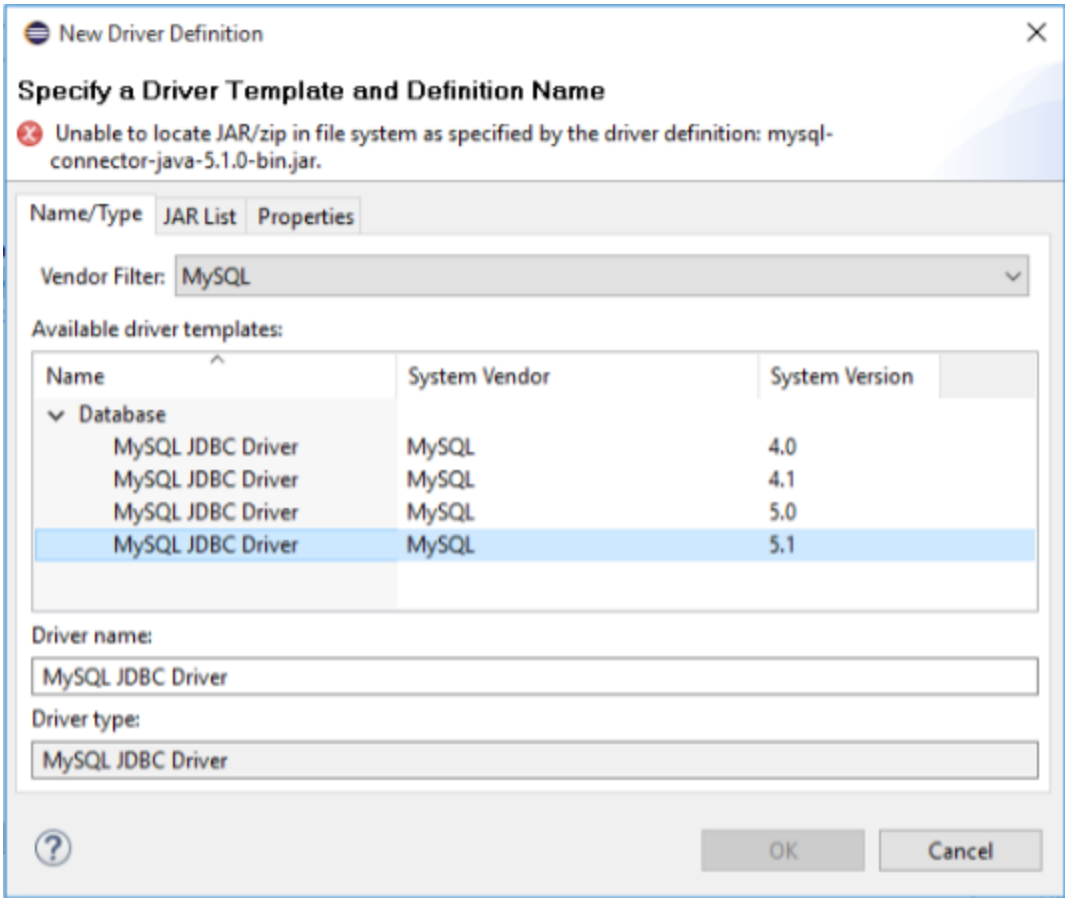
Luego en el menu Windows / Preferences

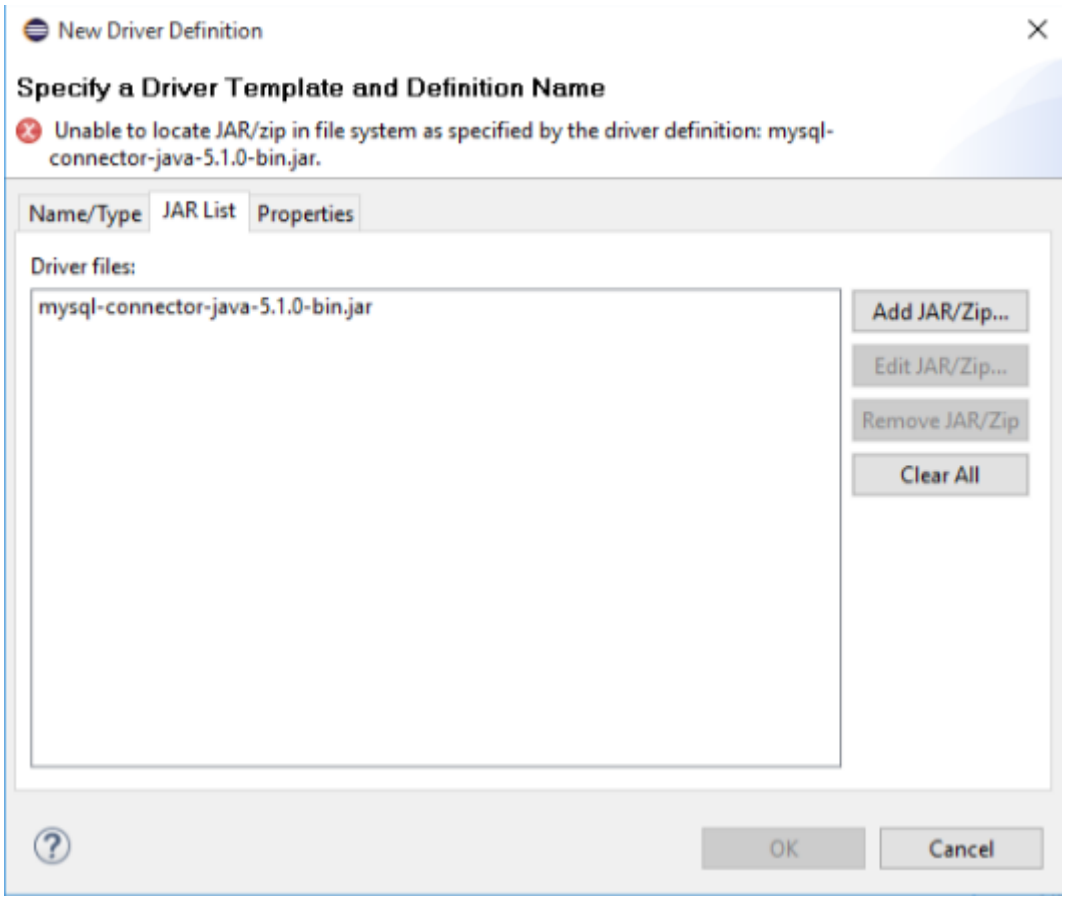


Esta es la configuracion de la conexion para el sistema.

Se selecciona MySQL como herramienta de trabajo


Preferences
type filter text
General
Ant
Code Recommenders
Data Management
Connectivity
Database Conn
Driver Definitio
Open Data Acc
Label Decorations
SQL Development
Help
HQL editor
Install/Update
Java
Java EE
Java Persistence
JavaScript
JBoss Tools
Maven
Mylyn
Oomph
Plug-in Development
Run/Debug
New Driver Definition
Specify a Driver Template and Definition Name
Specify a driver template, then modify details in the fields below to provide a unique name, a list of required jars, and set any available and applicable property values.
Name/Type
JAR List
Properties
Vendor Filter:
All
Available drive
Name
Database
Derby
Driver name:
Driver type:
All
DB2 for i5/OS
DB2 for Linux, UNIX, and Windows
DB2 for z/OS
Derby
Generic JDBC
HSQLDB
Informix
Ingres
MaxDB
MySQL
Oracle
PostgreSQL
SQL Server
SQLite
Sybase
OK
Cancel
Add...
Edit...
Remove
Copy
Cancel

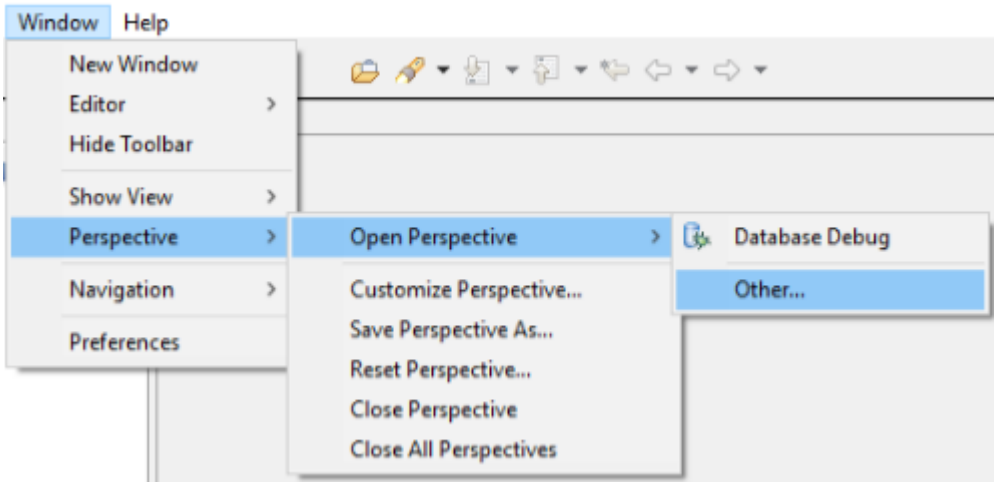

New Driver Definition
Specify a Driver Template and Definition Name
Unable to locate JAR/zip in file system as specified by the driver definition: mysql-connector-java-5.1.0-bin.jar.
Name/Type
JAR List
Properties
Vendor Filter:
MySQL
Available driver templates:
Name
System Vendor
System Version
Database
MySQL JDBC Driver
MySQL
4.0
MySQL JDBC Driver
MySQL
4.1
MySQL JDBC Driver
MySQL
5.0
MySQL JDBC Driver
MySQL
5.1
Driver name:
MySQL JDBC Driver
Driver type:
MySQL JDBC Driver
OK
Cancel

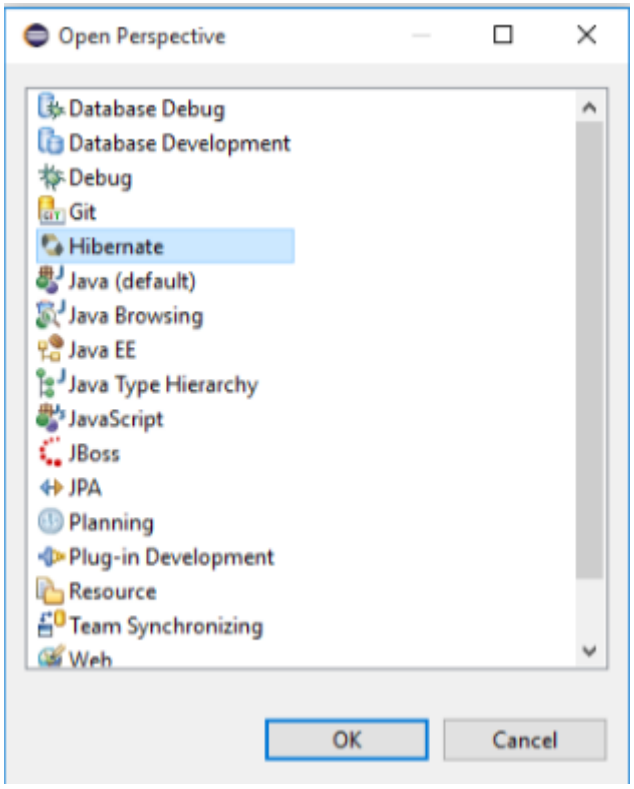


Aca se selecciona el conector que tengamos disponible o lo agregamos si no esta dentro de la lista.

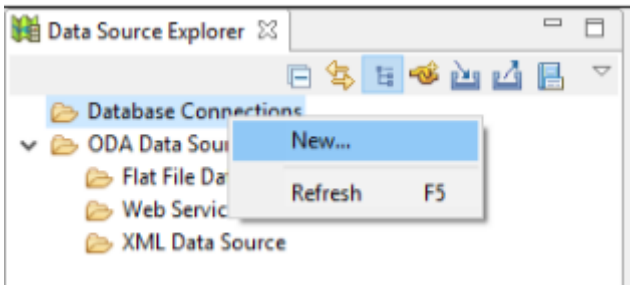
Posterior a esta configuracion basica se continua con las perspecticas

En el menu de windows / perspective

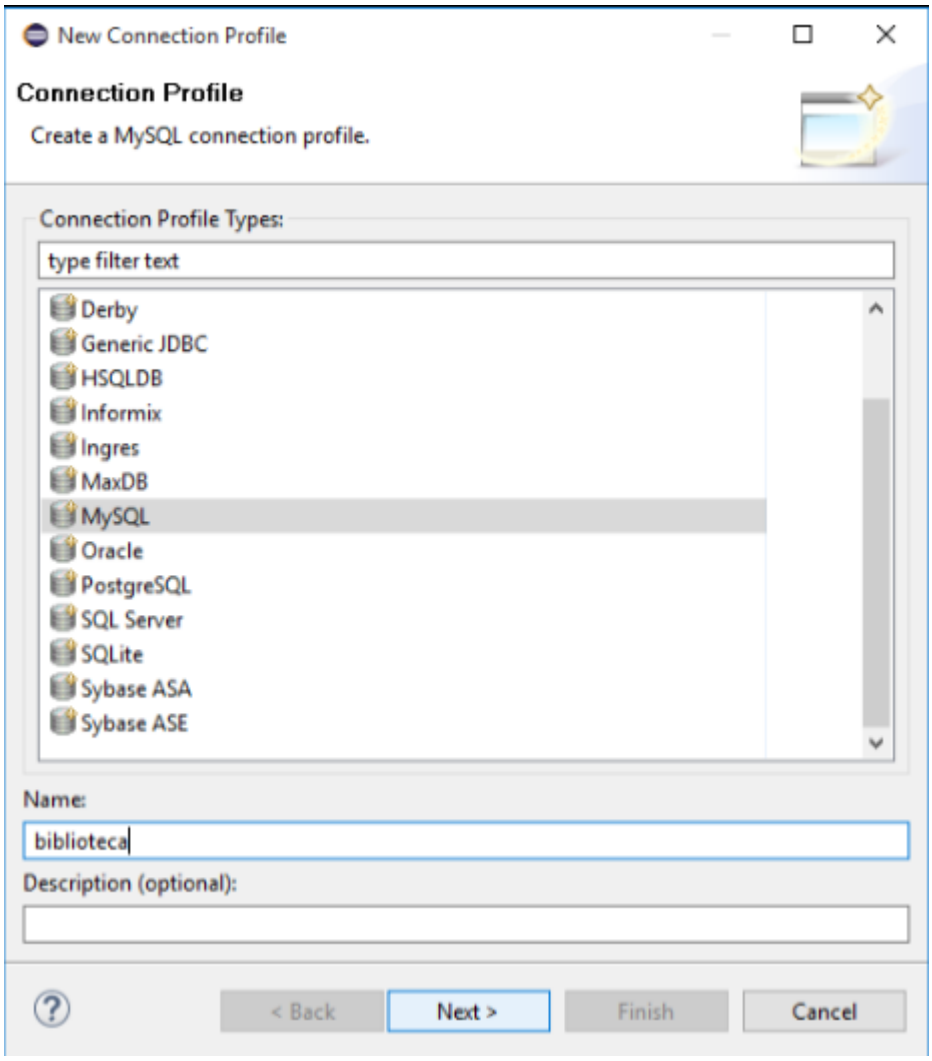


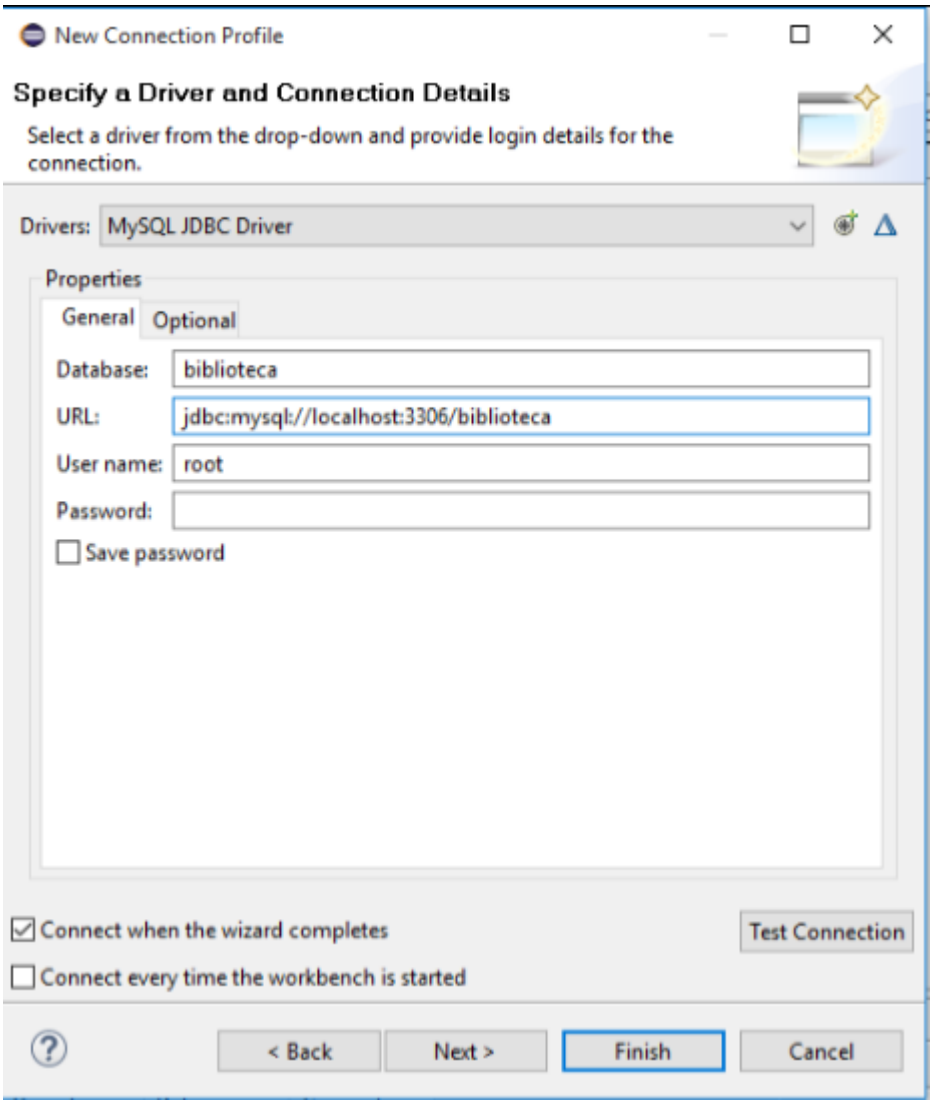


Luego de esto en la estructura del programa encontramos una serie de opciones nuevas

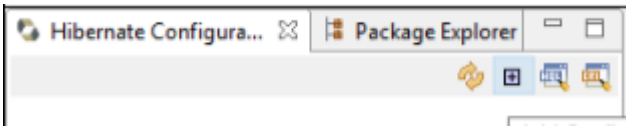


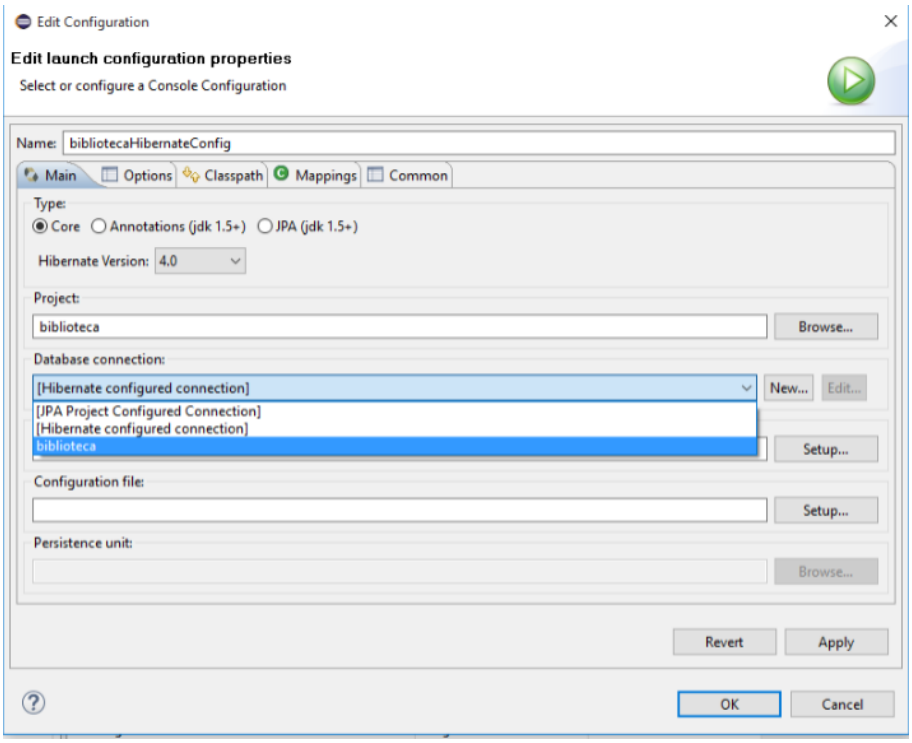
Se elecciona boton emergente en DataBase Connections, se selecciona MySQL

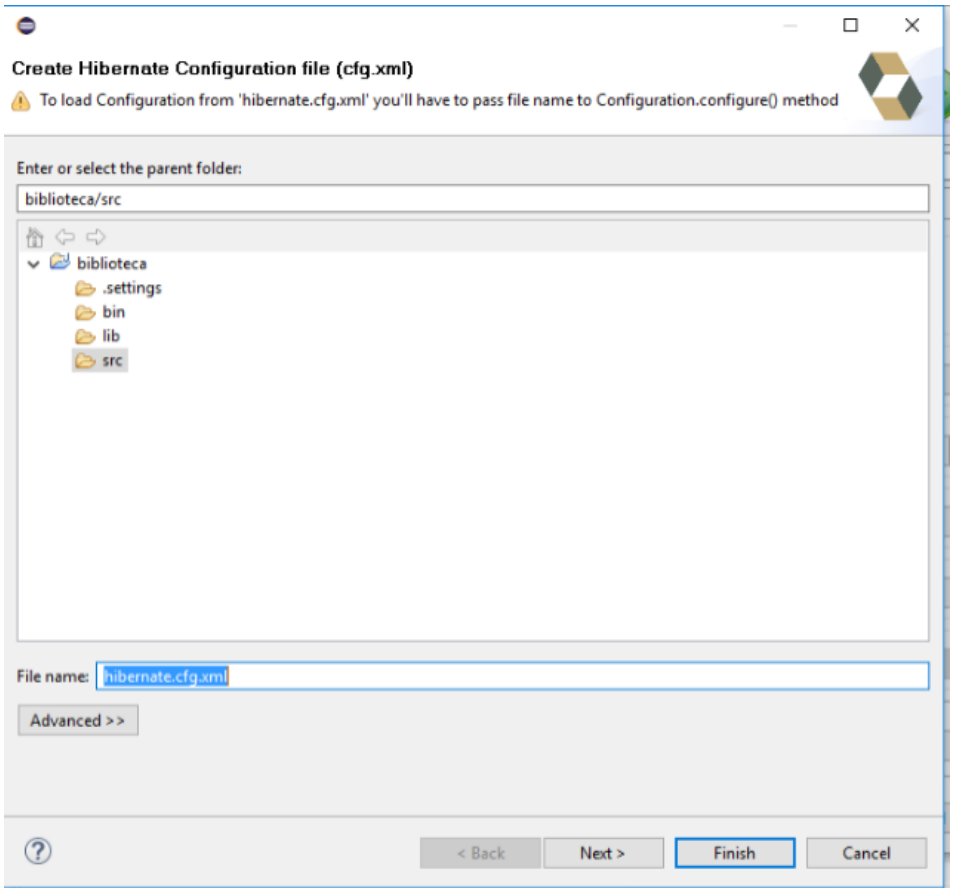

New Connection Profile
Connection Profile
Create a MySQL connection profile.
Connection Profile Types:
type filter text
Derby
Generic JDBC
HSQLDB
Informix
Ingres
MaxDB
MySQL
Oracle
PostgreSQL
SQL Server
SQLite
Sybase ASA
Sybase ASE
Name:
biblioteca
Description (optional):
< Back
Next >
Finish
Cancel

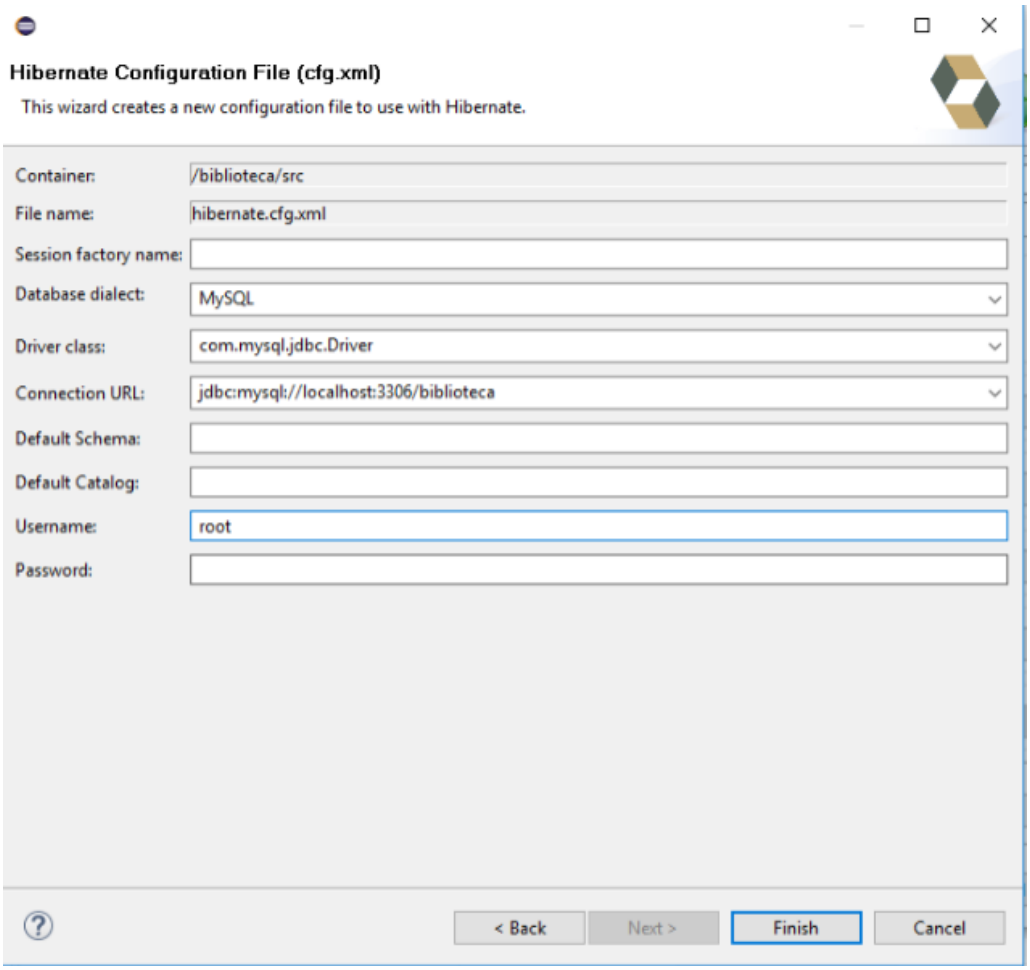


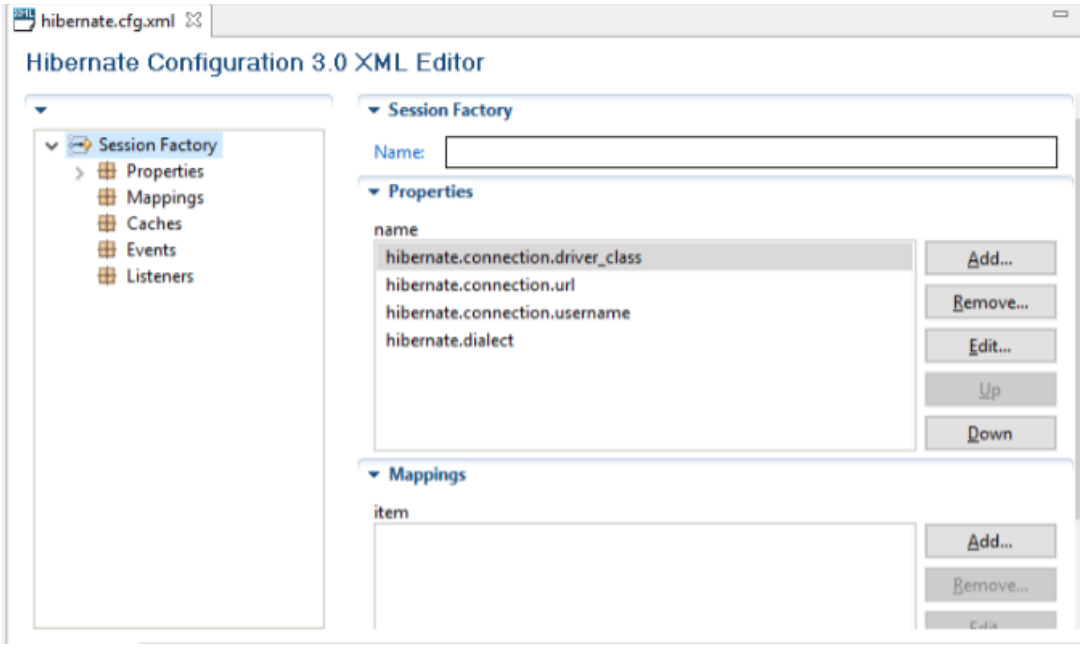
Luego en las opciones de configuracion del eclipse para android realizamos la siguiente tarea




Edit Configuration
Edit launch configuration properties
Select or configure a Console Configuration
Name: bibliotecaHibernateConfig
Main
Options
Classpath
Mappings
Common
Type:
Core
Annotations (jdk 1.5+)
JPA (jdk 1.5+)
Hibernate Version: 4.0
Project:
biblioteca
Browse...
Database connection:
[Hibernate configured connection]
New...
Edit...
[JPA Project Configured Connection]
[Hibernate configured connection]
biblioteca
Setup...
Configuration file:
Setup...
Persistence unit:
Browse...
Revert
Apply
OK
Cancel

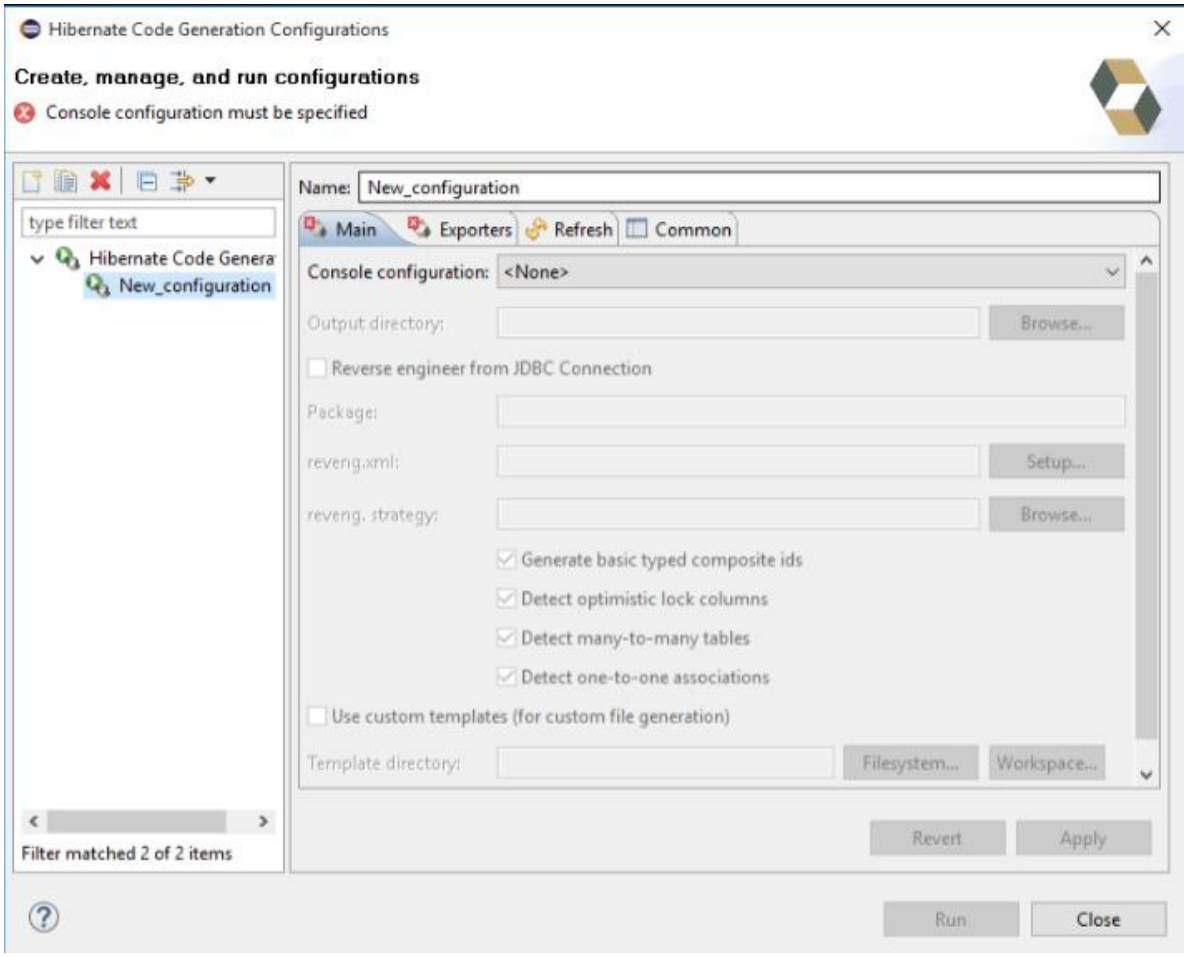

Create Hibernate Configuration file (cfg.xml)
To load Configuration from 'hibernate.cfg.xml' you'll have to pass file name to Configuration.configure() method
Enter or select the parent folder:
biblioteca/src
biblioteca
.settings
bin
lib
src
File name: hibernate.cfg.xml
Advanced >>
< Back
Next >
Finish
Cancel

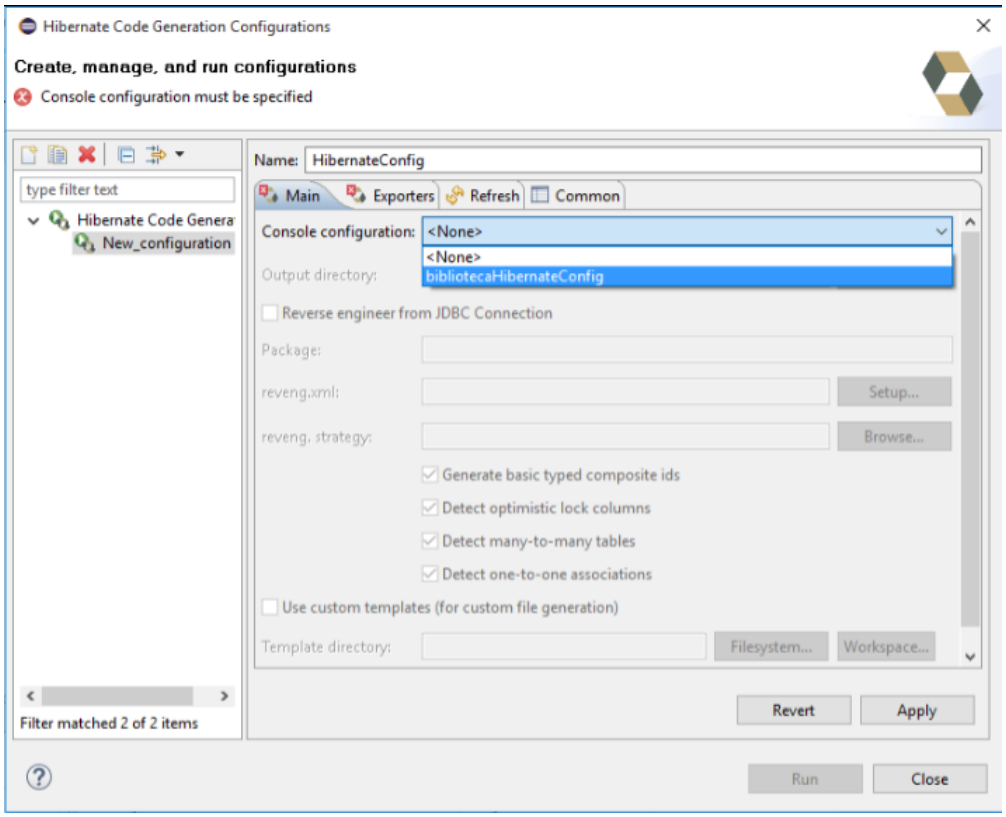

Hibernate Configuration File (cfg.xml)
This wizard creates a new configuration file to use with Hibernate.
Container:
/biblioteca/src
File name:
hibernate.cfg.xml
Session factory name:
Database dialect:
MySQL
Driver class:
com.mysql.jdbc.Driver
Connection URL:
jdbc:mysql://localhost:3306/biblioteca
Default Schema:
Default Catalog:
Username:
root
Password:
< Back
Next >
Finish
Cancel

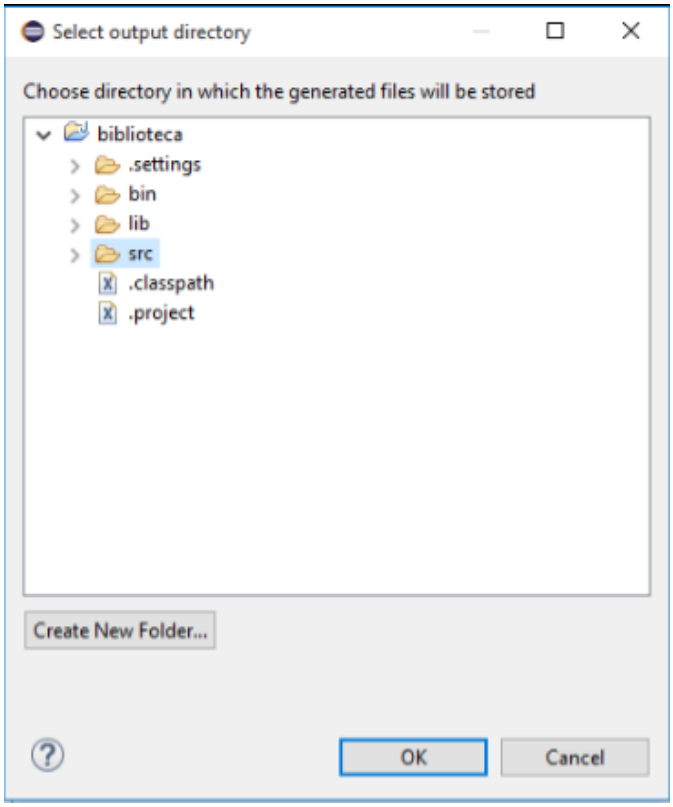

hibernate.cfg.xml
Hibernate Configuration 3.0 XML Editor
Session Factory
Properties
Mappings
Caches
Events
Listeners
Session Factory
Name:
Properties
name
hibernate.connection.driver_class
hibernate.connection.url
hibernate.connection.username
hibernate.dialect
Add...
Remove...
Edit...
Up
Down
Mappings
item
Add...
Remove...

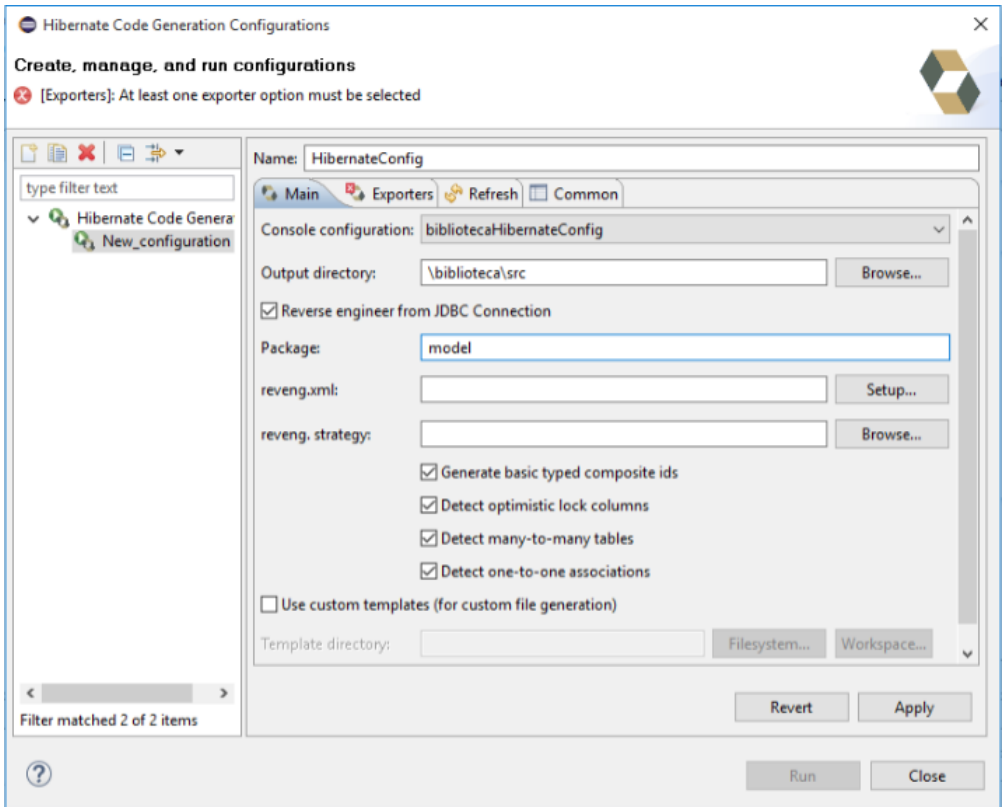
Despues de esta configuracion se procede a la activacion de la generacion de codigo por parte de Hibernate

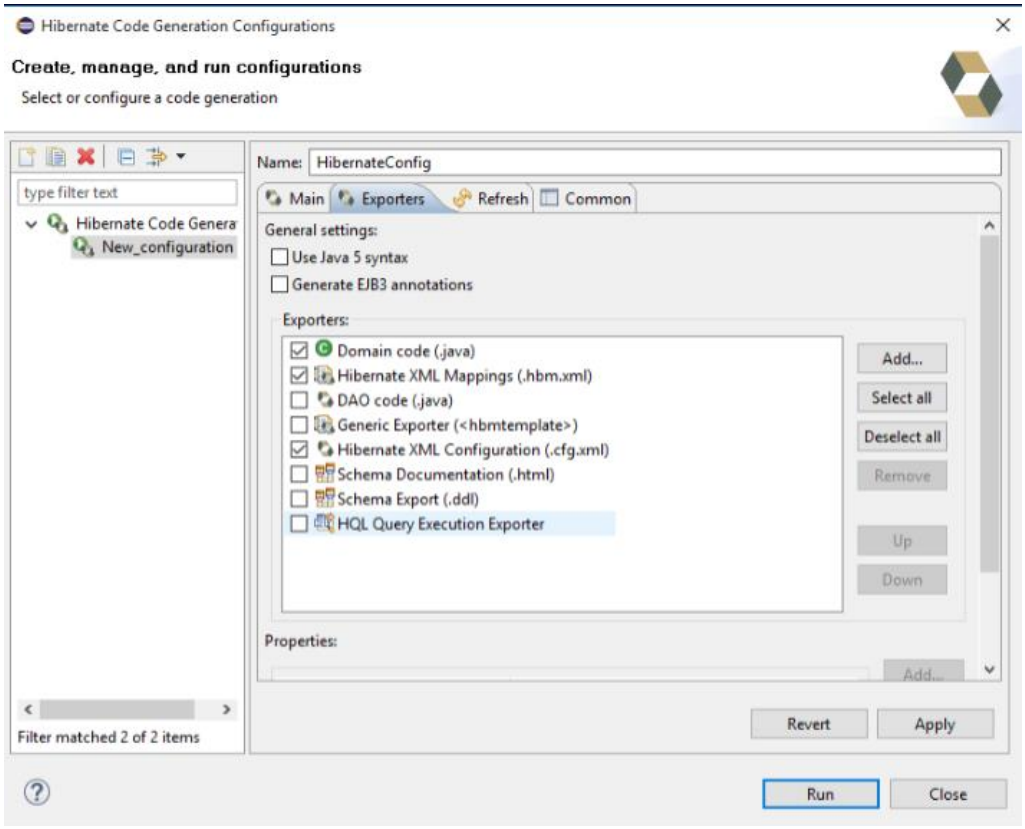
En el menu Run / Hibernate Code Generation / Hibernate Code Generation Configurations…




Hibernate Code Generation Configurations
Create, manage, and run configurations
Console configuration must be specified
type filter text
Hibernate Code Genera
New_configuration
Filter matched 2 of 2 items
Name: HibernateConfig
Main
Exporters
Refresh
Common
Console configuration:
<None>
<None>
bibliotecaHibernateConfig
Output directory:
Reverse engineer from JDBC Connection
Package:
reveng.xml:
Setup...
reveng. strategy:
Browse...
Generate basic typed composite ids
Detect optimistic lock columns
Detect many-to-many tables
Detect one-to-one associations
Use custom templates (for custom file generation)
Template directory:
Filesystem...
Workspace...
Revert
Apply
Run
Close


Select output directory
Choose directory in which the generated files will be stored
biblioteca
.settings
bin
lib
src
.classpath
.project
Create New Folder...
OK
Cancel


Hibernate Code Generation Configurations
Create, manage, and run configurations
[Exporters]: At least one exporter option must be selected
type filter text
Hibernate Code Genera
New_configuration
Filter matched 2 of 2 items
Name: HibernateConfig
Main
Exporters
Refresh
Common
Console configuration: bibliotecaHibernateConfig
Output directory: \biblioteca\src
Browse...
Reverse engineer from JDBC Connection
Package: model
reveng.xml:
Setup...
reveng. strategy:
Browse...
Generate basic typed composite ids
Detect optimistic lock columns
Detect many-to-many tables
Detect one-to-one associations
Use custom templates (for custom file generation)
Template directory:
Filesystem...
Workspace...
Revert
Apply
Run
Close

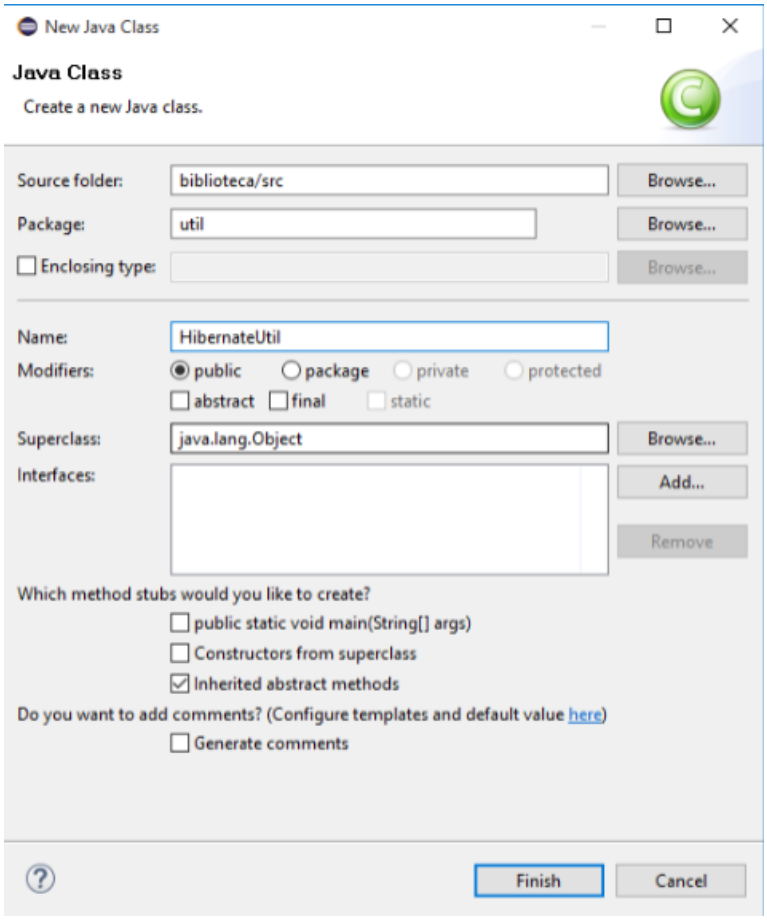

Hibernate Code Generation Configurations
Create, manage, and run configurations
Select or configure a code generation
type filter text
Hibernate Code Genera
New_configuration
Filter matched 2 of 2 items
Name: HibernateConfig
Main
Exporters
Refresh
Common
General settings:
Use Java 5 syntax
Generate EJB3 annotations
Exporters:
Domain code (.java)
Hibernate XML Mappings (.hbm.xml)
DAO code (.java)
Generic Exporter (<hbmtemplate>)
Hibernate XML Configuration (.cfg.xml)
Schema Documentation (.html)
Schema Export (.ddl)
HQL Query Execution Exporter
Add...
Select all
Deselect all
Remove
Up
Down
Properties:
Add...
Revert
Apply
Run
Close

En nuestra BD de biblioteca se agregara una tabla con los campos id, nombre y descripcion

Crearemos una clase Libro.java y se gereral los getters / setters, seguido a esto se crea un archivo libro.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 6/11/2015 04:09:56 PM by Hibernate Tools 4.0.0 -->
<hibernate-mapping>
    <class name="model.Libro" table="libro" catalog="biblioteca">
        <id name="id" type="string">
            <column name="id" length="10" />
            <generator class="assigned" />
        </id>
        <property name="nombre" type="string">
            <column name="nombre" length="25" not-null="true" />
        </property>
        <property name="descripcion" type="string">
            <column name="descripcion" length="45" not-null="true" />
        </property>
    </class>
</hibernate-mapping>
```

Luego de esto se crea un archivo HibernateUtil dentro de un paquete util



```
package util;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        try {
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable e) {
            System.out.println(e.getMessage());
            System.out.println(e.getCause());
            throw new ExceptionInInitializerError(e);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Complementamos con la creacion de una interfaz

New/ interface y la colocaremos facade dentro del paquete modelo y agregamos el siguiente codigo

```
package model;

import java.util.List;

public interface Facade<T> {
    public abstract boolean add(T entity);
    public abstract T find(Object id);
    public abstract boolean update(T entity);
    public abstract boolean delete(T entity);
    public abstract List<T> getAll();
}
```

A continuacion se crea una clase LibroDAO y se crearan todos los metodos a utilizar

```
package model;

import java.util.List;

public class LibroDAO implements Facade<Libro> {

    @Override
    public boolean add(Libro entity) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public Libro find(Object id) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public boolean update(Libro entity) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean delete(Libro entity) {
        // TODO Auto-generated method stub
        return false;
    }
```

```
@Override
public boolean add(Libro entity) {
    Session session = null;
    Transaction transaction = null;
    boolean resp;
    try {
        session = HibernateUtil.getSessionFactory().openSession();
        transaction = session.beginTransaction();
        transaction.begin();
        session.save(entity);
        transaction.commit();
        resp = true;
    } catch(HibernateException e) {
        transaction.rollback();
        resp = false;
    } finally {
        try {
            if(session != null) session.close();
        } catch(HibernateException e) {
            System.out.println(e.getMessage());
            System.out.println(e.getCause());
        }
    }
    return resp;
}
```

```
    @Override
    public Libro find(Object id) {
        Session session = null;
        Transaction transaction = null;
        Libro libro = null;
        try {
            session = HibernateUtil.getSessionFactory().openSession();
            transaction = session.beginTransaction();
            transaction.begin();
            libro = (Libro)session.get(Libro.class, (Serializable) id);
            transaction.commit();
        } catch(HibernateException e) {
            transaction.rollback();
        } finally {
            try {
                if(session != null) session.close();
            } catch(HibernateException e) {
                System.out.println(e.getMessage());
                System.out.println(e.getCause());
            }
        }
        return libro;
    }
```

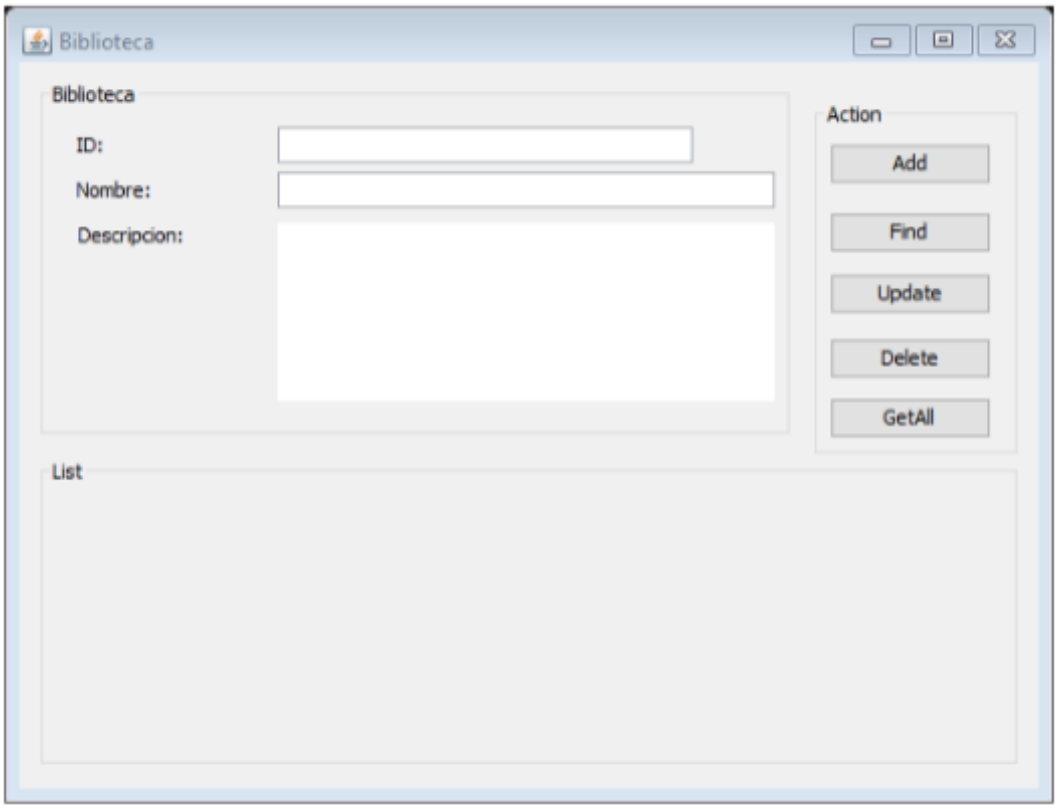
```
@Override
public boolean update(Libro entity) {
    Session session = null;
    Transaction transaction = null;
    boolean resp;
    try {
        session = HibernateUtil.getSessionFactory().openSession();
        transaction = session.beginTransaction();
        transaction.begin();
        session.update(entity);
        transaction.commit();
        resp = true;
    } catch(HibernateException e) {
        transaction.rollback();
        resp = false;
    } finally {
        try {
            if(session != null) session.close();
        } catch(HibernateException e) {
            System.out.println(e.getMessage());
            System.out.println(e.getCause());
        }
    }
    return resp;
}
```

```
    @Override
    public boolean delete(Libro entity) {
        Session session = null;
        Transaction transaction = null;
        boolean resp;
        try {
            session = HibernateUtil.getSessionFactory().openSession();
            transaction = session.beginTransaction();
            transaction.begin();
            session.delete(entity);
            transaction.commit();
            resp = true;
        } catch(HibernateException e) {
            transaction.rollback();
            resp = false;
        } finally {
            try {
                if(session != null) session.close();
            } catch(HibernateException e) {
                System.out.println(e.getMessage());
                System.out.println(e.getCause());
            }
        }
        return resp;
    }

@Override
public List<Libro> getAll() {
    Session session = null;
    Transaction transaction = null;
    List<Libro> list = null;
    try {
        session = HibernateUtil.getSessionFactory().openSession();
        transaction = session.beginTransaction();
        transaction.begin();
        list = (List<Libro>) session.createQuery("From Libro").list();
        transaction.commit();
    } catch(HibernateException e) {
        transaction.rollback();
    } finally {
        try {
            if(session != null) session.close();
        } catch(HibernateException e) {
            System.out.println(e.getMessage());
            System.out.println(e.getCause());
        }
    }
    return list;
}
```

Luego se crea una clase Controller.java

```
package controller;

import java.util.List;
import model.Libro;
import model.LibroDAO;

public class LibroController {
    private Libro libro;
    private LibroDAO libroDAO;

    public LibroController() {
        libro = new Libro();
        libroDAO = new LibroDAO();
    }
    public Libro get() {
        return libro;
    }
    public boolean add() {
        return libroDAO.add(libro);
    }
    public Libro find() {
        libro = libroDAO.find(libro.getId());
        return libro;
    }
    public boolean update() {
        return libroDAO.update(libro);
    }
    public boolean delete() {
        return libroDAO.delete(libro);
    }
    public List<Libro> getAll() {
        return libroDAO.getAll();
    }
```

Posterior a este archivo se creara el paquete vista y la clase FrmLibro



Programacion de los botones

```
JButton btnAdd = new JButton("Add");
btnAdd.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        capturar();
        if(libroController.find() == null) {
            if(libroController.add())
                JOptionPane.showMessageDialog(null, "La informacion se ha agregado correctamente",
                        "Biblioteca", JOptionPane.INFORMATION_MESSAGE);
            else
                JOptionPane.showMessageDialog(null, "La informacion no se ha podido guardar",
                        "Biblioteca", JOptionPane.ERROR_MESSAGE);;
        } else
            JOptionPane.showMessageDialog(null, "La informacion ya existe",
                    "Biblioteca", JOptionPane.INFORMATION_MESSAGE);;
    }
});
```

```
JButton btnFind = new JButton("Find");
btnFind.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        libroController.get().setId(jtxtID.getText());
        if(libroController.find() != null) {
            jtxtID.setText(libroController.get().getId());
            jtxtNombre.setText(libroController.get().getNombre());
            jtaDescripcion.setText(libroController.get().getDescripcion());
        }
    }
});
```

```
JButton btnUpdate = new JButton("Update");
btnUpdate.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        libroController.get().setId(jtxtID.getText());
        libroController.get().setNombre(jtxtNombre.getText());
        libroController.get().setDescripcion(jtaDescripcion.getText());
        if(libroController.update())
            JOptionPane.showMessageDialog(null, "La informacion se ha actualizado correctamente",
                    "Biblioteca", JOptionPane.INFORMATION_MESSAGE);
        else
            JOptionPane.showMessageDialog(null, "La informacion no se ha podido actualizar",
                    "Biblioteca", JOptionPane.ERROR_MESSAGE);;
    }
});

JButton btnDelete = new JButton("Delete");
btnDelete.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        libroController.get().setId(jtxtID.getText());
        libroController.get().setNombre(jtxtNombre.getText());
        libroController.get().setDescripcion(jtaDescripcion.getText());
        if(libroController.delete())
            JOptionPane.showMessageDialog(null, "La informacion se ha eliminado correctamente",
                    "Biblioteca", JOptionPane.INFORMATION_MESSAGE);
        else
            JOptionPane.showMessageDialog(null, "La informacion no se ha podido eliminar",
                    "Biblioteca", JOptionPane.ERROR_MESSAGE);
    }
});
```

Con estos procesos el ejemplo quedaria funcional.

5.7.1 EJERICICIO DE APRENDIZAJE

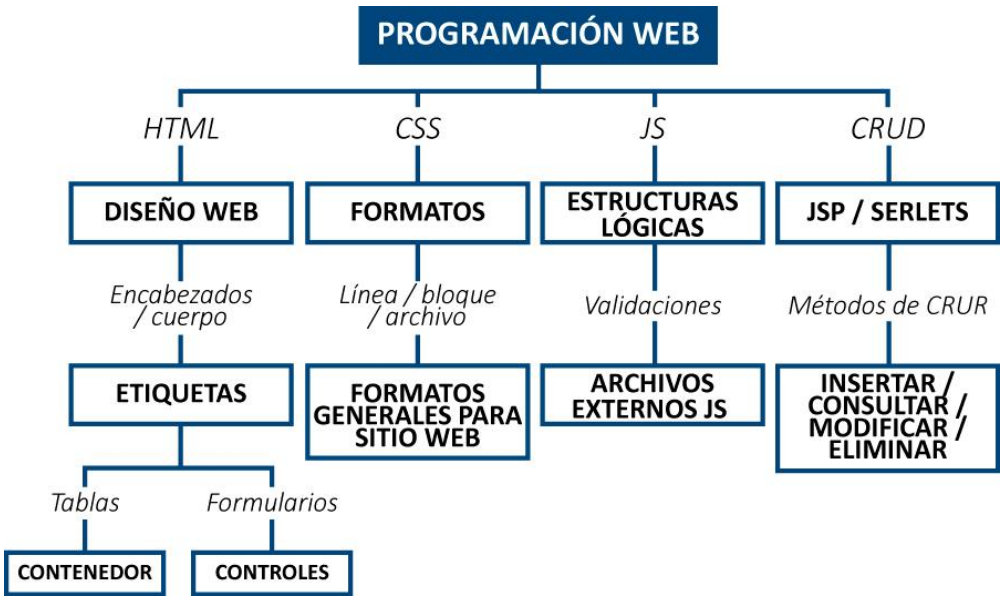
Nombre del taller de aprendizaje: Nomina	Datos del autor del taller: Cesar Augusto Jaramillo Henao
Escriba o plantee el caso, problema o pregunta: Realice un proceso de cálculo de nómina mediante procesos básicos de hibernate	
Solución del taller: Utilice las herramientas del trabajo realizado en este capitulo	

5.7.2 TALLER DE ENTRENAMIENTO

Nombre del taller: banco	Modalidad de trabajo: Individual
Actividad previa: Realice el trabajo conformado por Nomina	
Describa la actividad: Diseñar un programa en hibernate que cumpla las condiciones mínimas de un banco	

6 UNIDAD 5 INTRODUCCION A LA PROGRAMACION WEB

6.1.1 RELACIÓN DE CONCEPTOS



Contenedor herramienta que puede contener otros controles, ejemplo de tablas y formularios

Etiquetas "comandos" de HTML

Controles componentes de un ambiente de programación, cajas de texto, botones, combos, etc.

Formatos sentencias que permiten dar presentación, estilos y diseño a un sitio web

Estructuras son componentes de un lenguaje de programación tales como ciclos, preguntas, selectores y preguntas

CRUD descripción de Crear, Leer, actualizar y Eliminar información.

6.1.2 OBJETIVO GENERAL

Aprender los conceptos básicos de la programación web, las etiquetas básicas, los formatos y las validaciones, así como la construcción de un CRUD

6.1.3 OBJETIVOS ESPECÍFICOS

Conocer las principales características del HTML en su etapa de diseño para un CRUD

Aplicar formatos que le den un aspecto menos plano del que se trabaja habitualmente en HTML estándar mediante las herramientas de CSS

Aplicar las validaciones necesarias para controlar el ingreso de la información dentro un formulario HTML

Elaborar un CRUD mediante JSP y Servlets

6.2 TEMA 1 HTML / HTML5

HTML

El HTML (Hyper Text Markup Language) es el lenguaje con el que se escriben las páginas o estructuras web, un lenguaje que permite colocar texto de forma estructurada, y que está compuesto por etiquetas, también conocidas como tags o marcas, que indican el inicio y el fin de cada elemento del documento.

Un documento de hipertexto no sólo se compone de texto, puede contener sonido, vídeos, imágenes y otros elementos dinámicos, por lo que el resultado puede considerarse como un documento multimedia.

Los documentos HTML deben tener la extensión HTML o HTM, para que puedan ser visualizados en los navegadores web (Browser), sean estos los más comunes como Internet Explorer, Chrome, Mozilla, Safari, Opera, entre otros.

Los browsers se encargan de interpretar el código HTML de los documentos, y de mostrar a los usuarios las páginas web resultantes del código interpretado.

Estructura Basica

```
<html>
        <head>
...
        </head>
        <body>
...
        </body>
</html>
```

La gran mayoría de las etiquetas están compuestas por una apertura y un cerrado <html> </html>, la etiqueta que contiene el símbolo slash (/) es la que indica el cerrado, otras etiquetas no se componen por pares y se cierran al final de ella,
 esta es un típico caso.

Dentro de las páginas web existe una estructura como la vista al principio, la etiqueta <html> y </html> son la primera y la ultima de la página, es la etiqueta que enmarca lo que vamos a realizar, dentro de estas etiquetas se ubicaran dos áreas, la cabecera (head) y el cuerpo de la pagina (body).

Cabecera

<head>…</head>

Esta etiqueta alberga el título de la página y permite la invocación de otros elementos como los scripts y las hojas de estilo en cascada, elementos que se verán más adelante.

<title> primera página web </title>

<html>

<head>

<title>primera pagina web </title>

</head>

Cuerpo de la página

<body>…</body>

El cuerpo de la página alberga todo el contenido que se visualizará por parte del usuario, además el <body> podrá tener elementos como muchas otras etiquetas llamados parámetros, estos parámetros permiten darle un diseño o formato adicional

bgcolor=”color de fondo”, este se puede especificar de varias formas, el nombre del color como red, Green, yellow, etc, o se puede trabajar con un formato hexadecimal que nos da una combinación de más 16 millones de colores, este formato se representa asi #RRVVAA (Rojo, Verde, Azul), los valores que se utilizan para este caso son números de 0 a 9 y de A a F, en los formatos tradiciones se componen por parejas, las dos RR representan el rojo, GG verde y BB azul, de acá saldrán los 16 millones de colores, #FF0000 nos arroja rojo, #00FF00, verde y #0000FF azul

Background=”imagen de fondo”, para el manejo de fondos se podrá usar cualquier formato de imagen como JPG, PNG, GIF, tenga presente el tamaño y la resolución para hacer más agradable es espacio web.

Background=”fondo.jpg”

Comentarios en HTML

Con mucha frecuencia se requiere hacer comentarios o anular partes del código creado, para esto se utiliza una etiqueta que inhabilita esta área de trabajo

<!- - comentario //-->

Saltos de Línea

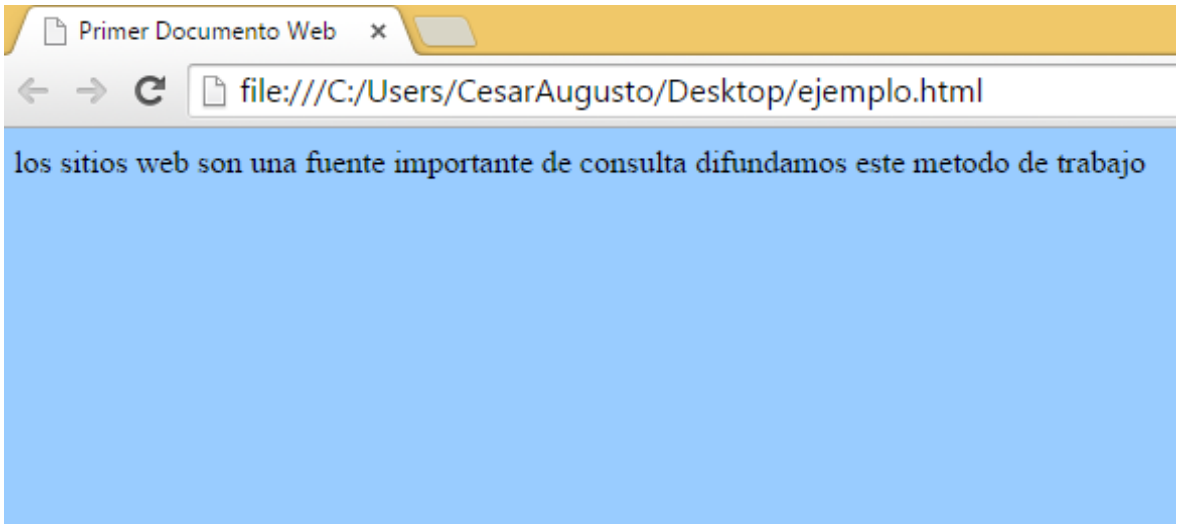
Es el equivalente a un enter, en HTML lo enter que especifiquemos presionando la tecla o la barra espaciadora no se vera al ejecutar la pagina para esto existe un grupo especifico de etiquetas que presentan estos caracteres

 representara este carácter

Ejemplo de representación

```
<html>
    <head>
        <title>Primer Documento Web</title>
    </head>

    <body bgcolor="#99CCFF">
        los sitios web son una fuente importante de consulta









        difundamos este metodo de trabajo
    </body>
</html>
```

Un código con las etiquetas básicas y da como resultado



Se observa el título “Primer Documento Web”, además en el cuerpo se ve el fondo azul que se especificó y el texto, pero se puede ver que el texto aparece en la misma línea y en el archivo el texto está separado por espacios, acá entra el funcionamiento de la etiqueta

```
<html>
    <head>
        <title>Primer Documento Web</title>
    </head>

    <body bgcolor="#99CCFF">
        los sitios web son una fuente importante de consulta
        <br />
        <br />
        <br />
        <br />
        difundamos este metodo de trabajo
    </body>
</html>
```

Resultado

los sitios web son una fuente importante de consulta

difundamos este metodo de trabajo

Link

Una de las razones principales de un sitio web es el manejo de los vínculos o links, con esta herramienta se podrán realizar comunicaciones o llamados con otras páginas o con otros sitios

La etiqueta <a> es la encargada de realizar esta tarea, se acompaña de múltiples parámetros, pero existe uno fundamental que es href que indica la dirección o ruta donde se encuentra el archivo o el sitio web a visitar

```
<html>
    <head>
        <title>Primer Documento Web</title>
    </head>

    <body bgcolor="#99CCFF">
        los sitios web son una fuente importante de consulta
        <br />
        <br />
        <br />
        <br />
        difundamos este metodo de trabajo
        <br />

        <a href="http://www.enter.co">Revista Enter</a>
    </body>
</html>
```

los sitios web son una fuente importante de consulta

difundamos este metodo de trabajo
Revista Enter

El hipervínculo mostrara la palabra “Revista Enter”, pero el llamado es , lo que se ubica en el href es una ruta o url, después de él se ubica una descripción del texto a llamar y cierra con la etiqueta

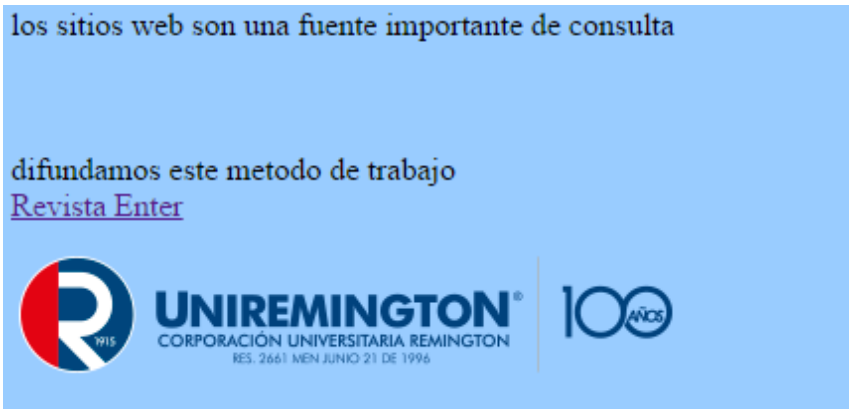
Imágenes

El diseño de un sitio web no se limita solo a el texto, los colores o los hipervínculos, las imagines hace parte fundamental de la presentación y de acercarse a las imágenes corporativas de las empresas.

La etiqueta que se utiliza es <img />

Se utiliza como parámetro fundamental src (source o ruta del archivo)

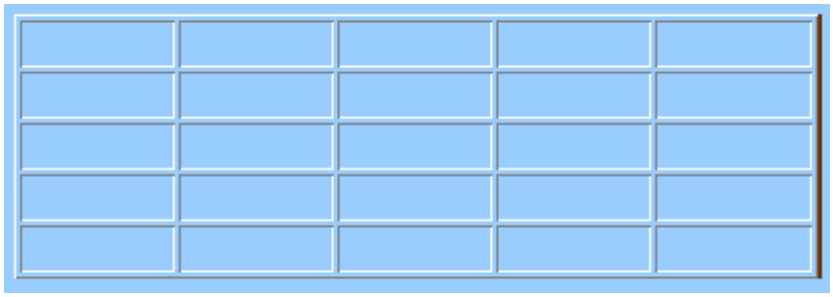
<img src="logo.png">



Estas imágenes se les pueden agregar bordes, se pueden convertir en hipervínculos.

Tablas

Las tablas son contenedores, son herramientas que permiten realizar distribución de los elementos y dentro de ellos ubicar texto, imágenes, hipervínculos y otros elementos incluyendo tablas anidadas



Este un caso típico de una tabla compuesto por 5 filas y 5 columnas, es una matriz

Para la construcción de ella se requiere de otras etiquetas como son

<tr> </tr> establece el inicio y fin de una fila

<td></td> establece las celdas de la fila

```
<table width="400" border="1">
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
</tr>
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
</tr>
</table>
```

Dentro de los parámetros más comunes están width (ancho) y border (grosor del borde)

Existe un carácter especial entre cada <td> </td> este carácter representa un espacio, en este código se mostrarán dos filas y 5 columnas, en el carácter especial hay que tener presente que existen 256 caracteres con este formato, algunos de los que son importantes representan las tildes y caracteres especiales que los browsers no reconocen y que muestran un símbolo que dañaría el formato original.

Existen algunos parámetros adicionales dentro de los <tr> y los <td>, entre ellos la posibilidad de colocarle formatos como colores e imágenes de fondo y la posibilidad de cambiar filas o columnas

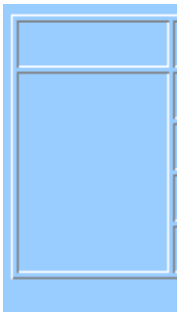
Combinar columnas

Para la combinación de columnas se utiliza el parámetro colspan y el número de columnas

```
<table width="400" border="1">
  <tr>
    <td> </td>
    <td colspan="4"> </td>
  </tr>
  <tr>
    <td rowspan="4"> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
</tr>
```

Combinación de Filas

La combinación de columnas se realiza con la sentencia rowspan y el número de filas



```
<table width="400" border="1">
  <tr>
    <td> </td>
    <td colspan="4"> </td>
  </tr>
  <tr>
    <td rowspan="4"> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
</tr>
```

Formularios

Los formularios son herramientas que permiten la interacción con el usuario, mediante estos se podrán solicitar datos, hacer cálculos y demás operaciones

Se conforma por la etiqueta <form></form>

Igual que las tablas es un contenedor, puede llevar distintos elementos como cajas de texto, botones, listas, etc., y contiene una serie de parámetros como son

Id	nombre para identificar el formulario
Name	nombre para identificar el formulario
Action	especifica el archivo o la función que se realizara a la hora de enviar los datos del formulario
Method	representa la forma de paso de la información, existen dos opciones

tradicionales, POST y GET

elementos de los formularios

cajas de texto

áreas de texto

botones de comando

botones de radio (botones de opción)

cajas de chequeo (casilla de verificación)

lista / menú (comboBox)

entre otros.

estructura

cajas de texto

```
<input name="caja" type="text" id="caja" size="20" maxlength="10" />
```

Se crea mediante la etiqueta input como muchos de los elementos de entrada de información, pero se especifica mediante el parámetro type que es un text, id y el name (nombre) permiten la identificación de la caja, size es el ancho que se ve y maxlength la cantidad de caracteres máximos que se pueden ingresar.

áreas de texto

```
<textarea name="comentario" id="comentario" cols="45" rows="5"></textarea>
```

Las áreas de texto son espacios mucho más amplios que las cajas de texto, se compone por id y name para identificarlas, cols para el número máximo de columnas que se mostraran y rows para el número máximo de filas visibles

botones de comando

```
<input type="submit" name="button" id="button" value="Almacenar">
```

```
<input type="reset" name="button2" id="button2" value="Restablecer">
```

Dos de los tipos de botones más comunes son los de envío y los de restablecer, igual de los demás elementos contienes un id y name, value para mostrar al usuario un resultado y type para determinar que elemento es, en este caso un submit para el envío y reset para limpiar los elementos del formulario.

botones de radio (botones de opción)

<input type="radio" name="radio" id="radio" value="radio">

Los botones de radio o de opción permiten seleccionar una de muchas opciones

cajas de chequeo (casilla de verificación)

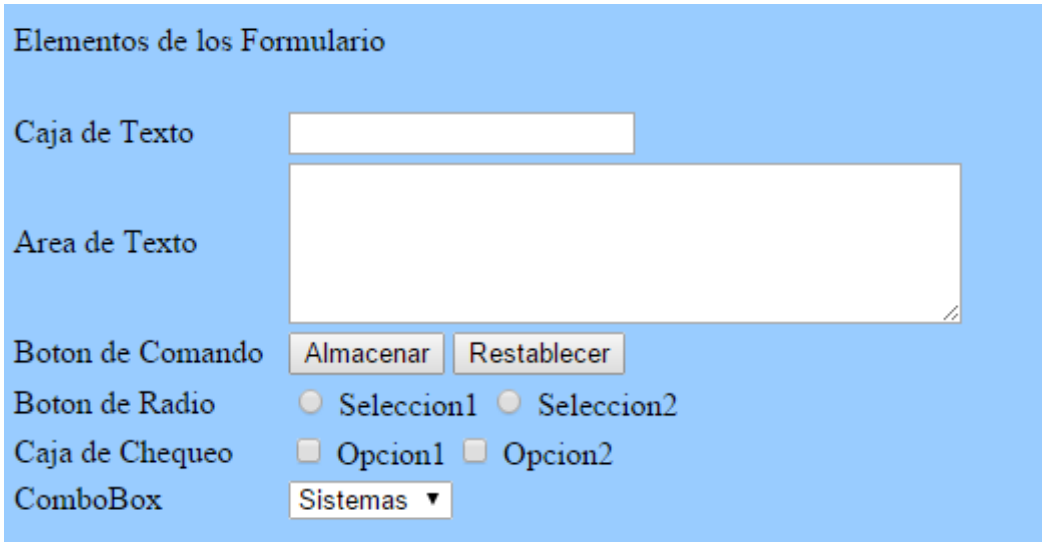
<input type="checkbox" name="checkbox" id="checkbox">

Las cajas de chequeo permiten la seleccione de uno, varios, todos o ningún elemento

lista / menú (comboBox)

```
<select name="select" id="select">
        <option value="1">Sistemas</option>
        <option value="2">Medicina</option>
        <option value="3">Derecho</option>
  </select>
```

Los comboBox permiten elegir de una lista de opciones, en value se especifica el valor a pasar y la otra información fuera de la etiqueta es lo que el usuario visualizara



HTML5

El HTML5 es una actualización del ambiente que por muchos años a estado al frente del desarrollo web, es probablemente el cambio más significativo que ha tenido el lenguaje, para este capítulo particular se enfocaran los cambios al manejo de formularios, teniendo en cuenta que en otras áreas también se presentaron cambios,

pero por efectos de que esta última unidad está enfocada al desarrollo y creación de CRUD se enfocara muy particularmente a los controles.

Formulario a diseñar

Creación del formulario y tabla para la ubicación de los elementos

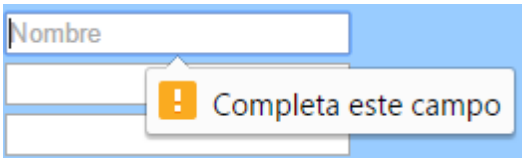
```
<form name="form1" method="post" action="." oninput="range_control_value.value=range_control.valueAsNumber" >
  <table width="550" border="0">
    <tr>
      <td colspan="2">Elementos de Formularios en HTML5</td>
    </tr>
    <tr>
      <td width="224"> </td>
      <td width="316"> </td>
  </tr>
```

Caja de texto con campos requeridos, foco y mensaje interno

```
    <tr>
      <td>Campos Requeridos y Foco</td>
      <td><input type="text" name="nombre" id="nombre" autofocus required placeholder = "Nombre"/> </td>
  </tr>
```

En las cajas de texto de HTML al igual que en este primer ejemplo se utiliza id y name para identificar el elemento según el browser, los demás elementos pueden cambiar según el alcance, además el HTML anterior a la versión 5 solo tenía en el type, las palabras text, hidden y password, en esta versión nueva encontramos mayor número de alternativas y se verán en los siguientes controles, para este caso particular de campos requeridos se utiliza

la sentencia required, con esto al momento de procesar la información si la caja de texto estuviera vacía mostraría un mensaje

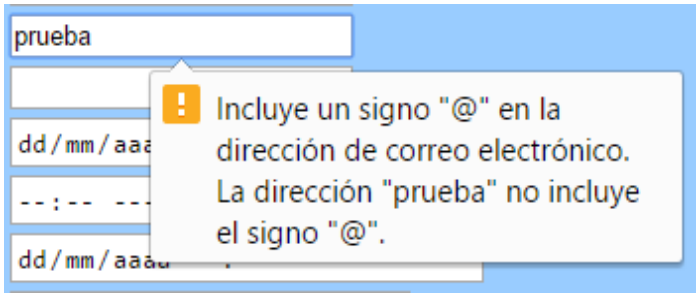


Además de esto se suma la propiedad autofocus, con esta propiedad lleva el cursor a esta caja con el fin de iniciar el proceso de digitación sin la ayuda del mouse, la recomendación para esta propiedad es que solo se utilice en una de las cajas de texto, la última propiedad que se va a trabajar para las cajas de texto tradicional es el placeholder, esta opción mostrara un mensaje en el interior de la caja de texto, en el momento de iniciar el ingreso de información esta desaparecerá, es ideal para ahorrar espacio y para dispositivos móviles.

Correo electrónico

```
<tr>
  <td>Correo Electronico</td>
  <td><input type="email" name="correo" id="correo" /></td>
</tr>
```

Se puede observar que el type contiene le valor email, este antes no se podía especificar, solo text o password, con esta instrucción el sistema validara que la información ingresada concuerde con el formato de un correo electrónico, se puede agregar required si se prefiere

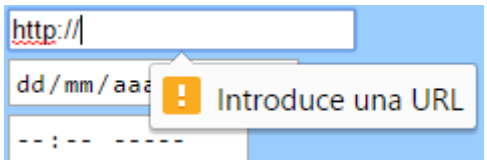


Desde el inicio de la digitación se indicará que incluya una arroba y los demás componentes de un correo electrónico

URL

Las cajas de texto para url tendrá esta palabra en el type, validaran que la dirección de un sitio sea cumpla las normas mínimas

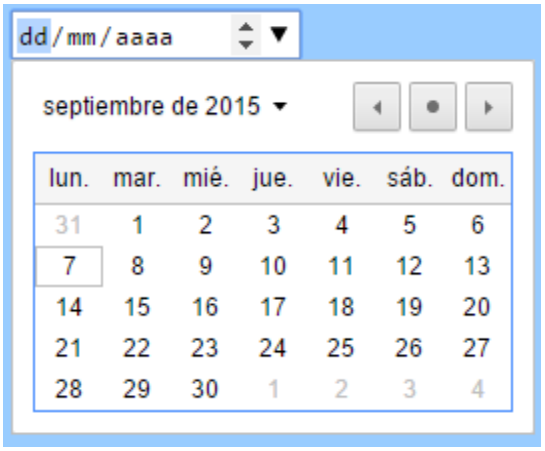
```
<tr>
  <td>URL</td>
  <td><input type="url" name="url" id="url" required /></td>
</tr>
```



Fechas

En la versión previa de HTML para crear un formato de fechas se recurría a herramientas como JavaScript, con el HTML5 y la instrucción date dentro del type se soluciona este impase.

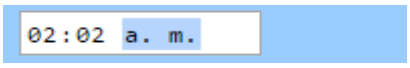
```
<tr>
  <td>Fecha</td>
  <td><input type="date" name="fecha" id="fecha" /></td>
</tr>
```



Hora

Se agrega la instrucción time dentro del type

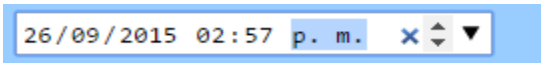
```
<tr>
  <td>Hora</td>
  <td><input type="time" name="hora" id="hora" /></td>
</tr>
```



Fecha y Hora

Esta etiqueta mezcla las dos anteriores

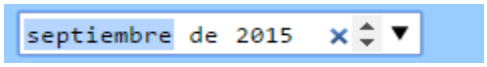
```
<tr>
  <td>Fecha y Hora</td>
  <td><input type="datetime-local" name="fechaHora" id="fechaHora" /></td>
</tr>
```



Meses

Con la instrucción month en el parámetro type

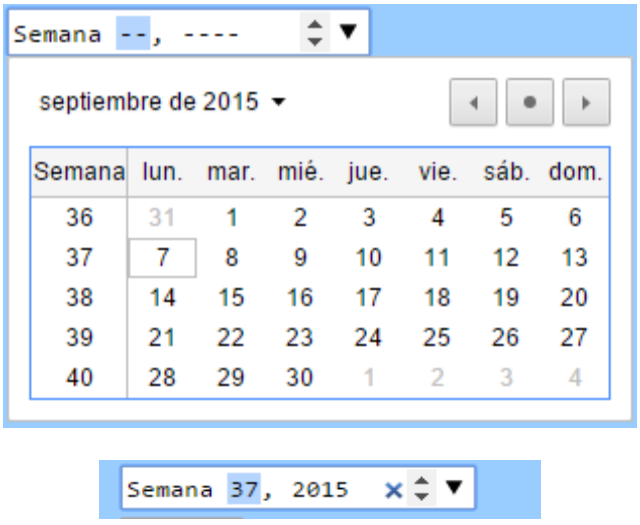
```
<tr>
  <td>Mes</td>
  <td><input type="month" name="mes" id="mes" /></td>
</tr>
```



Semana

Permite seleccionar el número de semana del año y representarlo en la fecha a la que corresponde

```
<tr>
  <td>Semana</td>
  <td><input type="week" name="semana" id="semana" /></td>
</tr>
```

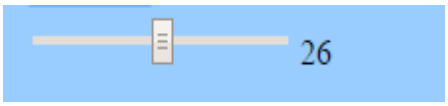


Rango de Números

```
<tr>
  <td>Rango de Numeros</td>
  <td><input type="number" name="rango" id="rango" min="1" max="50" /></td>
</tr>
```

Intervalos

```
<tr>
  <td>Intervalo</td>
  <td><input type="range" name="range_control" id="range_control" min="1" max="50" />
  <output for="range_control" name="range_control_value">0</output></td>
</tr>
```



6.3 TEMA 2 CSS HOJA DE ESTILO EN CASCADA

Con mucha frecuencia en la construcción de sitios web se presenta que los formatos no son uniformes o que se tienen que aplicar en cada página, cuando el sitio es considerablemente grande este tipo de formatos no son administrables y se puede recurrir a la construcción de un CSS (Cascading Style Sheet) Hojas de Estilo en Cascada.

Esta herramienta permite que de una manera simple se puedan administrar N cantidad de páginas de manera uniforme y con una codificación simple

Tipos de CSS

Existen 3 categorías para los CSS

CSS en línea: permite aplicar formatos a una etiqueta particular

CSS en bloque: permite aplicar formatos a una o varias etiquetas dentro del mismo archivo

CSS en archivo: permite la administración de múltiples paginas

Ejemplo de una pagina sin formatos

Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis mutat affert percipit cu, eirmod consectetuer signiferumque eu per. In usu latine equidem dolores. Quo no falli viris intellegam, ut fugit veritus placerat per.

Ius id vidit volumus mandamus, vide veritus democritum te nec, ei eos debet libris consulatu. No mei ferri graeco dicunt, ad cum veri accommodare. Sed at malis omnesque delicata, usu et iusto zzril meliore. Dicunt maiorum eloquentiam cum cu, sit summo dolor essent te. Ne quodsi nusquam legendos has, ea dicit voluptua eloquentiam pro, ad sit quas qualisque. Eos vocibus deserunt quaestio ei.

Estos bloques de código corresponden a un sitio web que no tiene formato alguno, la estructura es la siguiente

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Formatos CSS</title>
</head>
<body>
    <p>Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber
hendrerit an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis
reformidans eu, legimus senserit definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis
mutat affert percipit cu, eirmod consectetuer signiferumque eu per. In usu latine equidem dolores.
Quo no falli viris intellegam, ut fugit veritus placerat per.<p>
    <p>Ius id vidit volumus mandamus, vide veritus democritum te nec, ei eos debet libris consulatu.
No mei ferri graeco dicunt, ad cum veri accommodare. Sed at malis omnesque delicata, usu et iusto
zzril meliore. Dicunt maiorum eloquentiam cum cu, sit summo dolor essent te. Ne quodsi nusquam
legendos has, ea dicit voluptua eloquentiam pro, ad sit quas qualisque. Eos vocibus deserunt quaestio
 ei.</p>
</body>
</html>
```

Aplicación de formatos en línea

CSS se compone de una innumerable cantidad de opciones, algunas de ellas son

Font-family: especifica la fuente o familia de estas

Font-size: determina el tamaño de la fuente que se empleara, se puede especificar en pixeles (px), puntos (pt), pulgadas (in), centímetros(cm), milímetros(mm), picas (pc)

Text-align: alineación del texto a la derecha (right), izquierda (left), centrado (center), justificado (justify)

Font-wieght: intensidad de la fuente, los valores van entre 100 y 900, bold

Text-transform: se puede trasformar el texto, upper (mayúsculas), lower (minúscula), capitalize(primera letra en mayúscula)

Color: especifica el color de fuente, se puede especificar en formato hexadecimal, en formato RGB o con el nombre del color

Background-color: color de fondo

Background-image: imagen de fondo

Margin: en este formato se puede crear una margen de contorno, el valor que se especifique aplicara a la derecha, izquierda, arriba y abajo

Margin-left

Margin-right

Margin-top

Margin-button

Son complementos de la anterior

Border: especifica un borde en contorno

Border-left

Border-right

Border-top

Border-button

Text-decoration: aplica para colocar subrayados o para quitarlos

Line-heigth: especifica el espacio entre líneas

Width: ancho de un elemento

Los formatos en línea solo aplican a la etiqueta que lo requiera

Aplicando los formatos a la primera etiqueta

```
    <p style="font-family:Verdana, Geneva, sans-serif; font-size:10pt; color:#00F; background-color:#CCC; margin:8mm; line-height
:4mm; text-align:justify">Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit
an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit
definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis mutat affert percipit cu, eirmod consectetuer signiferumque eu
per. In usu latine equidem dolores. Quo no falli viris intellegam, ut fugit veritus placerat per.<p>
```

Arroja como resultado lo siguiente

Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis mutat affert percipit cu, eirmod consectetuer signiferumque eu per. In usu latine equidem dolores. Quo no falli viris intellegam, ut fugit veritus placerat per.

Ius id vidit volumus mandamus, vide veritus democritum te nec, ei eos debet libris consulatu. No mei ferri graeco dicunt, ad cum veri accommodare. Sed at malis omnesque delicata, usu et iusto zzril meliore. Dicunt maiorum eloquentiam cum cu, sit summo dolor essent te. Ne quodsi nusquam legendos has, ea dicit voluptua eloquentiam pro, ad sit quas qualisque. Eos vocibus deserunt quaestio ei.

Los dos párrafos están dentro de la etiqueta , pero solo se aplica el formato en línea a la primera, esta es la forma en la que trabaja el formato lineal.

Formato en bloque

Para la creación de un bloque de estilos se ubica en la cabecera de la página, se especifica la o las etiquetas, están aplicaran el formato a todas las etiquetas que se especifiquen.

```
<head>
<title>Formatos CSS</title>
<style type="text/css">

p   {

    font-family:Verdana, Geneva, sans-serif;
    font-size:10pt;
    color:#00F;
    background-color:#CCC;
    margin:8mm;
    line-height:6mm;
    text-align:justify;
}
</style>
</head>
```

La etiqueta <p> ya no tiene formatos en línea y el resultado es el siguiente

Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis mutat affert percipit cu, eirmod consectetuer signiferumque eu per. In usu latine equidem dolores. Quo no falli viris intellegam, ut fugit veritus placerat per.

Ius id vidit volumus mandamus, vide veritus democritum te nec, ei eos debet libris consulatu. No mei ferri graeco dicunt, ad cum veri accommodare. Sed at malis omnesque delicata, usu et iusto zzril meliore. Dicunt maiorum eloquentiam cum cu, sit summo dolor essent te. Ne quodsi nusquam legendos has, ea dicit voluptua eloquentiam pro, ad sit quas qualisque. Eos vocibus deserunt quaestio ei.

En este caso se observa que los dos párrafos tienen la misma distribución y los mismos formatos.

CSS en archivo

La limitante de aplicar formatos en bloque consiste en que solo serían formatos para una página, cuando se requiere que estos formatos se den en varias páginas debemos expórtalo

En un archivo independiente con extensión CSS aplicamos los formatos

La invocación de un archivo externo se aplica de la siguiente forma

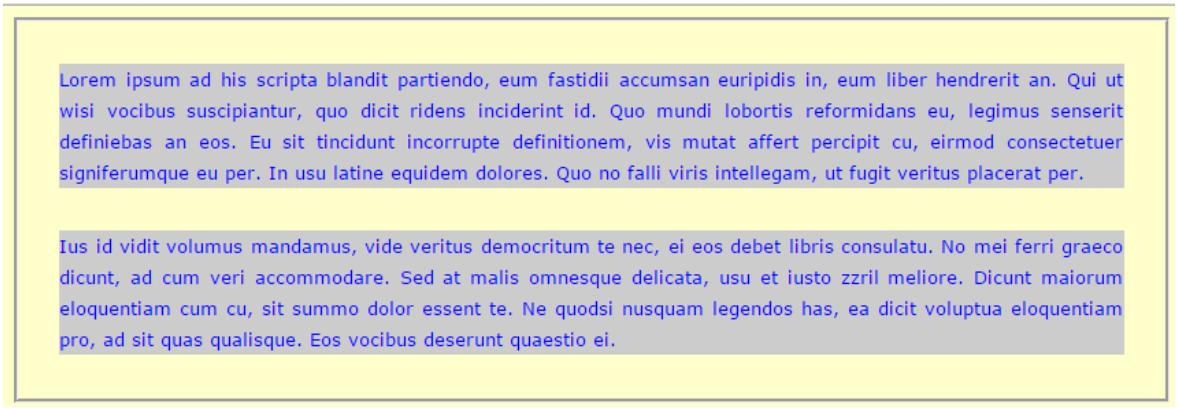
```
<head>
<title>Formatos CSS</title>
<link rel="stylesheet" type="text/css" href="formatos.css" />
</head>
```

En el archivo de ejemplo ya no hay formatos, existe el llamado a un archivo que contendrá formatos globales para todas las paginas asociadas. Todas las páginas del mismo sitio que contengan esta línea de código mostrara

el mismo formato, si se requiere un cambio de color, de fuente, de márgenes, etc., solo tendrá que ingresar al archivo, cambiarlo y al almacenar y ejecutar cualquiera de los archivos HTML mostrara dicha actualización.

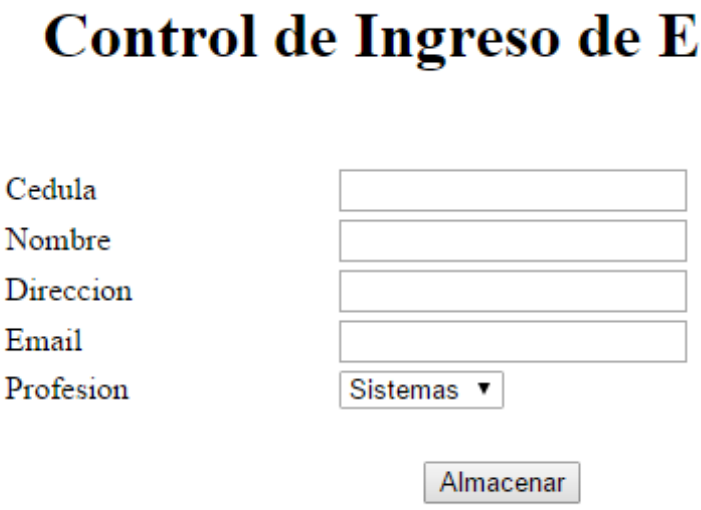
```
p    {

     font-family:Verdana, Geneva, sans-serif;
     font-size:10pt;
     color:#00F;
     background-color:#CCC;
     margin:8mm;
     line-height:6mm;
     text-align:justify;
}

body     {

     font-family:tahoma;
     font-size:9pt;
     background-color:#FFC;
     border:groove;
}
```

Resultado



Formatos en Formularios

Formulario sin formatos CSS



A este mismo formulario se le aplican algunos formatos de CSS.

Los formatos CSS son muchos y muy variados, aplique los mas esenciales que estamos tratando, implemente nuevas alternativas con caracteristicas nuevas de CSS2, CSS3.

6.4 TEMA 3 JAVASCRIPT

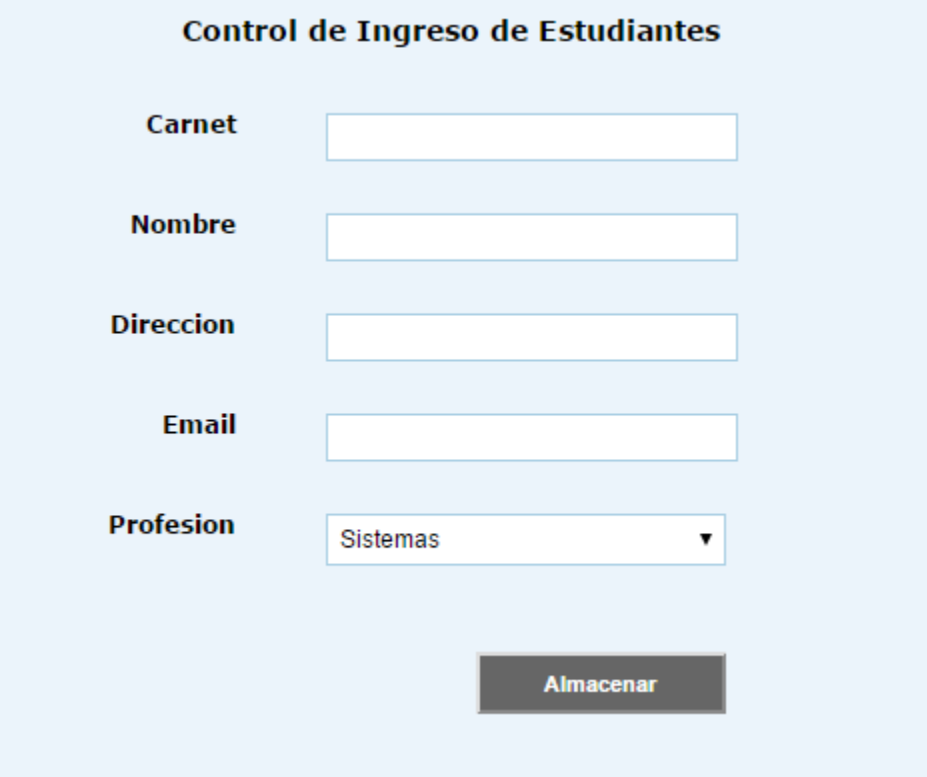
JavaScript es un lenguaje interpretado, tal vez uno de los mas conocidos desde hace muchos años por su versatilidad, su gran poder y por dejar una gran herencia con otros ambientes como JQuery, Ajax, JSon, etc., después de muchos años sigue como uno de los principales dentro de la programación web.

Estructura

En la estructura no se entrará mucho en detalles dado que tiene una similitud con Java y con C++, la forma de establecer ciclos paras o mientras, condicionales o selectores múltiples tienen las mismas estructuras.

JavaScript se va a utilizar principalmente como herramienta de validación, es un tema relativamente corto, pero de gran importación, se aplicará solo este tema por motivos de aplicabilidad en un tema posterior, la utilización del JavaScript (JS) es muy variado y muy amplio, a lo que se invita a seguir leyendo sobre el tema y no dejarlo solo en esta etapa inicial.

Formulario Inicial



Mediante este formulario se aplicarán los conceptos de validar que un campo no este vacío y que cumpla las condiciones mínimas solicitadas.

Para este caso se cuenta con carnet, nombre, dirección, email y profesión, se procederá con la validación de todos los campos que permitan el ingreso de datos.

Para el manejo de JavaScript hay que tener presente que se puede aplicar en condiciones similares a CSS, se pueden crear tareas en línea, en bloque o en archivos independientes, para este caso se aplicaran las validaciones en un archivo externo.

Para este ejemplo se creará un archivo llamado validar.js, para la vinculación de un archivo .js dentro de uno .HTML se procede a realizar la siguiente línea en la cabecera de la pagina

```
<head>
<title>Formatos CSS</title>
<link rel="stylesheet" type="text/css" href="formatos.css" />
<script type="text/javascript" src="validar.js"></script>
</head>
```

La instrucción del script especifica el tipo que es texto/ JavaScript y un parámetro src que indica la ruta y el archivo donde se encuentra la validación

Se utilizarán expresiones regulares para la validación, esto permitirá que de una forma sencilla y corta se pueda realizar múltiples procesos.

Declaración y asignación de los campos del formulario

```
function validacion () {

    var carnet = document.getElementById("carnet").value;
    var nombre = document.getElementById("nombre").value;
    var direccion = document.getElementById("direccion").value;
    var email = document.getElementById("email").value;
```

Al trabajar con un archivo externo para las validaciones en JavaScript se inicia con la sentencia function y el nombre de la función, acompañado de paréntesis que indican que pueden ir parámetros

En las líneas siguientes se declara cada una de las variables que contiene el formulario, y se asigna el valor de las cajas de texto a estas variables con la sentencia document.getElementById(“nombredelcontrol”).value

Posterior a este paso se procede con la primera validación, especificar que el carnet no este vacío

```
if (carnet == null || carnet == 0 || /^\s+$/.test(carnet)){

    alert ("Digite un numero de Carnet");
    return false;
}
```

Se especifica que el campo no sea nulo, no sea 0 y no esté compuesto solo por espacios

La sentencia alert muestra una ventana emergente, el return en false indica que no se cumplió la condición

Validación del contenido del carnet

```
else if (!(/^\d{12}$/.test(carnet))) {

    alert ("Carnet no Valido, Ingrese 12 Digitos, solo Numeros");
    return false;
}
```

En esta instrucción apreciamos las características de una expresión regular, algunas de sus características son

^ indica el inicio de una cadena

$ indica el final de una cadena

d indica valor entero

{12} indica que solo se pueden ingresar 12 caracteres

Validación de campo tipo texto (nombre)

Para la validación de un campo tipo texto se aplica la primera validación con el fin de que el campo no este vacío y luego se aplican las condiciones del campo

```
else if (!(/^[a-z A-Z]{7,40}$/.test(nombre))) {

    alert ("Nombre no Valido, Ingrese entre 7 y 40 Caracteres");
    return false;
}
```

En esta instrucción se valida que tenga un rango de letras de la “a” a la “z” tanto en minúscula como en mayúscula, un espacio y que tenga un rango de caracteres entre 7 y 40

Validación de una dirección (campos con texto y números)

```
else if (!(/^[a-z A-Z0-9-]{7,40}$/.test(direccion))) {

    alert ("Direccion no Validad, Ingrese entre 7 y 40 Caracteres");
    return false;
}
```

Tiene un contenido similar a el nombre, con la variación de 0-9 que indica que recibe numero de cero a nueve y guiones, además de permitir un rango de datos

Validación de campos email

```
else if (!(/^([\da-z_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/.test(email))) {

    alert ("E-Mail no Valido, Ingrese entre 7 y 40 Caracteres");
    return false;
}
else

    return true;
```

La validación de este tipo de campos comprenda una mayor cantidad de alternativas, permite números enteros, letras de la “a” a la “z” y underline (_) antes de la arroba, después de la arroba tiene una serie de caracteres similares y permite al final entre 2 y 6 caracteres para el domino de la dirección.

En el último else se aplica un return true, esto indica que cuando cumpla todas las condiciones la validación es valida

Por ultimo en el formulario se agrega el llamado a la función

```
<form id="form1" name="form1" method="post" action="." onsubmit="return validacion()">
```

Este evento onsubmit realiza el llamado de la función cada que se presiona el botón Almacenar, comprende return validación, con esta instrucción el sistema recibe el true o false según la validación de los campos, el valor que hay dentro del parámetro action es opcional, ahí se ubica el archivo que se va a trabajar si las condiciones se cumplen.

6.5 TEMA 4 JSP / SERVLETS

JSP (Java Server Page) es una herramienta complementaria de desarrollo web, la base de todo sitio web es y será HTML, que se complementa con herramientas como CSS, JS, entre otros elementos, JSP hace parte de un selecto grupo de opciones que permiten una mayor interacción, la comunicación y el acceso a las BD, se trabaja del lado del servidor y no del cliente como las otras herramientas.

Para la creación y utilización de un archivo JSP se trabaja con Eclipse EE, se puede descargar del sitio www.eclipse.org

También se requiere tomcat que es un complemento del Apache y permite la interpretación del sitio diseñado.

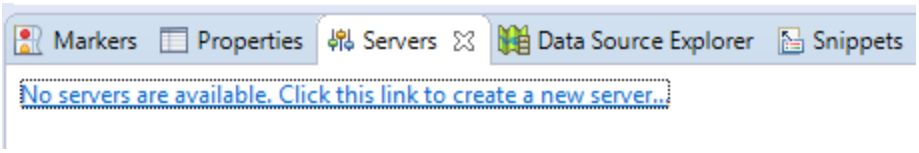
Apache Tomcat - Welcome!
tomcat.apache.org/ ▾ Traducir esta página

Para este capitulo se trabajará con la versión 8

Creación del servidor

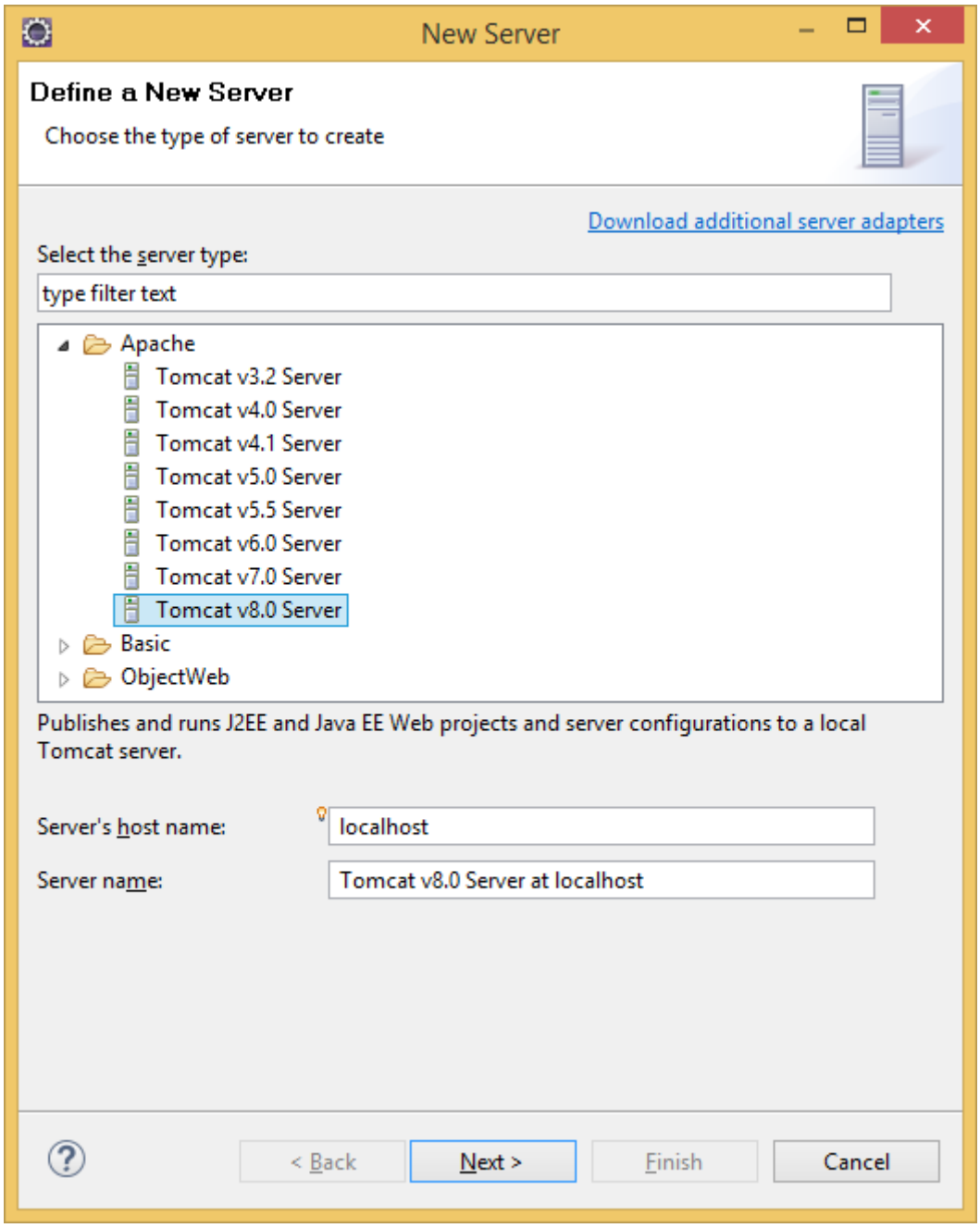
Para este proceso del JSP se requiere crear un servidor con la o las configuraciones necesarias para la interacción e interpretación de la nueva codificación.

Ubicados en la parte inferior del IDE en la pestana Server

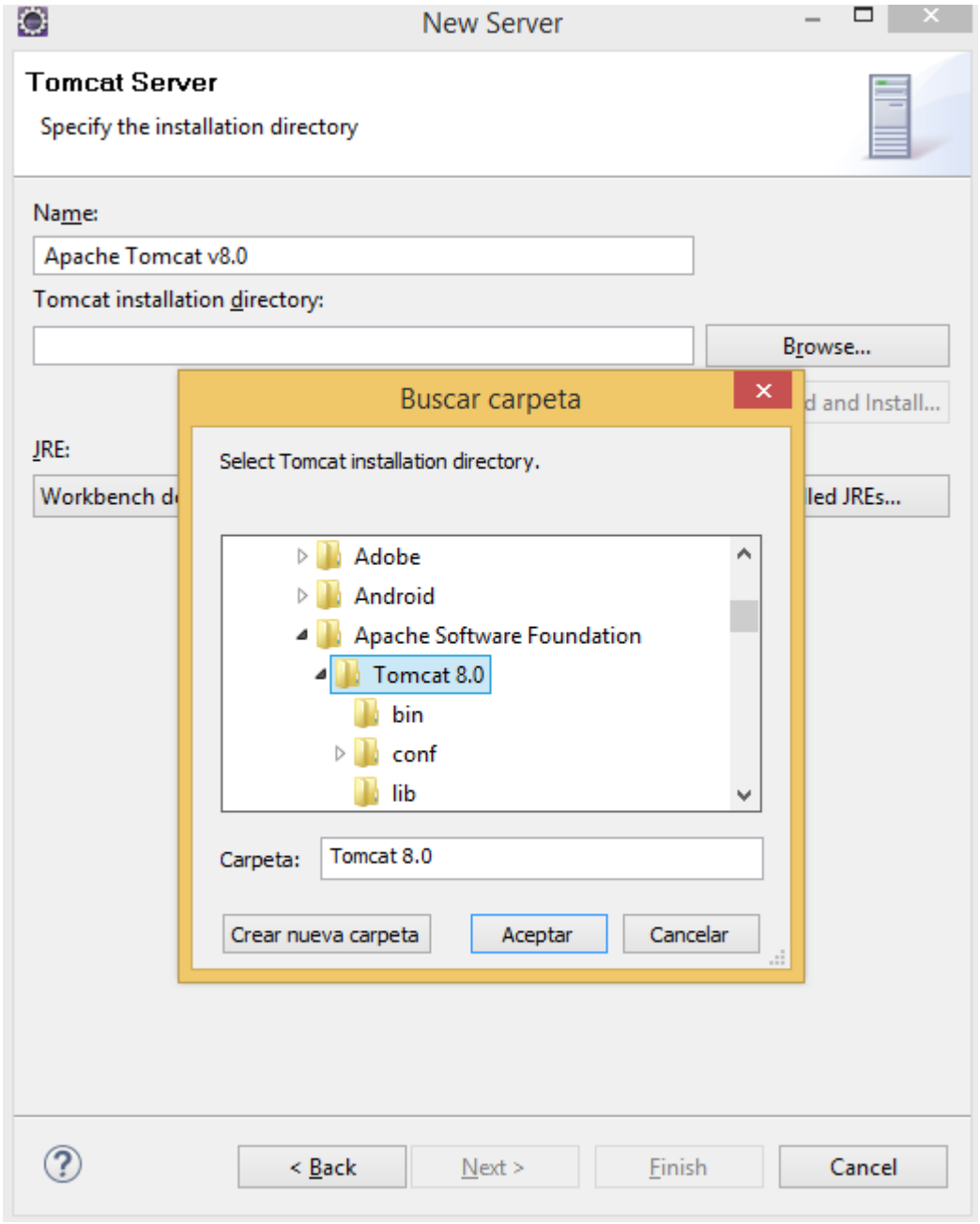


Se selecciona el vinculo No servers are available. Click this link to create a new server…

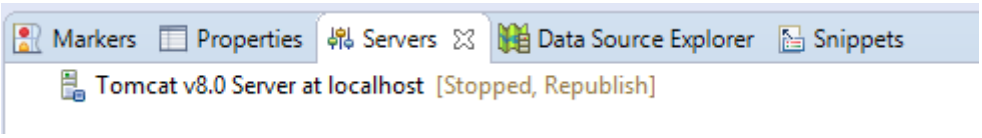
En la ventana contigua se selecciona apache y la versión mas reciente que se tenga disponible y / o se haya descargado



Se continua con el botón Next,

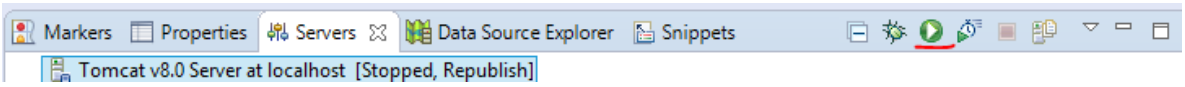


En la opción Tomcat Installation directory se busca la ubicación de tomcat, habitualmente se encuentra en archivos de programas, apaches Software Foundation y se selecciona la versión del tomcat, luego se acepta y finaliza

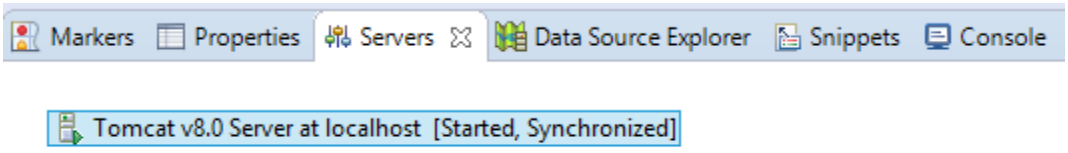


La pestana Servers tiene una configuración en este momento lista para ser usada

Probar la configuración



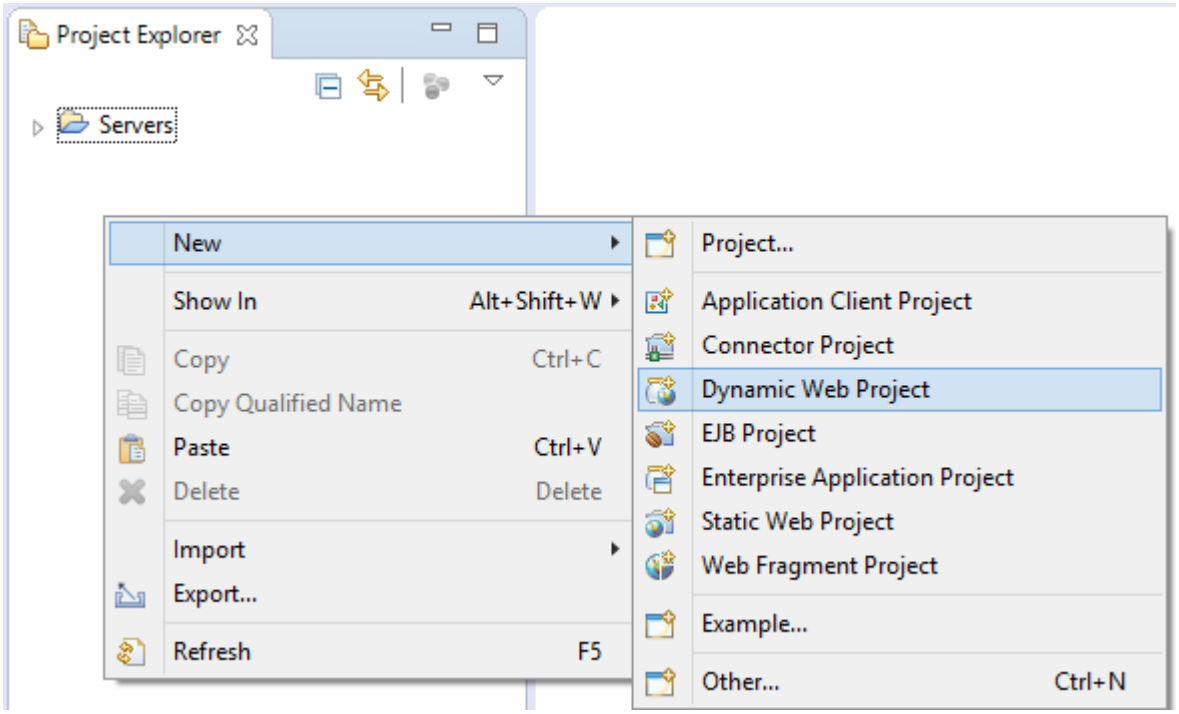
Al lado derecho de la opción server se encuentra un icono para la ejecución, verifique el buen funcionamiento del servidor



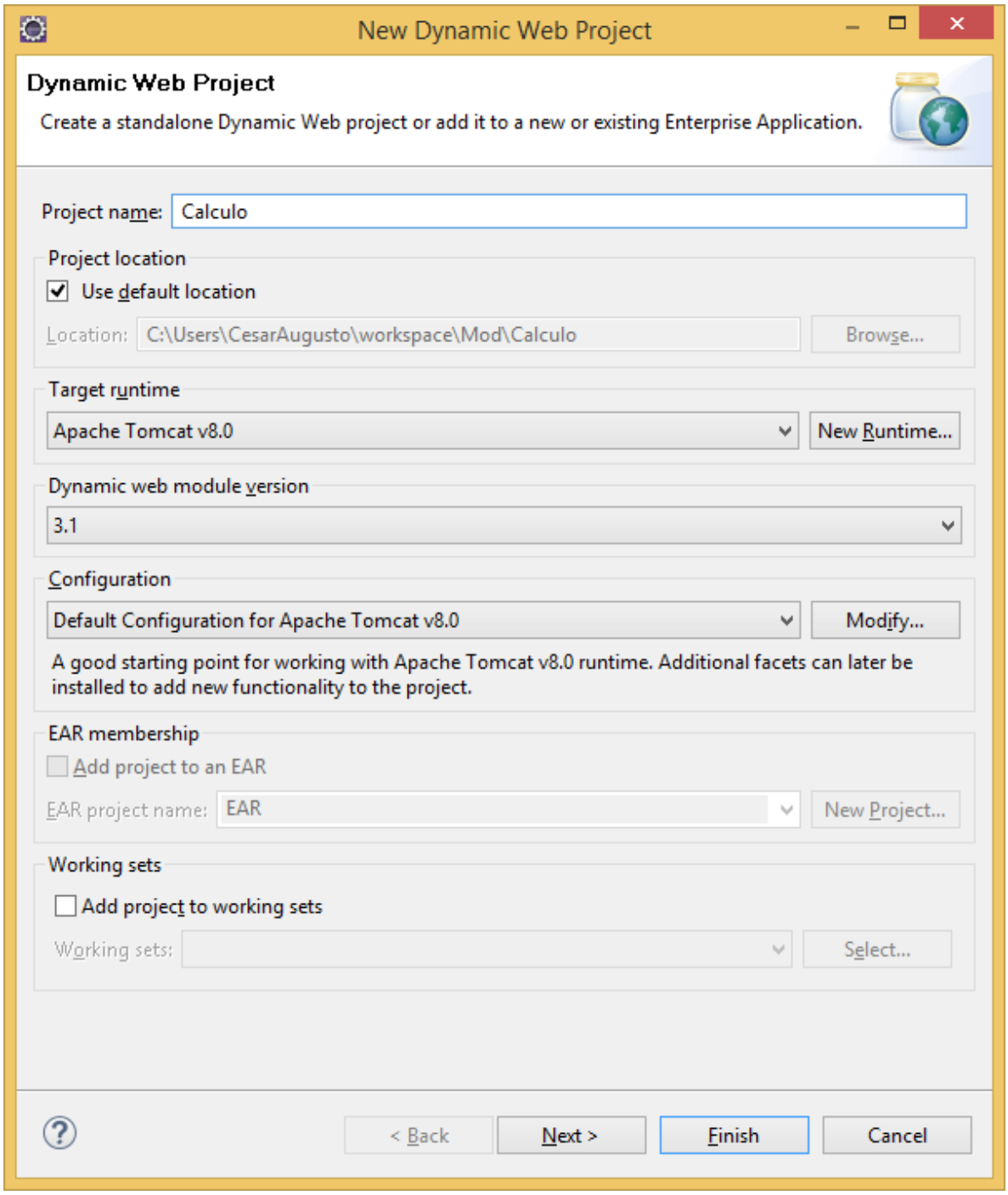
Se observa que el estado cambia de Stopped a Started y esta listo para su uso

Creación del primer proyecto

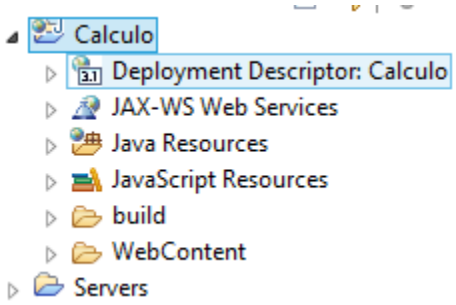
Ubicados en el Project Explorer y con botón emergente, se selecciona new / Dinamic Web Project



Se especifica el nombre del proyecto

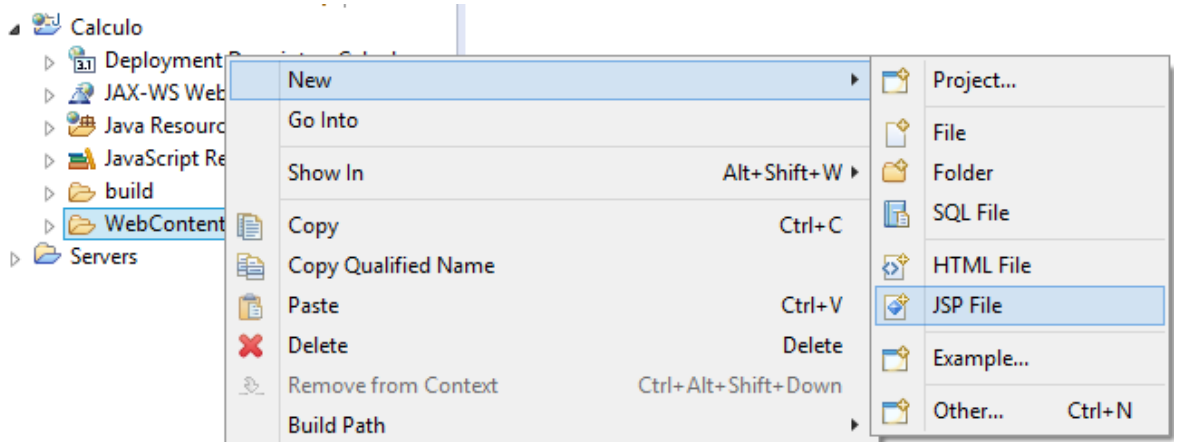


Y se finaliza

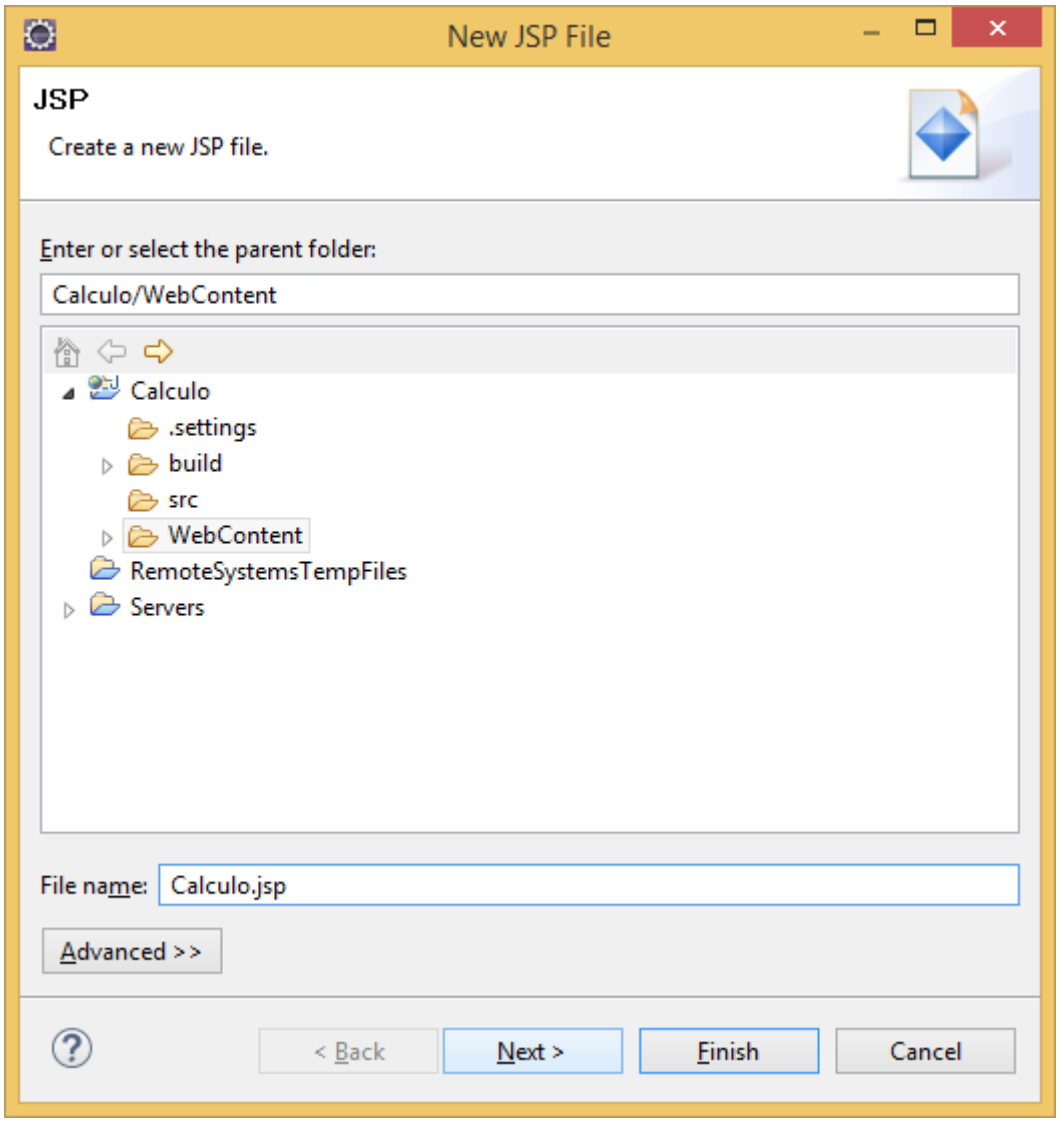


Esta es la estructura de un proyecto nuevo, dos de los aspectos mas comunes e importantes son el Java Resources donde se ubicarán los archivos con extensión .java y WebContent donde se ubicarán los archivos JSP

Creación de un archivo JSP



En el WebContent botón emergente, new / JSP File



Y finaliza la creación.

Apariencia Inicial

Calculo.jsp

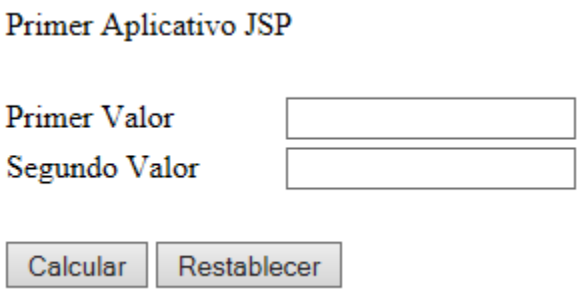
```
1  <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2      pageEncoding="ISO-8859-1"%>
3  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/h
4  <html>
5  <head>
6  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7  <title>Insert title here</title>
8  </head>
9  <body>
10
11 </body>
12 </html>
```

Contiene la mayor parte del código HTML pero en la primer línea de código se ve una serie de símbolos propios de JSP

<%@ %>

Estos son los símbolos que representan el trabajo con JSP, indica además el lenguaje, el contexto y una colección como es la ISO.

Creación de un formulario



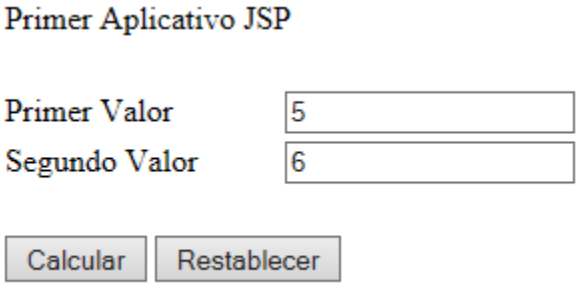
Este formulario tiene las mismas características de los temas anteriores (creación de formularios), se ubica dentro del body del archivo creado, se podrá ejecutar, aunque no arroje ningún resultado.

Aplicación de la operación mediante archivo JSP

Se crea un archivo nuevo llamado Resultado.jsp y se invoca en el formulario anterior en el parámetro action el nombre del archivo y la extensión de este.

```
<form id="form1" name="form1" method="post" action="Resultado.jsp">
```

```
<body>
    <%
        float pv = Float.parseFloat(request.getParameter("primerValor"));
        float sv = Float.parseFloat(request.getParameter("segundoValor"));
        float total = pv + sv;
        out.print ("resultado de la suma es " + total);
    %>
</body>
```



resultado de la suma es 11.0

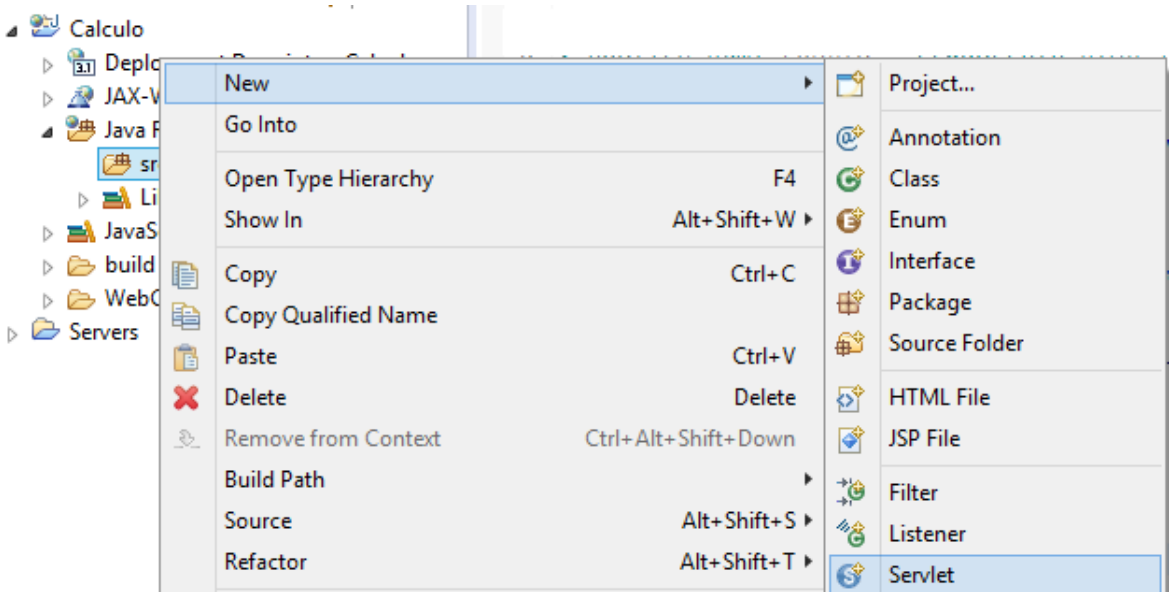
En el formulario creado inicialmente no tiene cambios de codificación salvo la línea de JSP que se establece automáticamente, pero se empieza viendo un parámetro como action que contiene un valor (Resultado.jsp), esta acción es el archivo o la función que se desea ejecutar.

Observe en el archivo Resultado.jsp que a una variable float pv se le hace una conversión y dentro de esta aparece la instrucción request.getParameter, esta instrucción toma el contenido de la caja de texto primerValor, lo mismo sucede para la segunda variable, tenga muy presente la escritura de las variables o campos, el java es sensible a mayúsculas y minúsculas.

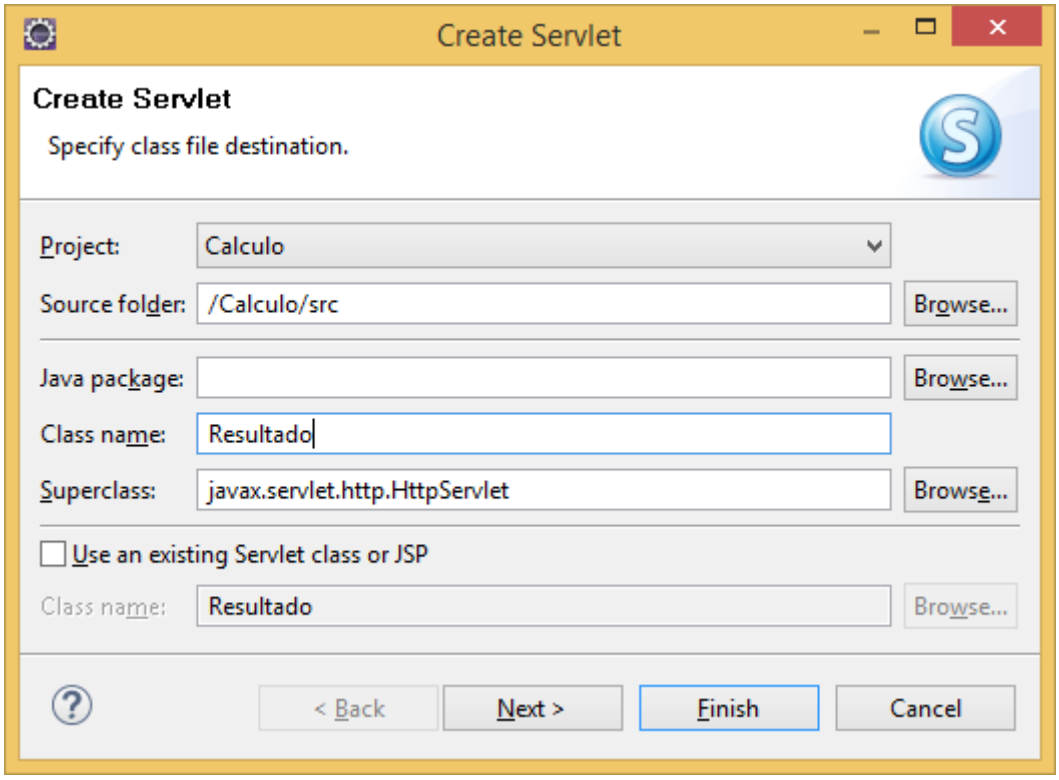
Aplicación del Mismo formulario mediante Servlets

Los servlets son archivos con extensión java diseñados para el manejo de los datos de un formulario mediante métodos como post (doPost) get (doGet)

Creación de un servlet



Con el botón emergente ubicados sobre el src del Java Resources, new / Servlet



Y se finaliza.

Componentes del Servlet

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Resultado
 */
@WebServlet("/Resultado")
public class Resultado extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Resultado() {
        super();
        // TODO Auto-generated constructor stub
    }
```

Se compone en su parte superior con los paquetes de uso, una anotación fundamental en la ejecución de este @WebServlet, nombre de la clase constructor

Metodo doPost y doGet

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // TODO Auto-generated method stub
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // TODO Auto-generated method stub
}
```

Útiles para la recepción de la información, esta llega inicialmente al doGet y al ser procesada pasa al doPost

Implementación

Se hace le llamado desde el action del formulario

```
<form id="form1" name="form1" method="post" action="Resultado">
```

Desarrollo del servlet

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // TODO Auto-generated method stub

    float pv = Float.parseFloat(request.getParameter("primerValor"));
    float sv = Float.parseFloat(request.getParameter("segundoValor"));
    float total = pv + sv;

    System.out.print ("Resultado de la suma :" + total);

}
```

Resultado

```
Resultado de la suma :15.0
```

6.6 TEMA 5 JAVABEANS

Dentro de los modelos nuevos de desarrollo cada día se encuentras más alternativas, una de ella son los Beans o JavaBeans, este modelo o patrón, cumple la tarea de “clase principal”, en una clase sin cara (sin diseño gráfico) pero permite el tránsito de la información, todos los procesos pasan por esta clase, la información se actualiza en esta y el proceso que lo requiera siempre y cuando tenga acceso podrá tomarlos procesarlos y devolverlos, así en un ciclo constante la información estará disponible, además de brindar seguridad y que no se tenga que acceder hasta el formulario o a otra clases más restringidas.

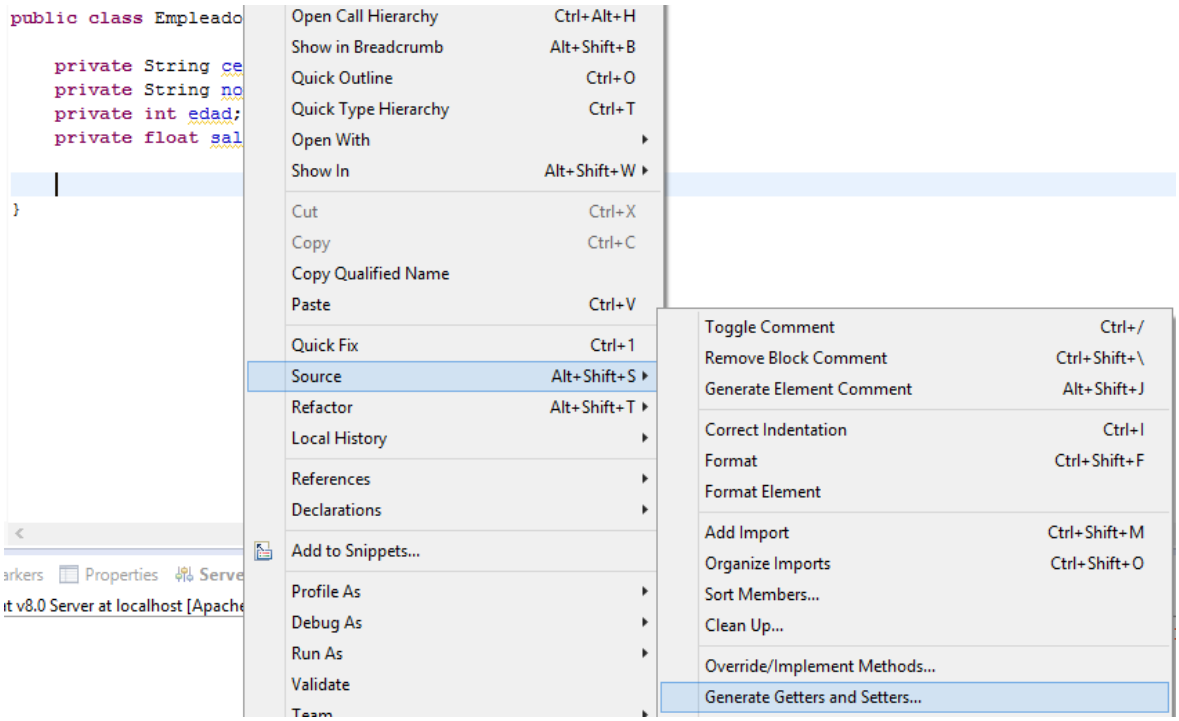
Creación de un JavaBean

Se crea una clase Empleado.java

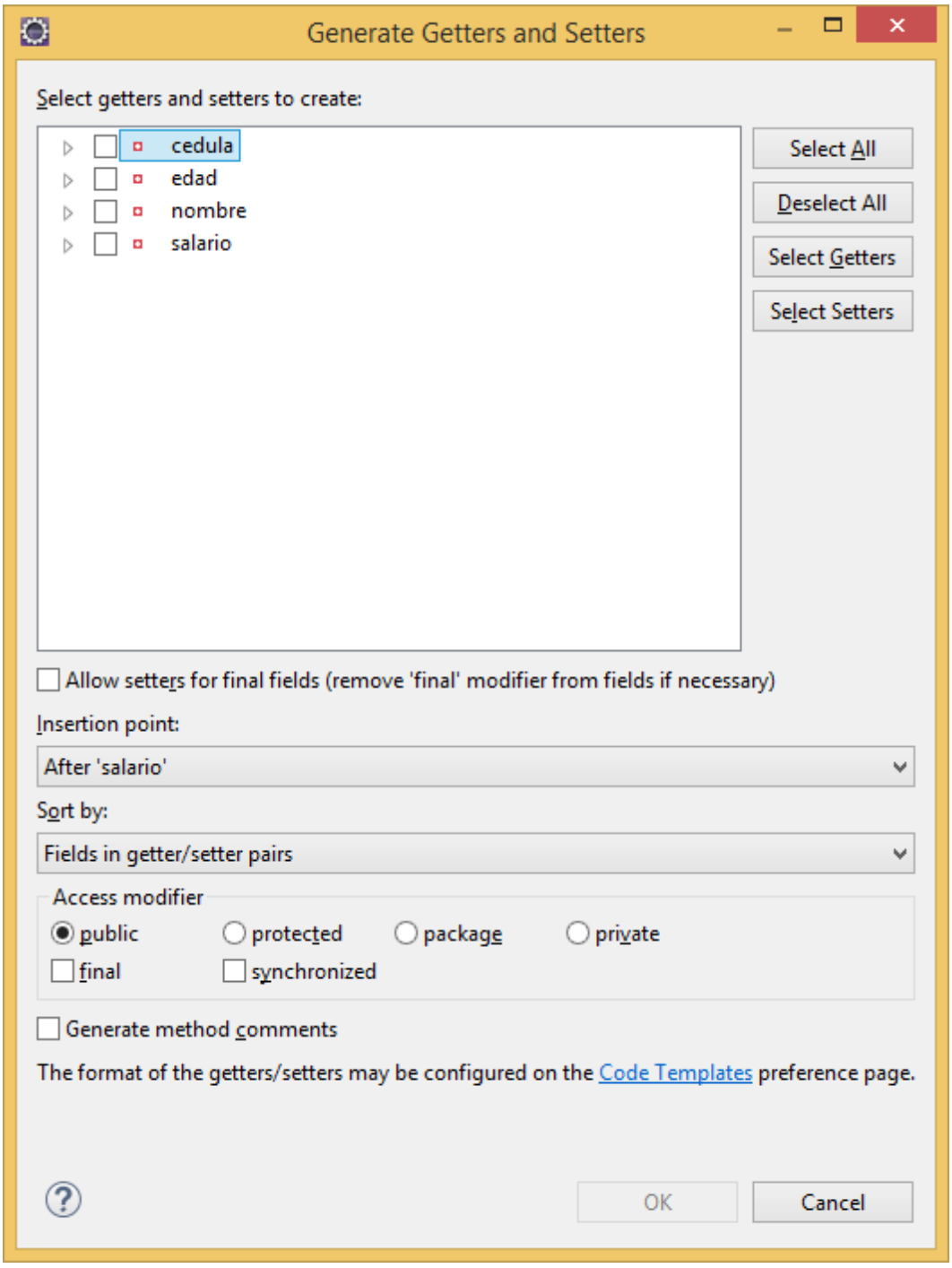
```
public class Empleado {

    private String cedula;
    private String nombre;
    private int edad;
    private float salario;

}
```

Este es el inicio típico de un bean en java, luego de este proceso se procede a crear o a generar los gettes/setters

Ubicados en cualquiera de los campos y con el botón emergente realizamos la generación de los getters / setters



Después de esto se procede a generar cuales son los campos que se desean incluir



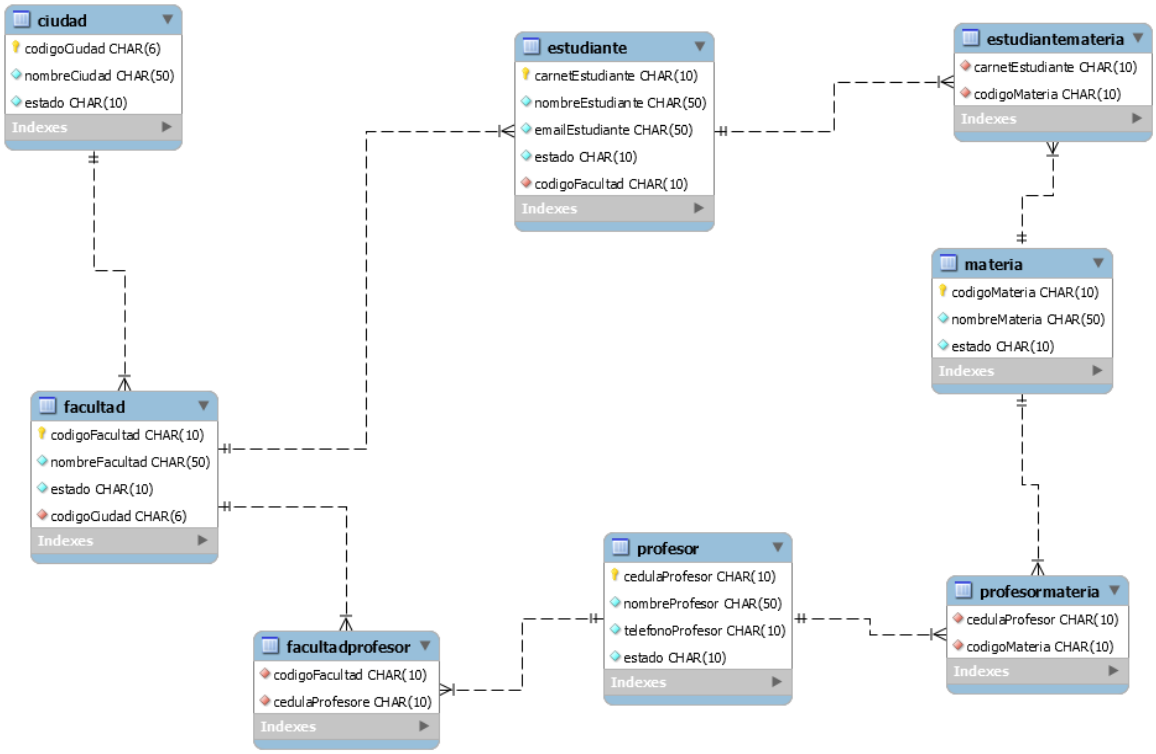
Acá se observan algunas de ellas ya generadas

```
private String cedula;
private String nombre;
private int edad;
private float salario;
public String getCedula() {
    return cedula;
}
public void setCedula(String cedula) {
    this.cedula = cedula;
}
public String getNombre() {
    return nombre;
}
public void setNombre(String nombre) {
    this.nombre = nombre;
}
public int getEdad() {
    return edad;
}
public void setEdad(int edad) {
    this.edad = edad;
}
```

En caso de agregar un campo nuevo, el proceso se repite, se selecciona con botón emergente y con source se realiza el proceso de getters / setters.

6.7 TEMA 6 CRUD

La creación de un CRUD mediante ambientes de programación siempre cumple unos mínimos requisitos, para este caso se iniciará con un MER (Modelo Entidad Relación)



Este MER se creará mediante MySQL y se aplicaran procedimientos almacenados para las tareas básicas de insertar, consultar, modificar, eliminar.

Posterior a esto se utilizará el conector de MySQL, este conector debe de ubicarse en la carpeta LIB de Tomcat.

Proyecto

Se creará un proyecto Universidad

Paquetes de trabajo

Control

Modelo

Utilidad

Utilidad

Se crea el archivo Conexion.java (Class)

```
package utilidad;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Conexion {

    public static Connection getConnection ()   {

        Connection con = null;

        try {

            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection("jdbc:mysql://localhost/universidad", "root", "admin");
        }
        catch (SQLException | ClassNotFoundException e) {

            System.out.print ("" + e);
        }

        return con;

    }
}
```

Creación de los javaBeans

Archivo Ciudad.java

```
package modelo;

public class Ciudad {

    private String codigoCiudad;
    private String nombreCiudad;
    private String estado;

    public String getCodigoCiudad() {
        return codigoCiudad;
    }

    public void setCodigoCiudad(String codigoCiudad) {
        this.codigoCiudad = codigoCiudad;
    }

    public String getNombreCiudad() {
        return nombreCiudad;
    }

    public void setNombreCiudad(String nombreCiudad) {
        this.nombreCiudad = nombreCiudad;
    }
}
```

Creación del archivo DAO, CiudadDAO.java

Este archivo contiene los métodos de trabajo para insertar, consultar, modificar y eliminar

Se especificarán los métodos y luego se aplicarán según su necesidad.

Este archivo CiudadDAO.java es una clase tradicional.

Inicio del archivo CiudadDAO

```
package control;

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

import modelo.Ciudad;
import utilidad.Conexion;

public class CiudadDAO {
```

Método de Insertar

```
public void insertar (Ciudad ciudad){

    Connection cnn = Conexion.getConnection();
    try{

        PreparedStatement registroCiudad = cnn.prepareStatement("call insertarCiudad (?, ?, ?)");
        registroCiudad.setString (1, ciudad.getCodigoCiudad());
        registroCiudad.setString (2, ciudad.getNombreCiudad());
        registroCiudad.setString (3, "Activo");
        registroCiudad.executeUpdate();
        System.out.print ("\nRegistro Almacenado");
    }
    catch (SQLException e) {

        System.out.print("" + e);
    }
}
```

Método de Listado

```
public ArrayList <Ciudad> listado ()    {

    ArrayList <Ciudad> arrayCiudad = new ArrayList <Ciudad>();
    Connection cnn = Conexion.getConnection();
    Ciudad ciudad;
    try {

        PreparedStatement registroCiudad = cnn.prepareStatement("call listarCiudad ()");
        ResultSet rs = registroCiudad.executeQuery();
        while (rs.next())   {

                ciudad = new Ciudad ();
                ciudad.setCodigoCiudad(rs.getString("codigoCiudad"));
                ciudad.setNombreCiudad(rs.getString("nombreCiudad"));
                arrayCiudad.add(ciudad);
        }
    }
    catch(SQLException e)   {

        System.out.print(""+ e);
    }

    return arrayCiudad;
}
```

Este método tiene como particularidad la creación de un arrayList, es un arreglo que permite el java y es aplicable para Java SE, EE y Móviles, permite almacenar los datos que cumplan una condición y posteriormente hacerlo visible, en este ejemplo se llena en la instrucción arrayCiudad.add, este método adiciona elemento por elemento y luego este se retorna para ser visualizado (véase JTable de la primera unidad).

Eliminar

```
public static void eliminar (String codigoCiudad)   {

    Connection cnn = Conexion.getConnection();
    try {

        PreparedStatement registroCiudad = cnn.prepareStatement("call eliminarCiudad (?)");
        registroCiudad.setString(1, codigoCiudad);
        registroCiudad.executeUpdate();
    }
    catch(SQLException e) {

        System.out.print("" + e);
    }
}
```

Consultar

```
public Ciudad consultar (String codigoCiudad) {

    Connection cnn = Conexion.getConnection();
    Ciudad ciudad = new Ciudad();
    try {

        PreparedStatement registroCiudad = cnn.prepareStatement("call consultarCiudad(?)");
        registroCiudad.setString(1, codigoCiudad);
        ResultSet rs= registroCiudad.executeQuery();
        if (rs.next())  {

            ciudad.setCodigoCiudad (rs.getString ("codigoCiudad"));
            ciudad.setNombreCiudad (rs.getString ("nombreCiudad"));
        }
    }
    catch(SQLException se)      {

        System.out.print("" + se);
    }

    return ciudad;

}
```

Modificar

```
public void modificar (Ciudad ciudad) {

    Connection cnn = Conexion.getConnection();
    try {

        PreparedStatement registroCiudad = cnn.prepareStatement("call modificarCiudad(?,?)");
        registroCiudad.setString(1, ciudad.getCodigoCiudad());
        registroCiudad.setString(2, ciudad.getNombreCiudad());
        registroCiudad.executeUpdate();
    }
    catch(SQLException se) {

        System.out.print("" + se);
    }
}
```

Todos estos procesos tienen el mismo origen del CRUD creado en java SE, las sentencias y los comandos son los mismos

Fachada

La fachada en el ambiente web cambia un poco, tiene una codificación más específica de cada tarea.

Se construirá una Fachada o Facade para la ciudad y se llamara FachadaCiudad.java, este archivo es un servlet.

```java
package control;

import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import modelo.Ciudad;

@WebServlet("/FachadaCiudad")
public class FachadaCiudad extends HttpServlet {
    private static final long serialVersionUID = 1L;

    private CiudadDAO ciudadDAO;
    private Ciudad ciudad;

    private static final String LISTAR = "/listarCiudad.jsp";
    private static final String MODIFICAR = "/modificarCiudad.jsp";

    public FachadaCiudad() {
        super();

        ciudadDAO = new CiudadDAO();
        ciudad = new Ciudad ();
    }
```

Esta fachada instancia CiudadDAO donde se encuentran los procesos previamente creados, instancia Ciudad que contiene los Bean, posteriormente se encuentran dos líneas que tienen constantes, una de ellas LISTAR y la otra MODIFICAR, hace referencia a sus respectivas URLs y posteriormente un constructor con la instancia definitiva de las dos clases antes mencionadas.

A continuación, se desarrolla el método doGET

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    String opc = request.getParameter("opc");
    String accion = null;

    if (opc.equalsIgnoreCase("listar"))  {

        accion=LISTAR;
        request.setAttribute("listarCiudad", ciudadDAO.listado());
    }
    else if (opc.equalsIgnoreCase("eliminar"))  {

        String codigoCiudad = request.getParameter("codigoCiudad");
        ciudadDAO.eliminar(codigoCiudad);
        accion=LISTAR;
        request.setAttribute("listarCiudad", ciudadDAO.listado ());
    }
    else if (opc.equalsIgnoreCase("consultar")) {

        String codigoCiudad = request.getParameter("codigoCiudad");
        request.setAttribute("listarCiudad", ciudadDAO.consultar(codigoCiudad));
        accion=MODIFICAR;
    }
    else

        accion="modificar";

    RequestDispatcher view = request.getRequestDispatcher(accion);
    view.forward(request, response);
}
```

Este método recibe las ordenes según la tarea que se pretende desarrollar, la variable opc recibirá una palabra clave, sea para insertar, consultar, modificar o eliminar, se evalúa cual de las opciones es la correcta y se implementa.

Ejemplo

```
if (opc.equalsIgnoreCase("listar"))  {

    accion=LISTAR;
    request.setAttribute("listarCiudad", ciudadDAO.listado());
}
```

Se evalúa si la opción que se recibe es listar, a la variable acción se le asigna una constante que a su vez tiene una URL asignada y por último se invoca el método ciudadDAO.listado, este es el método del archivo DAO, este proceso se lleva a un listar Ciudad que es un arreglo de datos.

Al final de este método se encuentra un par de líneas

```
RequestDispatcher view = request.getRequestDispatcher(accion);
view.forward(request, response);
```

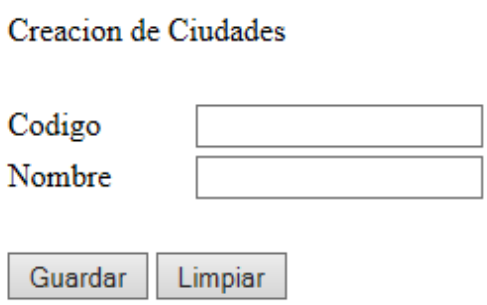
permiten la invocación del proceso que tenga asignado la variable acción, por ejemplo si se insertar o modifica después de hacer la tarea va a llamar el listado para corroborar que si esta funcionando.

Método doPost

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    String opc = request.getParameter("opc");
    ciudad.setCodigoCiudad(request.getParameter("codigoCiudad"));
    ciudad.setNombreCiudad(request.getParameter("nombreCiudad"));

    if (opc.equalsIgnoreCase("insertar"))

        ciudadDAO.insertar (ciudad);
    else if (opc.equalsIgnoreCase("modificar")) {

        String codigoCiudad = request.getParameter("codigoCiudad");
        ciudad.setCodigoCiudad (codigoCiudad);
        ciudadDAO.modificar(ciudad);
    }
    RequestDispatcher view = request.getRequestDispatcher(LISTAR);
    request.setAttribute("listarCiudad", ciudadDAO.listado());
    view.forward(request, response);
}
```

En este método se hace recepción de todos los valores que involucran el formulario, se recibe el codigoCiudad, nombreCiudad y se determina si va a ser una inserción o una modificación

Formulario de trabajo, Ciudad.jsp



Este formulario es tradicional a los vistos anteriormente, en el action tiene el siguiente llamado

```
<form id="form1" name="form1" method="post" action="FachadaCiudad?opc=insertar">
```

Esta primera parte es funcional, ya almacena información, se va a complementar realizando las demás tareas desde un archivo index.jsp y el listado de información.

Index.jps

El archivo de índex se va a utilizar como medio para llamar las opciones que se desean. Para el ejemplo inicial solo contendrá la inserción y el listado y desde esta última ira la eliminación y modificación.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<table width="200" border="0" align="center">
  <tr>
    <td>Menu Principal</td>
  </tr>
  <tr>
    <td> </td>
  </tr>
  <tr>
    <td><a href="ingresoCiudad.jsp">Ingresar Ciudad</a></td>
  </tr>
  <tr>
    <td><a href="FachadaCiudad?opc=listar">Listar Ciudad</td>
  </tr>
</table>
</body>
</html>
```

Archivo listarCiudad.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>


<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<table width="500" border="0" align="center">
<thead>
  <tr>
    <td colspan="4">Listado General de Ciudades</td>
  </tr>
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
  </tr>
```

Este archivo contiene algunas características de configuración, como las 3 primeras líneas en las que se realiza el llamado a la librería jstl, para un correcto funcionamiento del listado.

Los demás procesos son de una tabla tradicional de HTML

```
<tr>
  <td>Codigo</td>
  <td>Nombre</td>
  <td> </td>
  <td> </td>
</tr>
</thead>
 <tbody>
      <c:forEach items="${listarCiudad}" var="lciudad">
  <tr>
    <td> <c:out value="${lciudad.codigoCiudad}"/></td>
    <td> <c:out value="${lciudad.nombreCiudad}"/></td>
    <td> <a href="FachadaCiudad?opc=eliminar&codigoCiudad=<c:out value="${lciudad.codigoCiudad}"/>">Eliminar</a></td>
    <td> <a href="FachadaCiudad?opc=consultar&codigoCiudad=<c:out value="${lciudad.codigoCiudad}"/>">Modificar</a></td>
  </tr>
  </c:forEach>
  </tbody>

</table>
</body>
</html>
```

La segunda parte del archivo comprende

```
<c:forEach items="${listarCiudad}" var="lciudad">
```

La utilización del arreglo o matriz de datos, esta se especifico desde la fachadaCiudad y se agrega un parámetro var como alias de esta matriz

```
<td> <c:out value="${lciudad.codigoCiudad}"/></td>
```

Se hace uso del alias y del campo que se desea visualizar (nombre asignado en la clase principal de getters / setters), este proceso se repita para cuantos campos deseamos mostrar en pantalla.

El listar quedaría de esta manera.

Listado General de Ciudades

Codigo	Nombre		
101010	envigado	Eliminar	Modificar
202020	manizales	Eliminar	Modificar
303030	tunja	Eliminar	Modificar

Modificación

El archivo de modificacionCiudad.jsp, tiene un diseño similar al de insertarCiudad.jsp, adicionando las siguientes líneas en la parte superior

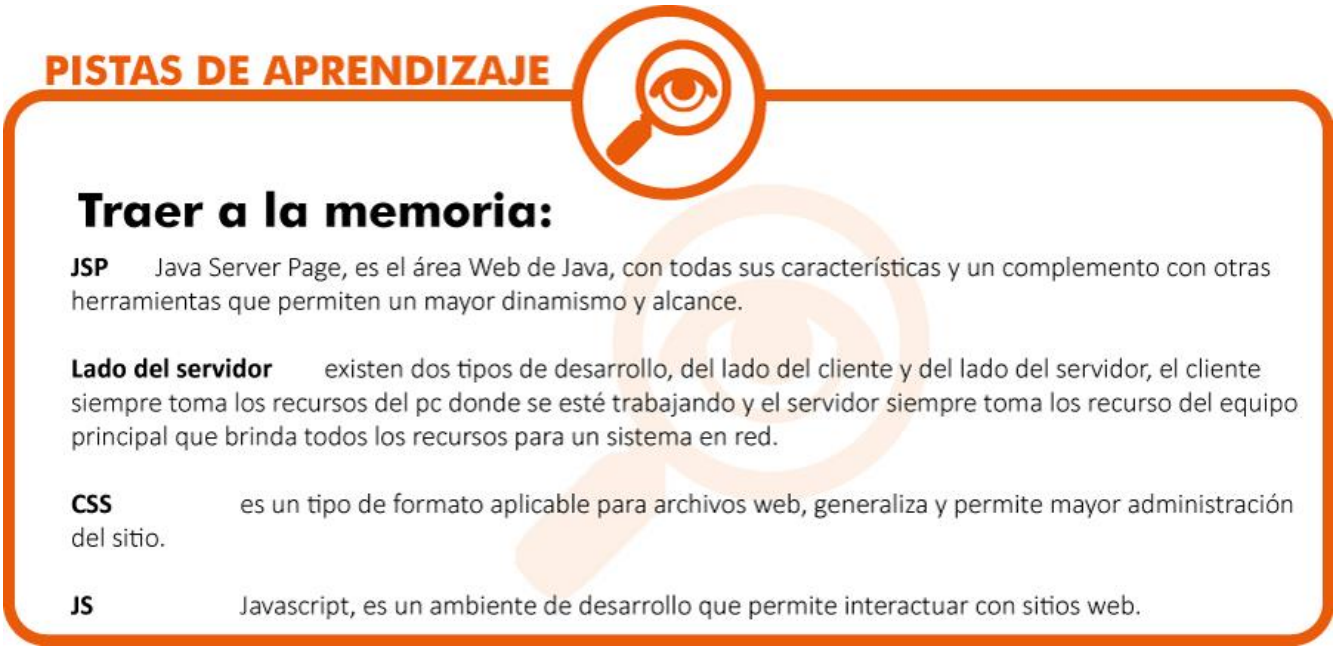
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
```

Y en cada caja de texto se aplicaría lo siguiente

```
<tr>
  <td>Codigo</td>
  <td><input name="codigoCiudad" type="text" id="codigoCiudad" value="${listarCiudad.codigoCiudad}" /></td>
```

Este proceso ya es operativo para una tabla maestra.

Tenga presente que el modificar puede ser el proceso más largo, luego de listar información, se consulta y posterior a esto se almacena.



6.7.1 EJERICICIO DE APRENDIZAJE

Nombre del taller de aprendizaje: nomina	Datos del autor del taller: Cesar Augusto Jaramillo Henao
Escriba o plantee el caso, problema o pregunta:	

Crear un proyecto que represente una nómina, se parte de un modelo relacional, diseño CSS, validaciones JS, implemente el proceso con JSP y Servlets.
Solución del taller: Mediante los conceptos vistos en la primera y última unidad aplique conceptos de CRUD y arroje resultados correspondientes al tema planteado.

6.7.2 TALLER DE ENTRENAMIENTO

Nombre del taller: Veterinaria	Modalidad de trabajo: Individual
Actividad previa: Realice completamente el CRUD de jsp y servlets visto en la última unidad, esto dará las bases necesarias para el trabajo posterior.	
Describa la actividad: Cree un proyecto que cubra todos los temas de la unidad 5, aplique formatos y validaciones y un CRUD que comprenda varias tablas incluyendo tablas maestras, referenciales e intermedias.	

7 PISTAS DE APRENDIZAJE

Recuerde que: programación web es un recurso muy amplio que maneja multiples lenguajes y elementos

Tenga en cuenta: la programación utilizada es basada en java tanto para java SE como para Java EE

Traiga a la memoria: que la mayor parte de comando e instrucciones son los mismos en java SE que en java EE

8 GLOSARIO

Java SE	es la versión estándar de java, esta versión es la base de todo el trabajo en java
Java EE	es la versión Enterprise o empresarial, es utilizada para la programación web
Eclipse	es un IDE de desarrollo que permite facilitar algunas tareas de la programación en Java
Proyecto	es un conjunto de archivos que componen una aplicación
Paquete	es un área de trabajo que permite la clasificación de archivos o clases
DAO	es un modelo de desarrollo o patrón de diseño, standard de trabajo
Getters/setters	hacen parte de una clase principal que permite accede a la información
JSP	Java Server Page, ambiente de trabajo web
HTML	Lenguaje de Marcas de Hipertexto
JavaScript	lenguaje similar en estructura a Java que se puede mezclar con aplicaciones web
CSS	Formatos de aplicación de aplicaciones web
Método	espacio de código que realiza una funciona especifica
Façade	patrón de diseño que administra un conjunto de clases
Hilo	herramienta de trabajo que permite realizar una tarea en procesos paralelos
Red	parte de la programación que permite que varios trabajen con elementos compartidos
Hibernate	FrameWork de java que permite realizar procesos standard o web de una forma mas simplificada
Propertie	extensión de archivo que permite acceder a recursos fuera de la compilación
Conector	archivo que contiene los elementos necesarios para vincular un proyecto con un motor de bases de datos
Reportes	herramienta de visualización de información general o especifica
Documentación	herramienta de ayuda para el desarrollador y el control de los procesos realizados en periodos de tiempo.

9 BIBLIOGRAFÍA

Eckel, Bruce. (2008). Piensa en Java, Madrid. ISBN: 978-84-8966-034-2

Villalobos, Jorge (2006), Fundamentos de Programación, Bogotá. ISBN: 970-26-0846-5

Deitel, Paul. (2012), Java, como programar, México. ISBN: 978-607-32-1150-5