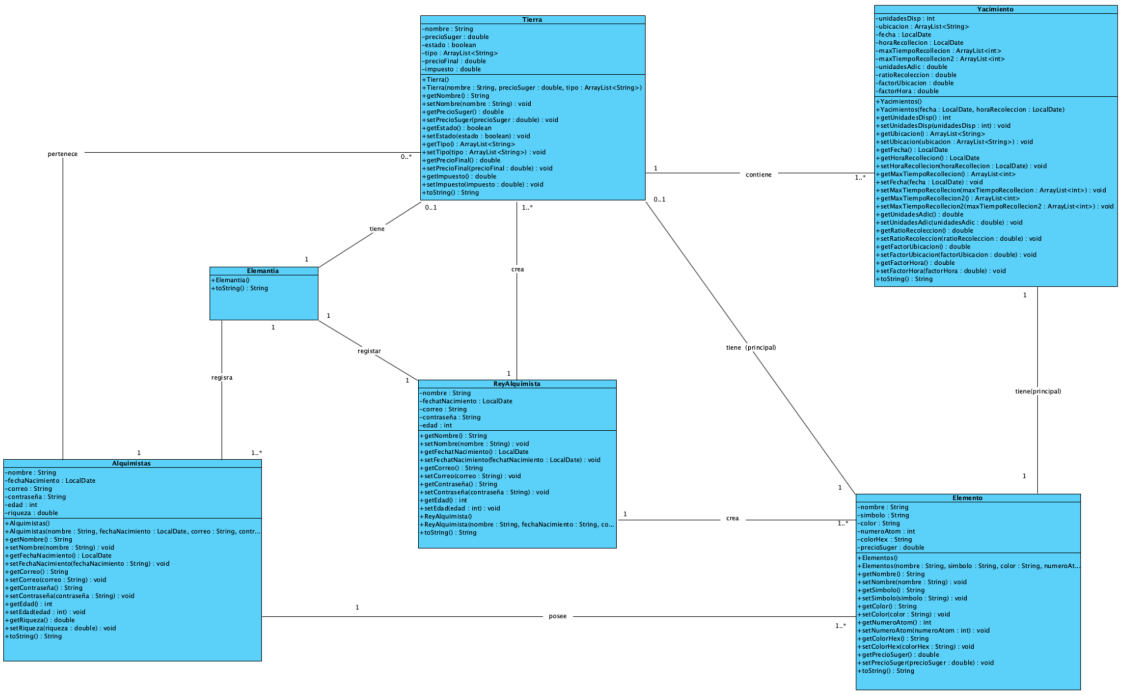
Bitácora
Programación orientada a objetos

Actividad	Fecha
Diagrama de clases siguiendo el estándar de UML	semana 4 - 11-06-23
Aplicando Herencia, Polimorfismo, Java Doc y manejo de excepciones en Java	semana 9 - 13-07-23
Arquitectura y Base de Datos	semana 12 - 15-08-23

Diagrama de clases siguiendo el estándar de UML



Como primera parte del proyecto al diseñar y crear un prototipo del juego debemos de empezar por la el diagramación del software, usando el estándar UML. Es importante que en este paso solo vamos a diagramar las partes fundamentales de la aplicación, es decir no diagramamos las funcionalidades por ejemplo “registros”.
Las siguientes entidades se representan de la siguiente forma:

Elementia:

- En el continente tiene o existen de 1 a muchos alquimistas ya que en un continente existen alquimistas en número indefinido ya previamente registrados.
- En el continente tiene 1 rey Alquimista porque un continente debe de tener un rey alquimista para que cumpla su función de crear características que el contiene también posee.
- El continente tiene muchas tierras de 1.* a muchos porque la idea principal es que el continente tenga varias instancias de tierras con diferentes características.

Tierra:

- Muchas tierras existen dentro de la entidad de continente (elementia) es decir, de 1 a muchas instancias de tierra pueden estar en un continente.
- Muchas tierras pueden ser creadas por un rey alquimista, por lo que muchas instancias de tierra pueden relacionarse con un solo rey para su creación de las mismas.
- Una tierra o instancia puede estar o no relacionadas con un único elemento principal, es decir una instancia de tierra tiene 1 o 0 elementos como principal.
- Una instancia de tierra contiene 1 o muchos yacimientos, por eso la relación entre ambas clases ya que según el tipo de tierra tendremos más de 2 yacimientos.
-

Yacimientos:

- Muchas instancias de yacimientos pueden estar relacionadas a una clase de tierra por eso de 1 a muchos puede asociarse a una tierra
- Una instancia o cada instancia de yacimiento está relacionada a un elemento, a un único elemento para darle una característica de *elemento principal*.

Elemento:

- Un elemento está relacionado a un yacimiento para identificarlo como elemento principal del yacimiento.
- Un elemento está asociado a la clase de de tierra, pero puede que no, por eso de 1 instancia a 0 instancias o 1 única instancia de tierra.
- Muchos elementos pueden ser creados por una instancia o un único rey Alquimista por eso el tipo de multiplicidad. 1 a muchas instancias puede ser creada por una instancia de rey Alquimista.
- Muchas instancias de la clase Elemento puede ser poseídas o tener un “owner” de la clase alquimistas regulares. Por eso la relación de 1 a muchos - 1.

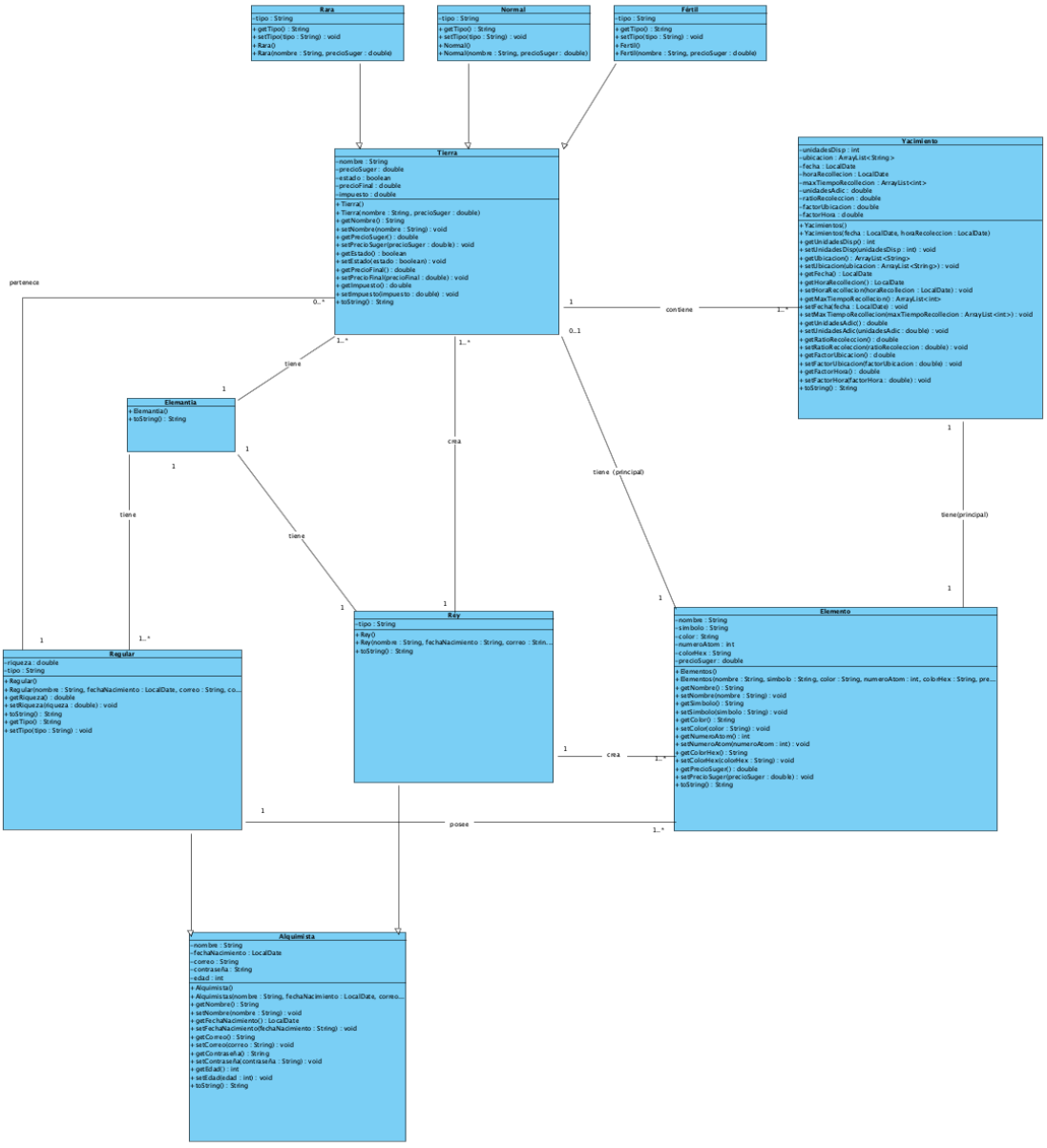
Alquimista

- Un alquimista posee múltiples instancias de la clase Elementos ya que los alquimistas pueden comprar o sustraer elementos de los yacimientos.
- 1 o muchos alquimistas existen en la única instancia de continente ya que en un continente pueden ser registrados N cantidad de alquimistas.
- A un alquimista le pertenecen N cantidad de Tierra ya que los alquimistas pueden comprar tierras ya previamente creadas por el rey alquimista, pero siguiendo algunas reglas mencionadas.

Rey Alquimista

- 1 rey alquimista existe en la única instancia de continente ya que en un continente puede registrar un rey para que cumpla la función de rey en el juego.
- 1 rey puede crear múltiples instancias de la clase Tierra por esa razón de la relación entre la clase rey y la tierra.
- 1 rey puede crear múltiples instancias de la clase Elemento por esa razón de la relación entre la clase rey y el Elemento de 1 a muchos.

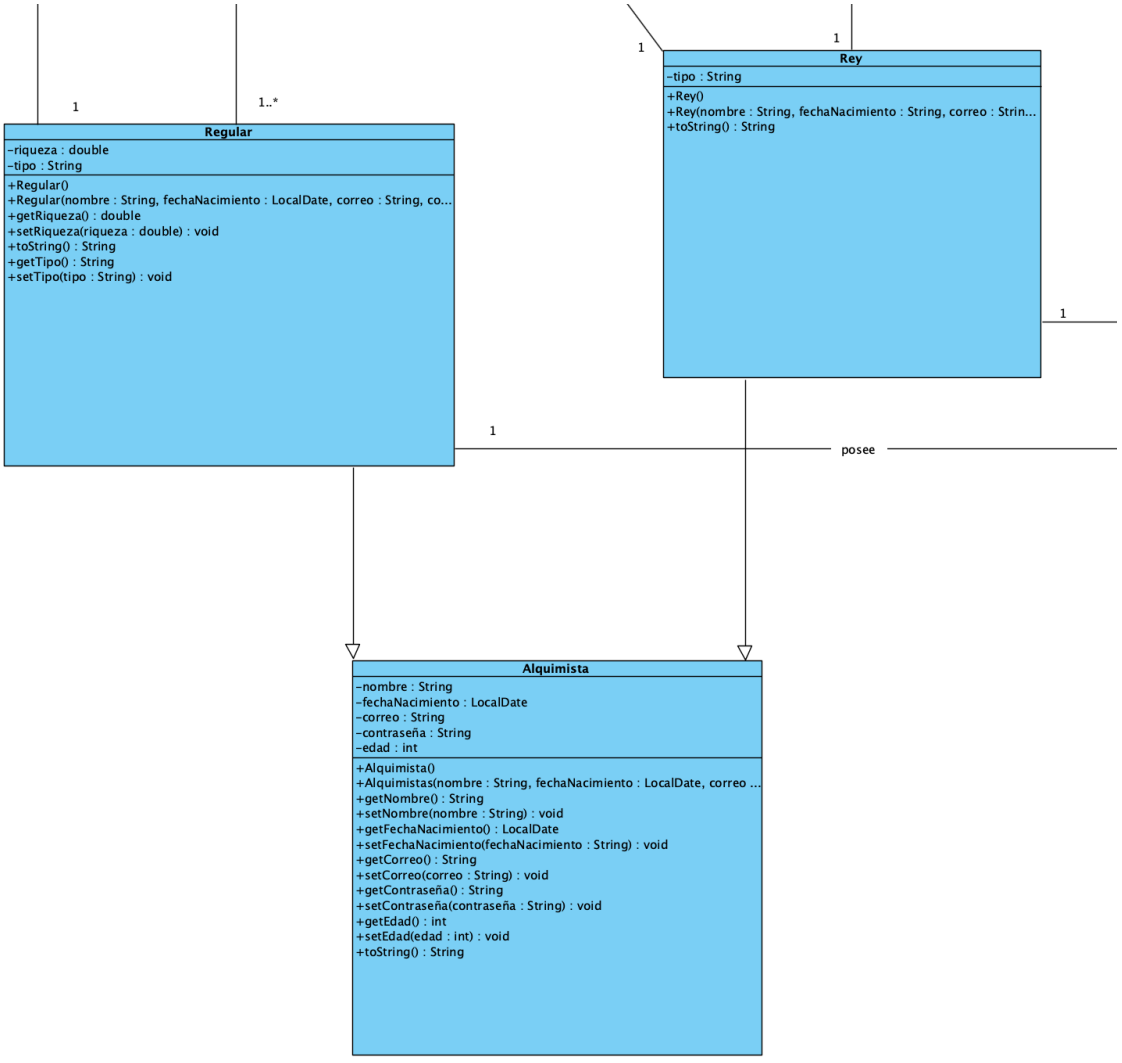
Aplicando herencia y polimorfismo



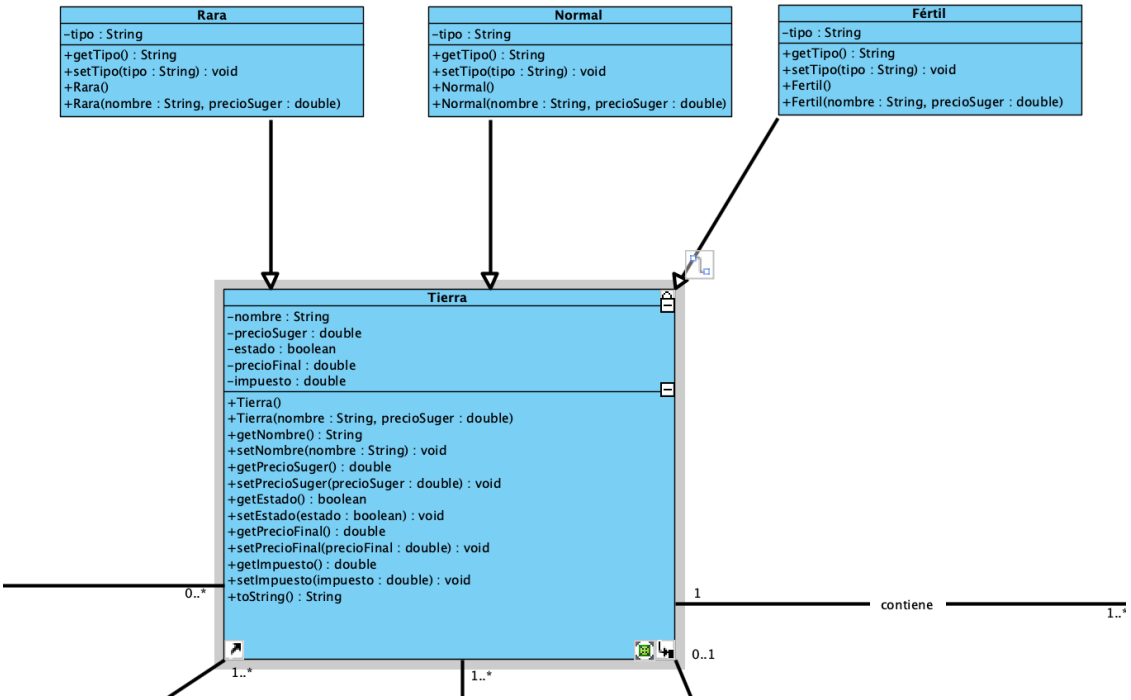
Como segunda parte del proyecto, tenemos que aplicar según lo visto en clase, herencia ya sea objetos o clase, previamente ya habíamos creado algunas jerarquías de tipo relación sencilla entre los personajes de elementia, de momento no he hecho ninguna composición o agregación entre objetos porque no lo veo necesario según mi abstracción, pero si aplique jerarquía en algunas clases, es decir herencia que se mencionara en los siguientes puntos.

Herencia

1. La primera herencia que se que se aplicó al proyecto, fue la de alquimistas ya que hay 2 tipos de alquimistas, un rey y aldeanos o comunes, por lo que se creo 2 clases Regular y Rey que heredan de Alquimista, para poder más adelante especificarlas con sus propios atributos o método que los diferencian entre ambos, por ejemplo un rey puede hacer acciones en el juego que un alquimista normal o común no, entonces estos son las características que se agregan en sus respectivas clases, pero ambas comparten cosas en común que las heredan del papa (clase Alquimista).

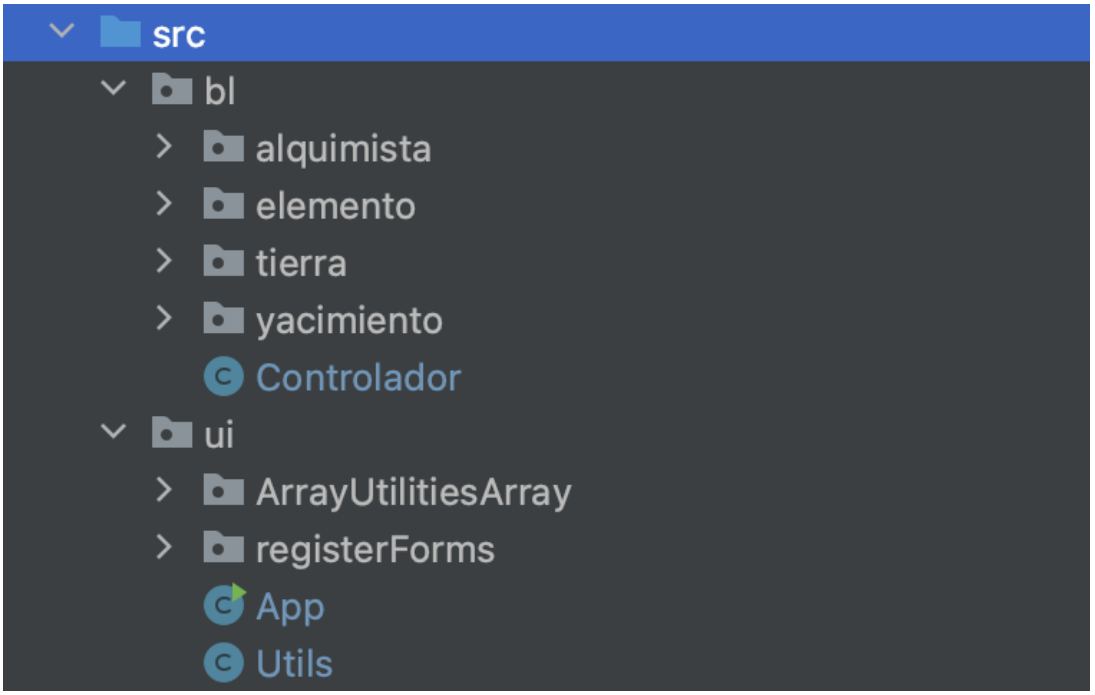


2. La segunda herencia que se aplicó al proyecto, fue para las tierras ya que cada una de los diferentes tipos de tierra tienen características similares pero cada una de ellas tienen otras que son específicas para cada una de ellas por lo que se necesita diferenciar responsabilidades.



Arquitectura

También como parte de este entregables tuvimos que aplicar la arquitectura del proyecto vista en clase que se parte de 2 paquetes, una llamada UI y otra BL
Resumidamente la UI se encarga de manejar toda funcionalidad de la aplicación que sea visual, por ejemplo el pintar e imprimir cosas en la pantalla y el BL su proposition es el que se encarga de toda la parte de lógica del negocio, es el que maneja toda la parte de registros y coordina que entra y que debería de salir, es decir: si el usuario necesita registrar un objeto el UI tiene que notificarle o hacerle saber al BL de dicho registro o petición del usuario y el controlador manipulara esos datos conforme sea la necesidad del negocio, el coordina y sabe que hacer con esos datos.


src
bl
alquimista
elemento
tierra
yacimiento
Controlador
ui
ArrayUtilitiesArray
registerForms
App
Utils

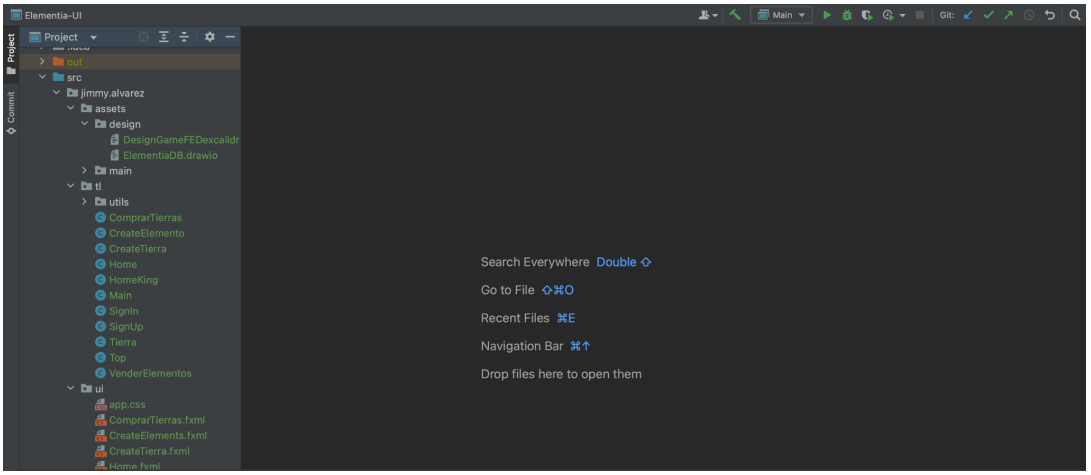
Arquitectura y Base de Datos

Ok, en esta semana nos centramos en finalizar los requerimientos del proyecto, esto conlleva a la implementación de los yacimientos y las otras funcionalidades, ya no solo la parte del registro. Entonces esto involucra muchos cambios de planes en la parte del manejo de muchas herencias que se estaban realizando en su momento para poder manipular más fácilmente las entidades con la aplicación.

Para en este entregable también involucrò una reestructuración del proyecto para poder seguir la arquitectura de proyecto que se sigue en el curso,
Desde separar el BL con el UI y hacer proyectos aparte para su escalabilidad y prevención de riesgos futuros, como una migración de Frontend o backend.

Para el UI tuvimos que usar JavaFX para la construcción de aplicaciones de escritorio y esculpir en una interfaz clara la lógica del proyecto.

la estructura fue tener una carpeta UI y Un TL para los controllers.

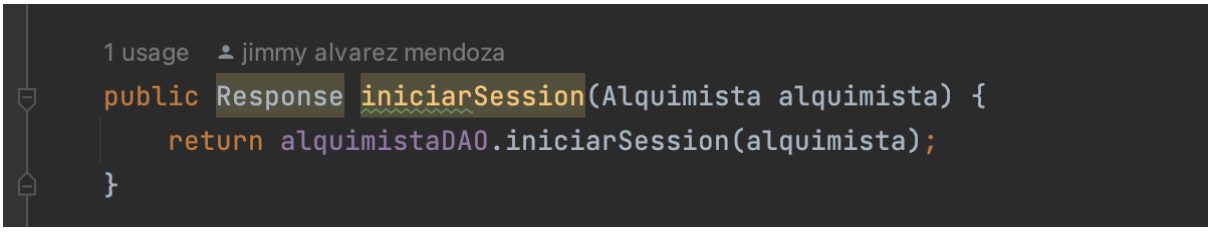


Para el BL o Business Logic se tuvo que seguir la siguiente estructura:

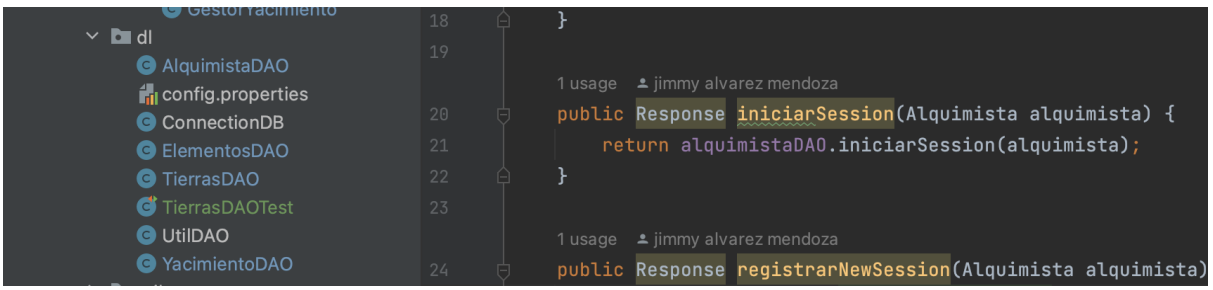
Crear un PKG con el nombre entidades para agregar todas las clases de los personajes, como si fuese un SKELETON y usarás en toda la aplicación para el recibiendo y devolución de respuestas tanto para el UI como en la parte del procesamiento de la parte del backend.

También se tuvo que crear un Gestor para cada entidad, y así crear otro pkg dentro del paquete BL para la creación de los mismos.

El único propósito del Gestor es comunicar el UI o la vista con la parte de acceso a dato, es un comunicador entre las 2 (un puente)



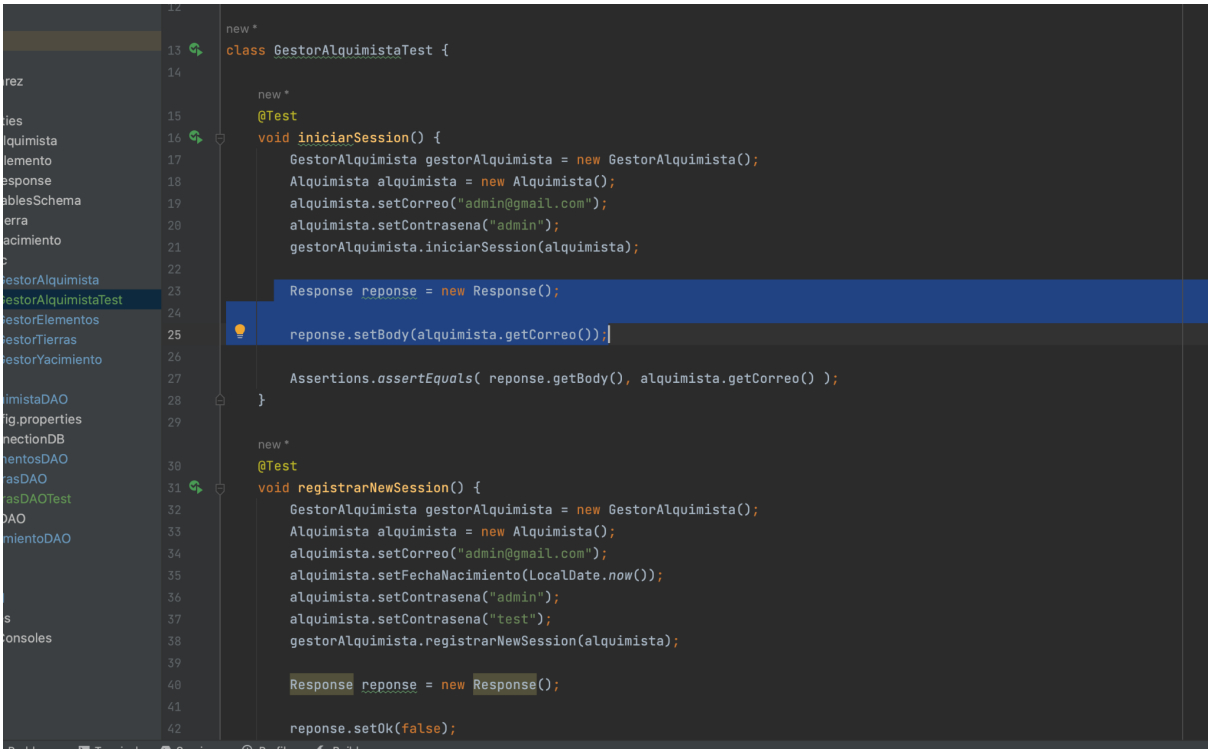
Y como última parte teníamos que crear el DL donde iban a estar cada de las DAO de las entidades. Para esto igual se creó un pkg llamada DL y dentro de ella se crearon las clases con la terminación DAO, ***data access object.***
El fin de estas clases era la comunicación y procesamiento de los objetos que recibimos desde el Vista controlador a la bases de datos, este mismo es el encargado de notificar y hacerle saber si hubo un error al procesar los datos.



Ahora para la parte de la conexión de igual manera se maneja en el DL, y su configuración se encuentra en el archivo .properties, con el fin de poder especificar que JDBC o driver queremos utilizar, en mi caso tuve que hacer la conexión con SQL server porque es de mi preferencia.

Para la conexión tuve que conectarme al server de una pc para poder trabajar el código en mac y la bases de datos en pc por comodidad. Tuve que hacer un nuevo usuario con todos los permisos y conectarse a la IP de mi computadora.

Y como parte final, se integraron varias pruebas unitarias con ayuda de una librería llamada UNIT 5. Esto con el fin de verificar que cada parte de mi programa está dando el resultado esperado.



```
new *
class GestorAlquimistaTest {

    new *
    @Test
    void iniciarSession() {
        GestorAlquimista gestorAlquimista = new GestorAlquimista();
        Alquimista alquimista = new Alquimista();
        alquimista.setCorreo("admin@gmail.com");
        alquimista.setContrasena("admin");
        gestorAlquimista.iniciarSession(alquimista);

        Response reponse = new Response();

        reponse.setBody(alquimista.getCorreo());

        Assertions.assertEquals( reponse.getBody(), alquimista.getCorreo() );
    }

    new *
    @Test
    void registrarNewSession() {
        GestorAlquimista gestorAlquimista = new GestorAlquimista();
        Alquimista alquimista = new Alquimista();
        alquimista.setCorreo("admin@gmail.com");
        alquimista.setFechaNacimiento(LocalDate.now());
        alquimista.setContrasena("admin");
        alquimista.setContrasena("test");
        gestorAlquimista.registrarNewSession(alquimista);

        Response reponse = new Response();

        reponse.setOk(false);
```