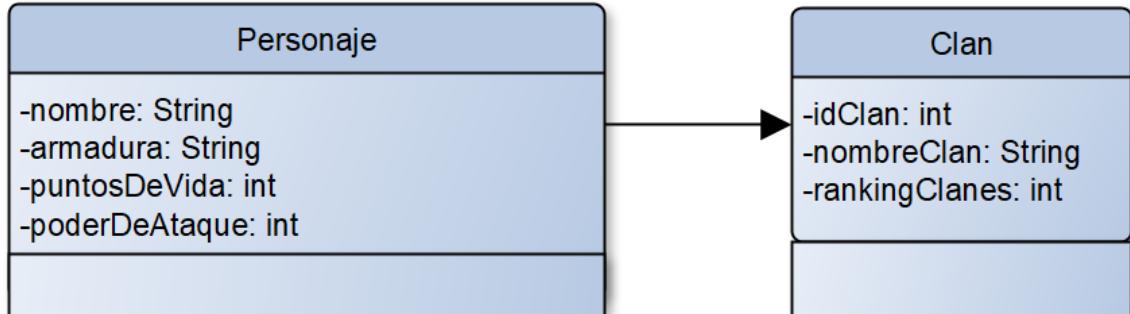


Examen Parcial 2°

Fecha: 20/10/2023

Nombre y Apellido : DNI:

Implementar en un CRUD de la entidad Personaje en Java con el modelo presentado en el siguiente diagrama:



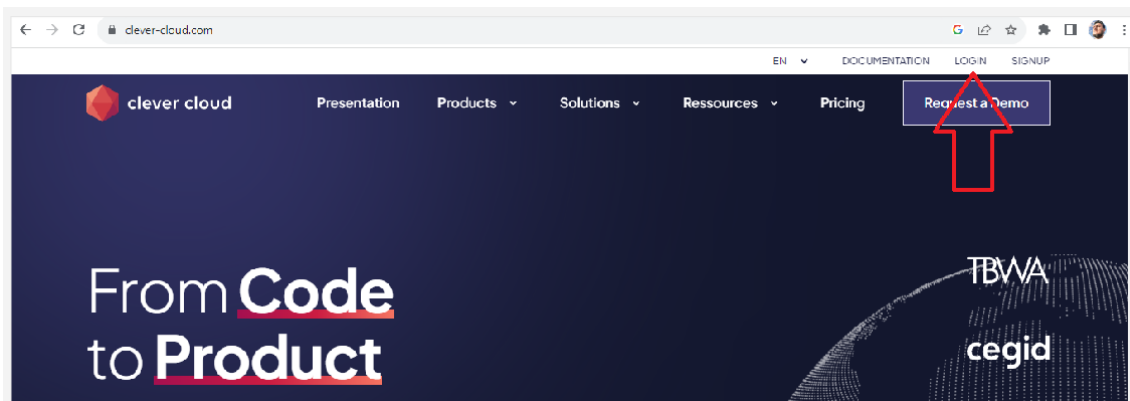
Consignas:

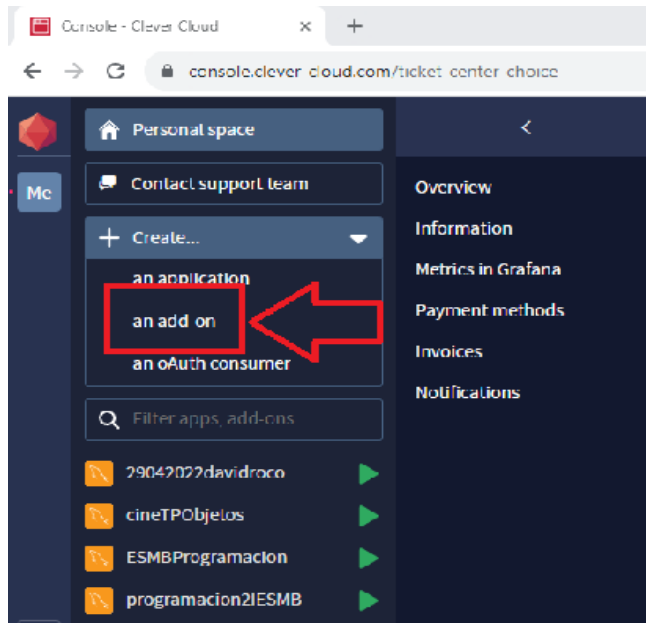
- Pude implementar preferiblemente las JPA.
- Base de datos relacionales SQL. (Ejemplo practicado por la clase en <https://www.clever-cloud.com>, puede ser en forma local).
- Modelo por capas (Interfaz/Lógica/Persistencia).
- Aplicar en la interfaz de usuario como mínimo:
 - Alta(crear un nuevo personaje).
 - Baja(elimina un nuevo personaje).
 - Editar(edita los atributos del personaje).
- Existen 5 Clanes:
 - Los Magios.
 - Los NoHomero.
 - Los Peces del infierno.
 - Los Santos Orantes.
 - Los Amigos de los Pinos.
- Generar 5 instancias. (Preferiblemente la base de datos que sea externa así se puede revisar este punto).

Una posible SOLUCIÓN

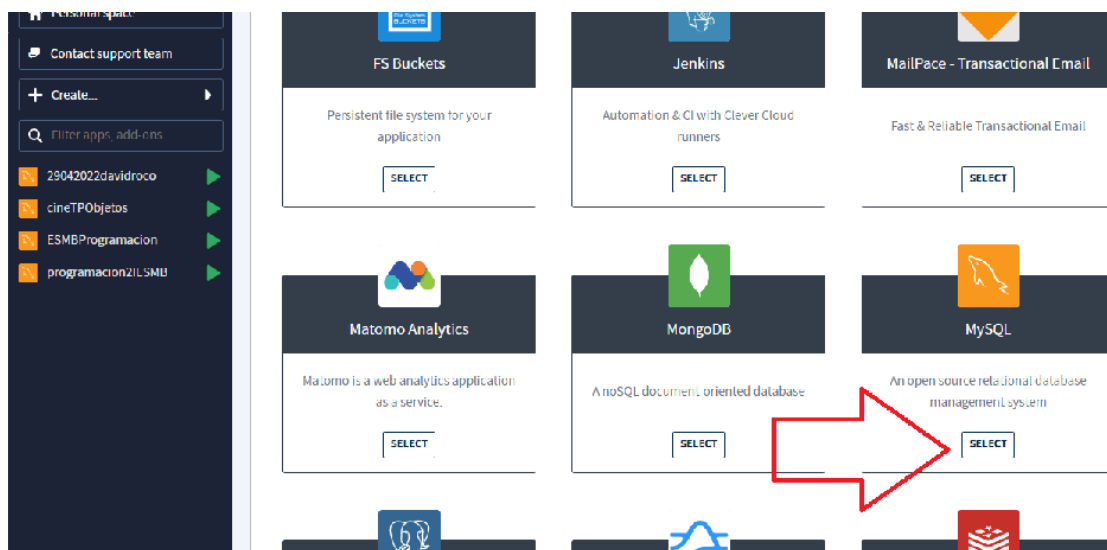
Base de Datos:

Iniciamos sesión en Clever-cloud, si no tenemos cuenta la generamos.





Creamos una aplicación MySQL.



Elegimos la básica DEV gratuita.

Add-on creation Provider Plan Applications Information Options Invoice

What kind of MySQL do you need?

PLAN NAME	BACKUPS	LOGS	MAX CONNECTION LIMIT	MAX DB SIZE	MEMORY	METRICS	TYPE	VCPUS	ESTIMATED PRICE (30 DAYS)
<input checked="" type="radio"/> DEV	Daily - 7 Retained	No	5	10 MB	Shared	No	Shared	Shared	€0.00
<input type="radio"/> XXS Small Space	Daily - 7 retained	Yes	15	512 MB	512 MB	Yes	Dedicated	1	€5.00
<input type="radio"/> XXS Medium Space	Daily - 7 retained	Yes	15	1 GB	512 MB	Yes	Dedicated	1	€5.90

Y le damos NEXT.

Seleccionamos el servidor y el nombre de la base de datos.

What is the name of your MySQL add-on? In which region should it be located?

NAME: *

ParcialObjetos

ZONE: *



Paris France
infratolever-cld

Montreal Canada
infratovh

NEXT

Con eso ya estaría la base de datos.

MySQL by Clever Cloud

TYPE	PLAN	CLUSTER	VERSION	REGION	STATUS	CREATION DATE	ID
MySQL	Dev	par-mysql-c6	8.0	par	ACTIVE	2023-10-23	mysql_b7f82ccc-838b-4264-8f51-b72fc4275bd7

Database Credentials

Get credentials for manual connections to this database.

Host
bmsnkrjdayiy2jtyfy0-mysql.services.clever-cloud.com

Database Name
bmsnkrjdayiy2jtyfy0

User
usjwq5bdeluw3eww

Password
oik9j8DpA00No0902kS3

Port
3306

Connection Url
mysql://usjwq5bdeluw3eww:oik9j8DpA00No0902kS3@bmsnkrjdayiy2jtyfy0-mysql.services.clever-cloud.com

Host: bmsnkrjdayiy2jtyfy0-mysql.services.clever-cloud.com

Data base Name: bmsnkrjdayiy2jtyfy0

User: usjwq5bdeluw3eww

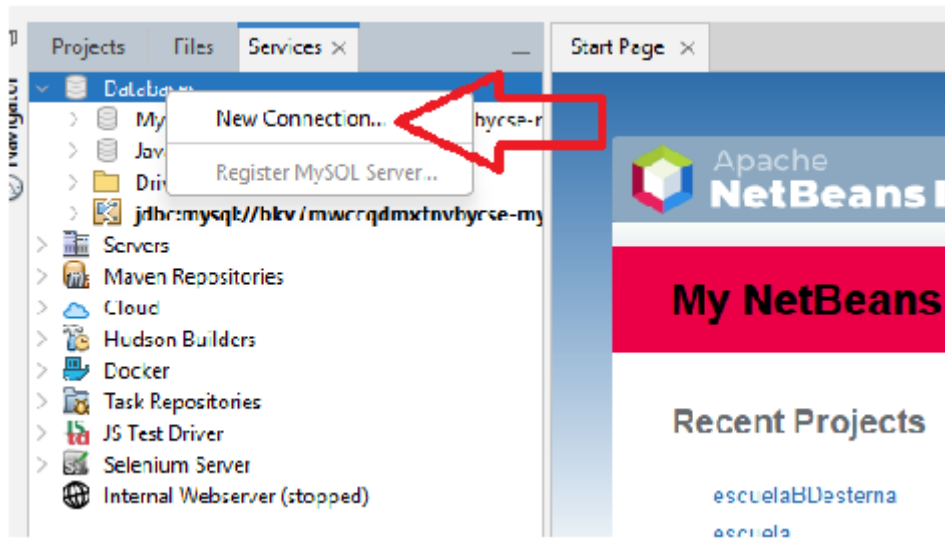
Password: oik9j8DpA00No0902kS3

Port: 3306

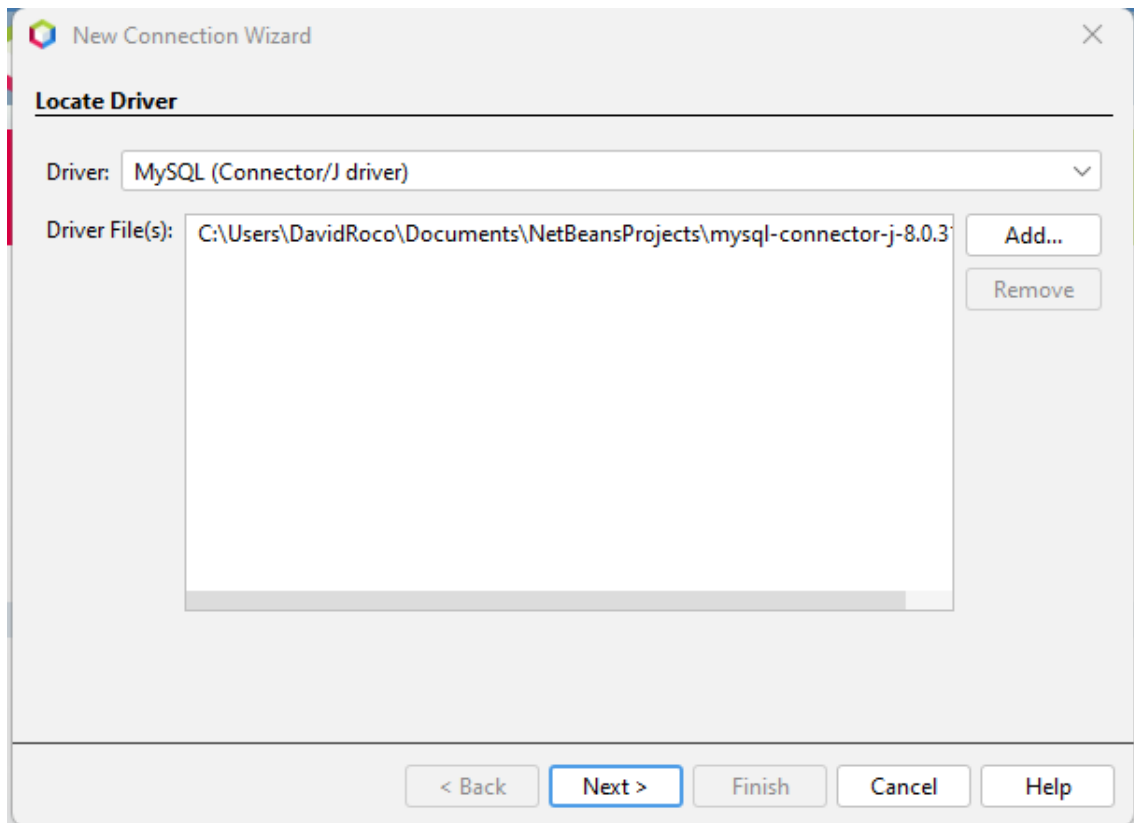
NOTA: En la versión gratuita no podemos cambiar las credenciales root

IDE Netbeans:

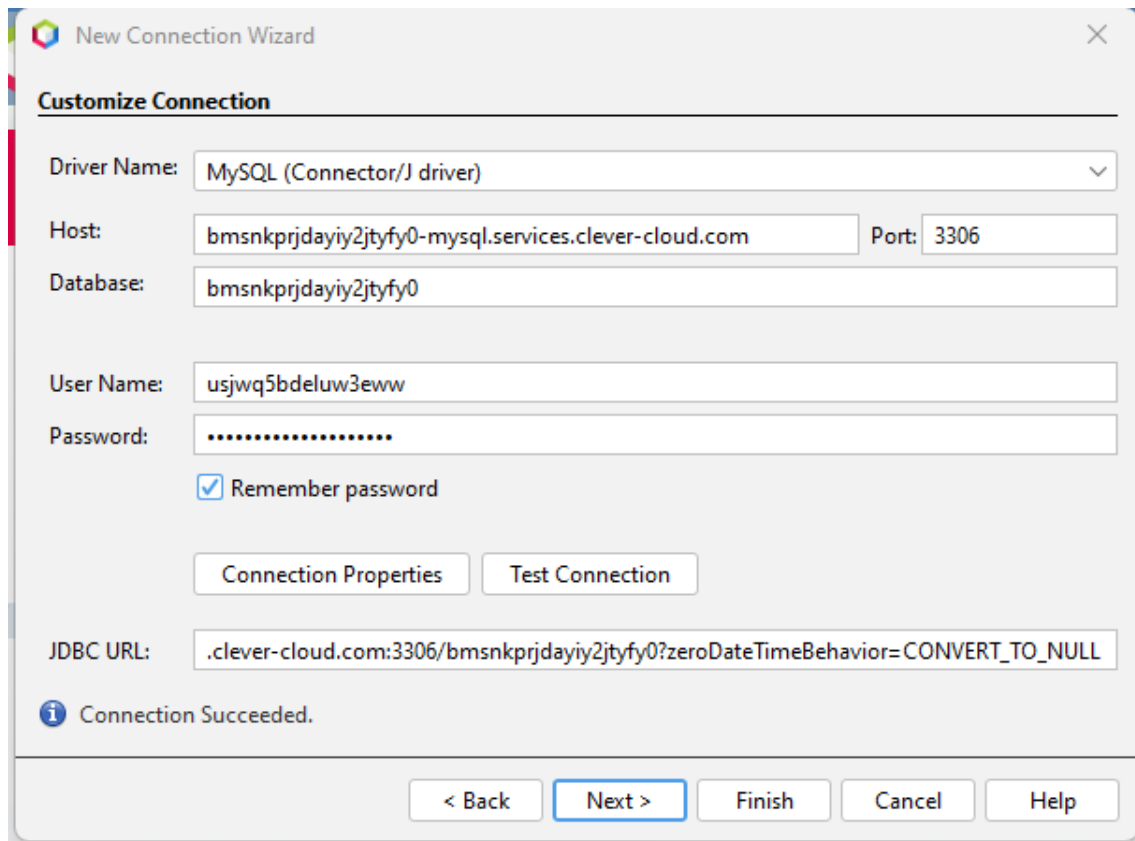
Abrimos Netbeans y lo primero que hacemos es generar la conexión con la base de datos que recién creamos.



Seleccionamos MySQL y si no tenemos el driver de mysql conector j lo podemos descargar desde: <https://dev.mysql.com/downloads/connector/j/?os=26> . Damos click en Next.



Completamos las credenciales y testeamos la conexión:



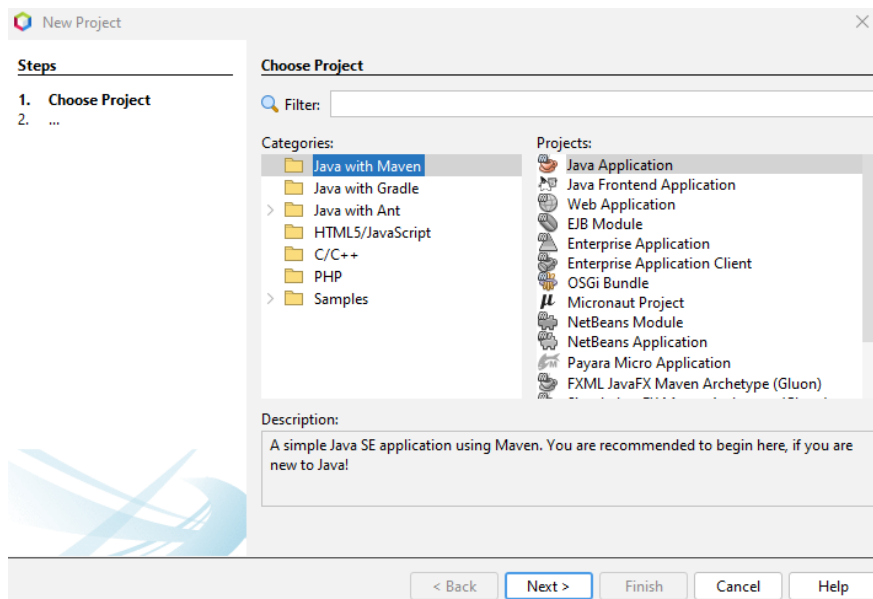
Sincronizamos con horario UTC

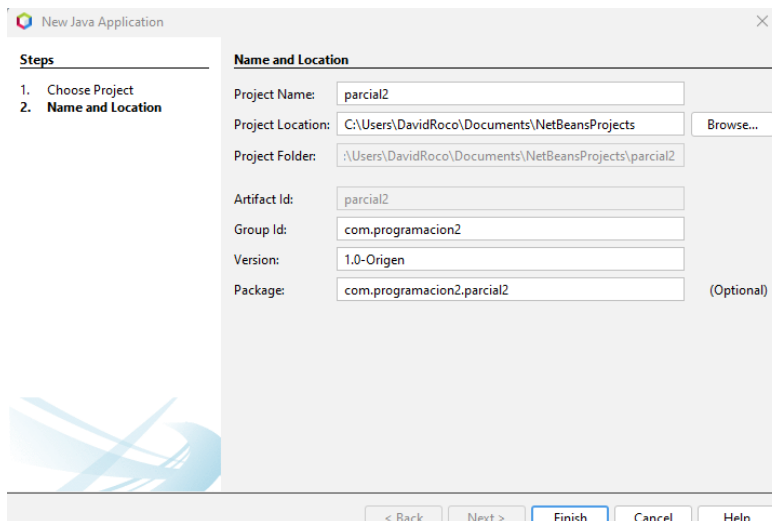
Podemos borrar: **zeroDateTimeBehavior=CONVERT_TO_NULL** [usjqw5bdeluw3eww on Default schema]

Y agregar: **serverTimezone=UTC**

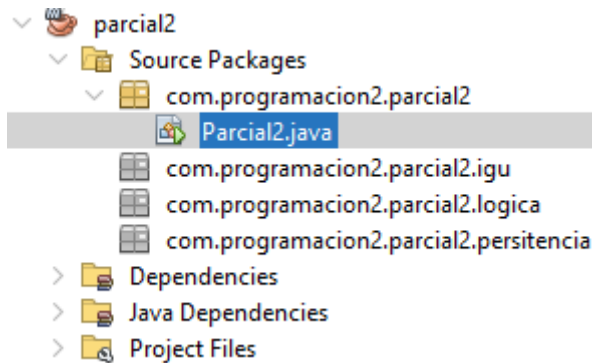
Debería quedarnos así: **jdbc:mysql://bmsnkprjdayiy2jtyfy0-mysql.services.clever-cloud.com:3306/bmsnkprjdayiy2jtyfy0?serverTimezone=UTC**

Ahora si, abrimos un nuevo proyecto Maven:

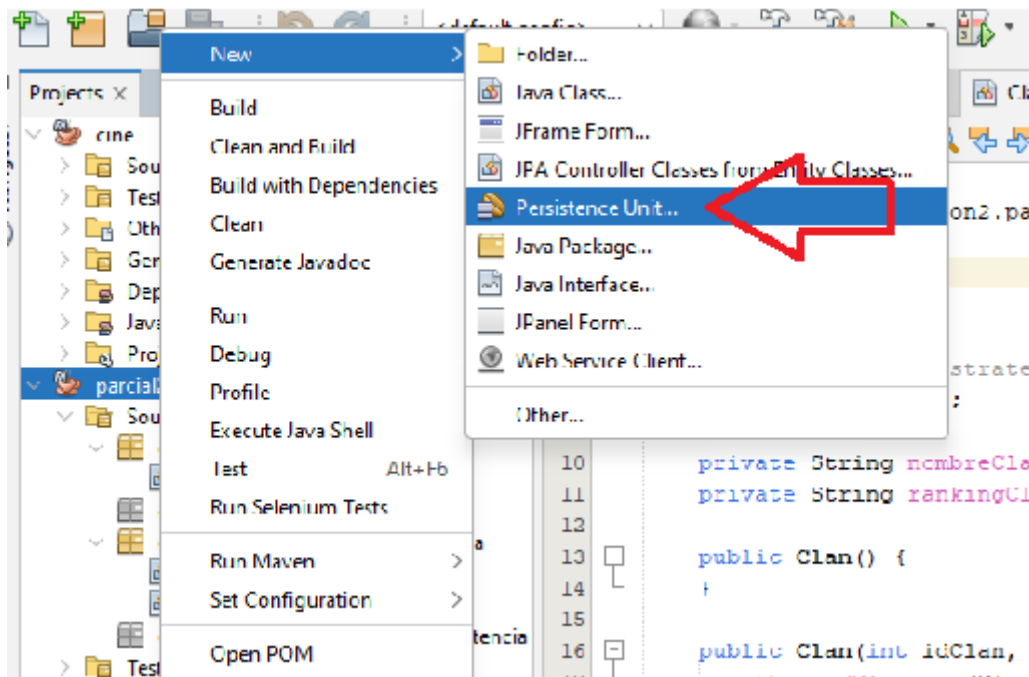




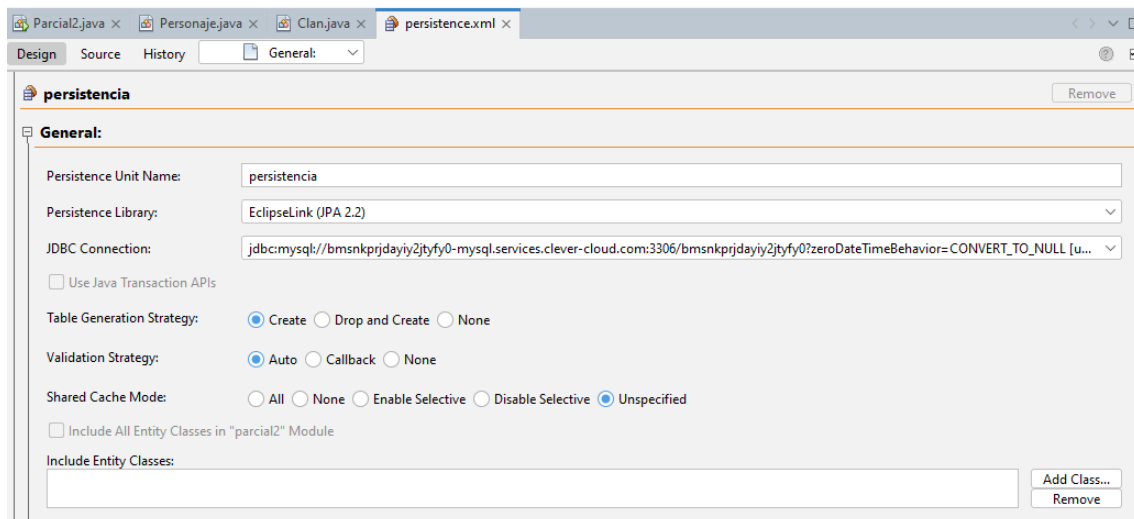
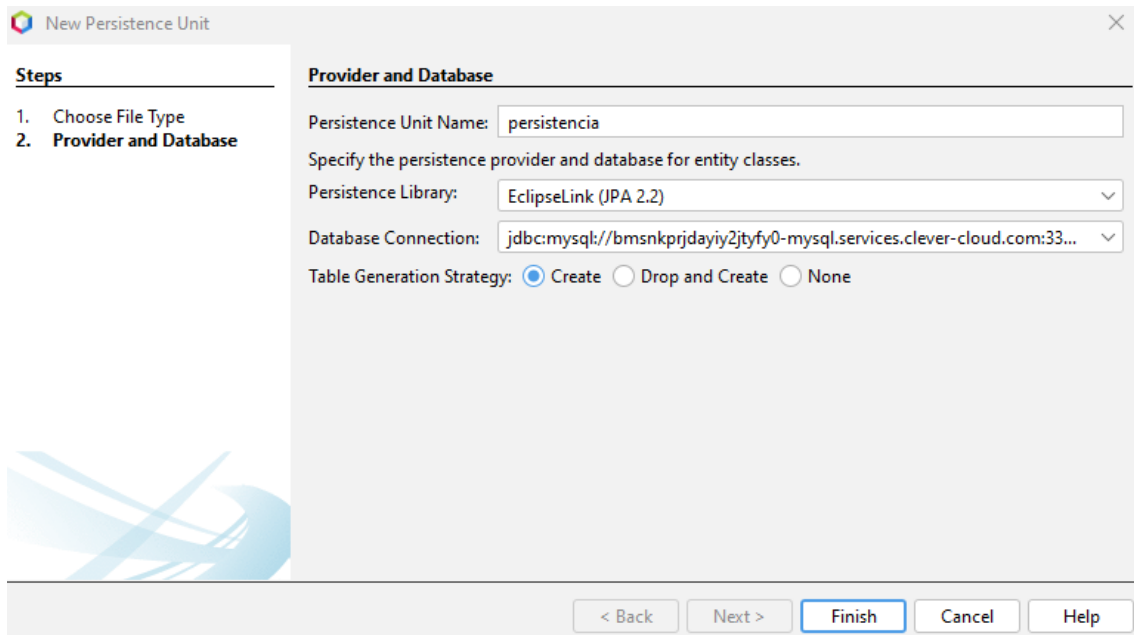
Creamos la estructura de directorios de tres capas, IGU, lógica y persistencia.



Luego vamos a agregar nuestra unidad de persistencia.



Cambiamos el nombre de la unidad de persistencia, seleccionamos EclipseLink (JPA 2.2) y nuestro String de conexión a la base de datos.



Todavía no podemos agregar las clases que van a machear con las tablas en nuestra base de datos porque no las hemos creado pero si ya se agregaron las dependencias a nuestro archivo de Maven pom.xml

pom.xml

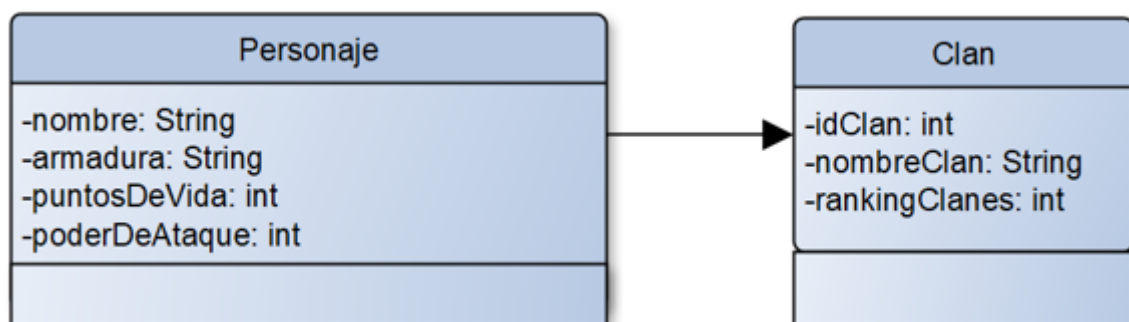
```
<dependency>
  <groupId>org.eclipse.persistence</groupId>
  <artifactId>org.eclipse.persistence.core</artifactId>
  <version>2.7.12</version>
</dependency>
<dependency>
  <groupId>org.eclipse.persistence</groupId>
  <artifactId>org.eclipse.persistence.asm</artifactId>
  <version>9.4.0</version>
</dependency>
<dependency>
  <groupId>org.eclipse.persistence</groupId>
  <artifactId>org.eclipse.persistence.antlr</artifactId>
  <version>2.7.12</version>
</dependency>
<dependency>
  <groupId>org.eclipse.persistence</groupId>
  <artifactId>org.eclipse.persistence.jpa</artifactId>
```

```

    <version>2.7.12</version>
  </dependency>
  <dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.jpa.jpql</artifactId>
    <version>2.7.12</version>
  </dependency>
  <dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.moxy</artifactId>
    <version>2.7.12</version>
  </dependency>
  <dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>jakarta.persistence</artifactId>
    <version>2.2.3</version>
  </dependency>
  <dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.jpa.modelgen.processor</artifactId>
    <version>2.7.12</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>8.0.33</version>
  </dependency>

```

Una vez ahí nos concentramos en crear las clases que representan nuestros objetos en la capa de lógica.



Creamos nuestros java class de Clan y luego de Personaje.

1. Generamos los atributos.
2. Constructores.
3. Getters and Setters.
4. toString.
5. Implementar la escritura JPA a los atributos y relaciones.

NOTA: Recordar agregar los import correspondientes.


```

2  package com.programacion2.parcial2.logica;
3
4  @Entity
5  @GeneratedValue(strategy = GenerationType.SEQUENCE)
6  private int idClan;
7  @Basic
8  private String nombreClan;
9  private String rankingClanes;

```

```

import javax.persistence.Basic;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

```

Clan.java

```

package com.programacion2.parcial2.logica;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Clan implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int idClan;
    @Basic
    private String nombreClan;
    private String rankingClanes;

    public Clan() {
    }

    public Clan(int idClan, String nombreClan, String rankingClanes) {
        this.idClan = idClan;
        this.nombreClan = nombreClan;
        this.rankingClanes = rankingClanes;
    }

    public int getIdClan() {
        return idClan;
    }

    public void setIdClan(int idClan) {
        this.idClan = idClan;
    }

    public String getNombreClan() {
        return nombreClan;
    }

```

```
}

public void setNombreClan(String nombreClan) {
    this.nombreClan = nombreClan;
}

public String getRankingClanes() {
    return rankingClanes;
}

public void setRankingClanes(String rankingClanes) {
    this.rankingClanes = rankingClanes;
}

@Override
public String toString() {
    return "Clan{" + "idClan=" + idClan + ", nombreClan=" + nombreClan + ",
rankingClanes=" + rankingClanes + "}";
}

}
```

Personaje.java

```
package com.programacion2.parcial2.logica;

import javax.persistence.Basic;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.OneToOne;

@Entity
public class Personaje {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int idPersonaje;
    @Basic
    private String nombre;
    private String armadura;
    private int puntosDeVida;
    private int poderDeAtaque;
    @ManyToOne
    private Clan clan;

    public Personaje() {
    }

    public Personaje(int idPersonaje, String nombre, String armadura, int puntosDeVida, int
poderDeAtaque, Clan clan) {
        this.idPersonaje = idPersonaje;
        this.nombre = nombre;
        this.armadura = armadura;
    }
}
```

```
this.puntosDeVida = puntosDeVida;
this.poderDeAtaque = poderDeAtaque;
this.clan = clan;
}

public int getIdPersonaje() {
    return idPersonaje;
}

public void setIdPersonaje(int idPersonaje) {
    this.idPersonaje = idPersonaje;
}

public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getArmadura() {
    return armadura;
}

public void setArmadura(String armadura) {
    this.armadura = armadura;
}

public int getPuntosDeVida() {
    return puntosDeVida;
}

public void setPuntosDeVida(int puntosDeVida) {
    this.puntosDeVida = puntosDeVida;
}

public int getPoderDeAtaque() {
    return poderDeAtaque;
}

public void setPoderDeAtaque(int poderDeAtaque) {
    this.poderDeAtaque = poderDeAtaque;
}

public Clan getClan() {
    return clan;
}

public void setClan(Clan clan) {
    this.clan = clan;
}

@Override
public String toString() {
```

```
        return "Personaje{" + "idPersonaje=" + idPersonaje + ", nombre=" + nombre + ",  
armadura=" + armadura + ", puntosDeVida=" + puntosDeVida + ", poderDeAtaque=" +  
poderDeAtaque + ", clan=" + clan + '}';  
    }  
  
}
```

Luego creamos los **DTOClan.java** y **DTOPersonaje.java** que tienen la funcionalidad de llevar la información del objeto a la capa gráfica con los atributos en STRING.

DTOClan.java

```
package com.programacion2.parcial2.logica;  
  
public class DTOClan {  
    private String idClan;  
    private String nombreClan;  
    private String rankingClanes;  
  
    public DTOClan() {  
    }  
  
    public DTOClan(String idClan, String nombreClan, String rankingClanes) {  
        this.idClan = idClan;  
        this.nombreClan = nombreClan;  
        this.rankingClanes = rankingClanes;  
    }  
  
    public String getIdClan() {  
        return idClan;  
    }  
  
    public void setIdClan(String idClan) {  
        this.idClan = idClan;  
    }  
  
    public String getNombreClan() {  
        return nombreClan;  
    }  
  
    public void setNombreClan(String nombreClan) {  
        this.nombreClan = nombreClan;  
    }  
  
    public String getRankingClanes() {  
        return rankingClanes;  
    }  
  
    public void setRankingClanes(String rankingClanes) {  
        this.rankingClanes = rankingClanes;  
    }  
  
    @Override  
    public String toString() {
```

```
        return "DTOClan{" + "idClan=" + idClan + ", nombreClan=" + nombreClan + ",  
        rankingClanes=" + rankingClanes + '}';  
    }  
  
}
```

DTOPersonaje

```
package com.programacion2.parcial2.logica;  
  
public class DTOPersonaje {  
    private String idPersonaje;  
    private String nombre;  
    private String armadura;  
    private String puntosDeVida;  
    private String poderDeAtaque;  
    private String nombreClan;  
  
    public DTOPersonaje() {  
    }  
  
    public DTOPersonaje(String idPersonaje, String nombre, String armadura, String  
    puntosDeVida, String poderDeAtaque, String nombreClan) {  
        this.idPersonaje = idPersonaje;  
        this.nombre = nombre;  
        this.armadura = armadura;  
        this.puntosDeVida = puntosDeVida;  
        this.poderDeAtaque = poderDeAtaque;  
        this.nombreClan = nombreClan;  
    }  
  
    public String getIdPersonaje() {  
        return idPersonaje;  
    }  
  
    public void setIdPersonaje(String idPersonaje) {  
        this.idPersonaje = idPersonaje;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public String getArmadura() {  
        return armadura;  
    }  
  
    public void setArmadura(String armadura) {  
        this.armadura = armadura;  
    }  
}
```

```
public String getPuntosDeVida() {
    return puntosDeVida;
}

public void setPuntosDeVida(String puntosDeVida) {
    this.puntosDeVida = puntosDeVida;
}

public String getPoderDeAtaque() {
    return poderDeAtaque;
}

public void setPoderDeAtaque(String poderDeAtaque) {
    this.poderDeAtaque = poderDeAtaque;
}

public String getNombreClan() {
    return nombreClan;
}

public void setNombreClan(String nombreClan) {
    this.nombreClan = nombreClan;
}

@Override
public String toString() {
    return "DTOPersonaje{" + "idPersonaje=" + idPersonaje + ", nombre=" + nombre + ",
armadura=" + armadura + ", puntosDeVida=" + puntosDeVida + ", poderDeAtaque=" +
poderDeAtaque + ", nombreClan=" + nombreClan + '}';
}

}
```

Luego creamos la controladora lógica, por ahora solo la clase y vamos a utilizar el **PATRÓN SINGLETÓN** porque queremos que solo exista una sola instancia de ese objeto, a posterior vamos a ir agregando los métodos cuando los necesitemos.

ControladoraLogica.java

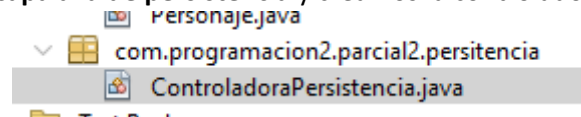
```
package com.programacion2.parcial2.logica;

public class ControladoraLogica {
    private static ControladoraLogica instance = null;

    public ControladoraLogica() {
    }

    public static ControladoraLogica getInstance() {
        if (instance == null) {
            instance = new ControladoraLogica();
        }
        return instance;
    }
}
```

Luego cambiamos de **capa a la de persistencia** y creamos la controladora de Persistencia



ControladoraPersistencia.java

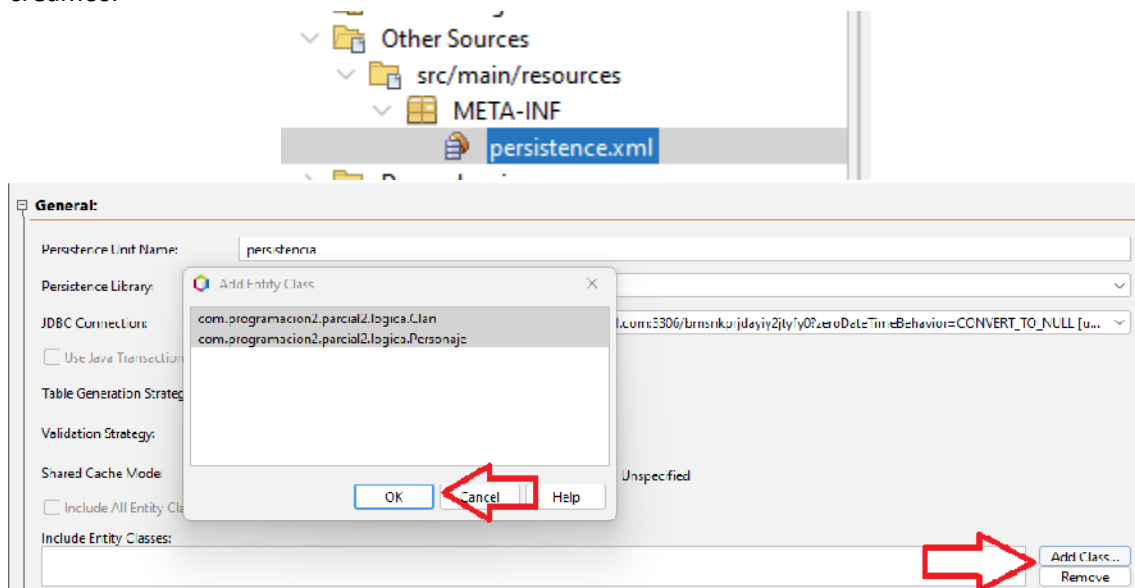
```
package com.programacion2.parcial2.persistencia;

public class ControladoraPersistencia {
    private static ControladoraPersistencia instance = null;

    public ControladoraPersistencia() {
    }

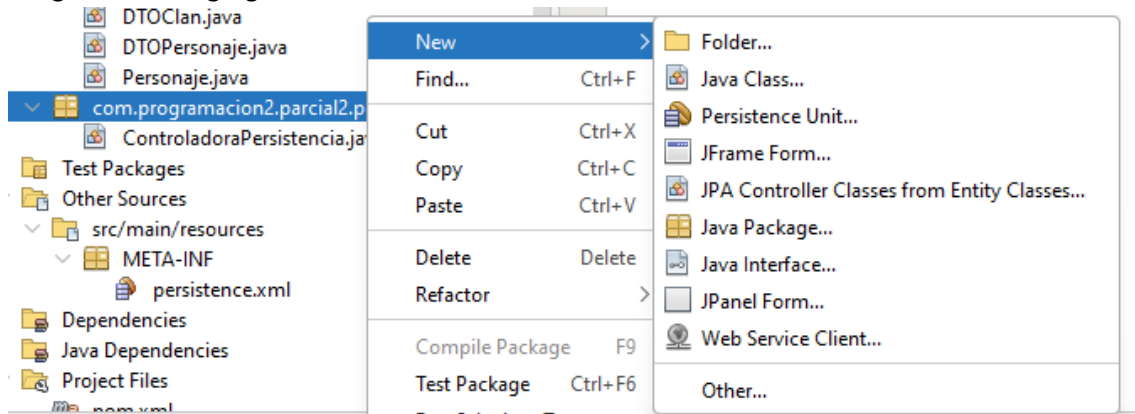
    public static ControladoraPersistencia getInstance() {
        if (instance == null) {
            instance = new ControladoraPersistencia();
        }
        return instance;
    }
}
```

Ahora volvemos a nuestra unidad de persistencia y agregamos las clases que recientemente creamos.

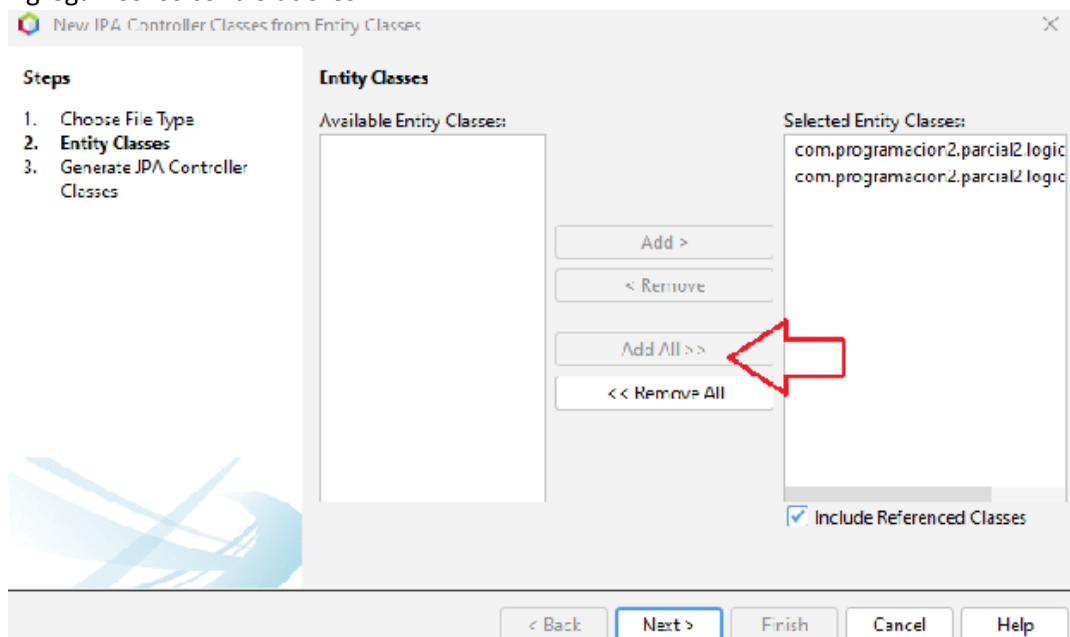


```
<?xml version="1.0" encoding="UTF 8"?>
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://
<persistence-unit name="persistencia" transaction-type="RESOURCE_LOCAL">
  <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
  <class>com.programacion2.parcial2.logica.Clan</class>
  <class>com.programacion2.parcial2.logica.Personaje</class>
  <properties>
    <property name="javax.persistence.jdbc.url" value="jdbc:mysql://bmsnkprjdayiy2jtyfy0-m
    <property name="javax.persistence.jdbc.user" value="usjwg5bdeuw3eww"/>
    <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
    <property name="javax.persistence.jdbc.password" value="01x9j5UpAU0NCU902KS3"/>
    <property name="javax.persistence.schema-generation.database.action" value="create"/>
  </properties>
</persistence-unit>
</persistence>
```

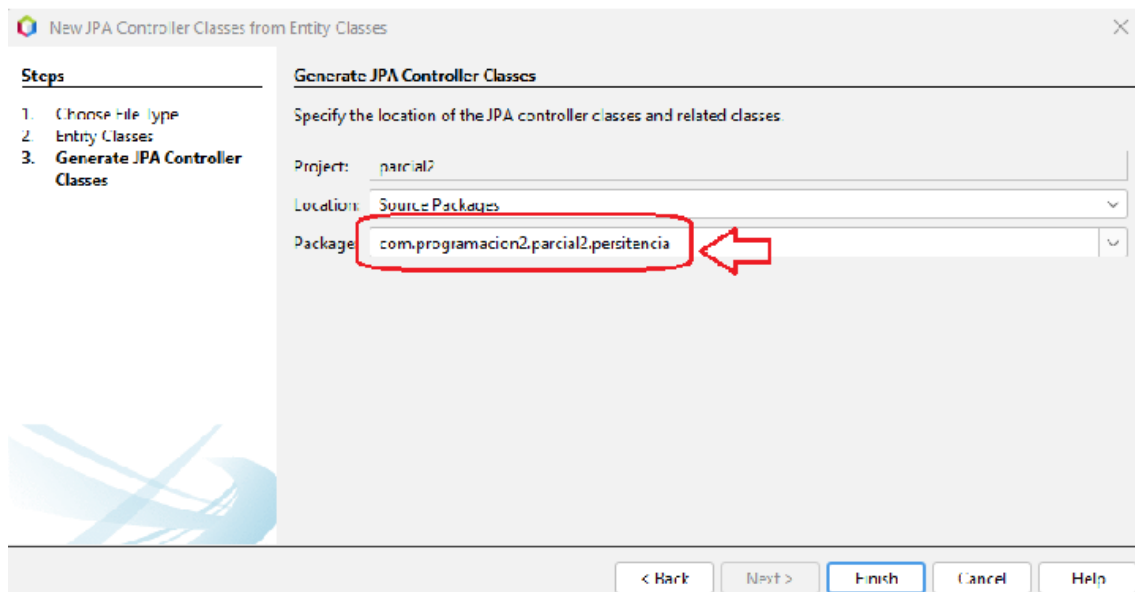
Luego vamos a agregar los controladores de las entidades de JPA.



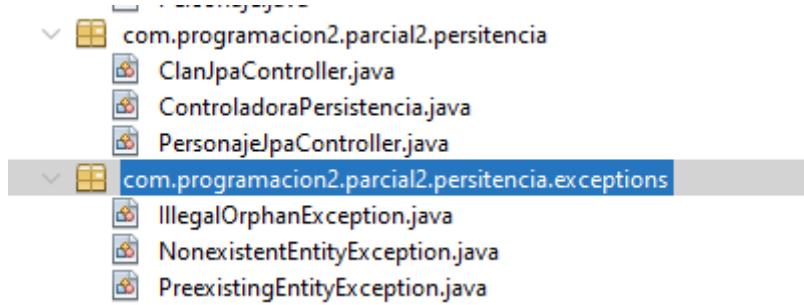
Agregamos los controladores



Prestar atención en ubicarlos en la carpeta de **Persistencia**.



NOTA: Se crearon los controladores y una nueva carpeta de excepciones.



Abrimos el controlador y borramos el constructor por defecto y agregamos la conexión a nuestra unidad de persistencia.

```
public class ClanJpaController implements Serializable {
    public ClanJpaController(EntityManagerFactory emf) {
        this.emf = emf;
    }
    private EntityManagerFactory emf = null;

    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }
}
```

```
...
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
...
public class ClanJpaController implements Serializable {

    private EntityManagerFactory emf = null;

    public ClanJpaController() {
        emf = Persistence.createEntityManagerFactory("persistencia");
    }

    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }
}
....
```

```
...
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
...
public class PersonajeJpaController implements Serializable {
    public PersonajeJpaController() {
        emf = Persistence.createEntityManagerFactory("persistencia");
    }

    private EntityManagerFactory emf = null;

    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }
}
```

Bien hasta ahora está implementada la configuración JPA con las entidades, ahora vamos a concentrarnos en la etapa dinámica de nuestro sistema.

Creamos un JFrame MAIN que va a direccionarnos a nuestros ABMs.

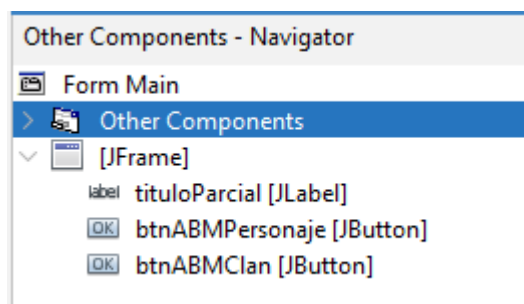
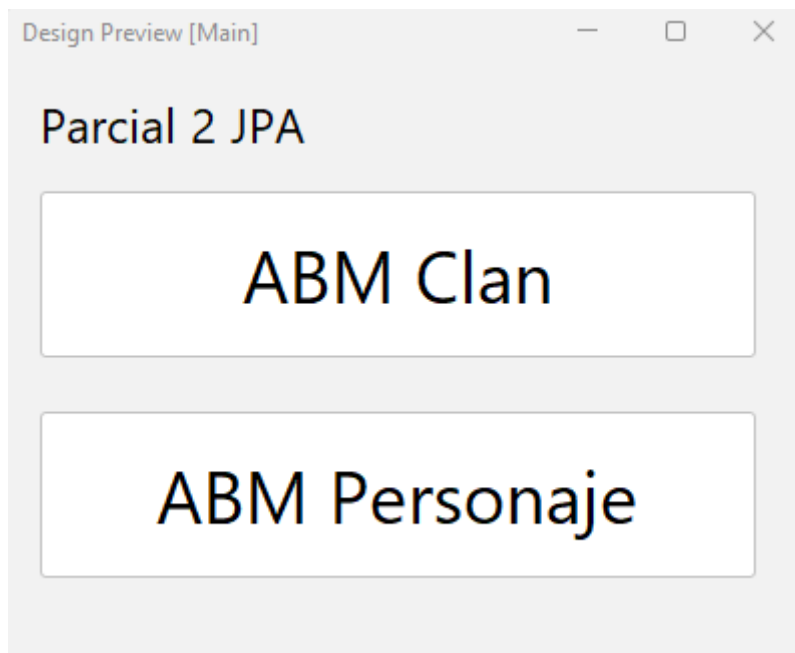
- Aplicamos el patrón singleton.
- Y borramos el método main arg del jfrom para borrar el símbolo de play verde.

```
...
public class Main extends javax.swing.JFrame {

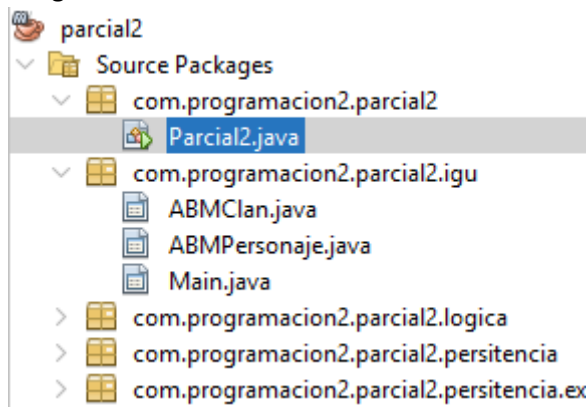
    private static Main instance = null;

    public Main() {
        initComponents();
    }

    public static Main getInstance() {
        if (instance == null) {
            instance = new Main();
        }
        return instance;
    }
}
...
```



Luego volvemos al inicio a nuestro método main inicial del sistema.



Parcial2.java

```
package com.programacion2.parcial2;

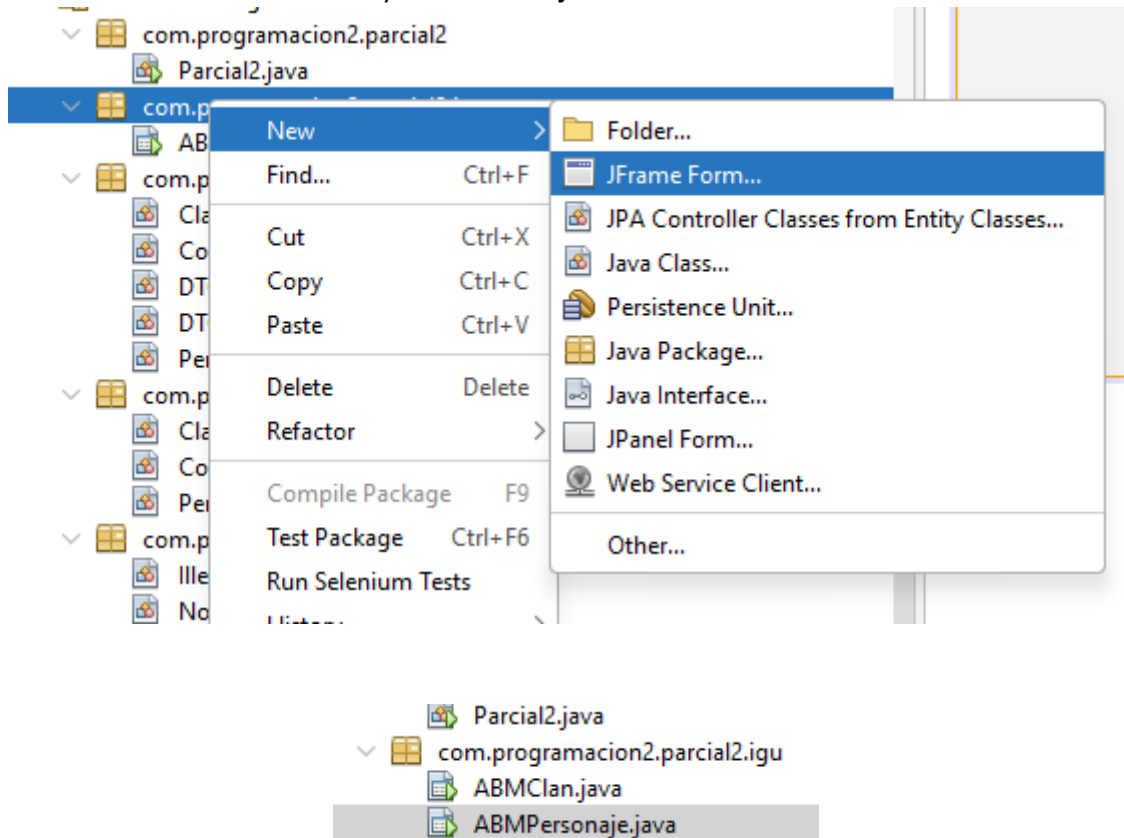
import com.programacion2.parcial2.igu.Main;

public class Parcial2 {

    public static void main(String[] args) {
        Main pantallaInicial = Main.getInstance();
        pantallaInicial.setVisible(true);
        pantallaInicial.setLocationRelativeTo(null);
    }
}
```

Con esto ya podemos probar la compilación.

Creamos dos JFrame **ABMClan** y **ABMPersonaje**



Vamos a notar que aparece el símbolo de play en verde, para borrarlo debemos borrar el método main.

```

1  /**
2  * @param args the command line arguments
3  */
4  public static void main(String args[]) {
5      /* Set the Nimbus look and feel */
6      Look and feel setting code (optional)
7
8      /* Create and display the form */
9      java.awt.EventQueue.invokeLater(new Runnable() {
10         public void run() {
11             new ABMPersonaje().setVisible(true);
12         }
13     });
14 }

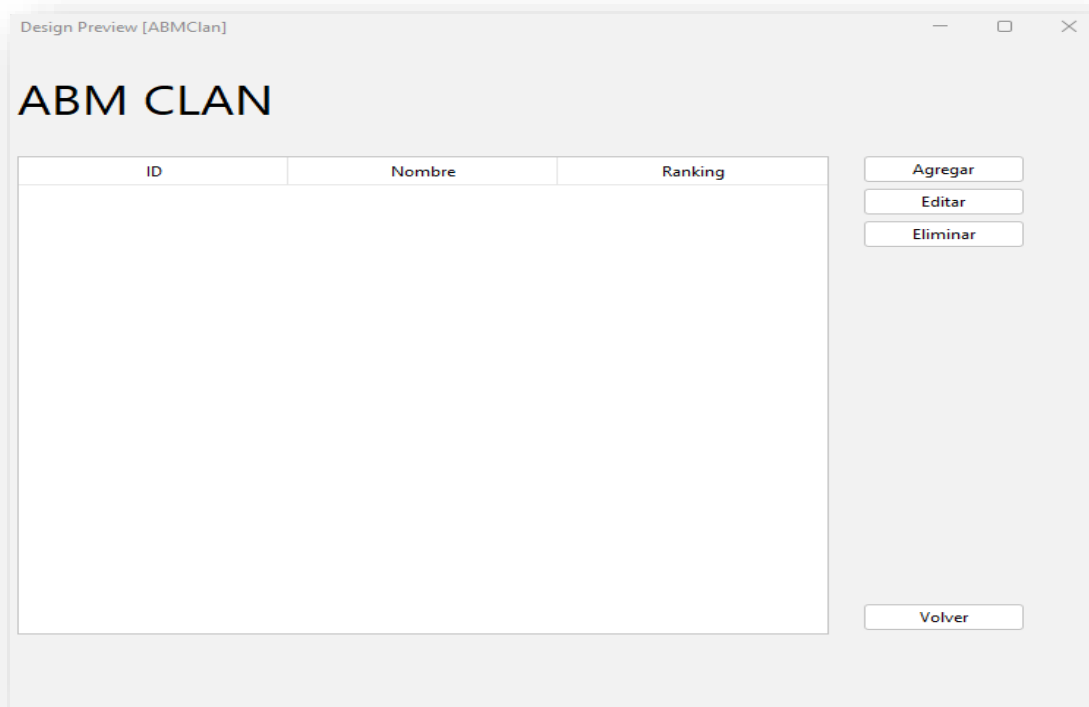
```

```

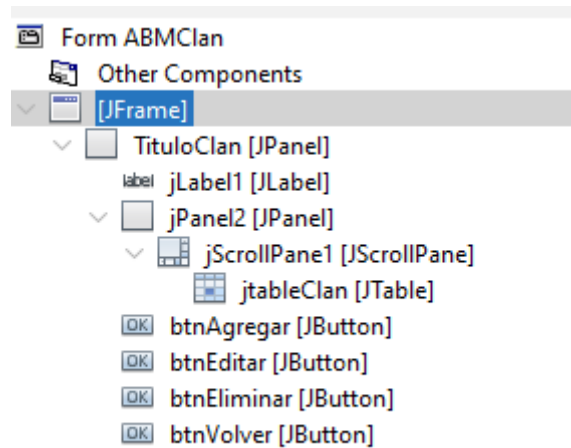
com.programacion2.parcial2.igu
├── ABMClan.java
├── ABMPersonaje.java
└── com.programacion2.parcial2.logica

```

Iniciamos por el ABMClan.



Creamos los elementos, la tabla y a funcionalidades de los 4 botones de acción.
Renombramos los componentes.



Luego aplicamos el **patrón singleton** a al JFROM para que sea única la ventana de la interfaz y vinculamos a la Controladora Lógica.

```
...
public class ABMClan extends javax.swing.JFrame {

    ControladoraLogica controla = new ControladoraLogica();
    private static ABMClan instance = null;

    public ABMClan() {
        initComponents();
    }

    public static ABMClan getInstance() {
        if (instance == null) {
            instance = new ABMClan();
        }
        return instance;
    }
}
...
```

El primer método que vamos a implementar es el de manejo volver al jfrom Main.

```
...
private void btnVolverActionPerformed(java.awt.event.ActionEvent evt) {
    this.dispose();
}
...
```

Luego vinculamos jfrom Main con el jfrom ABMClan con el método btnABMClan en dicho archivo origen.

```
...
private void btnABMClanActionPerformed(java.awt.event.ActionEvent evt) {
    ABMClan pantallaInicial = ABMClan.getInstance();
    pantallaInicial.setVisible(true);
    pantallaInicial.setLocationRelativeTo(null);
}
...
```

Ahora podemos probar la navegación entre ventanas con **RUN Project**

Luego nos vamos a concentrar en método agregarClan para crear nuestras primeras instancias. Para ello vamos a crear el jfrom de agregarClan donde vamos a ingresar nuestros atributos.



Borramos el método main inicial de jfrom y aplicamos el patrón de diseño singleton.

```
...
package com.programacion2.parcial2.igu;

import com.programacion2.parcial2.logica.ControladoraLogica;

public class AgregarClan extends javax.swing.JFrame {

    ControladoraLogica controla = new ControladoraLogica();
    private static AgregarClan instance = null;
    private DTOClan dtoClan= new DTOClan();

    public AgregarClan() {
        initComponents();
    }

    public static AgregarClan getInstance() {
        if (instance == null) {
            instance = new AgregarClan();
        }
        return instance;
    }
    ...
}
```

Generamos los componentes y el btn volver similar al manejo de interfaces anteriormente.

Importamos la clase DTOClan para completar la información ingresada por el usuario.

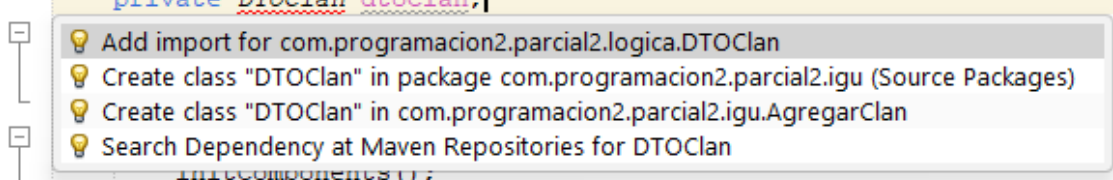
```
package com.programacion2.parcial2.igu;

import com.programacion2.parcial2.logica.ControladoraLogica;

public class AgregarClan extends javax.swing.JFrame {

    ControladoraLogica controla = new ControladoraLogica();
    private static AgregarClan instance = null;
    private DTOClan dtoClan;

    initComponents();
}
```



- ⚡ Add import for com.programacion2.parcial2.logica.DTOClan
- ⚡ Create class "DTOClan" in package com.programacion2.parcial2.igu (Source Packages)
- ⚡ Create class "DTOClan" in com.programacion2.parcial2.igu.AgregarClan
- ⚡ Search Dependency at Maven Repositories for DTOClan

Ahora vinculamos las ventanas ABMClan con AgregarClan

```
...
private void btnAgregarClanActionPerformed(java.awt.event.ActionEvent evt) {
    AgregarClan pantallaAgregarClan = AgregarClan.getInstance();
    pantallaAgregarClan.setVisible(true);
    pantallaAgregarClan.setLocationRelativeTo(null);
}
...
```

Bien luego vamos a cargar el DTO con la información agregada or el usuario según el workflow de nuestro escenario.

```
...
private void btnAgregarActionPerformed(java.awt.event.ActionEvent evt) {
    dtoClan.setNombreClan(txtNombreClan.getText());
    dtoClan.setRankingClanes(txtRankingClan.getText());
    controla.agregarClan(dtoClan);
    mostrarMensajes("Registro exitoso", "Info", "Se guardó El descuento en la BD");
    this.dispose();
}
...
```

NOTA: El método mostrar mensaje se utiliza para trabajar con interfaces de usuarios, se va a explicar a continuación pero se debe agregar en cada JFrame con interacción con el usuario.

```
...
public void mostrarMensajes(String mensaje, String tipo, String titulo){
    JOptionPane optionPane = new JOptionPane(mensaje);
    if (tipo.equals("Info")){
        optionPane.setMessageType(JOptionPane.INFORMATION_MESSAGE);
    }
    else if (tipo.equals("Error")){
        optionPane.setMessageType(JOptionPane.ERROR_MESSAGE);
    }
    JDialog dialog = optionPane.createDialog(titulo);
    dialog.setAlwaysOnTop(true);
    dialog.setVisible(true);
}
...
```

Como vemos `controla.agregarClan(dtoClan)`; ese método hay que implementarlo en nuestra controladora lógica que va a transformar ese DTOClan en un objeto Clan, que luego va a enviar a la controladora de persistencia y esta va a llamar al JPAcontroller indicado para posteriormente ubicarlo en la base de datos.

```

143         dtoClan.setNombreClan(nombreClan: txtNombreClan.getText());
144         dtoClan.setRankingClanes(rankingClanes: txtRankingClan.getText());
145         controla.agregarClan(dtoClan);
146         // Create method "agregarClan(com.programacion2.parcial2.logica.DTOClan)" in com.programacion2.parcial2.logica.ControladoraLogica
147         this.dispose();
148     }

```

ControladoraLogica

```

...
public void agregarClan(DTOClan dtoClan) {
    Clan clan = new Clan();
    clan.setNombreClan(dtoClan.getNombreClan());
    clan.setRankingClanes(dtoClan.getRankingClanes());

    ControladoraPersistencia persistencia = ControladoraPersistencia.getInstance();
    persistencia.guardarClan(clan);
}
...

```

Ídem al paso anterior debemos crear el método en la controladora de persistencia.

```

24
23         ControladoraPersistencia persistencia = ControladoraPersistencia.getInstance();
24         persistencia.guardarClan(clan);
25         // Create method "guardarClan(com.programacion2.parcial2.logica.Clan)" in com.programacion2.parcial2.persistencia.ControladoraPersistencia
26

```

ControladoraPersistencia

```

...
package com.programacion2.parcial2.persistencia;

import com.programacion2.parcial2.logica.Clan;

public class ControladoraPersistencia {
    private static ControladoraPersistencia instance = null;
    ClanJpaController clanJPA = new ClanJpaController();

    public ControladoraPersistencia() {
    }

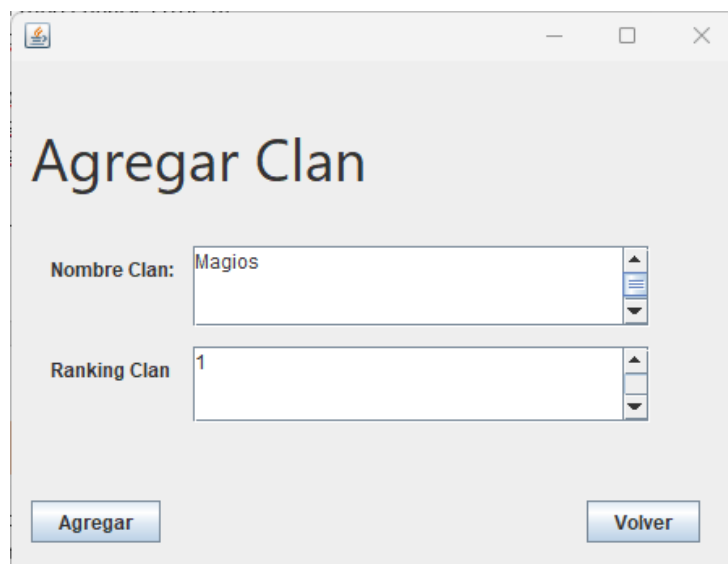
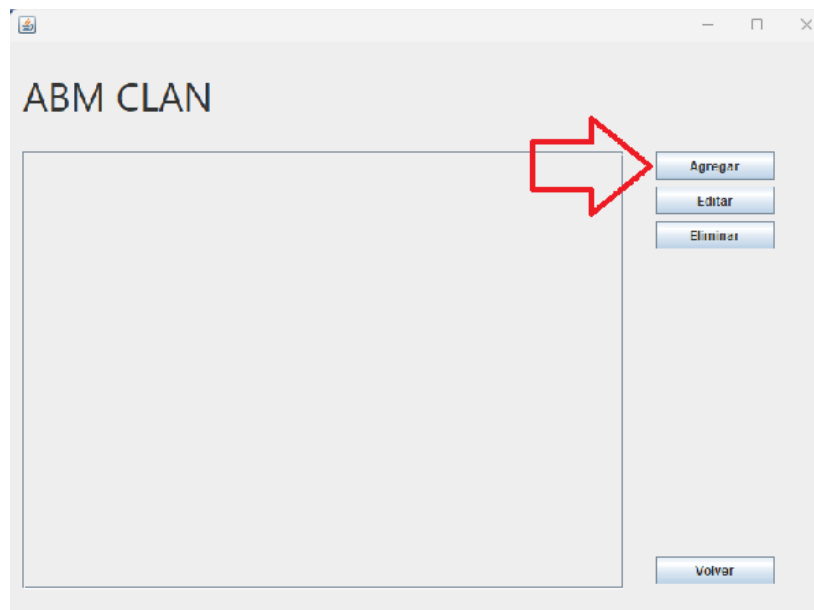
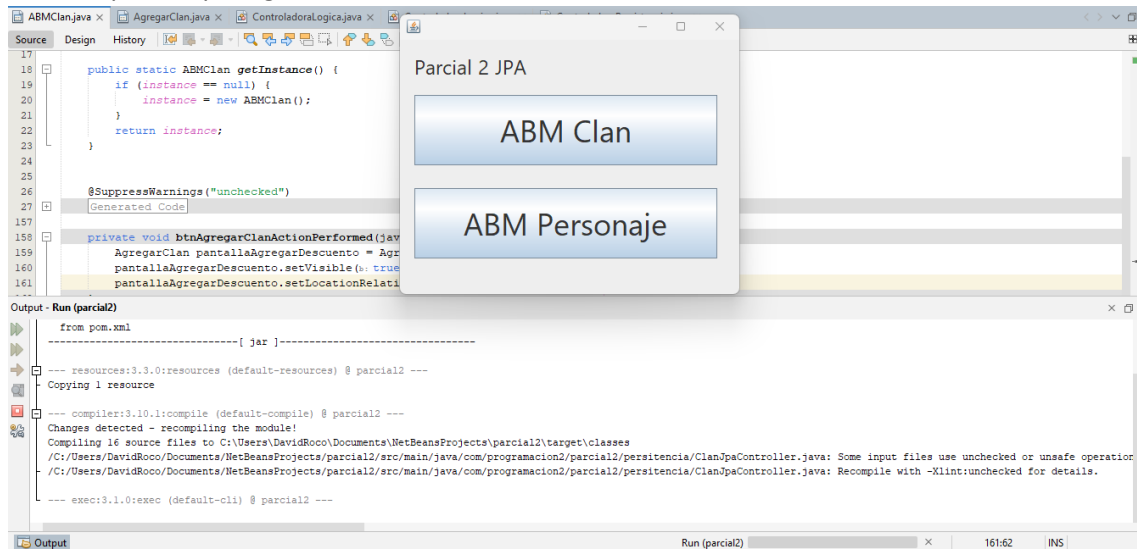
    public static ControladoraPersistencia getInstance() {
        if (instance == null) {
            instance = new ControladoraPersistencia();
        }
        return instance;
    }

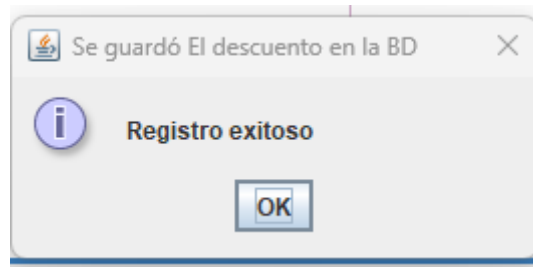
    public void guardarClan(Clan clan) {
        clanJPA.create(clan);
    }
}
...

```

Instanciamos una instancia del ClanJpaController y creamos el método en el guardarClan. Grabamos todo y probamos nuestro sistema.

Primera prueba para guardar Clan.



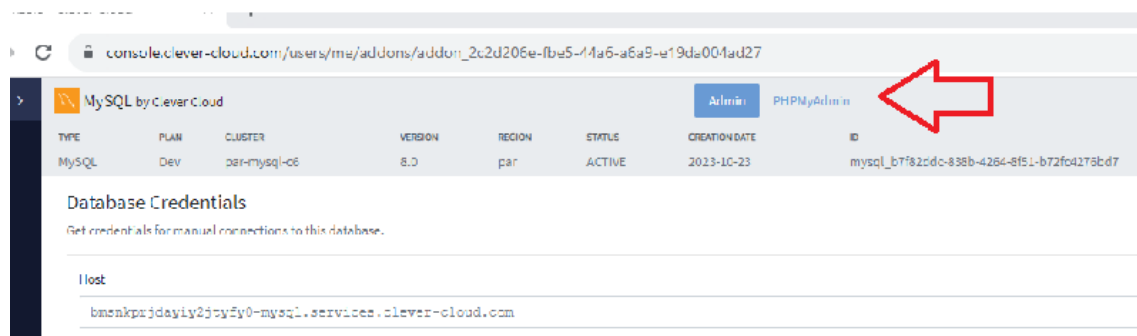


```

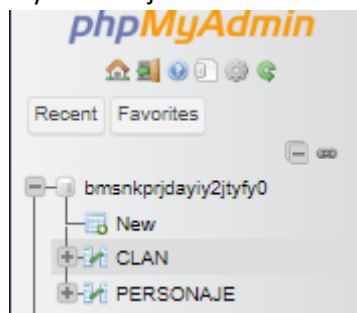
Output - Run (parcial2)
-----
Building parcial2 1.0-Origin
from pom.xml
-----[ jar ]-----
--- resources:3.2.0:resources (default-resources) @ parcial2 ---
Copying 1 resource
--- compiler:3.10.1:compile (default-compile) @ parcial2 ---
Changes detected - recompiling the module!
Compiling 16 source files to C:/Users/DavidRoco/Documents/NetBeansProjects/parcial2/target/classes
C:/Users/DavidRoco/Documents/NetBeansProjects/parcial2/src/main/java/com/programacion2/parcial2/persistencia/ClanJpaController.java: Some input files use unchecked or unsafe operations.
C:/Users/DavidRoco/Documents/NetBeansProjects/parcial2/src/main/java/com/programacion2/parcial2/persistencia/ClanJpaController.java: Recompile with -Xlint:unchecked for details.
--- exec:3.1.0:exec (default-cli) @ parcial2 ---
[EL Info]: 2023-10-24 10:50:57.622--ServerSession(1034172804)--EclipseLink, version: Eclipse Persistence Services - 2.7.12.v20230209-a5c4074e23

```

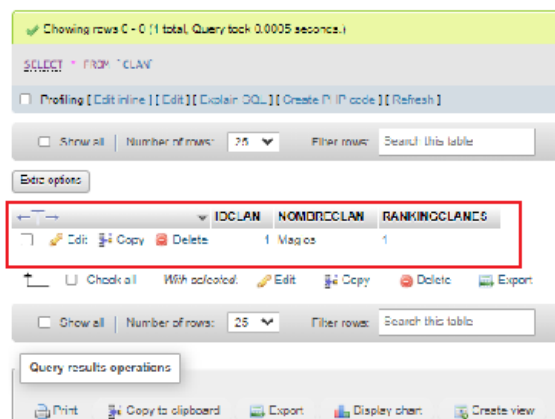
Ahora vamos al php MyAdmin de nuestra base de datos para comprobar que la Tabla y la instancia se hayan creado correctamente.



Vemos que se creó las tablas Clan y Personaje.



La instancia también se ha creado.



NOTA: Se originó un error anteriormente que se solucionó agregando la dependencia de jdbc driver en el archivo pom.xml

```
...
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>8.0.33</version>
</dependency>
...
```

De la misma forma creamos las instancias requerida por la consigna.

	IDCLAN	NOMBRECLAN	RANKINGCLANES
<input type="checkbox"/> Edit Copy Delete	1	Magios	1
<input type="checkbox"/> Edit Copy Delete	2	NoHomero	2
<input type="checkbox"/> Edit Copy Delete	3	Peces del Infierno	3
<input type="checkbox"/> Edit Copy Delete	4	Santos Orantes	4
<input type="checkbox"/> Edit Copy Delete	5	Amigos de los Pinos	5

☐ Check all With selected: Edit Copy Delete Export

Ahora vamos a implementar el método de cargar tabla que van a traer las instancias de los objetos de la tabla Clan que existen.

ABMClan.jffrom

```
...
public void cargarTabla() {

    DefaultTableModel tabla = new DefaultTableModel(){

        //filas y columnas no editables
        @Override
        public boolean isCellEditable (int row, int column){
            return false;
        }
    };
    //establecer nombre y columnas

    String titulos[] = {"ID CLAN", "NOMBRE CLAN ", "RANKING CLANES"};
    tabla.setColumnIdentifiers(titulos);

    //carga de los datos desde la base de datos
    List <DTOClan> listaClanes = controla.traerClanes();

    //recorrer la lista y mostrar cada uno de llos elementos en la tabla

    if (listaClanes !=null){
        for (DTOClan dtoClan : listaClanes){
            Object[] objeto =
            {dtoClan.getIdClan(),dtoClan.getNombreClan(),dtoClan.getRankingClanes()};

            tabla.addRow(objeto);
        }
    }
}
```

```

    }
    jTableClan.setModel(tabla);

    }
    ...

```

Ahora vamos a implementar el método traerClanes() en la controladora lógica.

ControladoraLogica.java

```

...
public List<DTOClan> traerClanes() {
    ControladoraPersistencia persistencia = ControladoraPersistencia.getInstance();
    List <Clan> listaClanes = persistencia.buscarClanes();
    ArrayList <DTOClan> listaDTOClanes = new ArrayList<>();
    int index = 0;
    for (Clan clan: listaClanes){
        DTOClan dtoClan = new DTOClan();
        String idClan = String.valueOf(clan.getIdClan());
        dtoClan.setIdClan(idClan);
        String nombreClan = clan.getNombreClan();
        dtoClan.setNombreClan(nombreClan);
        String rankingClanes = clan.getRankingClanes();
        dtoClan.setRankingClanes(rankingClanes);

        listaDTOClanes.add(index, dtoClan);
        index += 1;
    }
    return listaDTOClanes;
}
...

```

Ahora hay que implementar el método en la controladora de persistencia.

ControladoraPersistencia.java

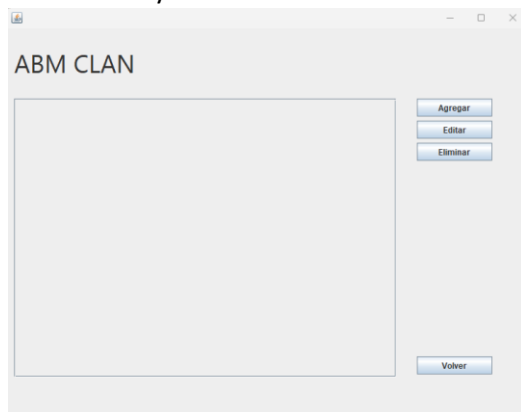
```

....
public List<Clan> buscarClanes() {
    List<Clan> listaClan = clanJPA.findClanEntities();
    return listaClan;
}
....

```

Listo a probar el método.

Guardamos y damos RUN a nuestro sistema.



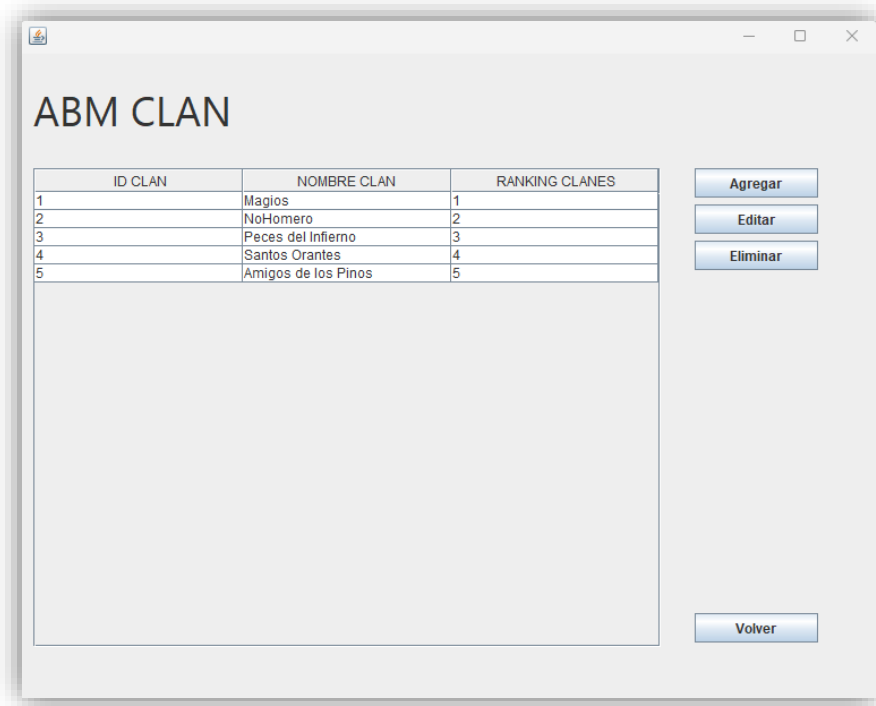
Vemos que no trae porque no hemos colocado el método cargarTabla() en el ABMClan.

ABMClan.jffrom

```
...
private void formWindowOpened(java.awt.event.WindowEvent evt) {
    cargarTabla();
}

private void formWindowGainedFocus(java.awt.event.WindowEvent evt) {
    cargarTabla();
}
...
```

_Ahora si probamos nuevamente



Bien ahora vamos a implementar el método de eliminar instancia de Clan.

ABMClan.jffrom

```
...
private void btnEliminarClanActionPerformed(java.awt.event.ActionEvent evt) {
    //hay mas de una fila en la tabla
    if (jtableClan.getRowCount() > 0) {
        //hay por lo menos una selección
        if (jtableClan.getSelectedRow() != -1) {
            //optengo la id a borrar
            int idClan =
                Integer.parseInt(String.valueOf(jtableClan.getValueAt(jtableClan.getSelectedRow(), 0)));

            controla.borrarClan(idClan);

            mostrarMensajes("Se Eliminó correctamente La Sala","Info", "Eliminación
Exitoso");
        }
        else{
            mostrarMensajes("No Seleccionó ningún alumno","Error", "No se eliminó el
registro");
        }
    }
}
```

```
    }  
  }  
  else{  
    mostrarMensajes("No hay registros para eliminar","Error", "No se eliminó el  
registro");  
  }  
  cargarTabla();  
}  
...
```

Ahora implementamos el método borrar Clan en la controladora Lógica.

ControladoraLogica.java

```
...  
public void borrarClan(int idClan) {  
    ControladoraPersistencia persistencia = ControladoraPersistencia.getInstance();  
    persistencia.borrarClan(idClan);  
}  
...
```

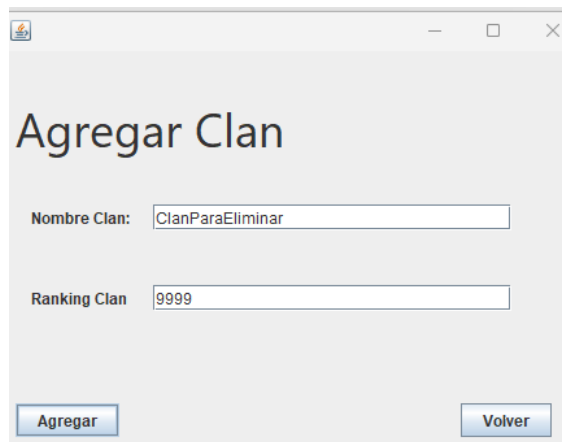
Ahora lo implementamos el método borrarClan(idClan) en la controladora de Persistencia.

ControladoraPersistencia.java

```
...  
public void borrarClan(int idClan) {  
    try {  
        clanJPA.destroy(idClan);  
    } catch (NonexistentEntityException ex) {  
        Logger.getLogger(ControladoraPersistencia.class.getName()).log(Level.SEVERE, null,  
ex);  
    }  
}  
...
```

Ahora a Probar!!!

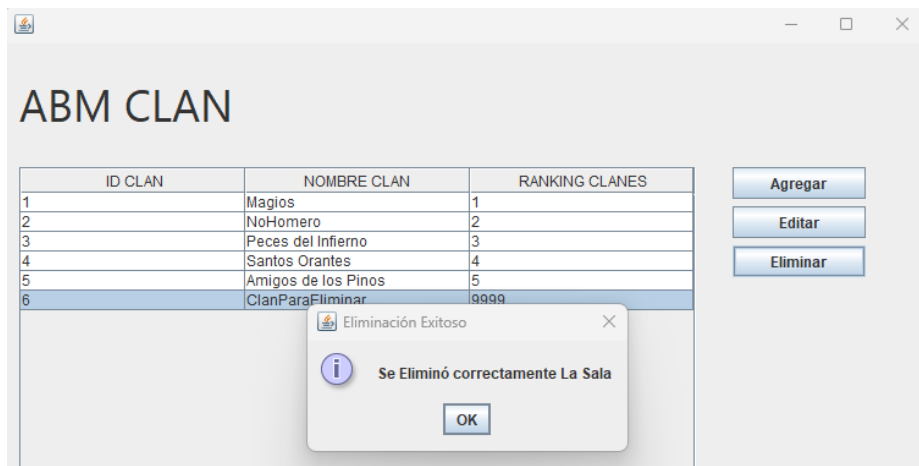
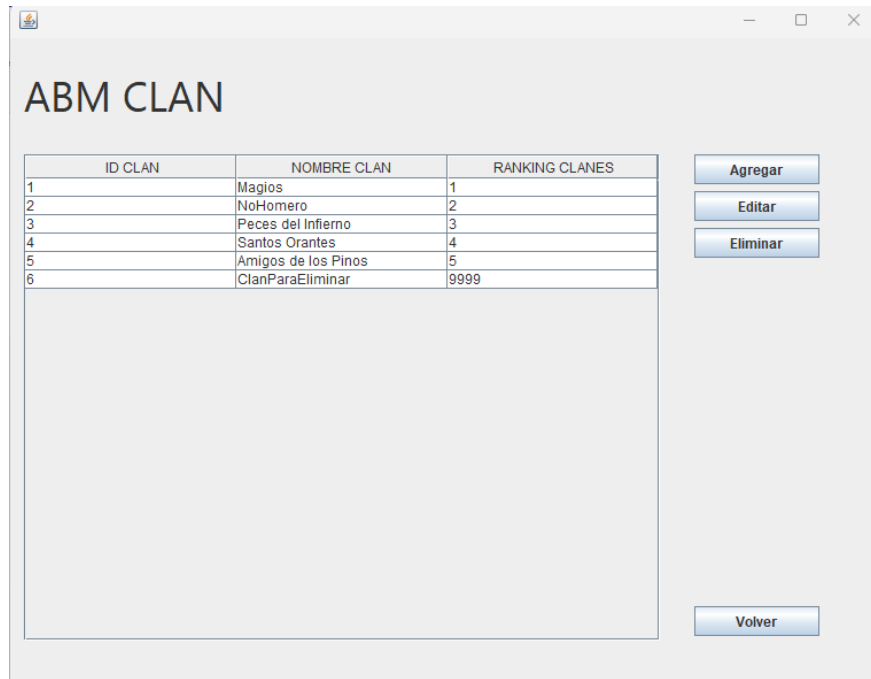
Para ello creamos una nueva instancia y luego la borramos.



Agregar Clan

Nombre Clan:

Ranking Clan:



ABM CLAN

ID CLAN	NOMBRE CLAN	RANKING CLANES	Agregar
1	Magios	1	Editar
2	NoHomero	2	Eliminar
3	Peces del Infierno	3	
4	Santos Orantes	4	
5	Amigos de los Pinos	5	

El método que nos queda por implementar es el de editar, para ello, se necesita crear un nuevo JFROM similar al agregar clan.

ABMClan.jfrom

```
...
private void btnEditarClanActionPerformed(java.awt.event.ActionEvent evt) {
    //hay mas de una fila en la tabla
    if (jtableClan.getRowCount() > 0) {
        //hay por lo menos una selección
        if (jtableClan.getSelectedRow() != -1) {
            //optengo la id a Editar.
        }
    }
}
```

```

        int idClan =
Integer.parseInt(String.valueOf(jtableClan.getValueAt(jtableClan.getSelectedRow(), 0)));

        EditarClan pantallaEditarClan = new EditarClan(idClan);
        pantallaEditarClan.setVisible(true);
        pantallaEditarClan.setLocationRelativeTo(null);

        mostrarMensajes("Se Editó correctamente El Clan","Info", "Edición exitosa");
    }
    else{
        mostrarMensajes("No Seleccionó ningún CLAN ","Error", "No se eliminó el
registro");
    }
}
else{
    mostrarMensajes("No hay registros para eliminar","Error", "No se eliminó el
registro");
}
cargarTabla();

}
...

```

Prestar atención que en este caso no es conveniente aplicar el patrón Singleton ya que al momento de crear la edición necesita enviar un idClan que va a depender de cada vez que se ejecute el método.

EditarClan.java

```

...
public class EditarClan extends javax.swing.JFrame {

    ControladoraLogica controla = new ControladoraLogica();
    private DTOClan dtoClan= new DTOClan();
    int idClan;

    public EditarClan(int idClan) {
        this.idClan = idClan;
        initComponents();
        cargarClan();
    }
    ...
    private void btnEditarActionPerformed(java.awt.event.ActionEvent evt) {

        dtoClan.setIdClan(String.valueOf(this.idClan));
        dtoClan.setNombreClan(txtNombreClan.getText());
        dtoClan.setRankingClanes(txtRankingClan.getText());
        controla.editarClan(dtoClan);
        this.dispose();
    }
    ...
    private void cargarClan() {
        dtoClan = controla.buscarClan(this.idClan);

        txtNombreClan.setText(dtoClan.getNombreClan());
        txtRankingClan.setText(dtoClan.getRankingClanes());
    }
    ...
}

```


El método `editarClan(dtoClan)` y `buscarClan(idClan)` vamos a implementarlo en la controladora lógica.

ControladoraLogica.java

```
...
public DTOClan buscarClan(int idClan) {
    ControladoraPersistencia persistencia = ControladoraPersistencia.getInstance();
    Clan clan = new Clan();
    DTOClan dtoClan = new DTOClan();

    clan = persistencia.buscarClan(idClan);

    dtoClan.setIdClan(String.valueOf(clan.getIdClan()));
    dtoClan.setNombreClan(clan.getNombreClan());
    dtoClan.setRankingClanes(clan.getRankingClanes());

    return dtoClan;
}
...
public void editarClan(DTOClan dtoClan) {
    ControladoraPersistencia persistencia = ControladoraPersistencia.getInstance();
    Clan clan = new Clan();

    clan.setIdClan(Integer.parseInt(dtoClan.getIdClan()));
    clan.setNombreClan(dtoClan.getNombreClan());
    clan.setRankingClanes(dtoClan.getRankingClanes());
    persistencia.editar(clan);

}
...
```

Ahora debemos implementar esos métodos en la controladora de persistencia.

ControladoraPersistencia.java

```
...
public Clan buscarClan(int idClan) {
    return clanJPA.findClan(idClan);
}
...
public void editar(Clan clan) {
    try {
        clanJPA.edit(clan);
    } catch (Exception ex) {
        Logger.getLogger(ControladoraPersistencia.class.getName()).log(Level.SEVERE, null,
ex);
    }
}
...
```

Ahora vamos a probar, vamos a realizar la integración con agregar una nueva instancia y la vamos a editar.

ABM CLAN

ID CLAN
1
2
3
4
5

Agregar Clan

Nombre Clan:

Ranking Clan:

ABM CLAN

ID CLAN	NOMBRE CLAN	RANKING CLANES
1	Magios	1
2	NoHomero	2
3	Peces del Infierno	3
4	Santos Orantes	4
5	Amigos de los Pinos	5
7	EditarClan	88888

ABM CLAN

ID CLAN
1
2
3
4
5
7

Editar Clan

Nombre Clan:

Ranking Clan:

ABM CLAN

ID CLAN	NOMBRE CLAN	RANKING CLANES
1	Magios	1
2	NoHomero	2
3	Peces del Infierno	3
4	Santos Orantes	4
5	Amigos de los Pinos	5
7	EditarClan2	7777777

AgregarEditarEliminar

Bien ahora funciona todo ok!!!

Ahora debemos hacer lo mismo pero con **ABMPersonaje.jf**from

El código está en el repositorio GITHUB:

<https://github.com/davidroco99/JPAPersonaje-Clanes.git>