FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Java

[7507/9502] Algoritmos y Programación III
Curso 1
Primer cuatrimestre de 2024

Grupo 05	
Alumno:	Número de padrón:
Alperovich, Denisse	110266
More, Danica	109502
Cruceño, Juan Ignacio	110286
Apaza, Axel	105006

Índice

1. Introducción

El presente informe reune la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación de un juego de preguntas y respuestas en java utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

En la especificación no estaban contempladas la modalidad específica del juego mas allá de ser de preguntas y respuestas por lo que decidimos que el juego tenga partidas y que cada una de estas contengan n rondas, especificadas por los jugadores, cada ronda corresponde a una pregunta. Además decidimos que el juego pueda ser jugado por N jugadores. También decidimos que no se pueden utilizar varios poderes a la vez, en un jugador y, por último, el juego denominará al ganador como el jugador con mayor puntaje a la hora de finalizar las preguntas, si en la selección de puntaje para ganar se encuentra un número mayor al máximo obtenido al finalizar la cantidad de preguntas elegidas.

3. Modelo de dominio

Para la realización del trabajo se dividió el dominio en tres grandes paquetes: modelo, vista y controlador.

Dentro del modelo tenemos la clase abstracta Pregunta que recopila los básicos comunes de una pregunta. Esta es heredada por PreguntaVF, PreguntaMC, PreguntaOC, PreguntaGC, PreguntaMCParcial, PreguntaMCPenalidad y PreguntaVFPenalidad, estas últimas tres reescriben el método calcularPuntaje para poder agregarle la penalidad o parcialidad al mismo.

Las preguntas a su vez tienen Opciones y Respuesta. La primera se comporta como un contenedor de Opcion para poder así manejar las posibles respuestas a la pregunta. Por otro lado, la Respuesta, a su vez es una clase abstracta heredada por RespuestaMC, RespuestaVF, RespuestaOC y RespuestaGC, estas heredan la firma del método que comprueba si dichas respuestas son iguales pero lo implementan cada una a su manera. Respuesta GC, por su lado, es compuesta por Grupos los cuales también tienen la capacidad de determinar si son iguales.

La clase Jugadores funciona de la misma manera que Opciones pero esta contiene Jugador, el jugador cuenta con Puntaje, uno opr ronday uno total, respuesta y Poderes la cual recopila a Poder. Poder es una clase abstracta heredada por los cuatro poderes del juego y un poder básico. A su vez, tanto básico como duplicador y triplicador implementan la interfaz PoderIndividual los cuales al aplicarse solo afectan a un Puntaje mientras que Anulador y Exclusividad implementan PoderGrupal ya que estas trabajan con Puntajes siendo estos los de cada jugador del juego.

La clase Pregunta y sus hijas son creadas por la clase PreguntaFactory, esta trabaja con la clase Parser, la cual se encarga de leer el archivo Json y trabaja con PreguntaFactory, la cual decide a partir de la lectura que pregunta se crea cuando y la crea.

Por último, creamo la clase Alerta para manejar las excepciones de una forma amigable con el usuario.

Por el lado de los Controladores y las Vistas tenemos la interfaz de ControladorPregunta, la cual es implementada por ControladorVF, ControladorMC, ControladorOC y ControladorGC. Estos controladores trabajan con parte del modelo como lo va a ser Jugador y pregunta y a su vez con su respectiva Vista(VistaVF, VistaMC, VistaOC y VistaGC), las cuales heredan todas de la clase abstracta VistaPregunta y PoderesVista. PoderesVista se encarga de mostrar y manejar los poderes de cada jugador a la hora de jugar el juego. Por último, así como los utilizamos con Pregunta, existe un ControladorFactory, el cual a partir del tema de la Pregunta decide que tipo de Controlador crear y lo fabrica.

4. Diagramas de clase

A continuación, se puede ver la interacción entre las clases a través de 2 diagramas de clase.

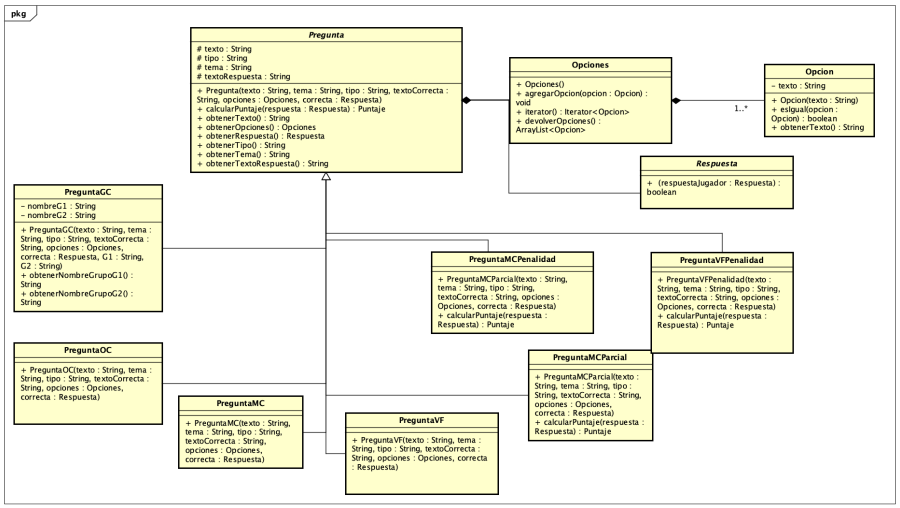

Figura 1: Relación de Preguntas.

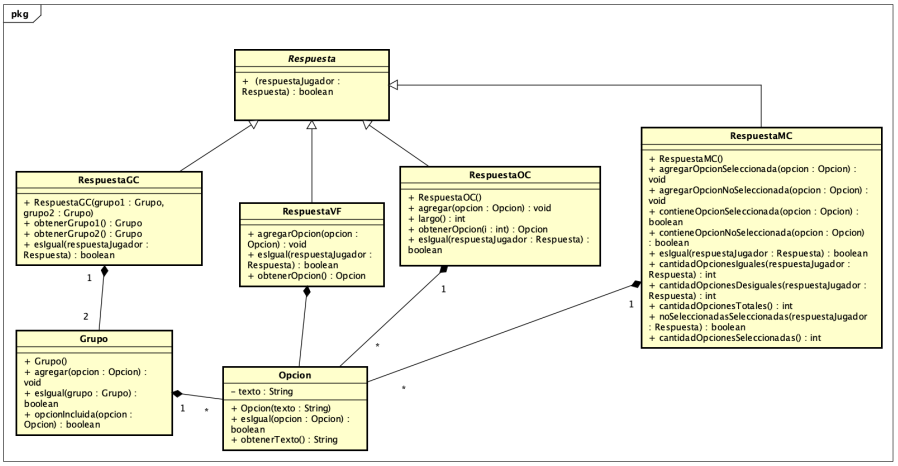

Figura 2: Relación de Respuestas.

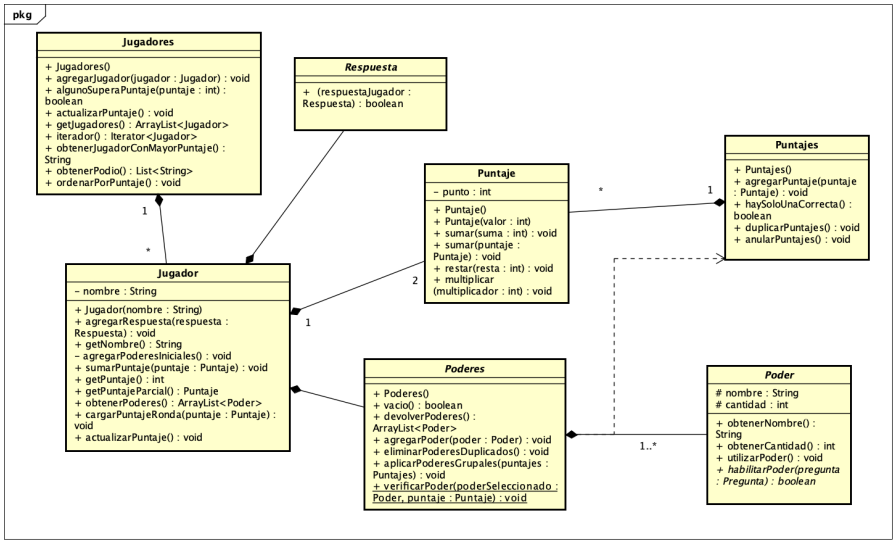

Figura 3: Relación de Jugadores con sus atributos.

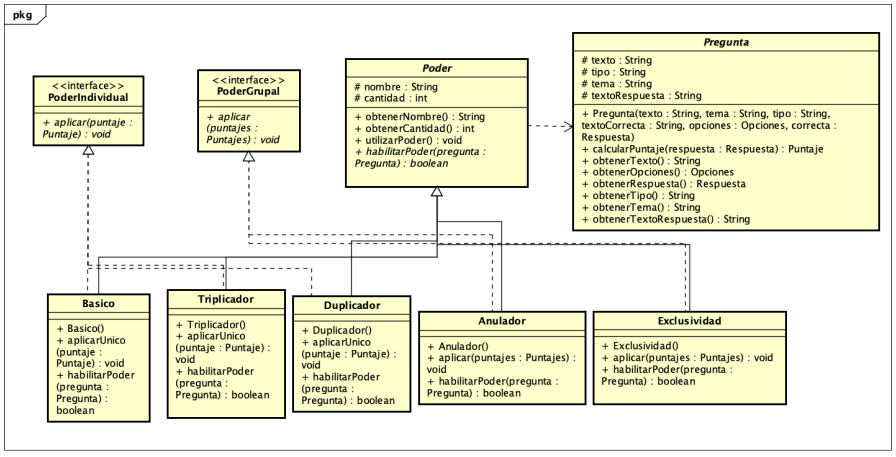

Figura 4: Relación de Poderes.

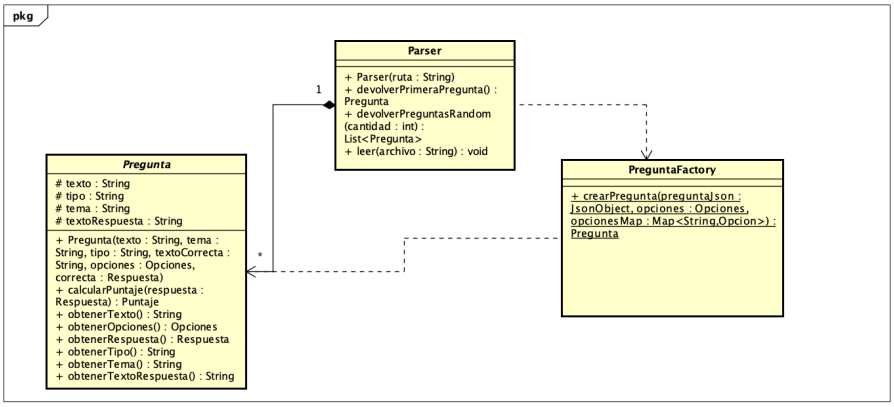

Figura 5: Relación Parser y Factory con Pregunta.

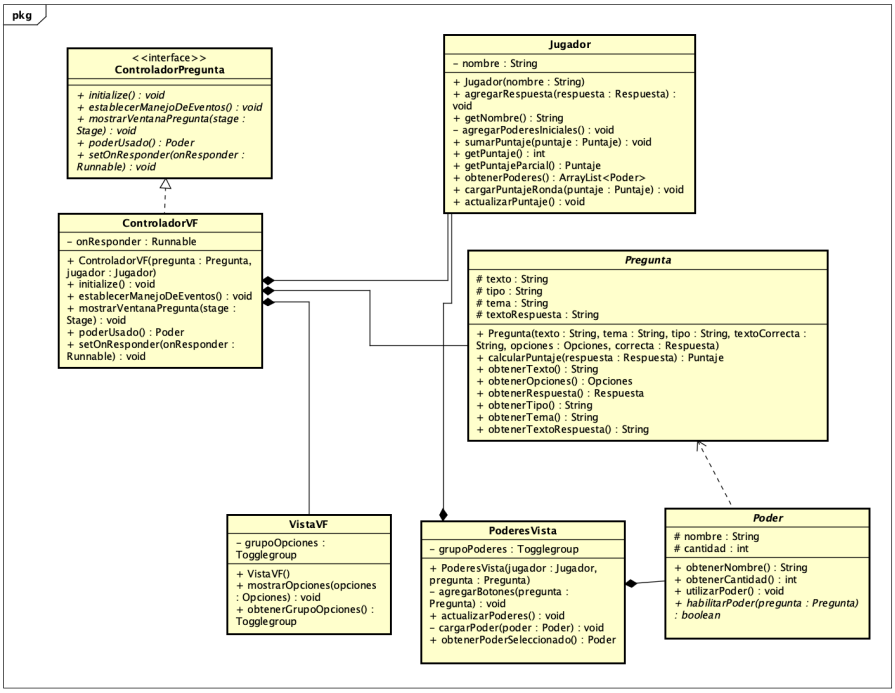

Figura 6: Se muestra ControladorVF.

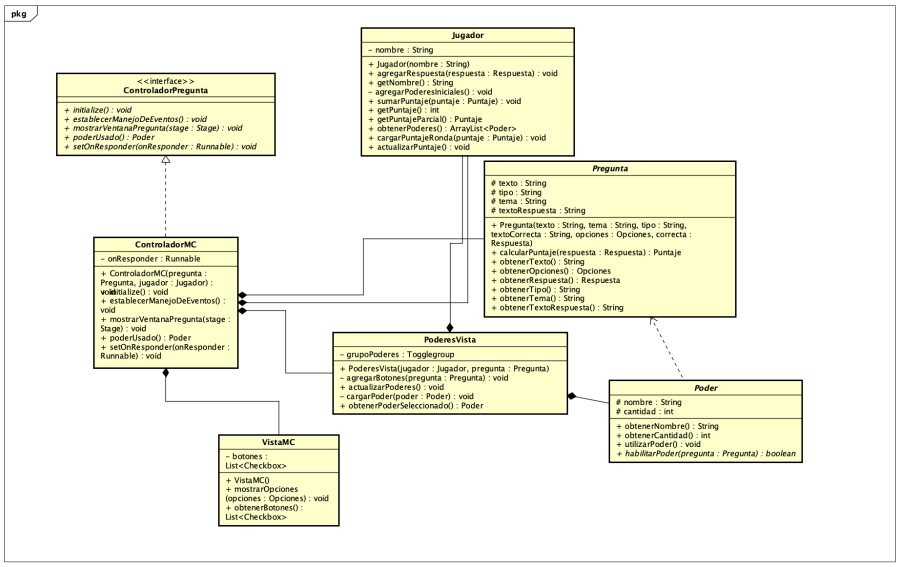

Figura 7: Se muestra ControladorMC.

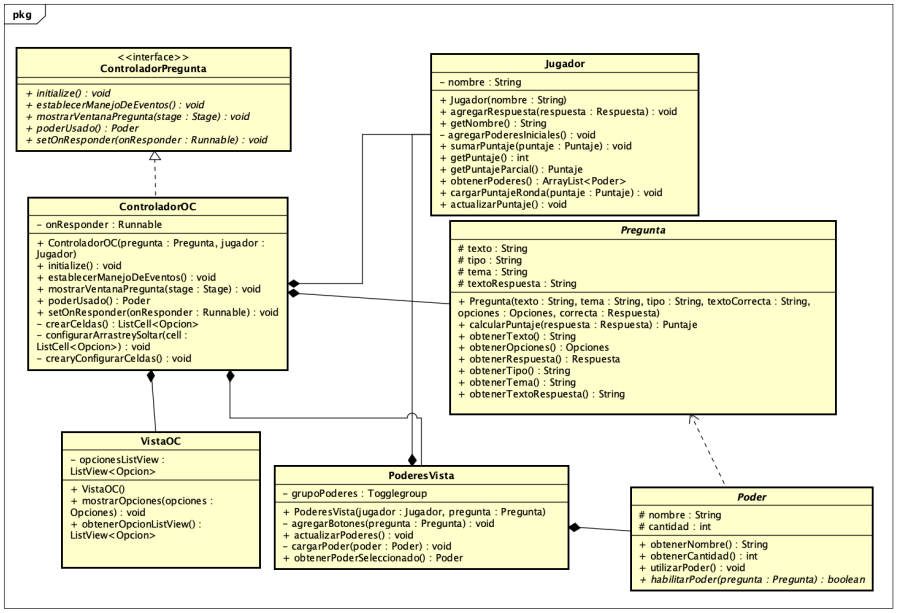

Figura 8: Se muestra ControladorOC.

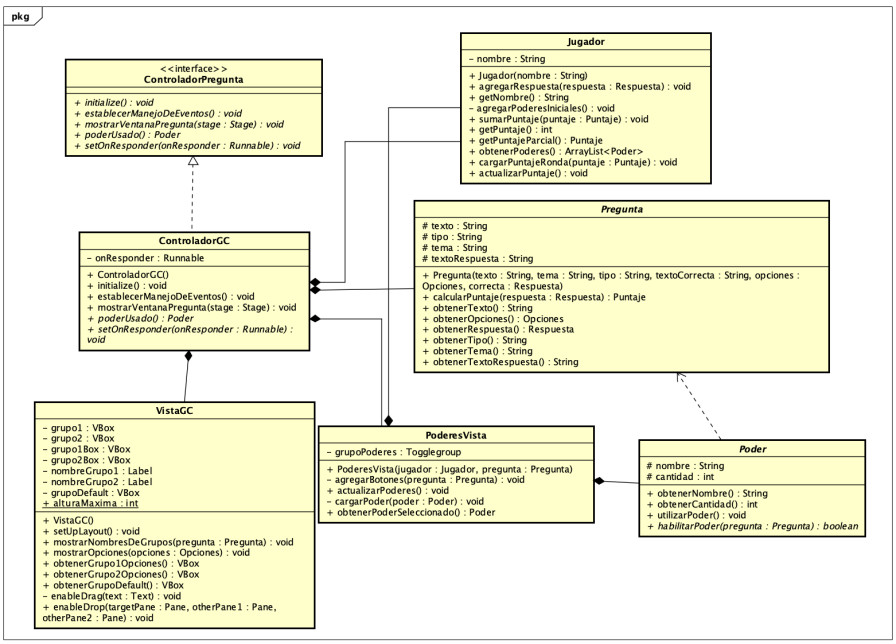

Figura 9: Se muestra ControladorGC.

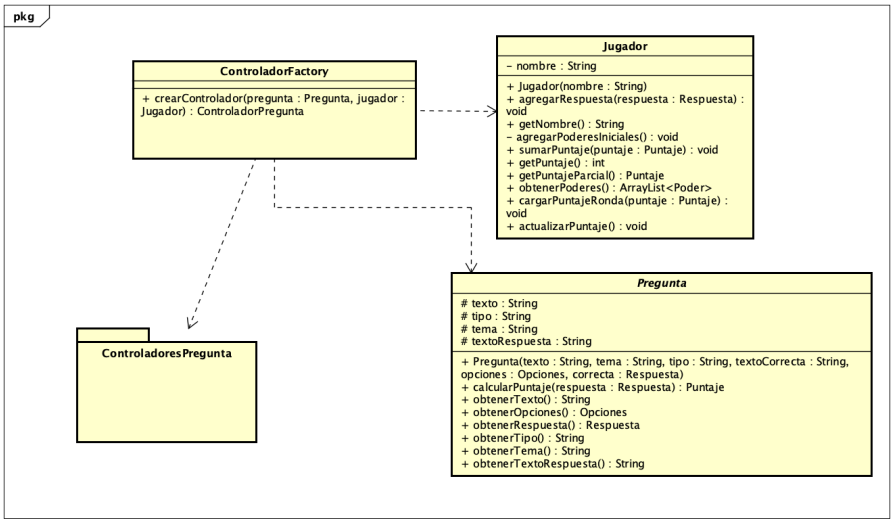

Figura 10: Funcionalidad de ControladorFactory.

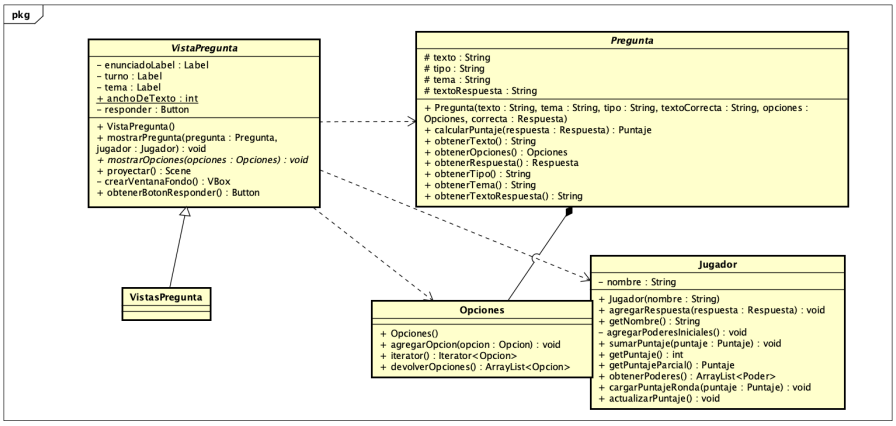

Figura 11: Relación de las Vistas de Pregunta.

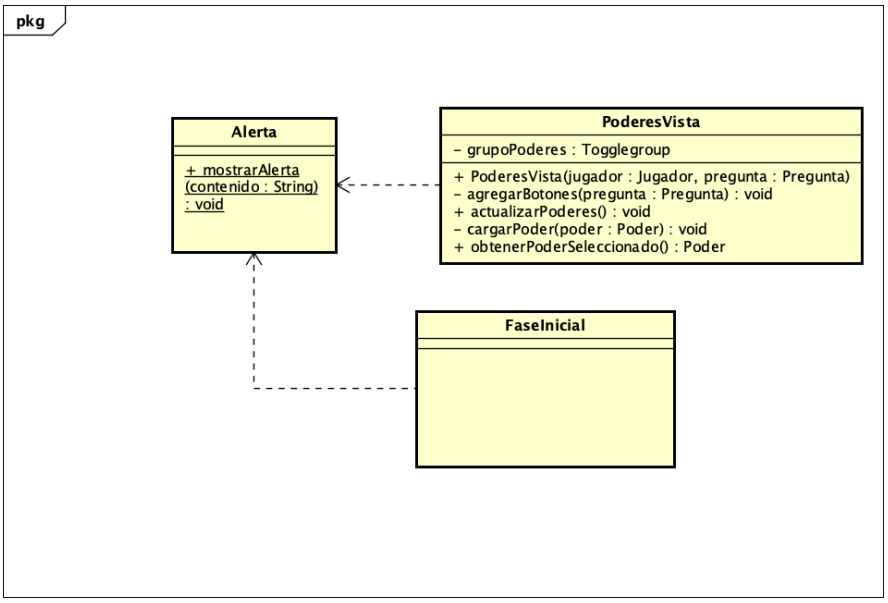

Figura 12: Relación de las Alertas con la Vista.

5. Detalles de implementación

5.1. ¿Herencia o delegación?

Para los objetos como las preguntas, respuestas y poderes decidimos utilizar herencia. Esto se debe a que como cada pregunta y por ende su respuesta tienen comportamientos distintos, consideramos que deberían ser distintos objetos, asi mismo, los poderes heredan de poder y dependiendo de si es un poder individual o grupal se comportan distinto, por eso consideramos que deberían ser objetos diferentes.

De igual manera, para crear los objetos como las preguntas decidimos delegar la responsabilidad en un builder para poder crear los distintos tipos de preguntas utilizando Pregunta.

5.2. Principios de diseño

Principios SOLID:

Single Responsibility Principle: Utilizamos muy claramente este principio en la clase Parser ya que esta se encarga de leer un JSON y luego delega las responsabilidades creacionales a otro objeto.

OpenClose Principle: Usamos este principio para diseñar el comportamiento de Puntaje frente a los distintos poderes. Ya que cada modificacion de poder está agregando nueva funcionalidad a la modificacion de Puntaje

Principio de segregacion de interfaces: Utilizamos este principio cuando hacemos la distincion entre el comportamiento de un poder grupal con el poder individual.

5.3. Patrón de diseño

Utilizamos el patrón Factory para resolver el problema creacional de los diferentes tipos de Pregunta en la clase Parser, ya que esto nos permitia tener menor acoplamiento entre clases y mantener el Single Responsibility Principle de Parser.

Utilizamos el patrón Factory para resolver el problema creacional de los controladores de ControladorPregunta durante la fase Intermedia del juego de esta manera dividimos la responsabilidad creacional de la responsabilidad de la FaseIntermedia de mostrar preguntas.

Utilizamos el patrón State para resolver el manejo de las fases que tiene FaseManejador, de esta manera mantenemos OpenClosed Principle vemos que esto disminuye el acoplamiento y aumenta la cohesion.

Utilizamos el patrón Null porque nos evitó tener valores nulos para los objetos que implementen Poder.

6. Excepciones

Para el manejo de excepciones decidimos utilizar alertas ya que consideramos que para un mejor manejo de interfaz de usuario, estas serían mas .ªmigablesz no cortarían la experiencia del usuario cada vez que hubiese un problema.

Alerta PoderAgotado: Esta Alerta es chequeada por PoderesVista cuando el jugador intenta usar un poder que ya agotó sus usos.

Alerta CantDeJugadores1: Esta Alerta la maneja FaseInicial y es mostrada cuando en vez de un entero se ingresa un String en el campo de cantidad de Jugadores.

Alerta CantDeJugadores2: Esta Alerta también es mostrada por FaseInicial cuando la cantidad de jugadores ingresada es menor o igual a cero ya que no permitiría empezar el juego.

Alerta CantdePreguntas1: Esta Alerta, manejada por FaseInicial, es mostrada cuando la cantidad de preguntas ingresadas es menor o igual a cero ya que al igual que con los jugadores, el juego no podría empezar.

Alerta CantDePreguntas2: Al igual que CantDePreguntas1, es manejada por la misma clase, solo que esta chequea que la cantidad de preguntas ingresada no supere a la cantidad de preguntas disponibles en el Json.

Alerta CantDePuntos: Por último, esta Alerta se muestra cuando el Puntaje necesario para ganar es menor o igual a cero ya que todos ganarían apenas empezado el juego, lo que no le dejaría jugabilidad.

7. Diagramas de secuencia

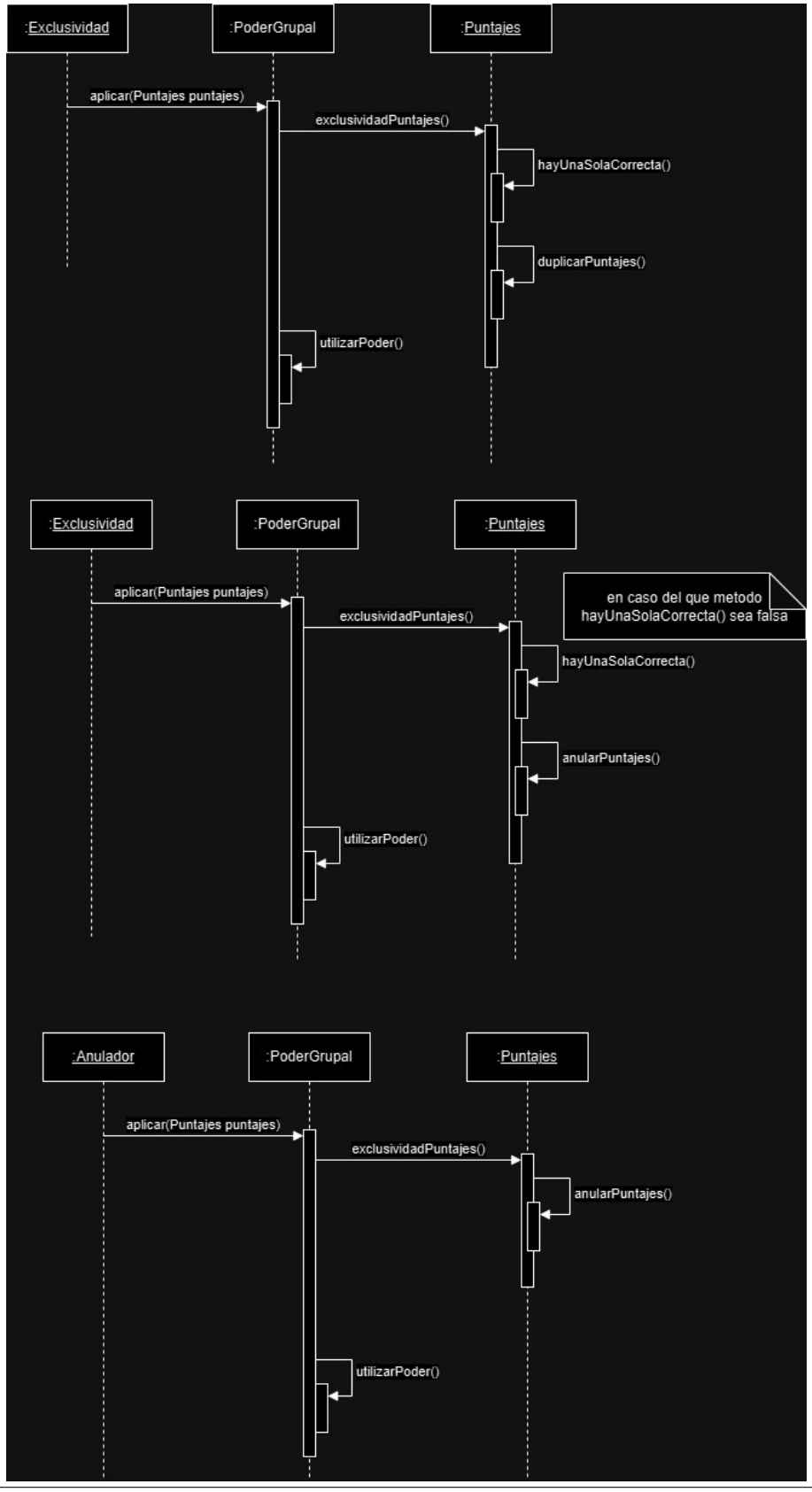

Figura 13: Diagrama de secuencia utilización de poderes.

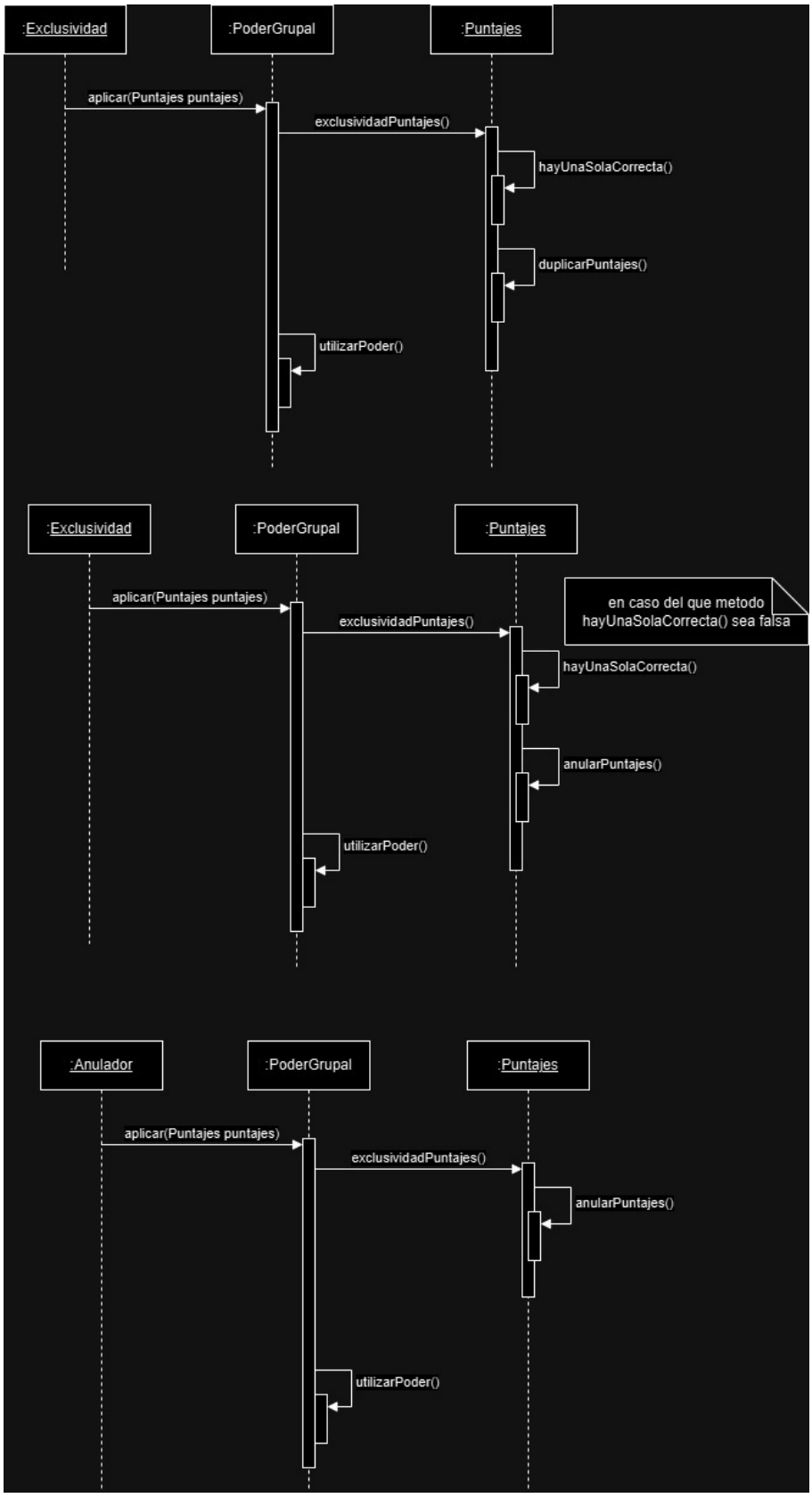

Figura 14: Otro caso de utilización de Poderes.

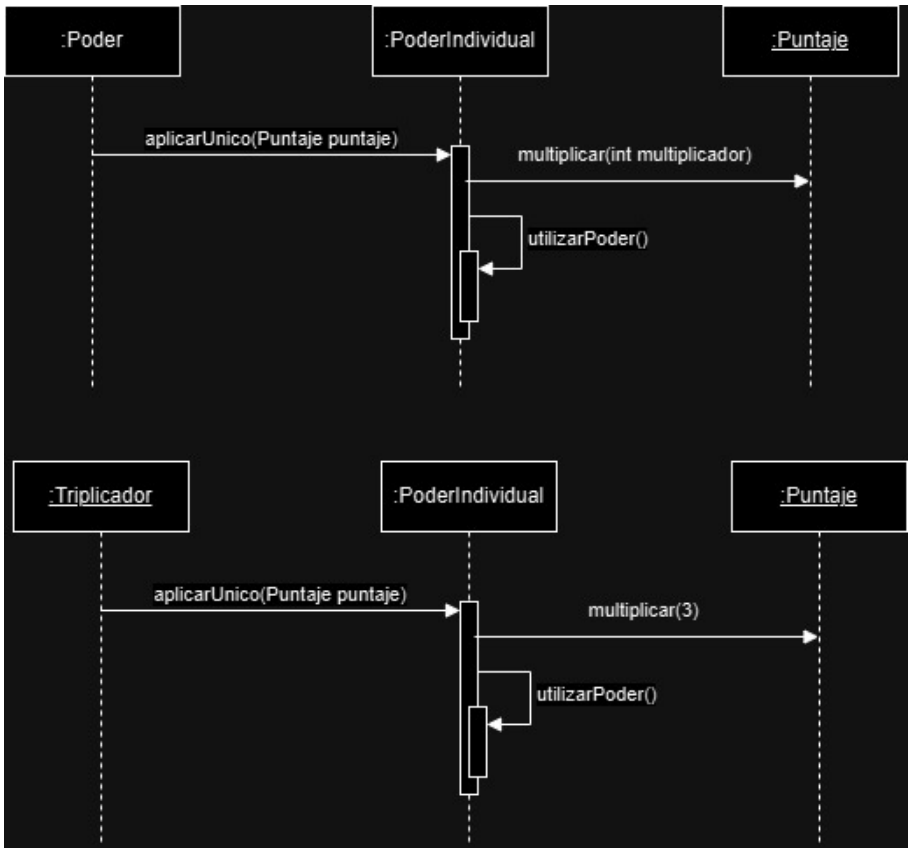

Figura 15: Aplicación de los poderes individuales.