Universidad Autónoma de Nuevo León
Facultad de Ciencias Físico Matemáticos

Licenciado en Seguridad en Tecnologías de Información

Diseño Orientado a Objetos

Práctica en el laboratorio

Practica 9

Lic. Miguel Ángel Salazar Santillán

Grupo: 007 Matrícula: 1732645

Diana Elizabeth Díaz Rodríguez

02/ 04/ 2017

PRACTICA

En esta práctica, solo tuvimos que usar la práctica anterior que sería el lab7

Con la finalidad de crear un patrón Singleton.

Lo primero que hicimos fue crear un nuevo proyecto con el nombre de Laboratorio 9.

El procedimiento fue similar al Lab7…

Después creamos la base de Datos en JavaDB.

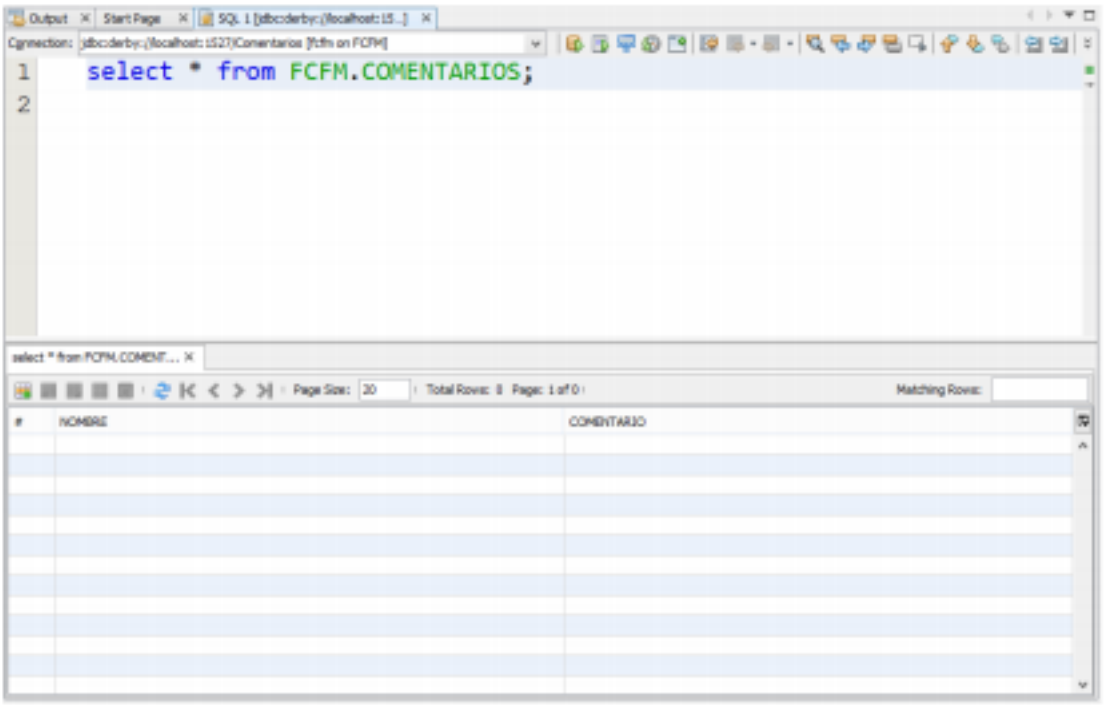
Los datos que tendría serán…

Database name: Comentarios

User name: fcfm

Password: lsti01

Luego lo conectamos que nos mostrara otro esquema llamado FCFM, y luego vendrán varias opciones una de ellas será la Tabla, le insertamos y creamos dos columnas una para nombre y otra para los comentarios, contestando sus especificaciones.



Los cuales se irán guardando todo lo que se vaya insertando cuando la aplicación esté terminada.

El siguiente paso es crear el modelo y una clase para que interactúe con la base de datos. Creamos una clase con para el acceso de los datos con el nombre de DAO y otra clase como objeto de transferencia de los mismos datos.

Con la clase de transferencia de datos recibirá el nombre de ComentariosPOJO que sirve para representar un registro de una tabla de base de datos, que tendrá dos variables de tipo String: Nombre y comentario.

Este es uno de los más sencillos, solo teníamos que hacer los get y los sets de cada uno de ellos.

```
public class ComentariosPOJO {

    private String nombre;
    private String comentario;

    /**...3 lines */
    public String getNombre() {
        return nombre;
    }

    /**...3 lines */
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    /**...3 lines */
    public String getComentario() {
        return comentario;
    }

    /**...3 lines */
    public void setComentario(String comentario) {
        this.comentario = comentario;
    }
```

Que olvide mencionar, se encuentra en un paquete llamado modelos.

Ahora sigue la clase de ComentariosDAO, que también se encuentra en el paquete de modelos.

Agregamos una variable de instancia privada con el nombre de conexión e importamos el java.sql.Connection.

Y abrimos una conexión, declarando una url para la conexión con la base de datos, daclarando usuario y contraseña y que obtenga un DriveManager con el método getConnection()… y cerramos la conexión.

```
/**...4 lines */
public class ComentariosDAO {
    Log log = Log.getInstance("G:\\lab7\\src\\Text.txt");
    private Connection conexion;
    private void abrirConexion() throws SQLException {
        String dbURI = "jdbc:derby://localhost:1527/Comentarios";
        String username = "fcfm";
        String password = "lsti01";
        conexion = DriverManager.getConnection(dbURI, username, password);
    }

    private void cerrarConexion() throws SQLException {
        conexion.close();
    }
```

Después hacemos otro método para insertar() que reciba los ComentariosPOJO, con la intención de que al insertar los datos tiene que pasar por el controlador y abrirá la conexión a la base de datos.

```
    public void insertar(ComentariosPOJO dto){
        try{
            abrirConexion();
            String sqlInsert = "insert into COMENTARIOS values ('"+dto.getNombre()+"','"+dto.getComentario()+"')";
            Statement stmt = conexion.createStatement();
            stmt.executeUpdate(sqlInsert);
            cerrarConexion();
        }catch(Exception ex){
            log.write("Ocurrio un error por favor vuelve a intentarlo");
        }
    }
```

Y después agregamos un método para buscar que regrese un objeto de tipo list. También devlaramos un ResultSet con el objetivo de almacenar los registros que coincidan con la búsqueda.

```
public ArrayList buscar(ComentariosPOJO dto){
    ResultSet regis;
    ArrayList<ComentariosPOJO> comentarios = new ArrayList();
    try{
        abrirConexion();
        String sqlBuscar = "select from comentario where nombre = '" + dto.getNombre() + "' y comentario '%" + dto.getComer
        Statement statement = conexion.createStatement();
        ResultSet result = statement.executeQuery(sqlBuscar);

        while(result.next()){
            String nombre = result.getString("Nombre");
            String comentario = result.getString("Comentario");
            ComentariosPOJO coment = new ComentariosPOJO();
            coment.setNombre(nombre);
            coment.setComentario(comentario);
            comentarios.add(coment);
        }
        cerrarConexion();
    }catch(Exception ex){
        log.write("Ocurrio un error por favor vuelve a intentarlo");
    }
    return comentarios;
```

Después creamos las paginas para la vista un index.html que ya viene por default, tiene que tener un POST y tiene que enlazar con el ervlet que se llamara ComentariosController.

Enviara el nombre, y el texto para los comentarios con un botón de submit.

```
<body>
    <form action="ComentarioController" method="POST">
        Nombre: <input type="text" value="" name="nombre" placeholder="inserte su nombre">
        Inserte su comentario
        <textarea rows="3" cols="5" name="comentario"></textarea>
        <input type="submit">
        <input type="hidden" name="action" value="buscar">
    </form>
```

Y también se agregara un campo oculto de las acciones y comentarios, es la tabla de base de datos.

```
    <% if (session !=null){
            List comentarios = (List) session.getAttribute("comentarios");
            if (comentarios != null){
    %>

        <table border="1">
            <tr>
                <th>Nombre</th>
                <th>Comentario</th>
            </tr>
            <%
                for (Object o : comentarios){
                    ComentariosPOJO comentario = (ComentariosPOJO) o;
            %>
                <tr>
                    <td><%=comentario.getNombre()%></td>
                    <td><%=comentario.getComentario()%></td>
                </tr>
            <% } %>
        </table>
    <%      }
        } %>
```

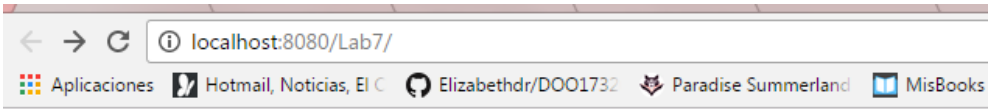
Luego crear un controlador, con el nombre que ya mencione anteriormente con un método de processRequest(), y la acción es comentar, que tengan como parámetros el nombre y el comentario. Agragar un POJO para los datos recibidos desde el formulario, e invocar el método para insertar pasándole el pojo como argumento.

Después de todo el proceso rediccionar la pagina a Buscar.jsp

```
ComentariosPOJO pojo = new ComentariosPOJO();
ComentariosDAO dao = new ComentariosDAO();

HttpSession session = request.getSession();

String action = request.getParameter("action");
String nombre = request.getParameter("nombre");
String comentario = request.getParameter("comentario");

pojo.setNombre(nombre);
pojo.setComentario(comentario);

if(action.equals("comentar")){
    dao.insertar(pojo);
    response.sendRedirect("buscar.jsp");
} else {
}
if(action.equals("search")){
    ArrayList<ComentariosPOJO> comentarios = dao.buscar(pojo);
    session.setAttribute("comentarios",comentarios);
    response.sendRedirect("buscar.jsp");
}
```

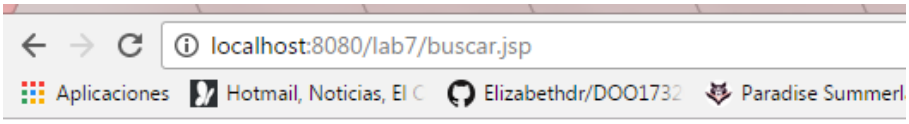
Después sigue la acción buscar, con los parámetros del nombre y comentario, construimos un POJO con lo mismo, invocar el método buscar del modelo y agregando la variable tipo List. Si hay un error enviarlo a la página error.jsp.

```
if(action.equals("comentar")){
    dao.insertar(pojo);
    response.sendRedirect("buscar.jsp");
} else {
}
if(action.equals("search")){
    ArrayList<ComentariosPOJO> comentarios = dao.buscar(pojo);
    session.setAttribute("comentarios",comentarios);
    response.sendRedirect("buscar.jsp");
}
else{
}
if(!"comentar".equals(action) & !"buscar".equals(action)){
response.sendRedirect("error.jsp");
    response.sendRedirect("error.jso");
}
```



Enviar comentario

Nombre: Diana Inserte su comentario Hola Enviar



Nombre: inserte su nombre Inserte su comentario Enviar

Nombre:	Comentario:
Diana	Hola
Diana	Hola

Pero como ultimo teníamos que agregar una clase Log, con los siguientes métodos

- `Log(String fileName)`: "Settear" la propiedad `fileName` en el constructor con el valor del argumento de entrada.

- `getInstance`: Validar si existe la instancia. Si existe, retornarla. De no existir, crear una nueva y retornarla.
- `write(String message)`

```
public class Log {
    final String fileName;
    private static Log instance;

    public Log(String fileName) {
        this.fileName = fileName;
    }

    public static  Log getInstance(String fileName){
        if(instance == null) {
            return new Log(fileName);
        }

            return instance;
    }
    public void write(String message){
        try (BufferedWriter br = new BufferedWriter(new FileWriter(fileName, true))) {
            DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            Calendar cal = Calendar.getInstance();

            //Create the name of the file from the path and current time
            String data = "\n" + dateFormat.format(cal.getTime()) + ": " + message ;
            br.write(data);
        }
        catch(Exception ex){

        }
    }
```

- ¿Qué ventajas identificas con el uso de un sistema de Logging de eventos?

Si lo hablamos aplicadamente, pues creo que nos ayudaría a un buen monitoreo de la entrada y los datos que se ingresan.

- ¿Qué ventajas tienes al utilizar una clase singleton?

Garantiza que en una sola clase tenga solo el uso de una instancia y podemos acceder por esa misma.

- ¿Qué "pros" y "contras" identificas al utilizar singleton vs clases estáticas?

Supongo que uno de ellos es el sobreescribir, ya que en el singleton podemos obtener los argumentos, y en la clase estática no ya que no podemos acceder a ella,

La clase singleton proporciona acceso global a los métodos, por así decirlo. Y las clases estáticas se dice que son más recomendadas (o al menos eso leí en internet) ya que cuando se quiere agrupar métodos siempre y cuando no se requieran un único acceso a algún recurso, en todo caso es así, se tendría que usar el singleton.