TÓPICOS AVANZADOS DE PROGRAMACIÓN
TEMA – Java + Patrones de Diseño (MVC)

OBJETIVO

- ✓ *Utilizar MySQL para manipulación de datos*
- ✓ *Trabajar con SQL como lenguaje de consulta*
- ✓ *Creación de la conexión de la base de datos en un proyecto NetBeans*
- ✓ *Uso de patrones de diseño en la construcción de aplicaciones*

REQUERIMIENTOS

SOFTWARE	VERSIÓN	LINK
Java Development Kit (JDK)	6	http://java.sun.com
NetBeans	6.1	http://www.netbeans.org
MySQL	5.0	http://dev.mysql.com/
Conector MySQL	-	http://dev.mysql.com/

DESCRIPCIÓN

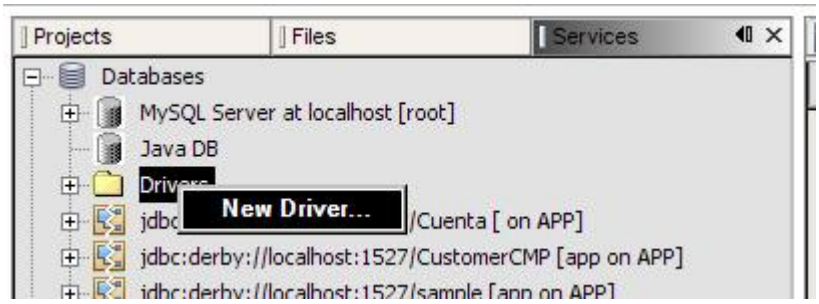
El siguiente tutorial tratará de explicar la construcción de aplicaciones usando patrones de diseño, específicamente, el Modelo Vista Controlador (MVC) aplicado a la construcción de una aplicación que permite consultar los datos de unas tablas en una base de datos. Se va a construir una aplicación de consola en Java dado que el énfasis está en la explicación de cómo trabajar con MVC. Junto a este tutorial se entrega el proyecto NetBeans que se obtiene siguiendo las actividades que se detallan.

ACTIVIDAD 1 – CREANDO EL PROYECTO

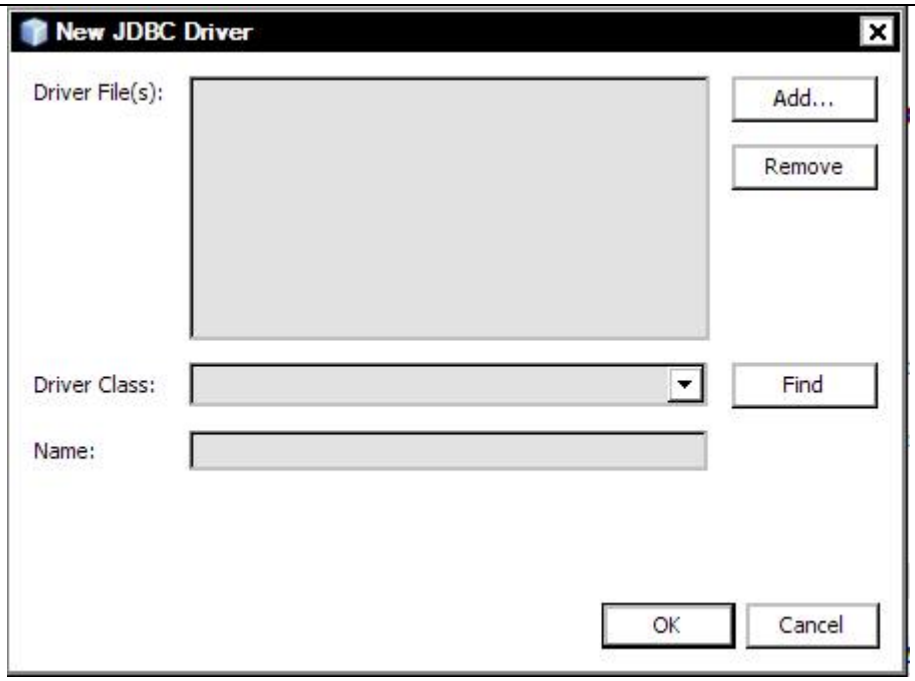
Crear un proyecto NetBeans del tipo Aplicación Java.

ACTIVIDAD 2 – VERIFICANDO LA EXISTENCIA DEL DRIVER DE CONEXIÓN

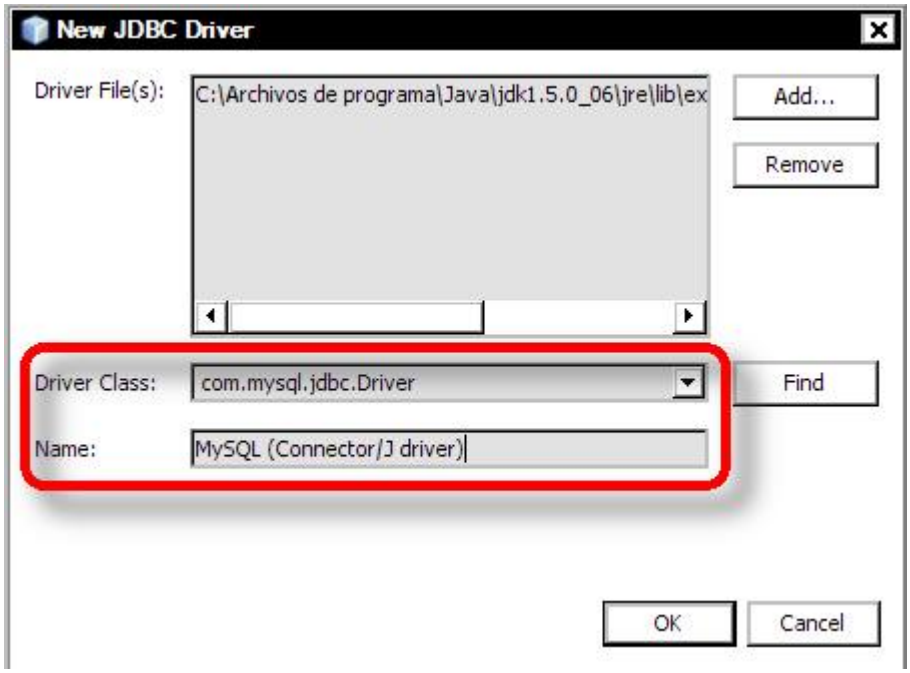
Para poder conectar una aplicación Java con la base de datos se requiere la existencia de un driver (conector). Primero que todo se debe verificar que este driver exista. En el caso de que no exista se debe agregar de acuerdo a como se indica en la siguiente figura (y como fue revisado en detalle en el tutorial anterior):



Al hacer clic en New Driver se despliega la siguiente ventana:



Se necesita el archivo que contiene el driver, normalmente es un archivo con extensión .JAR. Una vez que haya seleccionado el .JAR aparece el nombre del archivo .class y el nombre con el que se va a identificar el conector, tal y como lo muestra en la siguiente figura:

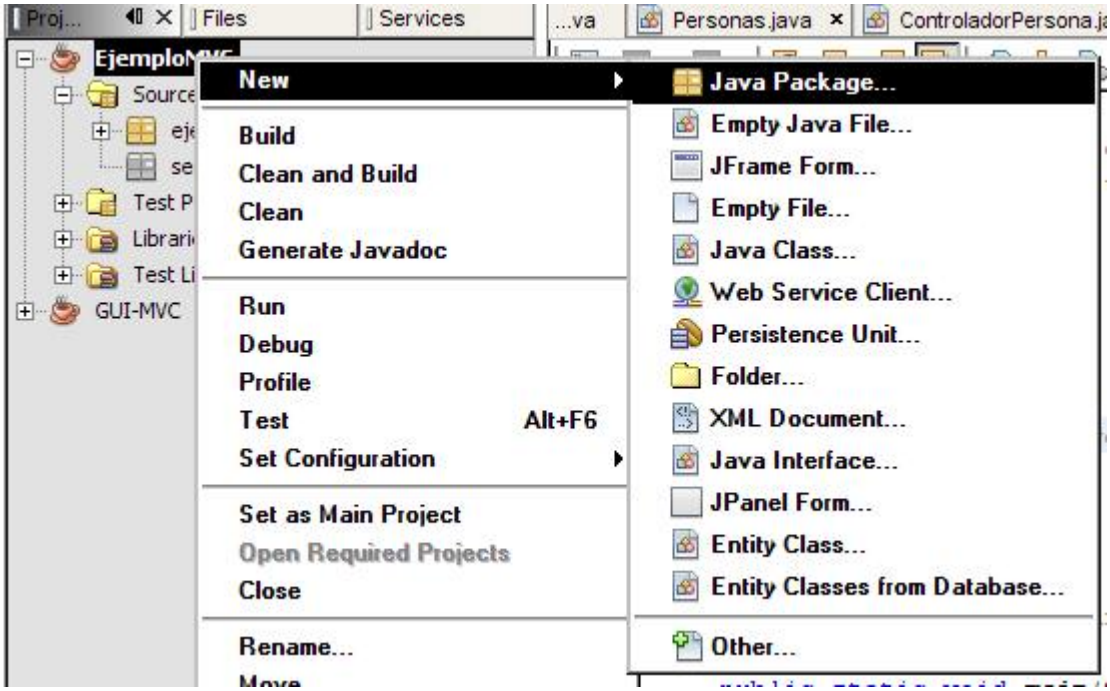


ACTIVIDAD 2 – CREACIÓN DE PAQUETES

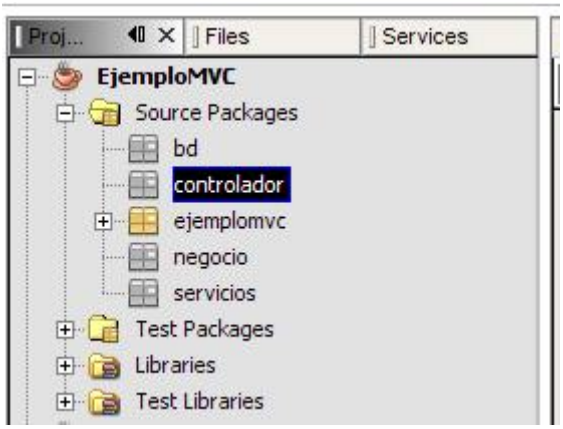
Vamos a dividir la aplicación en varios paquetes cuya existencia se justifica por agrupar las clases de acuerdo a los servicios que prestan. Se necesita agregar a la aplicación los siguientes paquetes:

servicios
bd
negocios
controlador
ui

La manera de crear el paquete es haciendo clic derecho sobre el proyecto, seleccionado New y luego Java Package tal y como se indica en la siguiente figura:



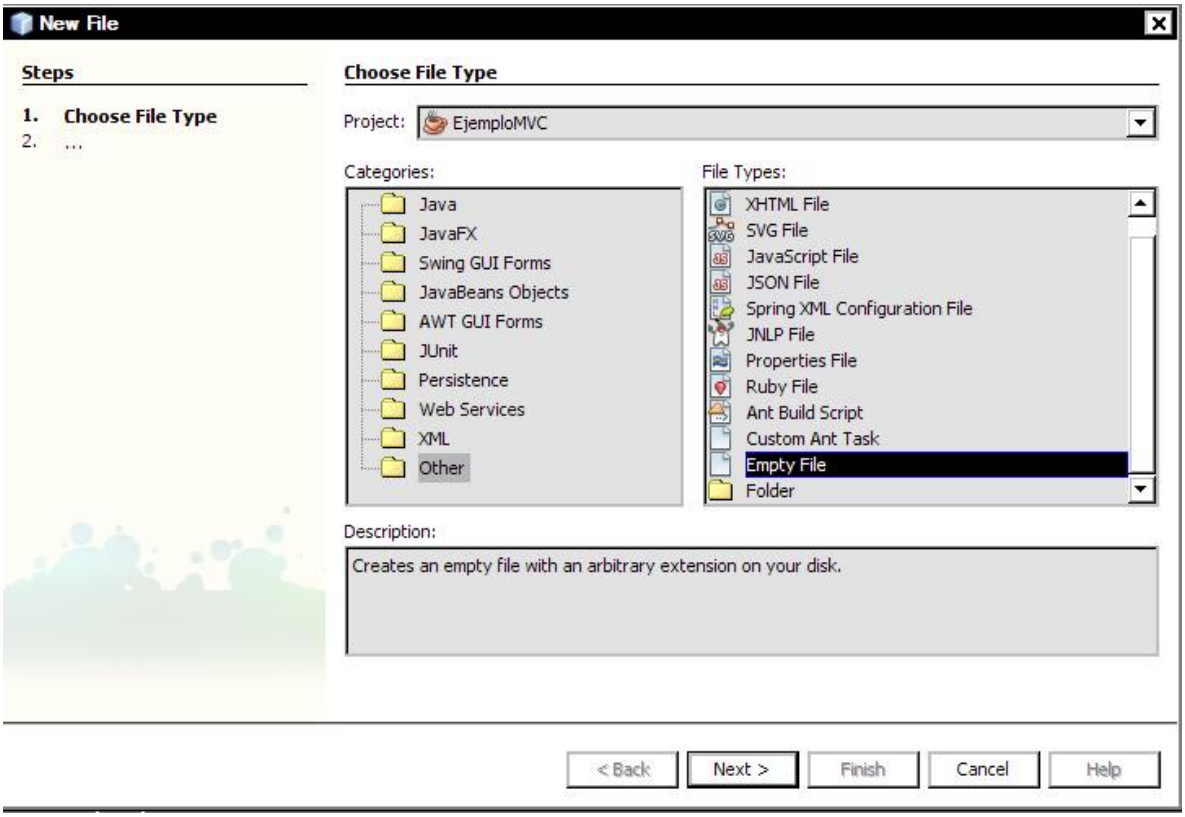
Una vez que se hayan creado todos los paquetes la estructura del proyecto debería estar como lo indica la siguiente figura:



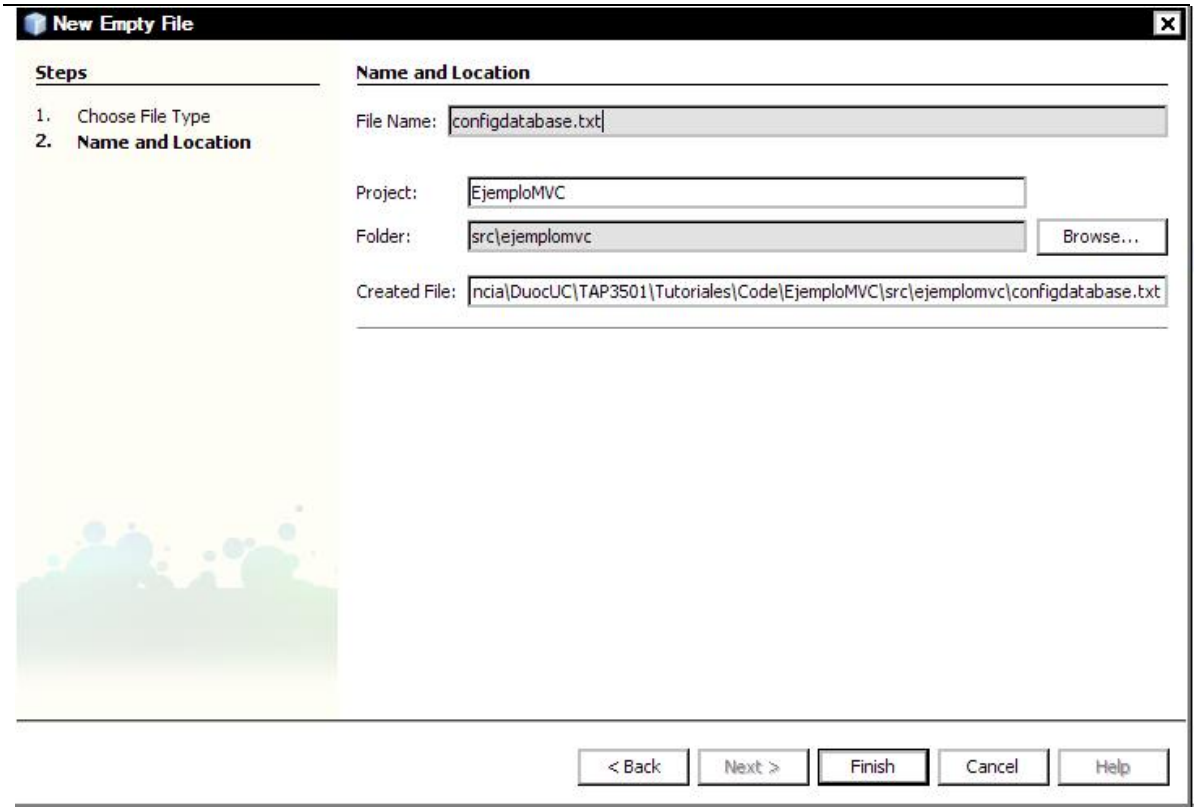
ACTIVIDAD 3 – CREACIÓN DE ARCHIVO DE CONFIGURACIÓN

De forma de independizar los parámetros de conexión vamos a crear un archivo llamado configdatabase.txt.

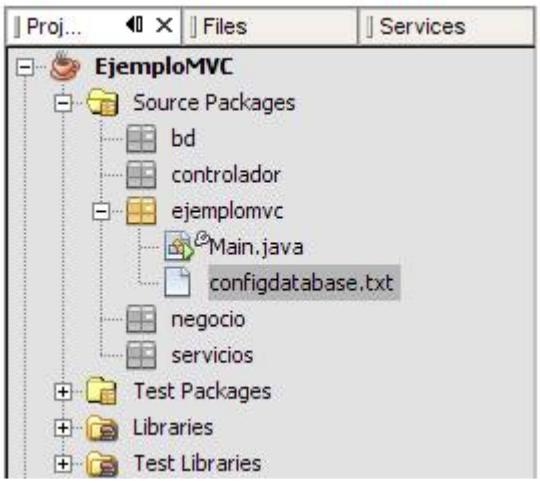
La forma de crear este archivo es haciendo clic derecho sobre el paquete principal (donde se encuentra la clase Main.java) del proyecto; luego seleccionar la opción New – Other, luego y tal como se indica en la siguiente figura seleccionar la categoría Other y luego seleccionar la opción Empty File:



Luego de hacer clic en el botón Next se obtiene una pantalla como la que se indica en la siguiente figura:



Al hacer clic en el botón Finís se crea el archivo dentro del paquete principal del proyecto. La estructura del proyecto ahora queda como se indica a continuación:



La idea es establecer en este archivo los valores de los parámetros de conexión con los cuales va a trabajar la aplicación, los parámetros son los que se indican en la siguiente tabla:

PARÁMETRO	VALOR
databaseclass	servicios.Conexion_MySQL
jdbc	jdbc:mysql://

driver	com.mysql.jdbc.Driver
databasehost	localhost
database	*<<nombre de la base de datos>>*
username	*<<usuario de conexión de MySQL>>*
password	*<<password de conexión de MySQL>>*
options	charSet=LATIN1
mostrarproperties	si

Es importante hacer notar la distinción con las mayúsculas y minúsculas. Debe definir los parámetros, tanto el nombre como el valor, tal y como se indica en la tabla anterior.

En el sector derecho se tiene el editor donde vamos a definir los parámetros que se han explicado en la tabla anterior y debería quedar como se indica a continuación:

```
1 databaseclass=servicios.Conexion_mySQL
2 jdbc=jdbc:mysql://
3 driver=com.mysql.jdbc.Driver
4 databasehost=localhost
5 database=persona
6 username=root
7 password=
8 options=charSet=LATIN1
9 mostrarproperties=si
```

Para efectos de este tutorial se va a utilizar la base de datos persona que contiene una sola tabla, cuya estructura se indica a continuación.

```
CREATE TABLE persona
(
      rut VARCHAR(12) NOT NULL,
      nombre VARCHAR(25),
      fono VARCHAR(15),
      edad int4,
      PRIMARY KEY (rut)
);
```

Se recomienda revisar el tutorial Java + MySQL para recordar la forma de crear la base de datos y ejecutar el script de creación que se acaba de entregar.

Hay algunos parámetros que se explican por sí solos y algunos de ellos tienen relación con lo que se revisó en el tutorial anterior, relacionado con el tema de crear una conexión a una base de datos en MySQL.

ACTIVIDAD 4 – ARQUITECTURA DE LA APLICACIÓN

La aplicación se organiza en varios paquetes cada uno de los cuales representa las capas en las cuales se organiza la aplicación. La idea de dividir la aplicación en capas obedece al hecho de independizar algunas tareas comunes a todas las aplicaciones, como son por ejemplo, las clases que se encargan de generar las conexiones con una base de datos, independiente del motor en la cual se encuentra.

Cada uno de los paquetes que fueron creados en la actividad anterior representa una capa que va a ofrecer servicios a las capas que se encuentran en los niveles superiores, de esta forma se obtiene cierto grado de independencia respecto de las tareas comunes a toda aplicación.

En las siguientes actividades se explica cada una de las clases que forman parte de estos paquetes y a medida que se avanza se va explicando la forma de actuar de la arquitectura que se ha definido.

ACTIVIDAD 5 – EXPLICANDO LAS CLASES – CAPA DE SERVICIOS

La capa de servicios cuenta con las siguientes clases:

CLASE	OBJETIVO
Conexion	Representa la conexión universal a cualquier motor de base de datos.
Conexion_mySQL	Representa la conexión específica a una base de datos que opera bajo MySQL. Es una extensión de la clase Conexion.
Error	Representa los errores de negocio que se van a definir dentro de la aplicación
factoriaServicios	Representa las instancias de servicio de conexión a la base de datos.

CLASE CONEXIÓN

Los atributos con los que cuenta la clase son los que se indican en la siguiente figura:

```
24  import java.sql.*;
25  public class Conexion {
26      public Connection conexion = null;
27      public Statement un_st = null;
28      public DatabaseMetaData dbmd;
29      public String s_conexion = null;
30      public ResultSet resultado = null;
31      public String un_sql = null;
32
33      protected String jdbc;
34      protected String driver;
35      protected String host;
36      protected String database;
37      protected String username;
38      protected String password;
39
```

Como pueden darse cuenta los atributos del tipo protected que se encuentran definidos tienen relación con los parámetros que se han definido en el archivo de configuración que fue definido en la actividad anterior.

Para obtener mayor información acerca de las clases/interfaces que aparecen en la definición de esta clase se sugiere revisar la API: http://java.sun.com/javase/6/docs/api/index.html

Al revisar el constructor se tiene:

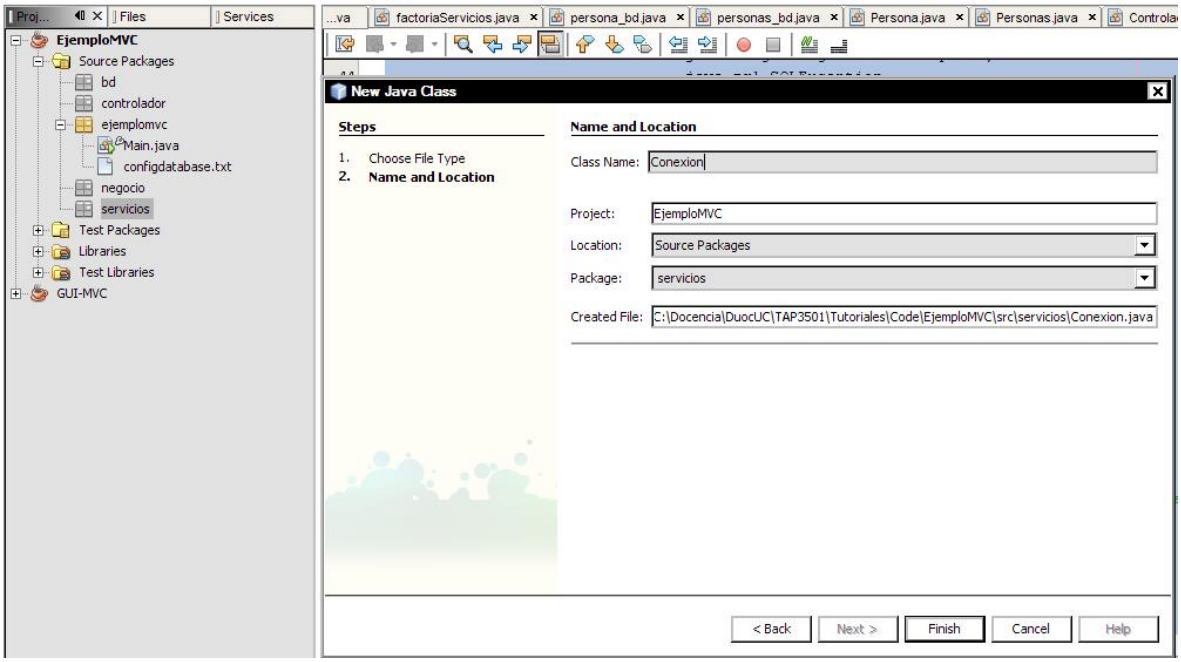
```
40      /** Creates a new instance of Conexion */
41      public Conexion() throws java.lang.ClassNotFoundException,
42                               java.lang.InstantiationException,
43                               java.lang.IllegalAccessException,
44                               java.sql.SQLException
45      {
46          jdbc =System.getProperty("jdbc");
47          driver = System.getProperty("driver");
48          host= System.getProperty("databasehost");
49          database = System.getProperty("database");
50          username = System.getProperty("username");
51          password = System.getProperty("password");
52
53          initdb();
54
55      }
```

Lo anterior va a permitir generar una instancia de conexión. Nuevamente se debe notar la relación que existe con los parámetros que fueron definidos dentro del archivo de configuración definido en la actividad anterior. Notar por ahora el uso del método ***getProperty()*** que permite obtener una propiedad (***property***) que se haya definido. Más adelante se explica la forma de definir y crear las ***properties*** con las cuales va a trabajar la aplicación.

Finalmente, se presenta el método ***initdb()*** el cual es invocado en el constructor para completar la definición de la clase.

```
57      public void initdb() throws java.lang.ClassNotFoundException,
58                                  java.lang.InstantiationException,
59                                  java.lang.IllegalAccessException,
60                                  java.sql.SQLException
61      {
62          s_conexion =jdbc + host + "/" + database;
63          Class.forName(driver).newInstance();
64          conexion = DriverManager.getConnection(jdbc + host + "/" + database, username, password);
65          dbmd = conexion.getMetaData();
66          un_st = conexion.createStatement();
67      }
```

Para crear la clase Conexion se debe hacer clic derecho en el paquete servicios y seleccionar New Java Class, tal y como se indica en la siguiente figura:



Haciendo clic en el botón Finish se crea el editor donde se agrega el código que se ha analizado en los párrafos anteriores.

CLASE Conexion_mySQL

Esta clase hereda de la clase Conexión, lo que se interpreta como un tipo de conexión especifica. El constructor de esta clase es el que se indica a continuación:

```
20  import java.sql.*;
21
22  public class Conexion_mySQL extends Conexion {
23
24      private String opciones;
25      /** Creates a new instance of Conexion_mySQL */
26      public Conexion_mySQL() throws java.lang.ClassNotFoundException,
27                                     java.lang.InstantiationException,
28                                     java.lang.IllegalAccessException,
29                                     java.sql.SQLException
30      {
31          opciones = System.getProperty("options");
32          initdb();
33
34      }
35
```

En el sólo se agrega el seteo del atributo opciones de acuerdo al valor de la propiedad que se obtiene y luego se llama al método ***initdb()*** el cual se encuentra sobreescrito tal y como se indica en la siguiente figura:

```
@Override
public void initdb() throws java.lang.ClassNotFoundException,
                            java.lang.InstantiationException,
                            java.lang.IllegalAccessException,
                            java.sql.SQLException
{
    s_conexion =jdbc + host + "/" + database; // + "?" + opciones;

    Class.forName(driver).newInstance();

    //aqui se usan las options, por eso este medio se redefine
    conexion = DriverManager.getConnection(jdbc + host + "/" + database + "?" +
                                           opciones, username, password);
    dbmd = conexion.getMetaData();
    un_st = conexion.createStatement();
}
```

Lo anterior permite sobrescribir el método que se encuentra en la clase padre de forma de definir el string de conexión de acuerdo al motor de base de datos que se está tratando (cada motor de base de datos maneja un string de conexión distinto, normalmente, se deben revisar los manuales para conocer la forma de definir el string de conexión en cada caso).

Analizando el código que se muestra en la figura anterior se tienen que en la línea 44 se genera una instancia anónima considerando el driver que se había definido previamente en el archivo de configuración.

En la línea 47 se considera la creación de la conexión, tomando como argumentos los valores de los atributos de la clase.

CLASE Error

La clase Error es bastante sencilla y sólo se limita a representar a los distintos errores de negocio que se van a definir dentro de la aplicación. El código de la clase se encuentra en la siguiente figura y no debería tener mayores explicaciones.

```java
package servicios;

/**
 *
 * @author Yasna Meza Hidalgo
 * Se crea una clase Error (en realidad una excepcion) especifica, que hereda de
 * Exception, para poder detectar y configurar nuestros propios errores y
 * para hacerlos "flotar" a las clases a nivel de GUI, si es que estas
 * exepciones ocurren a un nivel mas bajo.
 */
public class Error extends Exception{

    private int codigo;
    private String mensaje;

    /** Creates a new instance of Error */
    public Error(int codigo, String mensaje) {
        this.codigo = codigo;
        this.mensaje = mensaje;
    }

    public int codigo(){ return codigo; }

    public String mensaje(){ return mensaje; }

}
```

CLASE factoriaServicios

Esta clase representa la generación de las conexiones a la base de datos. La idea es mantener una sola conexión, evitando así el uso indebido de recursos al utilizar, por ejemplo, una conexión independiente para cada petición que se tenga.

```java
package servicios;

/**
 *
 * @author Yasna Meza Hidalgo
 */
public class factoriaServicios {

    /*
        Instancia del servicio a base de datos.
        este puntero puede aceptar cualquier clase que herede de Conexion, por
        ejemplo conexion mySQL
     */
    private Conexion cdb;

    /* Un atributo compartido para todas las instancias de la clase factoria de servicios */
    private static factoriaServicios instancia;

    /** Creates a new instance of factoriaServicios */
    public factoriaServicios() {
    }
```

En la figura anterior es posible destacar la definición de los dos atributos ***cdb*** e ***instancia*** que corresponden a la conexión sobre la cual se va a trabajar y a la instancia del servicio que se va a ofrecer.

A continuación se describen dos métodos que se han definido en la clase que permiten obtener la instancia y la conexión a la base de datos.

```
32      /* El constructor se asegura que se cree una sola instancia de la propia factoria */
33      public static synchronized factoriaServicios getInstancia()
34      {
35          if (instancia == null)
36          {
37              instancia = new factoriaServicios();
38          }
39          return instancia;
40      }
```

En este método se verifica si la instancia existe previamente, en caso de que exista se retorna el objeto que se tiene y en caso contrario se crea.

```
42      public Conexion getConexionDb() throws java.lang.ClassNotFoundException,
43                                     java.lang.InstantiationException,
44                                     java.lang.IllegalAccessException
45
46      {
47          if (cdb == null)
48          {
49              /*
50                  Esta propiedad se lee desde el archivo de propiedades
51                  y se mantiene en memoria. Esto se hizo en la main class
52               */
53              String nombreClase = System.getProperty("databaseclass");
54
55              /*
56                  Aqui se crea la instancia de la clase especifica de base
57                  de datos con la que queremos trabajar (las cuales
58                  deben heredar de Conexion).
59               */
60              cdb = (Conexion) Class.forName(nombreClase).newInstance();
61          }
62          return cdb;
63      }
```

En este caso se verifica que la conexión exista en caso de que así sea entonces se retorna, de lo contrario se crear la conexión de acuerdo a lo definido en la propiedad ***databaseclass*** definida dentro del archivo configdatabase definido en una de las actividades anteriores.

CLASE PersonaBD

Esta clase representa a la persona como un registro en una tabla de la base de datos. En la siguiente figura se aprecia la definición de atributos y el constructor de la clase:

```
10  package bd;
11
12  /**
13   *
14   * @author Yasna Meza Hidalgo
15   */
16
17  import servicios.Error;
18  import servicios.factoriaServicios;
19  import servicios.Conexion;
20
21  import negocio.Persona;
22
23  public class PersonaBD {
24
25      private Persona p;
26
27      public PersonaBD(Persona p)
28      {
29          this.p = p;
30      }
```

Esta clase cuenta con tres métodos que dicen relación con las operaciones asociadas a un registro de la tabla: grabar, leer y borrar. Las siguientes figuras muestran la implementación de cada método.

```
33      public void grabar() throws java.lang.ClassNotFoundException,
34                                  java.lang.InstantiationException,
35                                  java.lang.IllegalAccessException,
36                                  java.sql.SQLException
37      {
38          Conexion cdb = factoriaServicios.getInstancia().getConexionDb();
39
40          cdb.un_sql = "SELECT rut FROM persona WHERE rut = '"+ p.getRut() +"'";
41          cdb.resultado = cdb.un_st.executeQuery(cdb.un_sql);
42          if (cdb.resultado != null){
43              if (cdb.resultado.next()){
44                  cdb.un_sql = "UPDATE persona SET nombre ='" + p.getNombre() + "', fono ='" + p.getFono() +
45                               "', edad =" + p.getEdad()+" WHERE rut='" +  p.getRut()+ "'";
46                  cdb.un_st.executeUpdate(cdb.un_sql);
47              }
48              else{
49                  cdb.un_sql = "INSERT INTO persona VALUES ('" + p.getRut() + "', '" + p.getNombre() +
50                               "', '" + p.getFono() + "', " + p.getEdad() + ")";
51                  cdb.un_st.executeUpdate(cdb.un_sql);
52              }
53          }
54          else{
55              cdb.un_sql = "INSERT INTO persona VALUES ('" + p.getRut() + "', '" + p.getNombre() + "', '" +
56                           p.getFono() + "', " + p.getEdad() + ")";
57              cdb.un_st.executeUpdate(cdb.un_sql);
58          }
59      }
```

Vamos a comentar algunas líneas de código de este método.

En la línea 38 se crea el objeto asociado a la conexión a la base de datos, se utiliza la clase factoriaServicios que se encuentra en el paquete servicios que se ha definido anteriormente.

Las líneas 40 y 41 setean los atributos de la clase conexión que tienen relación con la consulta que se va a ejecutar y el resultado de la misma. La consulta SQL no es más que un simple string que contiene código SQL. Para poder ejecutar la consulta se utiliza el método ***executeQuery()***.

En la línea 42 se comprueba el éxito de la consulta SQL para ver si el proceso de grabar el registro se debe traducir como una operación de inserción (INSERT) o como una operación de actualización (UPDATE). Dependiendo del tipo de grabación del que se trata se procede a setear la consulta SQL y a ejecutarla a través del método ***executeUpdate()***.

```
62      public Persona leer() throws java.lang.ClassNotFoundException,
63                                   java.lang.InstantiationException,
64                                   java.lang.IllegalAccessException,
65                                   java.sql.SQLException,
66                                   Error
67
68      {
69          Conexion cdb = factoriaServicios.getInstancia().getConexionDb();
70
71          cdb.un_sql = "SELECT * FROM persona WHERE rut = '"+ p.getRut() +"'";
72          cdb.resultado = cdb.un_st.executeQuery(cdb.un_sql);
73          if (cdb.resultado != null){
74              if (cdb.resultado.next()){
75                  p.setNombre(cdb.resultado.getString("nombre"));
76                  p.setFono(cdb.resultado.getString("fono"));
77                  p.setEdad(cdb.resultado.getInt("edad"));
78              }
79              else{
80                  /* Generamos nuestro propio error, luego se activa el catch quien lo lanza nuevamente */
81                  throw new Error(-1, "Registro " + p.getRut() + " no encontrado en tabla persona. Ubicación=" +
82                                      this.getClass().getName());
83              }
84          }
85          else{
86              /* Generamos nuestro propio error, luego se activa el catch quien lo lanza nuevamente */
87              throw new Error(-1, "Consulta a Registro " + p.getRut() +
88                                  " ha devuelto un recordset null Ubicación=" + this.getClass().getName());
89          }
90          return p;
91      }
```

El método ***leer()*** tiene como objetivo obtener, a partir del RUT, los datos asociados a una persona. La explicación de las líneas 69, 71 y 72 tienen la misma explicación que se entregó en el método grabar.

Las líneas 75 a la 77 obtienen los datos a partir de los resultados de la consulta. Se hace uso del método ***getString()*** de la interfaz ***Statement*** (revisar la API para obtener mayores detalles). Observar que se lanzan las excepciones a partir de la clase Error que se encuentra definida en el paquete servicios.

Finalmente, se tiene el método ***borrar()*** cuya implementación se encuentra en la siguiente figura:

```
94      public void borrar() throws java.lang.ClassNotFoundException,
95                                  java.lang.InstantiationException,
96                                  java.lang.IllegalAccessException,
97                                  java.sql.SQLException
98      {
99          Conexion cdb = factoriaServicios.getInstancia().getConexionDb();
100
101         cdb.un_sql = "DELETE FROM persona WHERE rut = '"+ p.getRut() +"'";
102         cdb.un_st.executeUpdate(cdb.un_sql);
103     }
```

En este caso, nuevamente, se setean los valores de los atributos de la clase Conexión. En este caso la consulta que se va a ejecutar corresponde a la sentencia SQL DELETE asociada a la eliminación de un registro de la tabla.

En este paquete se encuentra incluida la clase PersonasBD cuyo objetivo es representar a un conjunto de personas. Esta clase cuenta con dos métodos asociados a leer al grupo completo de registros y a leer a un grupo de registros de acuerdo a un filtro asociado a un campo específico

de la tabla. Ambos método retornan un ArrayList con los objetos que han sido creados a partir de los datos que han sido extraídos de la tabla.

En las siguientes figuras se muestra una porción de código de cada método. El código completo lo puede analizar en el proyecto que le fue entregado junto a este tutorial.

En el caso del método ***leer()*** se tiene una consulta SQL general que permite retornar a todas las personas ordenadas por nombre.

```
27      public ArrayList leer() throws java.lang.ClassNotFoundException,
28                                     java.lang.InstantiationException,
29                                     java.lang.IllegalAccessException,
30                                     java.sql.SQLException,
31                                     Error
32      {
33          ArrayList r;
34          Persona p;
35
36          Conexion cdb = factoriaServicios.getInstancia().getConexionDb();
37
38          cdb.un_sql = "SELECT * FROM persona ORDER BY nombre";
39          cdb.resultado = cdb.un_st.executeQuery(cdb.un_sql);
```

Al sobrecargar el método ***leer()*** se tiene la aplicación de un filtro de acuerdo a un determinado campo y se tiene:

```
66      public ArrayList leer(String filtro, String campo) throws java.lang.ClassNotFoundException,
67                                     java.lang.InstantiationException,
68                                     java.lang.IllegalAccessException,
69                                     java.sql.SQLException,
70                                     Error
71      {
72          ArrayList r;
73          Persona p;
74
75          Conexion cdb = factoriaServicios.getInstancia().getConexionDb();
76
77          cdb.un_sql = "SELECT * FROM persona WHERE " + campo + " like '%" + filtro + "%' ORDER BY nombre";
78          cdb.resultado = cdb.un_st.executeQuery(cdb.un_sql);
```

En el caso del método leer sobrecargado, que utiliza filtro, se compone la consulta SQL con los parámetros que recibe.

CLASE Persona

Esta clase representa a la persona desde el punto de vista de objeto. Contiene el tradicional grupo de métodos setter y getter. Además cuenta con los métodos ***grabar()***, ***leer()*** y ***borrar()*** que dicen relación con las operaciones relacionadas con la tabla en la base de datos.

```
194        public void grabar() throws java.lang.ClassNotFoundException,
195                                    java.lang.InstantiationException,
196                                    java.lang.IllegalAccessException,
197                                    java.sql.SQLException,
198                                    Error
199        {
200            PersonaBD pdb = new PersonaBD(this);
201            pdb.grabar();
202        }
203
204
205        public Persona leer() throws java.lang.ClassNotFoundException,
206                                    java.lang.InstantiationException,
207                                    java.lang.IllegalAccessException,
208                                    java.sql.SQLException,
209                                    Error
210        {
211            PersonaBD pdb = new PersonaBD(this);
212            return pdb.leer();
213        }
214
215
216        public void borrar() throws java.lang.ClassNotFoundException,
217                                    java.lang.InstantiationException,
218                                    java.lang.IllegalAccessException,
219                                    java.sql.SQLException
220        {
221
222            PersonaBD pdb = new PersonaBD(this);
223            pdb.borrar();
224        }
```

En la figura anterior se aprecia que cada uno de los métodos crea un objeto del tipo PersonaBD e invoca al método asociado a la operación en la tabla; en términos simples, está haciendo uso de los servicios que la capa de BD ofrece.

En esta clase además se encuentra la clase Personas que representa al conjunto de objetos que pueden ser obtenidos a partir de los registros de la tabla de la base de datos. Cuenta con un método sobrecargado que hace uso de los servicios que provee la capa de BD.

```
27      public ArrayList leer() throws java.lang.ClassNotFoundException,
28                                     java.lang.InstantiationException,
29                                     java.lang.IllegalAccessException,
30                                     java.sql.SQLException,
31                                     Error
32      {
33          PersonasBD psdb = new PersonasBD();
34          return psdb.leer();
35      }
36
37
38      public ArrayList leer(String filtro, String campo) throws java.lang.ClassNotFoundException,
39                                     java.lang.InstantiationException,
40                                     java.lang.IllegalAccessException,
41                                     java.sql.SQLException,
42                                     Error
43      {
44          PersonasBD psdb = new PersonasBD();
45          return psdb.leer(filtro, campo);
46      }
```

ACTIVIDAD 8 – EXPLICANDO LAS CLASES – CAPA CONTROLADOR

Hasta ahora se han explicado las clases de los paquetes subyacentes de la aplicación, clases destinadas a prestar servicios y que modelan el problema. Estas clases tienen relación con la capa llamada MODELO dentro del patrón de diseño MVC (Modelo Vista Controlador). Por otro lado, debería existir una capa relacionada con la interfaz gráfica, lo que el usuario va a ver, a esta capa se le conoce con el nombre de VISTA (se revisará en la siguiente actividad).

En esta capa se incluye la clase ControladorPersona que hace las veces de vínculo entre el modelo y la interfaz gráfica (independiente del tipo de interfaz gráfica de la que se trate). Con esta clase es con la que se debería conectar las clases de la interfaz gráfica, esto a través de los servicios que se van a ofrecer. Esta clase cuenta con cuatro métodos que son los que se describen a continuación:

```
29      public ArrayList buscarpersonas(String filtro,
30                                      String campo)
31                                      throws java.lang.ClassNotFoundException,
32                                             java.lang.InstantiationException,
33                                             java.lang.IllegalAccessException,
34                                             java.sql.SQLException,
35                                             Error
36      {
37          Personas ps;
38          ps = new Personas();
39          return ps.leer(filtro, campo);
40      }
```

El método ***buscarpersonas(String, String)*** permite, conociendo el nombre del campo y el valor por el cual desea filtrar los registros, retornar una lista con los objetos que cumplen con ese requisito dentro de la tabla. Notar el uso de las clases que se encuentran en la capa de negocio las que, como se vio con anterioridad, van a utilizar los servicios de la capa de BD, las que a su vez van a utilizar las clases definidas dentro

de la capa de servicios. De esta forma se arma la cadena de interacción entre las capas lo que va a permitir el uso de la aplicación.

```
44      public void agregarpersona(String prut,
45                                 String pnombre,
46                                 String pfono,
47                                 String pedad)
48                                 throws java.lang.ClassNotFoundException,
49                                        java.lang.InstantiationException,
50                                        java.lang.IllegalAccessException,
51                                        java.sql.SQLException,
52                                        Error
53      {
54          Persona p = new Persona(prut, pnombre, pfono, pedad);
55          p.grabar();
56      }
```

El método ***agregarpersona(String, String, String, String)*** recibe como argumentos los antecedentes para crear a una persona. Notar, nuevamente, que hace uso de las clases de negocio para poder cumplir con su funcionalidad.

```
60      public void modificarpersona(String prut,
61                                   String pnombre,
62                                   String pfono,
63                                   String pedad)
64                                   throws java.lang.ClassNotFoundException,
65                                          java.lang.InstantiationException,
66                                          java.lang.IllegalAccessException,
67                                          java.sql.SQLException,
68                                          Error
69      {
70          Persona p = new Persona(prut);
71          p.actualizar(pnombre, pfono, pedad);
72          p.grabar();
73      }
```

Por su parte el método modificarpersona(String, String, String, String) es bastante similar al método anterior.

Finalmente, están los métodos asociados a eliminar a una persona y a recargar la lista de acuerdo a lo que se tiene en la tabla de la base de datos.

```
77      public void  eliminarpersona(String rut)
78                                   throws java.lang.ClassNotFoundException,
79                                          java.lang.InstantiationException,
80                                          java.lang.IllegalAccessException,
81                                          java.sql.SQLException,
82                                          Error
83      {
84          Persona p;
85          p = new Persona(rut);
86          p.borrar();
87      }
88
89
90
91      public ArrayList recargarlistapersona()
92                                          throws java.lang.ClassNotFoundException,
93                                                 java.lang.InstantiationException,
94                                                 java.lang.IllegalAccessException,
95                                                 java.sql.SQLException,
96                                                 Error
97      {
98          PersonasBD psdb = new PersonasBD();
99          return psdb.leer();
100     }
```

ACTIVIDAD 9 – EXPLICANDO LAS CLASES – CAPA UI

En este punto se hace necesaria la existencia una aplicación que permita ver el funcionamiento de las clases. A esta capa se le conoce con el nombre de VISTA dentro del modelo MVC (Modelo Vista Controlador).

En esta capa se encuentra la clase PersonaConsola. Esta clase tiene como objetivo mostrar a través de la salida estándar el resultado de la operatoria común sobre los datos de la tabla. Cuenta con un objeto de la clase ControladorPersona sobre el cual va a operar.

```
13  import java.util.ArrayList;
14  import java.util.Iterator;
15
16  import controlador.ControladorPersona;
17  import negocio.Persona;
18  import servicios.Error;
19
20  public class PersonaConsola {
21      private ControladorPersona cp;
22
23      public PersonaConsola(ControladorPersona cp) throws
24                                      java.lang.ClassNotFoundException,
25                                      java.lang.InstantiationException,
26                                      java.lang.IllegalAccessException,
27                                      java.sql.SQLException,
28                                      Error
29
30      {
31          this.cp = cp;
32          this.recargarpersonas();
33      }
```

Cuando se crea la instancia de la clase PersonaConsola se va a desplegar en la salida estándar el contenido de la tabla, esto lo logra a través de la llamada al método ***recargarpersonas()*** cuya implementación se entrega a continuación:

```
73      private void recargarpersonas() throws
74                                      java.lang.ClassNotFoundException,
75                                      java.lang.InstantiationException,
76                                      java.lang.IllegalAccessException,
77                                      java.sql.SQLException,
78                                      Error
79      {
80          ArrayList r;
81          /* Obtiene la lista de los datos almacenados en la tabla */
82          r = cp.recargarlistapersona();
83          Iterator it = r.iterator();
84          while(it.hasNext()){
85              /* Extrae el objeto de la colección y muestra sus datos en la salida estándar */
86              Persona p = (Persona) it.next();
87              System.out.println(p.getRut() + "-" + p.getNombre() + "-" + p.getFono() + "-" +
88                                      String.valueOf(p.getEdad()));
89          }
90      }
```

Notar que en la línea 82 se está haciendo uso de los servicios que provee el controlador para poder cumplir con la funcionalidad pedida.

```
35      /* Muestra toda la funcionalidad asociada a la aplicación */
36      public void accionar() throws
37                                      java.lang.ClassNotFoundException,
38                                      java.lang.InstantiationException,
39                                      java.lang.IllegalAccessException,
40                                      java.sql.SQLException,
41                                      Error
42      {
43          ArrayList r;
44          Iterator iterador;
45          Persona p;
46
47          /* Prueba la funcionalidad de agregar una persona */
48          System.out.println("Agrega un nuevo registro...");
49          cp.agregarpersona("88.888.888-8", "Nueva Persona", "123456", "31");
50
51          /* Muestra la nueva lista de personas */
52          this.recargarpersonas();
53
54          /* Prueba la funcionalidad asociada a la búsqueda */
55          r = cp.buscarpersonas("33.333.333-3", "rut");
56          System.out.println("Resultado de la busqueda ... " + r.size() + " registros");
57          iterador = r.iterator();
58          while (iterador.hasNext()){
59              p = (Persona) iterador.next();
60              System.out.println(p.getNombre() + " " + p.getFono() + " " + p.getEdad());
61          }
62
63          /* Prueba la funcionalidad de eliminar registros */
64          System.out.println("Eliminando el registro que acaba de agregar ....");
65          cp.eliminarpersona("88.888.888-8");
66
67          /* Muestra la nueva lista de personas */
68          this.recargarpersonas();
69      }
```

El método ***accionar()*** es el encargado de mostrar el uso de toda la funcionalidad asociada. Agrega un nuevo registro, muestra el contenido de la tabla después de la inserción, busca los antecedentes de una persona dado el rut y luego elimina el registro que había insertado al principio de la prueba.

ACTIVIDAD 10 – EJECUTANDO PROYECTO

Para finalizar se va a incluir en la clase Main.java el código necesario para lanzar la vista de la aplicación a partir de la cual se comienza a generar las llamadas.

Lo primero que se debe hacer es cargar los datos del archivo de configuración que se había creado en unas de las primeras actividades de este tutorial.

```
28          /* Lee archivo de configuracion */
29          try{
30              FileInputStream propFile = new FileInputStream(System.getProperty("user.dir")+
31                                                             "\\src\\ejemplomvc\\configdatabase.txt");
32              Properties p = new Properties(System.getProperties());
33              p.load(propFile);
34              System.setProperties(p);
35              if (System.getProperty("mostrarproperties").compareTo("si")==0){
36                  System.getProperties().list(System.out);
37              }
38          }
39          catch(java.io.FileNotFoundException e){
40              System.out.println("No se encuentra el archivo de configuracion configdatabase.txt ");
41              System.exit(-1);
42          }
43          catch(java.io.IOException e){
44              System.out.println("Ocurrio algun error de I/O.");
45              System.exit(-1);
46          }
```

Las líneas de código anteriores cargan las propiedades definidas en el archivo de configuración configdatabase.txt.

La línea 33 es la encargada de cargar las propiedades (properties) que se encuentran en el archivo.

La línea 35 verifica si la propiedad mostrarpropierties se encuentra activada, en caso de que sea así entonces va mostrando por la salida estándar el valor de cada una de esas propiedades.

Las líneas 39 a la 46 muestran el tratamiento de las excepciones respecto de los errores que se pueden producir al tratar de acceder al archivo.

Lo que corresponde hacer ahora es tratar de acceder a la base de datos. Para ello se utiliza la clase factoriaServicios que se encuentra en la capa de servicios y se tiene:

```
48          /* Comprueba acceso a base de datos */
49          try{
50              Conexion cdb = factoriaServicios.getInstancia().getConexionDb();
51          }
52          catch(java.lang.ClassNotFoundException e){
53              System.out.println("Ocurrio la exception " + e.toString());
54              System.out.println("Es probable que no se puede encontrar la clase del conector jdbc " +
55                                 System.getProperty("driver")+
56                                 ". Agreguela a su classpath con la opcion -cp.");
57              System.exit(-1);
58          }
59          catch (java.lang.InstantiationException e){
60              System.out.println("Ocurrio un error de Instanciacion. " + e.toString());
61              System.exit(-1);
62          }
63          catch (java.lang.IllegalAccessException e){
64              System.out.println("Ocurrio un error de acceso ilegal. " + e.toString());
65              System.exit(-1);
66          }
```

En la línea 50 se encuentra la llamada a los métodos para obtener la conexión a la base de datos.

Las líneas 52 a la 66 se encargan de tratar las excepciones que pudieran darse al tratar de obtener la conexión con la base de datos.

```
68          /* Comienza a ejecutar la aplicación */
69          try{
70              ControladorPersona cp = new ControladorPersona();
71
72              /* Creamos el objeto que representa la aplicación
73               * En caso de que se trate de una aplicación GUI acá debería crear la instancia del JFrame
74               */
75              PersonaConsola pc = new PersonaConsola(cp);
76              pc.accionar();
77          }
```

En esta primera parte se crea el controlador y la clase que utiliza la funcionalidad.

```
78            catch(Error e){
79                System.out.println("Ocurrio el error "+  e.mensaje());
80                System.exit(-1);
81            }
82            catch (java.sql.SQLException e){
83                System.out.println("Ocurrio una excepcion SQL. " + e.toString());
84                System.exit(-1);
85            }
86            catch(java.lang.ClassNotFoundException e){
87                System.out.println("Ocurrio la exception " + e.toString());
88                System.exit(-1);
89            }
90            catch (java.lang.InstantiationException e){
91                System.out.println("Ocurrio un error de Instanciacion. " + e.toString());
92                System.exit(-1);
93            }
94            catch (java.lang.IllegalAccessException e){
95                System.out.println("Ocurrio un error de acceso ilegal. " + e.toString());
96                System.exit(-1);
97            }
```

Finalmente, se tratan las excepciones que pudieran generarse cuando se trata de acceder a los datos almacenados en la tabla.

CONTACTO Y COMENTARIOS

Cualquier duda o comentario dirigirlo a yasna.meza@gmail.com