Lógica de negocios

I- Objetivos

- Diseñar la lógica de negocios para el desarrollo de una aplicación.
- Implementar la lógica de negocios en formularios de inicios de sesión.
- Establecer opciones con permisos para usuarios de la aplicación.

II- Contenido

Introducción

El desarrollo de aplicaciones requiere que se dedique tiempo para el manejo de usuarios y permisos que estos tendrán con respecto a las opciones que el sistema ejecutará. De manera que los accesos se manejen en función de perfiles previamente definidos por el administrador de la aplicación. Esto se traduce en la creación de tablas dentro de la base de datos que almacenarán información sobre los usuarios, grupos de usuarios, opciones del menú y permisos que los usuarios tendrán con respecto a las opciones.

Por otro lado el manejo de las operaciones con la base de datos (Lógica de Negocio), debe estructurarse de manera que satisfaga las exigencias de los procesos. Atendiendo a los principios que demandan las arquitecturas distribuidas.

Arquitecturas Distribuidas

Una aproximación elemental al JDBC implica que se realiza una conexión a la base de datos en cada servlet. Se repite el esquema **conexión – operación – desconexión.** Esta forma de trabajar es perfectamente válida, pero resulta **ineficiente,** ya que están desperdiciando ciclos de ejecución en cada conexión y desconexión.

En vez de esta orientación hay otra alternativa: crear un pool de conexiones, es decir, mantener un conjunto de conexiones. Cuando un servlet/JSP requiere una conexión la solicita al conjunto de conexiones disponibles (desocupadas, “idle”) y cuando no la va a usar la devuelve al pool. De esta forma se ahorra el consumo de tiempo de conexión y desconexión. La mayor parte de servicios de aplicaciones incluyen clases que implementan un pool de conexiones. Concretamente con DataBase Common Pooling (DBCP) de Tomcat. El pool debe gestionar situaciones anómalas, como el cierre involuntario de una conexión después de cerrar un ResultSet o un Statement.

Definir Origen de datos

El DataSource u origen de datos se define en el JNDI, gracias a este objeto se puede acceder a otros objetos definidos en el archivo de configuración del proyecto web denominado “web.xml”. Una vez

agregado el objeto **JNDI para acceder a un pool de conexiones.** El tipo de dato que utilizamos en web.xml es un **DataSource,** una representación genérica de una fuente de datos.

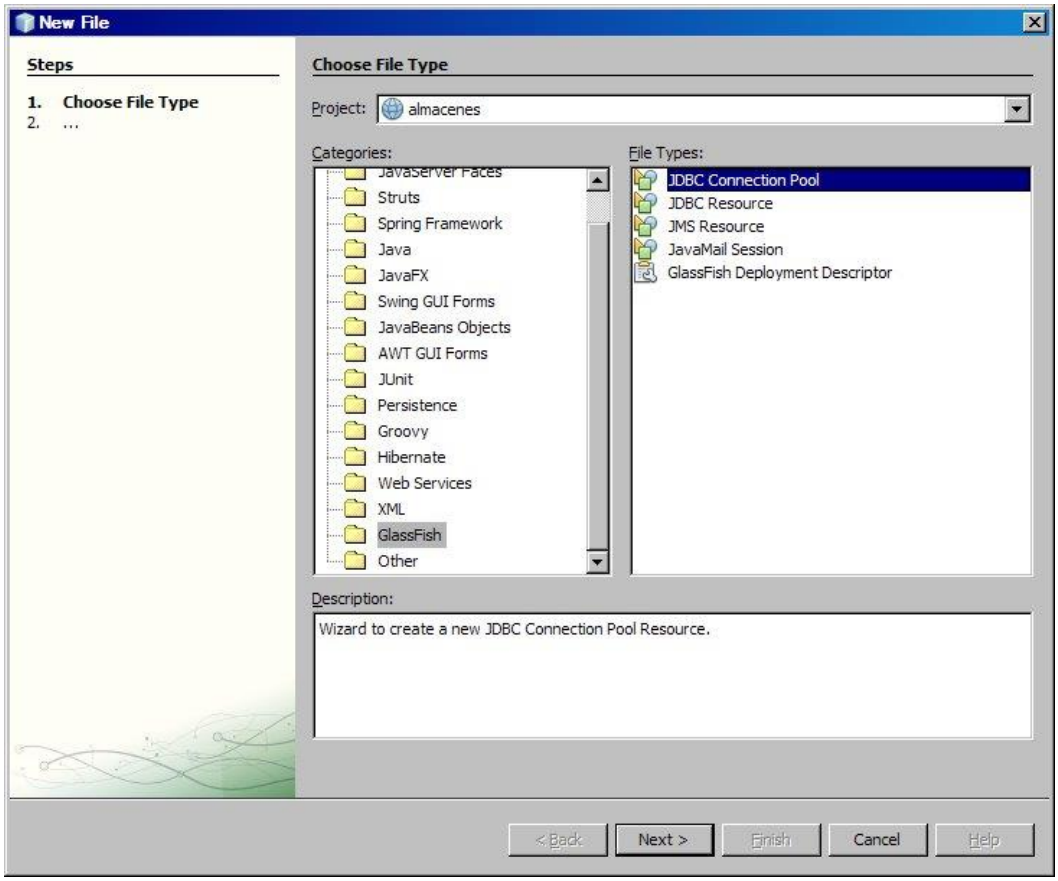
Nota: para continuar deberá estar creada la base de datos aerolínea proporcionada en el script. Y el proyecto creado llamado AerolineaProject.

Creando el Pool de Conexiones

Desde NetBeans

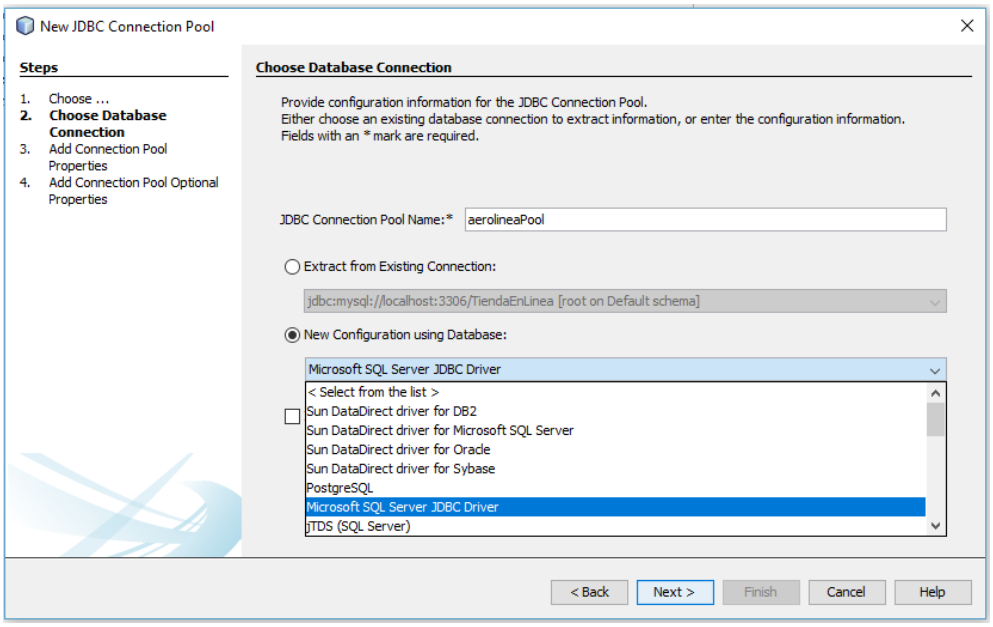
Para realizarlo desde NetBeans, necesitamos tener el proyecto que usará la base de datos abierto. Luego, presionamos Ctrl+N (File > New File)

1. Seleccionamos la categoría *Glassfish* y el tipo de archivo *JDBC Connection Pool*



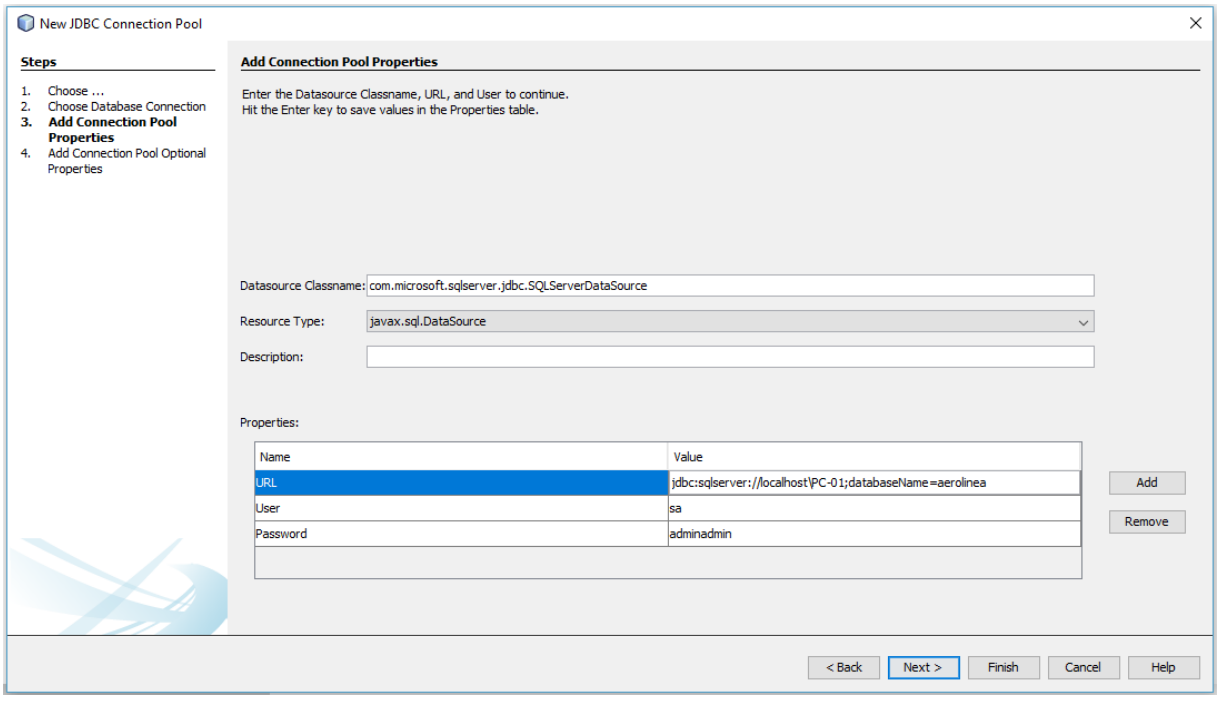
Clic en *Next*

2. Escribimos el nombre de nuestro pool de conexiones. De igual manera, cualquier no es válido, pero se recomienda que sea en base a letras y números. Además seleccionamos cual es la base de datos que usaremos. Como es una conexión nueva, entonces seleccionamos "New configuration using database", y seleccionamos para este caso para Microsoft SQL Server JDBC Driver:



Clic en *Next*

3. Seleccionamos el tipo de recurso (por omisión es java.sql.DataSource, escribimos el URL de nuestra conexión JDBC, el usuario y la contraseña).



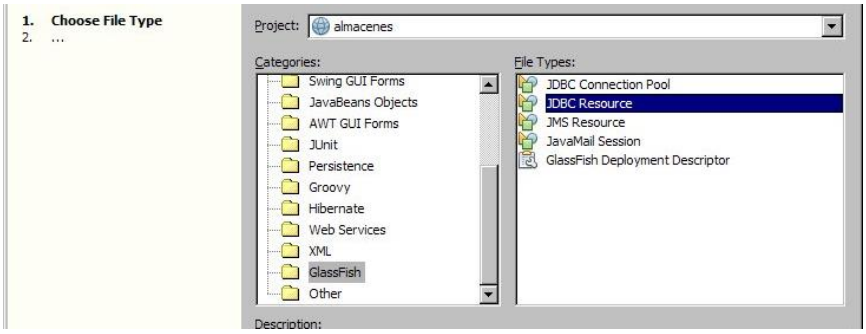
URL: jdbc:sqlserver://localhost\PC-01;databaseName=aerolinea
Usuario: sa
Clave: adminadmin

Clic en ***Finish***

Ya tenemos nuestro pool de conexiones creado para nuestro proyecto desde NetBeans

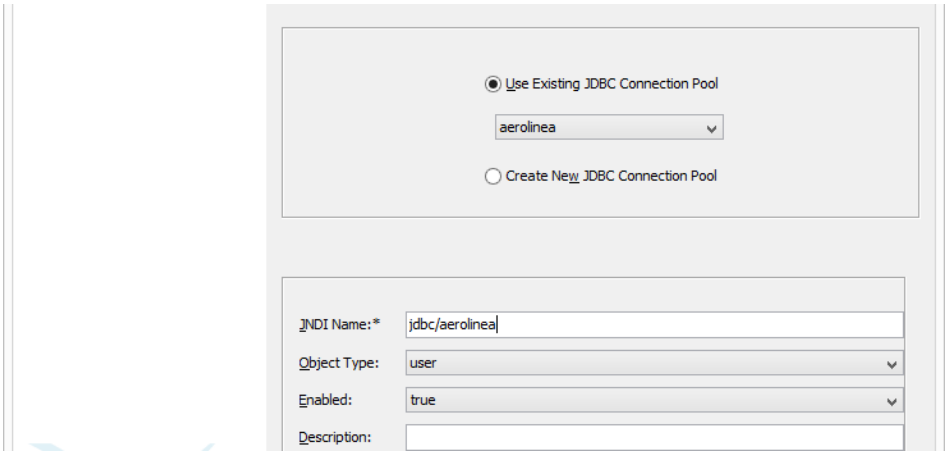
Creando el recurso JDBC

1. Presionamos Ctrl+N (File > New File), seleccionamos la categoría *Glassfish* y el tipo de archivo *JDBC Resource*



Clic en *Next*

2. Seleccionamos el pool de conexiones que estará asociado a nuestro recurso JDBC. En este caso es aerolínea_pool, y se escribe el nombre en formato JNDI de nuestro recurso jdbc.

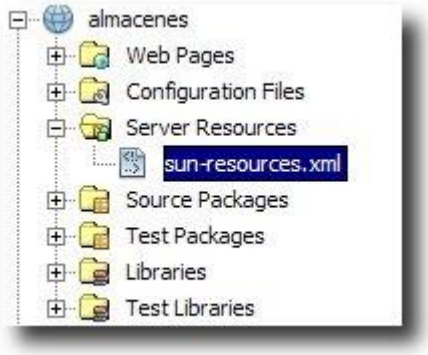


Clic en *Finish*

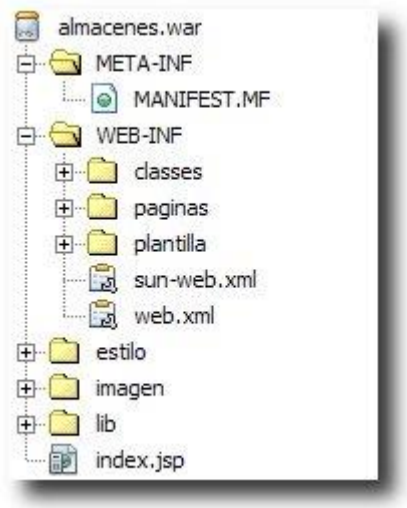
Ya tenemos nuestro recurso JDBC.
El recurso JDBC y el pool de conexiones creados desde NetBeans **se crearán en el glassfish unicamente cuando se despliegue el proyecto.**

¿Cuál es la diferencia en hacer desde Glassfish o desde NetBeans la configuración?

Cuando se hace desde NetBeans, **es solo para fines de desarrollo**. Notemos que se ha creado el archivo `sun-resources.xml` en nuestro proyecto.



Dentro tiene toda la configuración que NetBeans usará para crear los recursos JDBC para que nuestra aplicación funcione en nuestro glassfish de desarrollo. Para nada más.
Cuando se crea el archivo .war para desplegar nuestra aplicación en un glassfish de **producción**, el archivo `sun-resources.xml` no existe dentro del paquete de instalación. Simplemente no hay.



Entonces ¿cómo nuestra aplicación puede conectarse a la base de datos estando en el glassfish de producción? Pues bien, en este glassfish de producción, creamos el JDBC resource directamente en el mismo glassfish.
Esto es bueno, porque cuando desarrollamos usamos una base de datos que es solo para desarrollo. Mientras que para poner en producción, usaremos otra conexión, otra base de datos que será para producción.

Pasos para configuración:
http://wiki.netbeans.org/PoolConexionesGlassfishNetBeans

Construcción de la lógica de negocios

Construcción de clases y paquetes

Una vez se ha definido el mecanismo de conexión a la base de datos, es necesario crear un diseño de clases que sea capaz de administrar toda la lógica del negocio. Las operaciones con la base de datos, generación de Sentencias SQL para cualquier tabla y otras operaciones subyacentes a la aplicación.

El proyecto web actual incorporará las siguientes clases y paquetes para este proceso:

Agregar los siguientes paquetes y clases

- com.aerolinea.conexion
  - Conexion.java
  - ConexionPool.java
- com.aerolinea.anotaciones **(descargarlas del aula virtual)**
  - AutoIncrement.java
  - Entity.java
  - FiledName.java
  - NotNull.java
  - PrimaryKey.java
- com.aerolinea.operaciones
  - AuxiliarPersistencia.java
  - FieldClass.java
  - ManejadorSentencias.java
  - Operaciones.java
  - Persistencia.java
- com.aerolinea.utilerias
  - Hash.java

- **Conexión**

Es una interfaz que contiene los métodos abstractos de la conexión, se declaran los métodos utilizados para la conexión a la base de datos.

- **ConexionPool**

Es una clase que representa la conexión a la base de datos, implementa los métodos de la interfaz **Conexión. E**stablece una conexión utilizando como datos de configuración los establecidos en el pool de conexiones de Glassfish.

Código de las clases:

Nombre de la Interface: Conexion (crear un objeto interface, no clase)

```
package com.aerolinea.procesos;

import java.sql.Connection;

public interface Conexion {

    public void conectar();

    public Connection getConexion();

    public void desconectar();
}
```

Nombre de Clase: ConexionPool

```
package com.aerolinea.procesos;

import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

public class ConexionPool implements Conexion{
    private Connection conn;

    public ConexionPool() {
        conn = null;
    }

    @Override
    public synchronized void conectar(){
        try {
            Context initCtx;
            initCtx = new InitialContext();
    DataSource ds = (DataSource)initCtx.lookup("jdbc/aerolineaDatasource");
            conn=ds.getConnection();

        }catch(NamingException | SQLException e){
            System.out.println("db: " + e.getMessage());
        }
    }


    @Override
    public Connection getConexion(){
        return conn;
```

```
    }

    @Override
    public synchronized void desconectar(){
        try {
            conn.close();
        } catch (SQLException ex) {

Logger.getLogger(ConexionPool.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

}
```

Nombre de Clase: AuxiliarPersistencia.java

```
package com.aerolinea.operaciones;

import com.aerolinea.anotaciones.*;
import java.lang.annotation.Annotation;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.sql.Date;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;

public class AuxiliarPersistencia {

    public String getEntityName(Object entity) throws Exception {
        String nombre = "";
        Class clase = entity.getClass();
        Annotation[] anotaciones =
clase.getAnnotationsByType(Entity.class);
        if(anotaciones.length == 0) {
            throw new Exception(String.format("Debe agregar la anotación
Entity a la clase %s.", clase.getSimpleName()));
        }
        for(Annotation a : anotaciones) {
            Entity e = (Entity) a;
            nombre = e.table();
            if("".equals(nombre))
                nombre = clase.getSimpleName();
        }
        return nombre;
    }
    public List<FieldClass> getCampos(Object entity) throws Exception {
        ArrayList<FieldClass> campos = new ArrayList<>();
        Class clase = entity.getClass();
        Field[] fields = clase.getDeclaredFields();
        int nauto_increment = 0;
        for(Field f : fields) {
            FieldClass campo = new FieldClass();
```

```
            if (f.isAnnotationPresent(FieldName.class)){
                FieldName fc = f.getAnnotation(FieldName.class);
                campo.setNombre(fc.name());
            }else
                campo.setNombre(f.getName());

            //Annotation[] anotacionesPrimaryKey =
f.getAnnotationsByType(PrimaryKey.class);
            if (nauto_increment==0)
                if (f.isAnnotationPresent(PrimaryKey.class)){
                    campo.setPrimary(true);
                    nauto_increment++;
                }
            else
                throw new Exception(String.format("La clase %s solo debe
contener un campo como llave primaria (PrimaryKey).",
clase.getSimpleName()));

            if (f.isAnnotationPresent(AutoIncrement.class))
                if(campo.isPrimary())
                    campo.setAutoincrement(true);
                else
                    throw new Exception("La anotación AutoIncrement solo
puede asociarse a una llave primaria.");

            if (f.isAnnotationPresent(NotNull.class)){
                campo.setNotnull(true);
            }
            try {
                    Field field = clase.getDeclaredField(f.getName());
                    field.setAccessible(true);
                    campo.setValor(field.get(entity));

            } catch (IllegalAccessException | IllegalArgumentException ex)
{
                throw ex;
            }
            campos.add(campo);
        }
        return campos;
    }
}
```

Nombre de Clase: FieldClass.java

```
package com.aerolinea.operaciones;

public class FieldClass {

    private String Nombre;
    private Object valor;
    private boolean primary;
    private boolean autoincrement;
```

```
    private boolean notnull;
    private Class tipo;
    public Class getTipo() {
        return tipo;
    }
    public void setTipo(Class tipo) {
        this.tipo = tipo;
    }

    public String getNombre() {
        return Nombre;
    }

    public void setNombre(String Nombre) {
        this.Nombre = Nombre;
    }
    public Object getValor() {
        return valor;
    }
    public void setValor(Object valor) {
        this.valor = valor;
    }

    public boolean isPrimary() {
        return primary;
    }

    public void setPrimary(boolean primary) {
        this.primary = primary;
    }

    public boolean isAutoincrement() {
        return autoincrement;
    }

    public void setAutoincrement(boolean autoincrement) {
        this.autoincrement = autoincrement;
    }

    public boolean isNotnull() {
        return notnull;
    }

    public void setNotnull(boolean notnull) {
        this.notnull = notnull;
    }
}
```

Nombre de Clase: ManejadorSentencias.java

```
package com.aerolinea.operaciones;

import java.util.ArrayList;
import java.util.List;

public class ManejadorSentencias {

    private final List<FieldClass> campos;
    private final List<String> nombreCampos;
    private final List<Object> valorCampos;
    private final AuxiliarPersistencia aux;
    private final List<String> paramString;
    private final Object entidad;

    public ManejadorSentencias(Object entidad) throws Exception {
            this.entidad = entidad;
            this.aux = new AuxiliarPersistencia();
            campos = aux.getCampos(entidad);
            nombreCampos=new ArrayList();
            valorCampos=new ArrayList();
            paramString=new ArrayList();
    }

    public List<FieldClass> getCampos() {
        return campos;
    }

    public List<Object> getValorCampos() {
        for (FieldClass campo: campos)
            if (!campo.isAutoincrement())
                valorCampos.add(campo.getValor());
        return valorCampos;
    }

    public List<String> getNombreCampos() {
        for (FieldClass campo: campos)
            nombreCampos.add(campo.getNombre());
        return nombreCampos;
    }

    private String getListaCampos(){
        for (FieldClass campo: campos)
            if (!campo.isAutoincrement()){
                nombreCampos.add(campo.getNombre());
            }
        return String.join(",", nombreCampos);
    }

    private String getTodosCampos(){
        return String.join(",", getNombreCampos());
    }

    private String getListaParametros() {
```

```
        for (FieldClass campo: campos)
            if (!campo.isAutoincrement()){
                paramString.add("?");
            }
        return String.join(",", paramString);
    }
    private String getPrimaryKey(){
        String pk="";
        for (FieldClass campo: campos)
            if (campo.isPrimary()){
                pk = campo.getNombre();
                break;
            }
        return pk;
    }
    public Object getPrimaryKeyValue(){
        Object pkvalue="";
        for (FieldClass campo: campos)
            if (campo.isPrimary()){
                pkvalue = campo.getValor();
                break;
            }
        return pkvalue;
    }
    public  String getSelect() throws Exception {
        String sqlSelect = String.format("SELECT * FROM %s WHERE %s = ?",
                    aux.getEntityName(entidad.getClass().newInstance()),
getPrimaryKey());
        return sqlSelect;
    }

    public  String getSelectAll() throws Exception {
        String sqlSelect = String.format("SELECT * FROM %s",
                    aux.getEntityName(entidad.getClass().newInstance()));
        return sqlSelect;
    }

    public String getInsert() throws Exception {
        String sqlInsertar = String.format("INSERT INTO %s(%s) VALUES(%s)",
                    aux.getEntityName(entidad), getListaCampos(),
getListaParametros());
        return sqlInsertar;
    }

    public String getUpdate() throws Exception {
        String lista="";
        for (FieldClass campo: campos){
            if (!campo.isPrimary())
                lista+= campo.getNombre()+"=?,";
        }
        lista=lista.substring(0, lista.length()-1);

        String sqlUpdate = String.format("UPDATE %s SET %s WHERE %s = ?",
                    aux.getEntityName(entidad), lista, getPrimaryKey());
        return sqlUpdate;
```

```
    }

    public String getDelete() throws Exception {
        String sqlDelete = String.format("DELETE FROM %s WHERE %s = ?",
                    aux.getEntityName(entidad.getClass().newInstance()),
getPrimaryKey());
        return sqlDelete;
    }
}
```

Nombre de Clase: Persistencia.java

```
package com.aerolinea.operaciones;

import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Time;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Persistencia extends ManejadorSentencias{

    public Persistencia(Object entity, Connection cn) throws Exception {
        super(entity);
        this.conexion = cn;
    }

    public final <T> T insertar(T entity) throws Exception {
        try {
            List<Object> valores = super.getValorCampos();
            PreparedStatement stmt =
this.conexion.prepareStatement(super.getInsert(),
Statement.RETURN_GENERATED_KEYS);

            for(int i = 1; i <= valores.size() ;i++) {
                    stmt.setObject(i, valores.get(i-1));
            }
            int filasAfectadas = stmt.executeUpdate();

            if (filasAfectadas == 0) {
                throw new SQLException("Creating user failed, no rows
affected.");
            }
```

```
            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    Object pk = generatedKeys.getObject(1);
                    return getRegistro(pk, entity.getClass());
                }
                else {
                    return getRegistro(getPrimaryKeyValue(),
entity.getClass());
                }
            }
        } catch (Exception ex) {

Logger.getLogger(Persistencia.class.getName()).log(Level.SEVERE, null, ex);
            throw ex;
        }
    }

    public final <T> T actualizar(Object id, T entity) throws Exception {
        try {
            List<Object> valores = super.getValorCampos();
            PreparedStatement stmt =
this.conexion.prepareStatement(super.getUpdate(),
Statement.RETURN_GENERATED_KEYS);
            for(int i = 0; i < valores.size() ;i++) {
                    stmt.setObject(i+1, valores.get(i));
            }
            stmt.setObject(valores.size()+1, id);
            stmt.executeUpdate();
            return getRegistro(id, entity.getClass());
        } catch (Exception ex) {

Logger.getLogger(Persistencia.class.getName()).log(Level.SEVERE, null, ex);
            throw ex;
        }
    }

    public final <T> T getRegistro(Object pk, Class tipo) throws Exception
{
        T entity = (T)tipo.newInstance();
        try {
            PreparedStatement stmt =
this.conexion.prepareStatement(super.getSelect());
            stmt.setObject(1, pk);
            ResultSet rs = stmt.executeQuery();
            Field[] fs = entity.getClass().getDeclaredFields();
            while(rs.next()) {
                for(int i = 0; i<super.getCampos().size(); i++) {
                    Object valor =
rs.getObject(super.getCampos().get(i).getNombre());
                    if(fs[i].getModifiers() != Modifier.PRIVATE) {
                        fs[i].set(entity, valor);
                    } else {
                        Field field =
entity.getClass().getDeclaredField(fs[i].getName());
                        field.setAccessible(true);
```

```
                        System.out.println(valor);
                        if (valor instanceof Long)
                            valor =Integer.parseInt(valor.toString());
                        field.set(entity, valor);
                    }
                }
            }
        } catch(Exception ex) {

Logger.getLogger(Persistencia.class.getName()).log(Level.SEVERE, null, ex);
            throw  ex;
        }
        return entity;
    }

    public final <T> T eliminar(Object id, Class entity) throws Exception{
        try {
            Object result = getRegistro(id, entity);
            PreparedStatement stmt =
this.conexion.prepareStatement(super.getDelete());
            stmt.setObject(1, id);
            stmt.executeUpdate();
            return (T)result;
        } catch (Exception ex) {

Logger.getLogger(Persistencia.class.getName()).log(Level.SEVERE, null, ex);
            throw ex;
        }
    }

    public final <T> ArrayList<T> getTodos(Class entity) throws Exception {
        ArrayList<T> rows = new ArrayList<>();
        try {
            PreparedStatement stmt =
this.conexion.prepareStatement(super.getSelectAll());
            ResultSet rs = stmt.executeQuery();
            Field[] fs = entity.getDeclaredFields();
            while(rs.next()) {
                Object obj = entity.newInstance();
                for(int i = 0; i<super.getCampos().size(); i++) {
                    Object valor =
rs.getObject(super.getCampos().get(i).getNombre());
                    if(fs[i].getModifiers() != Modifier.PRIVATE) {
                        fs[i].set(obj, valor);
                    } else {
                        Field field =
entity.getDeclaredField(fs[i].getName());
                        field.setAccessible(true);
                        if (valor instanceof Long)
                            valor =Integer.parseInt(valor.toString());
                        field.set(obj, valor);
                    }
                }
                rows.add((T)obj);
            }
```

```
            return rows;
        } catch(Exception ex) {

Logger.getLogger(Persistencia.class.getName()).log(Level.SEVERE, null, ex);
            throw  ex;
        }
    }
}
```

Nombre de Clase: Operaciones.java

```
package com.aerolinea.operaciones;

import com.aerolinea.conexion.Conexion;
import java.sql.SQLException;
import java.util.ArrayList;
import java.math.BigDecimal;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Time;
import java.sql.Timestamp;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Operaciones {

    private static Conexion conexion;

    public static void abrirConexion(Conexion conexion){
        Operaciones.conexion = conexion;
    }
    public static void iniciarTransaccion() throws SQLException, Exception
{
        conexion.getConexion().setAutoCommit(false);
    }

    public static void commit() throws SQLException {
        conexion.getConexion().commit();
    }

    public static void rollback() throws SQLException {
        conexion.getConexion().rollback();
    }

    public static void cerrarConexion() throws SQLException {
        if(!conexion.getConexion().isClosed())
conexion.getConexion().close();
    }

    public static <T> T insertar(T entity) throws SQLException, Exception {
        try {
```

```
            Persistencia accion = new Persistencia(entity,
conexion.getConexion());
            return accion.insertar(entity);
        } catch(Exception ex) {
            conexion.getConexion().rollback();
            throw ex;
        }
    }

    public static <T> T get(Object id, T entity) throws SQLException,
Exception {
        try {
            Persistencia accion = new Persistencia(entity,
conexion.getConexion());
            return accion.getRegistro(id, entity.getClass());
        } catch(Exception ex) {
            conexion.getConexion().rollback();
            throw ex;
        }
    }

    public static <T> T actualizar(Object id, T entity) throws
SQLException, Exception {
        try {
            Persistencia accion = new Persistencia(entity,
conexion.getConexion());
            return accion.actualizar(id, entity);
        } catch(Exception ex) {
            conexion.getConexion().rollback();
            throw ex;
        }
    }

    public static <T> T eliminar(Object id, T entity) throws SQLException,
Exception {
        try {
            Persistencia accion = new Persistencia(entity,
conexion.getConexion());
            return accion.eliminar(id, entity.getClass());
        } catch(Exception ex) {
            conexion.getConexion().rollback();
            throw ex;
        }
    }

    public static <T> ArrayList<T> getTodos(T entity) throws Exception {
        try {
            Persistencia accion = new Persistencia(entity,
conexion.getConexion());
            return accion.getTodos(entity.getClass());
        } catch(Exception ex) {
            conexion.getConexion().rollback();
            throw ex;
        }
    }
```

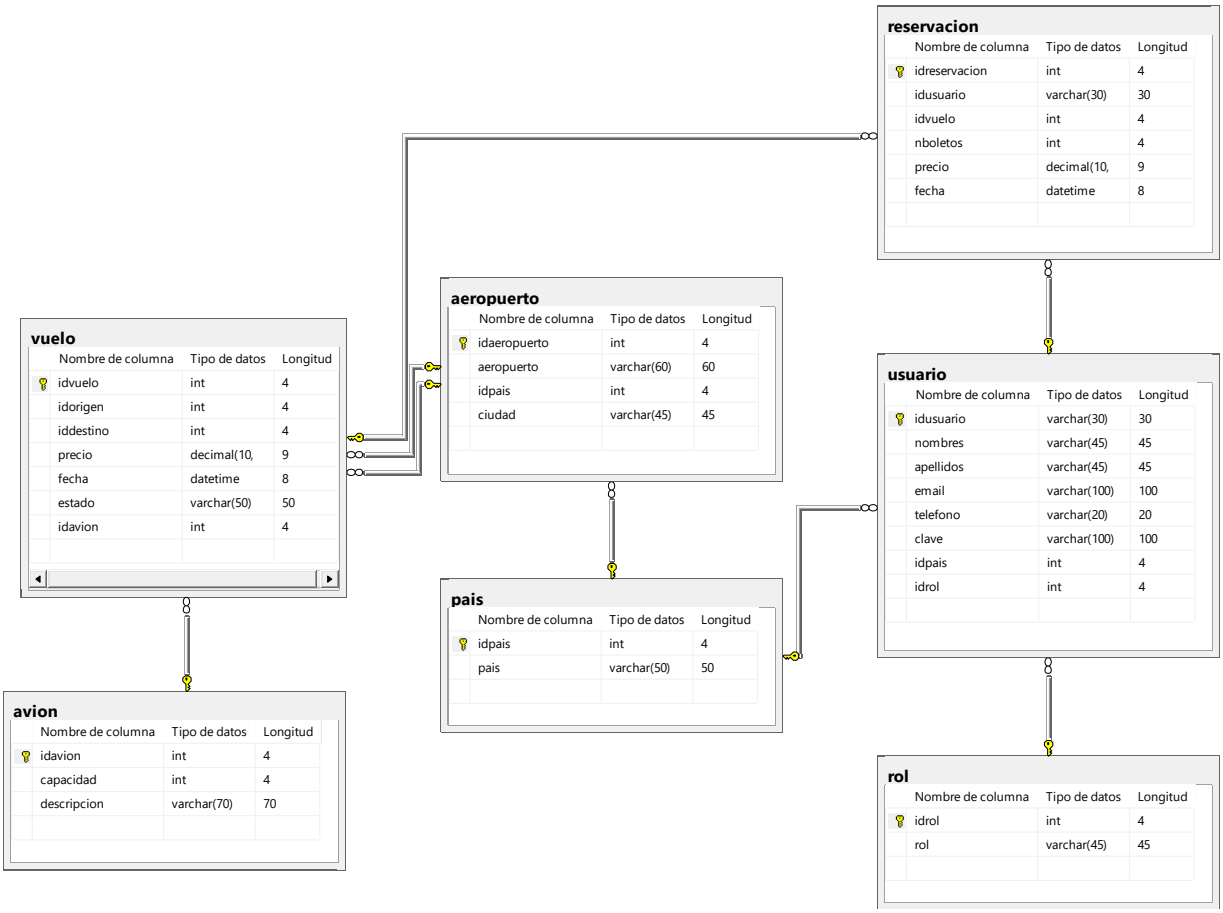
```
    public static String[][] consultar(String sqlQuery, List<Object>
params) throws Exception {
        String [][] resultados = null;
        try {
            PreparedStatement stmt =
conexion.getConexion().prepareStatement(sqlQuery,
ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
            if(params!=null && params.size()>0) {
                List<Object> cleanParams= params;
                for(int i=1;i<= params.size();i++) {
                    Object valor = cleanParams.get(i-1);
                    if(valor == null)
                        stmt.setNull(i, 0);
                    else if(valor instanceof Integer)
                        stmt.setInt(i, (int)valor);
                    else if(valor instanceof Double)
                        stmt.setDouble(i, (double)valor);
                    else if(valor instanceof BigDecimal)
                        stmt.setBigDecimal(i, (BigDecimal)valor);
                    else if(valor instanceof Float)
                        stmt.setFloat(i, (float)valor);
                    else if(valor instanceof Long)
                        stmt.setLong(i, (long)valor);
                    else if(valor instanceof Boolean)
                        stmt.setBoolean(i, (boolean)valor);
                    else if(valor instanceof Byte)
                        stmt.setByte(i, (byte)valor);
                    else if(valor instanceof Short)
                        stmt.setShort(i, (short)valor);
                    else if(valor instanceof String)
                        stmt.setString(i, valor.toString());
                    else if(valor instanceof Date)
                        stmt.setDate(i, (Date)valor);
                    else if(valor instanceof Time)
                        stmt.setTime(i, (Time)valor);
                    else if(valor instanceof Timestamp)
                        stmt.setTimestamp(i, (Timestamp)valor);
                }
            }
            ResultSet rs = stmt.executeQuery();
            if(rs.next()) {
                rs.last();
                ResultSetMetaData rsmd = rs.getMetaData();
                int numCols = rsmd.getColumnCount();
                int numFils =rs.getRow();
                resultados=new String[numCols][numFils];
                int j = 0;
                rs.beforeFirst();
                while (rs.next())
                {
                    for (int i=0;i<numCols;i++)
                    {
                        resultados[i][j]=rs.getString(i+1);
                    }
```

```
                    j++;
                }return resultados;
            }
            return resultados;
        } catch(Exception ex) {

Logger.getLogger(Persistencia.class.getName()).log(Level.SEVERE, null, ex);
            throw  ex;
        }
}
```

Nombre de Clase: Hash.java

```
package com.almacen.utilerias;

import java.security.MessageDigest;

public class Hash {
    public static String MD2 = "MD2";
    public static String MD5 = "MD5";
    public static String SHA = "SHA-1";
    public static String SHA224 = "SHA-224";
    public static String SHA256 = "SHA-256";
    public static String SHA384 = "SHA-384";
    public static String SHA512 = "SHA-512";

    //Message Digest
    public static String generarHash(String clave, String algoritmo) {
        String digested;
        try {
            MessageDigest md = MessageDigest.getInstance(algoritmo);
            md.update(clave.getBytes());
            byte byteData[] = md.digest();

            digested = convertirHex(byteData);
        } catch (Exception ex) {
            return "Error: " + ex.getMessage();
        }

        return digested;
    }

    //convertidores
    private static String convertirHex(byte[] byteData) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < byteData.length; i++) {
            sb.append(String.format("%02x", byteData[i]));
        }

        return sb.toString();
    }
}
```

Diagrama Entidad Relación Aerolínea.



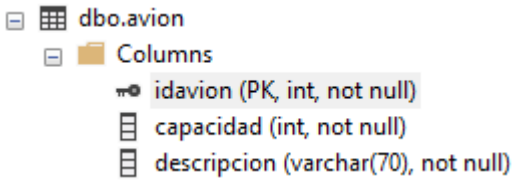
Construcción de paquete de clases Entidades

Agregar al proyecto un paquete con el nombre *com.aerolinea.entidad* el cual contendrá las clases encargadas para el manejo de las entidades de cada una de las tablas.

Una vez creadas las clases que se encargarán de los procesos se procede a crear las clases entidades para cada una de las tablas de la base de datos para manejar las consultas. Seguir indicaciones del instructor

Ya que tenemos creado nuestro paquete *com.aerolinea.entidad*, procedemos a agregar las entidades.

Una entidad es una clase Java que representa a cabalidad la estructura de campos de una tabla, por lo tanto, si tenemos la tabla avión, así:



La entidad (clase) debe quedar así:

```java
package com.aerolinea.entidad;

import com.aerolinea.anotaciones.AutoIncrement;
import com.aerolinea.anotaciones.Entity;
import com.aerolinea.anotaciones.FieldName;
import com.aerolinea.anotaciones.NotNull;
import com.aerolinea.anotaciones.PrimaryKey;

@Entity(table = "avion")
public class Avion {
    @PrimaryKey
    @AutoIncrement
    @FieldName(name = "idavion")
    private int idavion;
    @NotNull
    private int capacidad;
    @NotNull
    private String descripcion;

    public Avion() {
    }


    public Avion(int idavion, int capacidad, String descripcion) {
        this.idavion = idavion;
        this.capacidad = capacidad;
        this.descripcion = descripcion;
    }

    public int getIdavion() {
        return idavion;
    }
```

```
    public void setIdavion(int idavion) {
        this.idavion = idavion;
    }

    public int getCapacidad() {
        return capacidad;
    }

    public void setCapacidad(int capacidad) {
        this.capacidad = capacidad;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

}
```

Explicación:

Las entidad debe llamarse exactamente igual a la tabla que representa, y si ocurre el caso que queremos que la entidad se llame diferente a la tabla, entonces usamos la anotación **@Entity** con su variable **table**, al igual que en la entidad avion queda así: **@Entity(table=”avion”). NOTA:** Se recomienda utilizar siempre la variable table de la anotación para definir el nombre la tabla que representa.

Luego vemos en la tabla que el campo **idavion** es la llave primaria, y además se genera de manera automática por SQL Server, por eso es que la variable **idavion** de la entidad tiene la anotación **@PrimaryKey** para indicar que es llave primaria, y también la anotación **@AutoIncrement**, para indicar que su valor se genera automáticamente. Como agregado dicho campo **idavion**, también tiene la anotación **@FieldName**, el cual se utiliza junto a la variable **name** para indicar el nombre del campo de la tabla, cuando queremos que la variable de la entidad se llame de manera diferente al de la tabla. **NOTA**: Si la variable de la entidad se llama igual al campo de la tabla, entonces la anotación **@FieldName** es opcional.

Vemos también que en la tabla **avion**, los campos **capacidad** y **descripción** indican que no pueden ser nulos (not null), por lo tanto en la entidad a cada campo se les agrega la anotación **@NotNull,** de esa manera evitamos que el usuario intente insertar valores nulos a la tabla.

En resumen, ya vimos que una entidad representa a una tabla de la base de datos, por eso es que dicha entidad debe llamarse igual que la tabla, y cada una de las variables de la entidad debe llamarse igual a los campos de la tabla que representa.

También se recomienda que a las entidades se les defina un constructor por defecto, y otro que inicialice todas las variables de la entidad, justamente como se muestra en el ejemplo de arriba.

Procedimiento de uso de las clases

Ejemplo de inserción:

```
    try {
        ConexionPool con = new ConexionPool(); // inicializamos el pool de
conexiones
        con.conectar();
        Avion a = new Avion(); // creamos la entidad Avion
        a.setCapacidad(200); // definimos la capacidad de pasajeros del avión
        a.setDescripcion("Boeing 747"); // definimos la descripción o modelo
del avión
        Operaciones.abrirConexion(con); // abrimos la conexión de la bd
        Operaciones.iniciarTransaccion(); // iniciamos la transacción
        a = Operaciones.insertar(a); // insertamos la entidad avion en la bd
        // el método insertar retorna la entidad que acabamos de insertar
        Operaciones.commit(); // confirmamos los cambios de la transacción
    } catch(Exception ex) {
        Logger.getLogger(Principal.class.getName()).log(Level.SEVERE, null,
ex);
    } finally {
        try {
            Operaciones.cerrarConexion(); // se cierra la conexión al final de
todo
        } catch (SQLException ex) {
            Logger.getLogger(Principal.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
```

Ejemplo de actualización:

```
    try {
        ConexionPool con = new ConexionPool(); // inicializamos el pool de
conexiones
        con.conectar();
        Avion a = new Avion(); // creamos la entidad Avion
        a.setCapacidad(250); // definimos la capacidad de pasajeros del avión
        a.setDescripcion("Boeing 747 v1"); // definimos la descripción o
modelo del avión
        Operaciones.abrirConexion(con); // abrimos la conexión de la bd
        Operaciones.iniciarTransaccion(); // iniciamos la transacción
        // pasamos el id del registro de la bd que queremos actualizar,
        // y luego pasamos la entidad avion que queremos actualizar
        a = Operaciones.actualizar(1, a); // actualizamos la entidad avion en
la bd
        // el método actualizar retorna la entidad con todos los campos
actualizados
        Operaciones.commit(); // confirmamos los cambios de la transacción
    } catch(Exception ex) {
        Logger.getLogger(Principal.class.getName()).log(Level.SEVERE, null,
ex);
    } finally {
        try {
            Operaciones.cerrarConexion(); // se cierra la conexión al final de
todo
        } catch (SQLException ex) {
            Logger.getLogger(Principal.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
```

Ejemplo de borrado:

```
    try {
        ConexionPool con = new ConexionPool(); // inicializamos el pool de
conexiones
        con.conectar();
        Avion a = new Avion(); // creamos la entidad Avion
        Operaciones.abrirConexion(con); // abrimos la conexión de la bd
        Operaciones.iniciarTransaccion(); // iniciamos la transacción
        a = Operaciones.eliminar(41, new Avion()); // eliminamos la entidad
avion en la bd
```

```
        // el método eliminar retorna la entidad que acabamos de eliminar
        Operaciones.commit(); // confirmamos los cambios de la transacción
    } catch(Exception ex) {
        Logger.getLogger(Principal.class.getName()).log(Level.SEVERE, null,
ex);
    } finally {
        try {
            Operaciones.cerrarConexion(); // se cierra la conexión al final de
todo
        } catch (SQLException ex) {
            Logger.getLogger(Principal.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
```

Ejemplo de consulta de una entidad:

```
    try {
        ConexionPool con = new ConexionPool(); // inicializamos el pool de
conexiones
        con.conectar();
        Avion a = new Avion(); // creamos la entidad Avion
        Operaciones.abrirConexion(con); // abrimos la conexión de la bd
        a = Operaciones.get(42, new Avion()); // obtenemos la entidad avion
con el id 42 de la bd
        // el método get retorna la entidad que acabamos de consultar
    } catch(Exception ex) {
        Logger.getLogger(Principal.class.getName()).log(Level.SEVERE, null,
ex);
    } finally {
        try {
            Operaciones.cerrarConexion(); // se cierra la conexión al final de
todo
        } catch (SQLException ex) {
            Logger.getLogger(Principal.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
```

Ejemplo de consulta de todas las entidades:

```
    try {
        ConexionPool con = new ConexionPool(); // inicializamos el pool de
conexiones
        con.conectar();
        ArrayList<Avion> listado = new ArrayList<Avion>(); // creamos el
listado de entidades Avion
        Operaciones.abrirConexion(con); // abrimos la conexión de la bd
        listado = Operaciones.getTodos(new Avion()); // obtenemos las
entidades avion de la bd
        // el método getTodos retorna todas las entidades que acabamos de
consultar de la tabla
    } catch(Exception ex) {
        Logger.getLogger(Principal.class.getName()).log(Level.SEVERE, null,
ex);
    } finally {
        try {
            Operaciones.cerrarConexion(); // se cierra la conexión al final de
todo
        } catch (SQLException ex) {
            Logger.getLogger(Principal.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
```

Ejemplo de consulta personalizada a múltiples tablas:

```
    try {
        ConexionPool con = new ConexionPool(); // inicializamos el pool de
conexiones
        con.conectar();
        Operaciones.abrirConexion(con); // abrimos la conexión de la bd
        String consulta = "SELECT vu.estado, av.descripcion FROM avion AS
av\n" +
                          "  INNER JOIN vuelo AS vu ON vu.idavion =
av.idavion\n" +
                          "  WHERE av.descripcion = ? AND vu.estado = ?"; //
consulta SQL
        List<Object> params = new ArrayList<Object>();
        params.add("Delta");
        params.add("DISPONIBLE");
```

```
        String[][] listado  = Operaciones.consultar(consulta, params); //
obtenemos los registros de la consulta
        // el método consultar retorna todos los registros de la consulta
        // NOTA: Si la consulta no recibe parámetros entonces el segundo
parámetro
        // del método consultar debe ser null.
        // El método consultar retorna null si no encuentra ningún registro de
la búsqueda
    } catch(Exception ex) {
        Logger.getLogger(Principal.class.getName()).log(Level.SEVERE, null,
ex);
    } finally {
        try {
            Operaciones.cerrarConexion(); // se cierra la conexión al final de
todo
        } catch (SQLException ex) {
            Logger.getLogger(Principal.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
```

III- Tarea

1. Elaborar el correspondiente diagrama de clases con una herramienta CASE de la lógica de negocios
2. Crear el formulario de autenticación para la aerolínea. Al autenticarse, deberá dirigirse si el rol es 1(usuario) a una página de ejemplo1, y si es 2(administrador) a una página ejemplo2. Crear un Servlet llamado Auth para la autenticación.
3. Aplicar la conexión y clases para trabajarla en el **Proyecto**.
4. Mapear base de datos del proyecto
5. Agregar metodo para ejecutar procedimientos almacenados