FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — AlgoHoot

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

Alumnos:	SANTELLAN, Felipe. CENICEROS, Valentino. ORGEIRA, Ignacio. ALDONATE, Lucas. FUSCHETTO, Iván.
Números de padrón:	109022 111054 110335 100030 110632
Emails:	fsantellan@fi.uba.ar vceniceros@fi.uba.ar iorgeira@fi.uba.ar laldonate@fi.uba.ar ifuschetto@fi.uba.ar

Índice

1. Introducción

El presente informe reúne la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III, que consiste en el modelado e implementación de un juego de preguntas y respuestas. Dicho desarrollo se realizó usando el lenguaje Java, y aplicando los conceptos del paradigma de la orientación a objetos aprendidos a lo largo del curso.

2. Supuestos

Para el desarrollo fueron adoptados los siguientes supuestos:
(1) Los jugadores deben tener distintos nombres.
(2) Si en el contexto de un turno al menos un jugador utiliza exclusividad, y otro utiliza anulador, el que usó anulador se lleva los puntos multiplicados. Esta estrategia es válida ya que consideramos una respuesta correcta como una respuesta que recibe puntos.
(3) Si más de un jugador utiliza anulador de puntaje en un mismo turno, la ronda finalizará con 0 puntos para todos.
(4) Cuando finaliza el juego se muestra por pantalla una lista con los nombres jugadores ordenados decrecientemente según sus puntajes (se muestran también los puntajes).
(5) El orden en que responden los jugadores va rotando en cada ronda.

3. Diagramas de clase

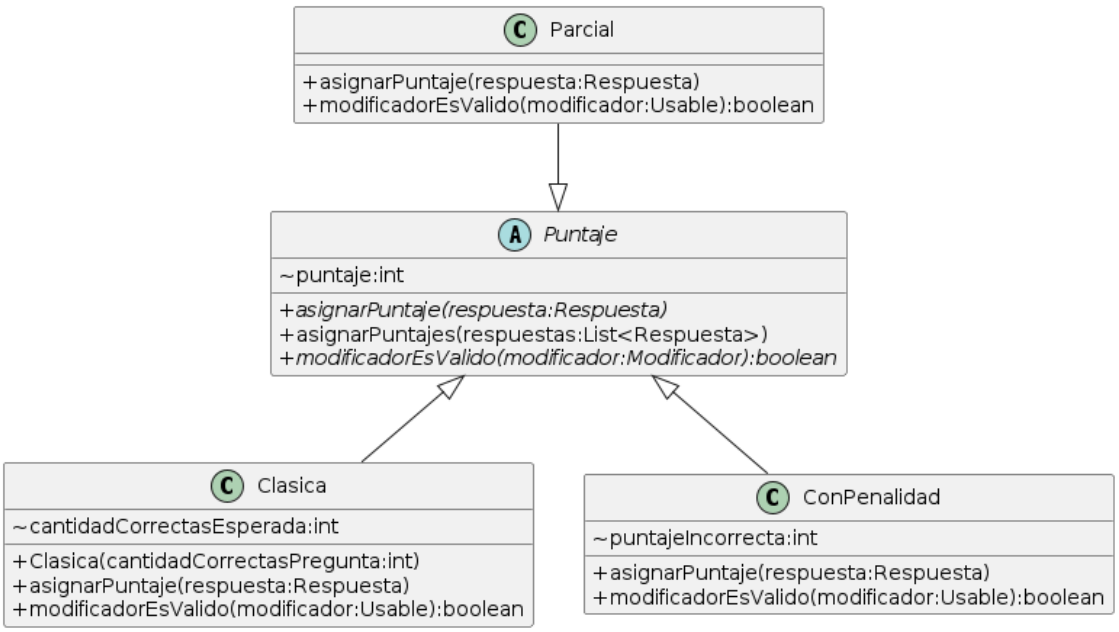

Figura 1: Las clases concretas que heredan de la abstracta Puntaje.

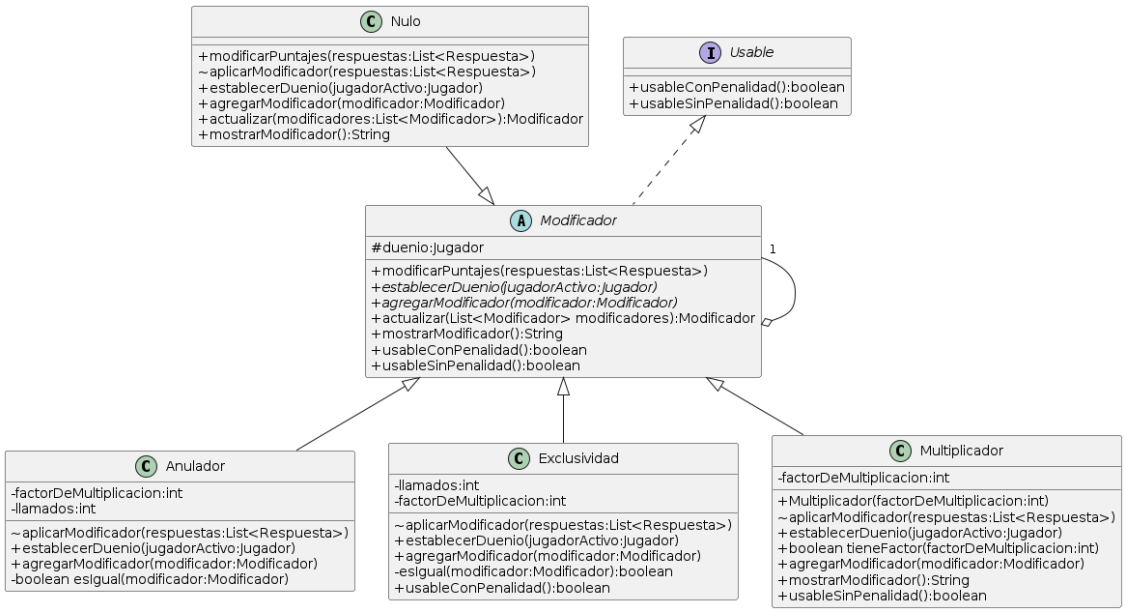

Figura 2: Las distintas subclases concretas de Modificador. Nótese la relación de agregación de la clase abstracta consigo misma.

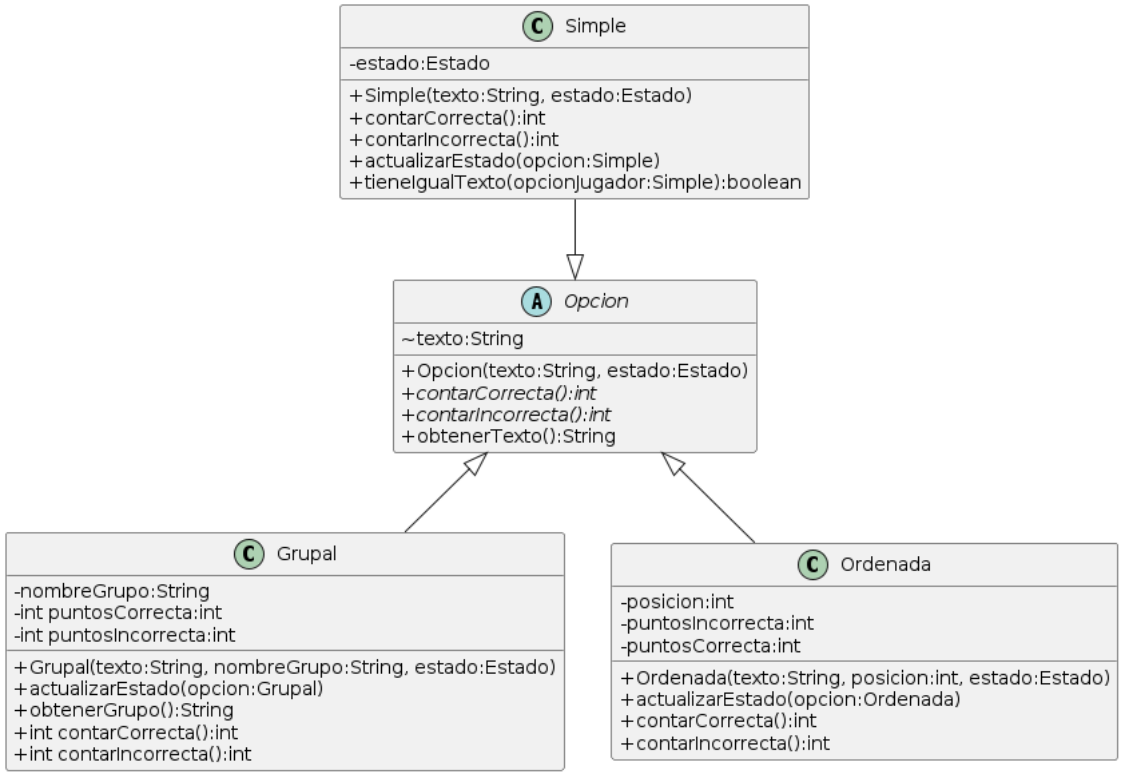

Figura 3: Las diferentes opciones.

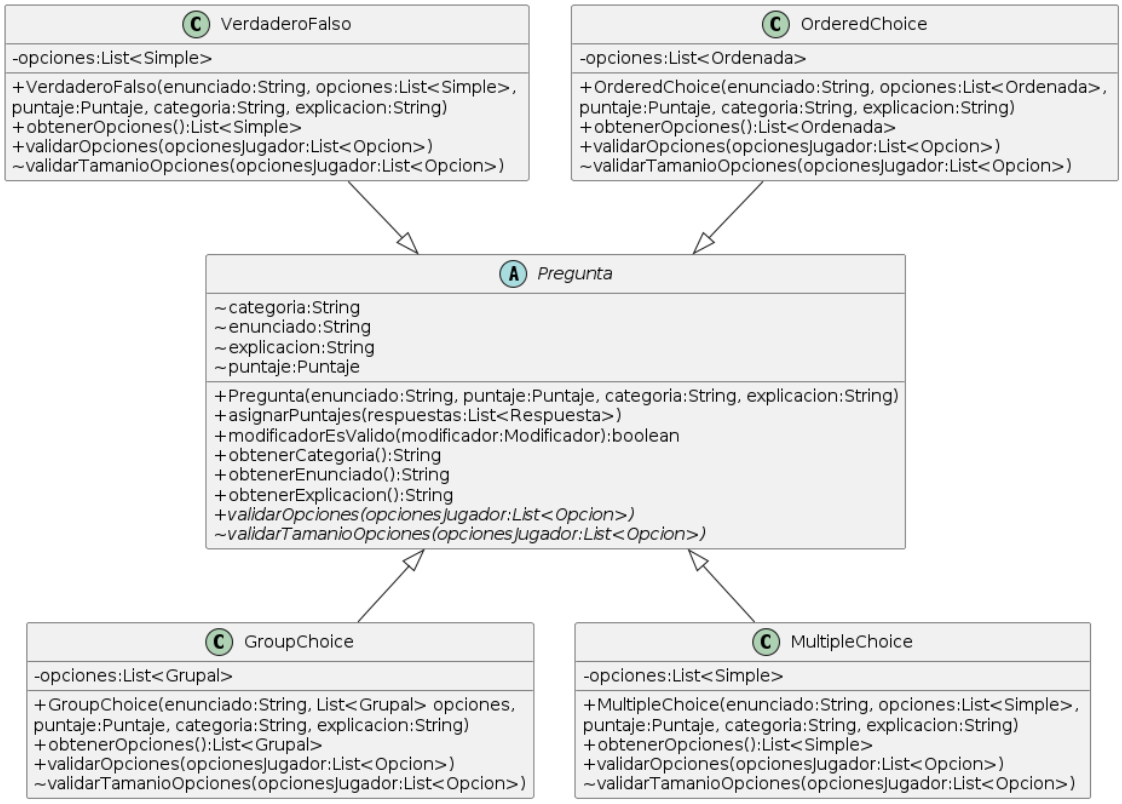

Figura 4: Las distintas hijas de Pregunta. Nótese en los atributos de cada una su relación con las opciones.

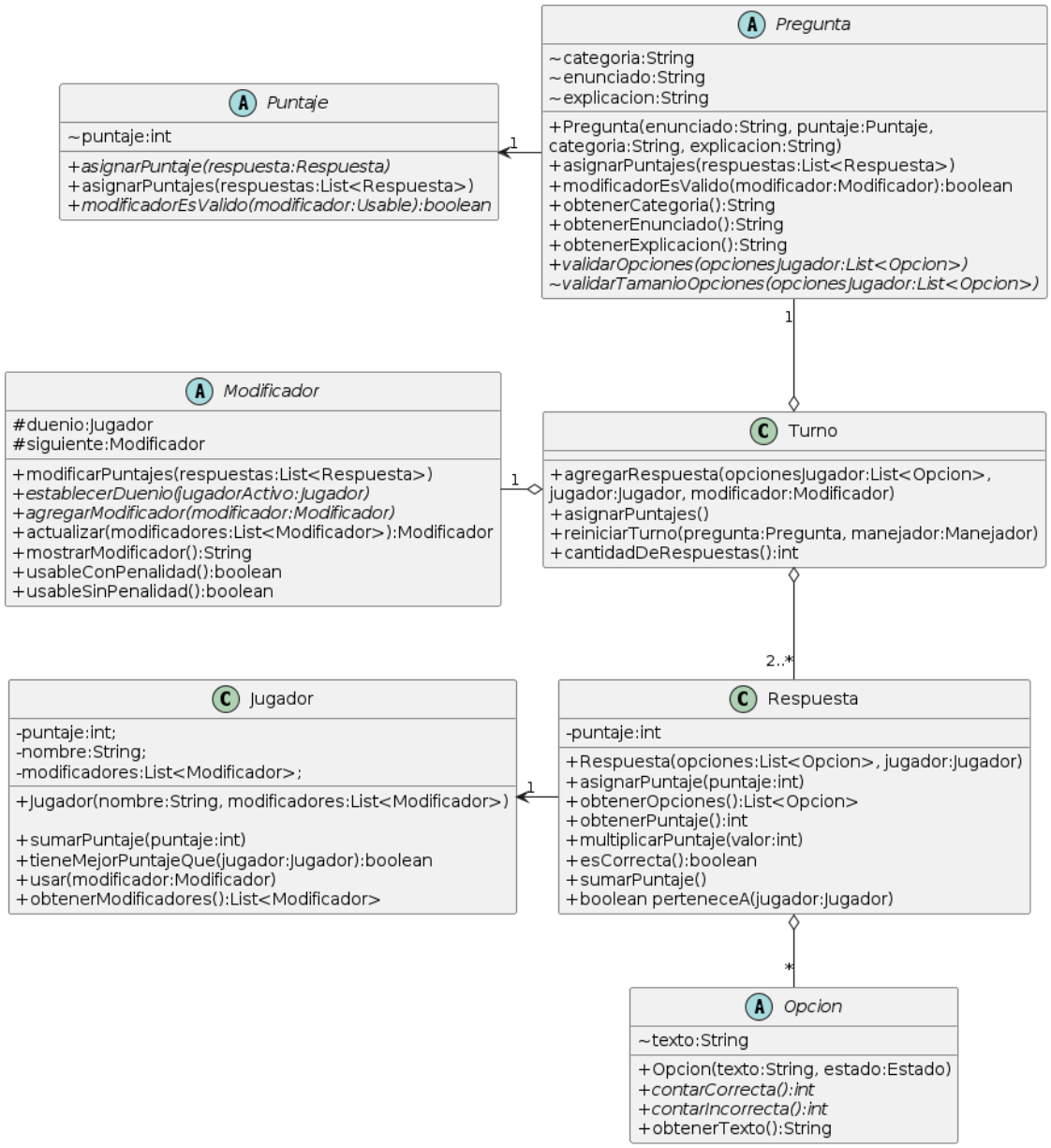

Figura 5: Un diagrama más general. Quedan de manifiesto algunas de las relaciones más importantes entre las con las clases que veníamos viendo y se observan las relaciones de la clase Turno.

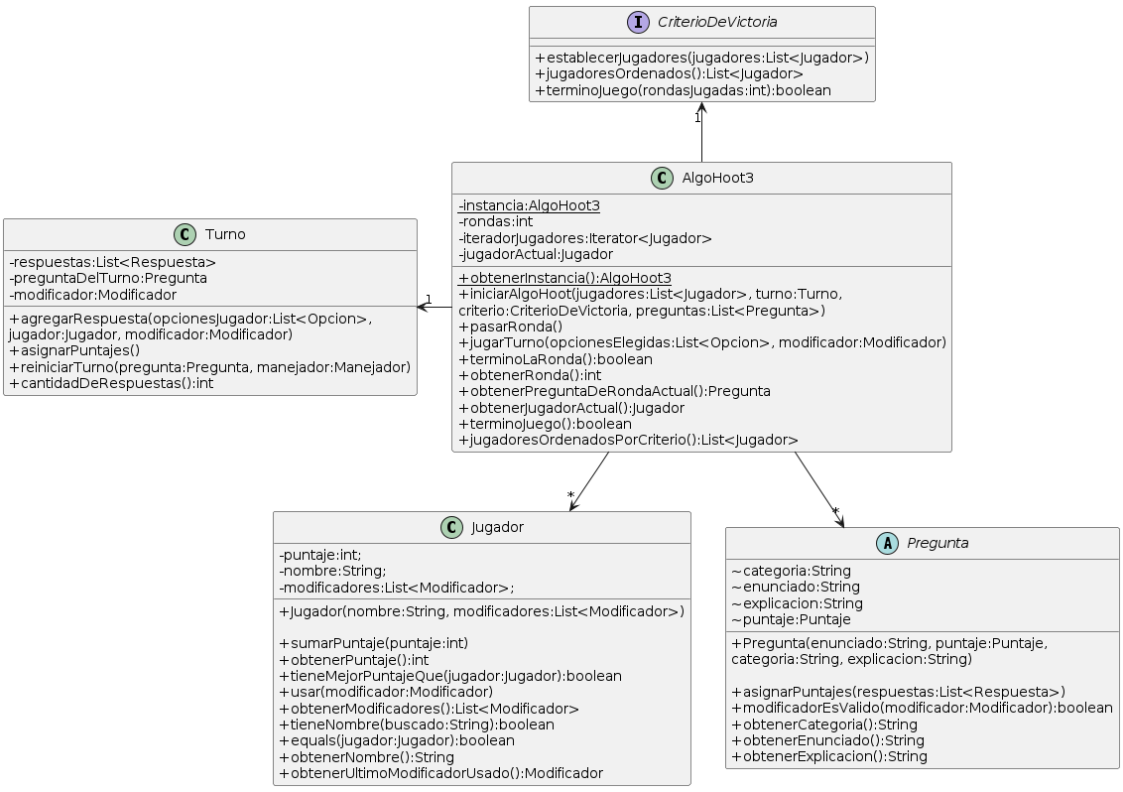

Figura 6: Las relaciones de la clase AlgoHoot3, entidad principal de nuestro modelo.

4. Diagramas de secuencia

En los diagramas decidimos mostrar los casos de uso de un test donde se puede observar las relaciones dinámicas entre entidades más interesantes de la solución y de un caso de uso hipotetico. Éstos son el primer test de TurnoTest, y el quinto de AlgohootTest. Además mostramos un caso de uso donde se observa el comportamiento del Anulador.

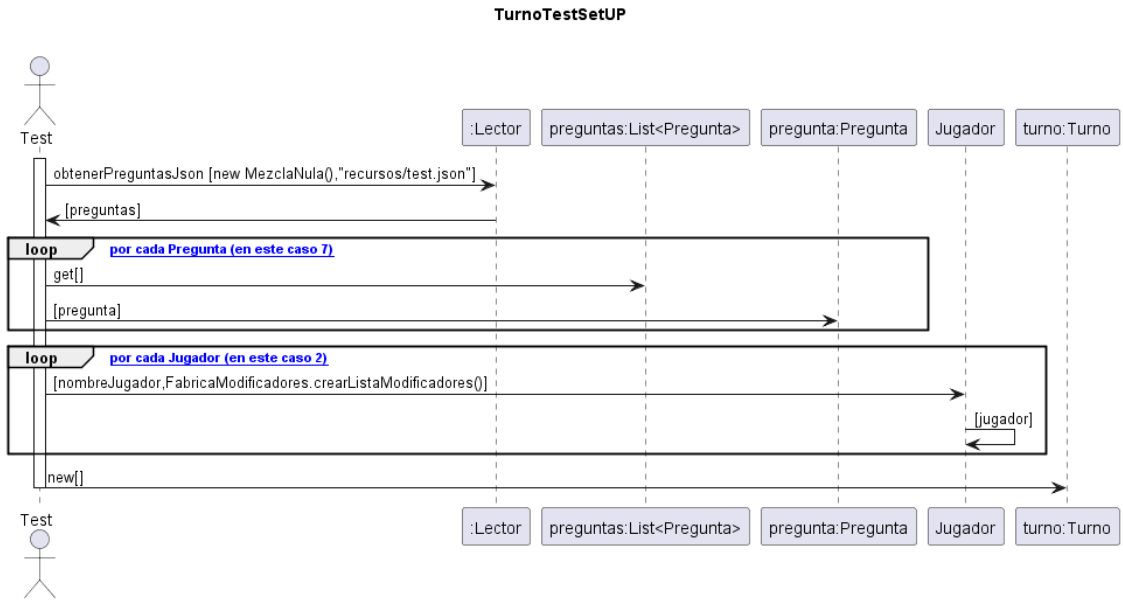

Figura 7: Secuencia para inicializar cada test en TurnoTest.

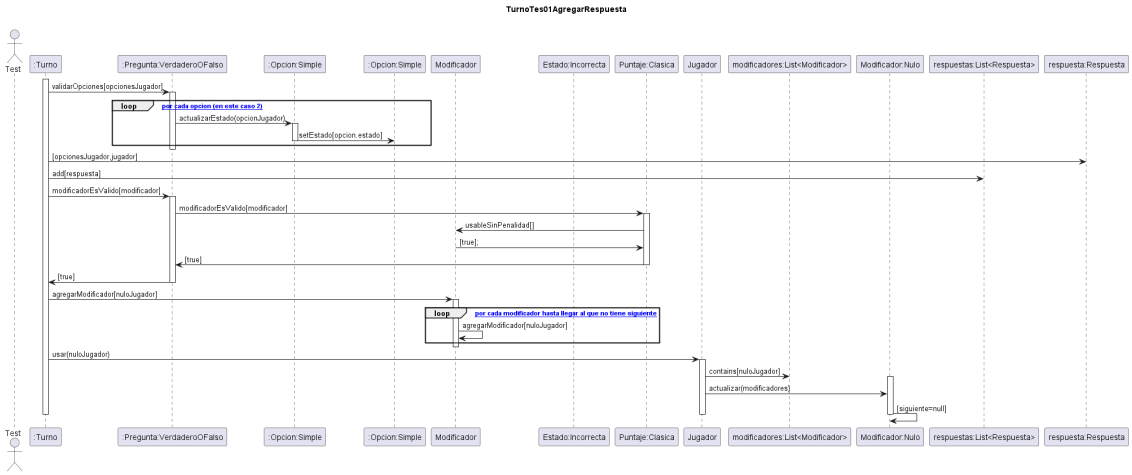

Figura 8: Secuencia de AgregarRespuesta en un Turno, particularmente en el Test01.

El siguiente diagrama muestra la ejecución del test01 (a partir de la asginacion de las opciones a cada jugador, para facilitar el entendimiento). Naturalmente, tampoco se desarrolla lo ya mostrado en los diagramas anteriores.

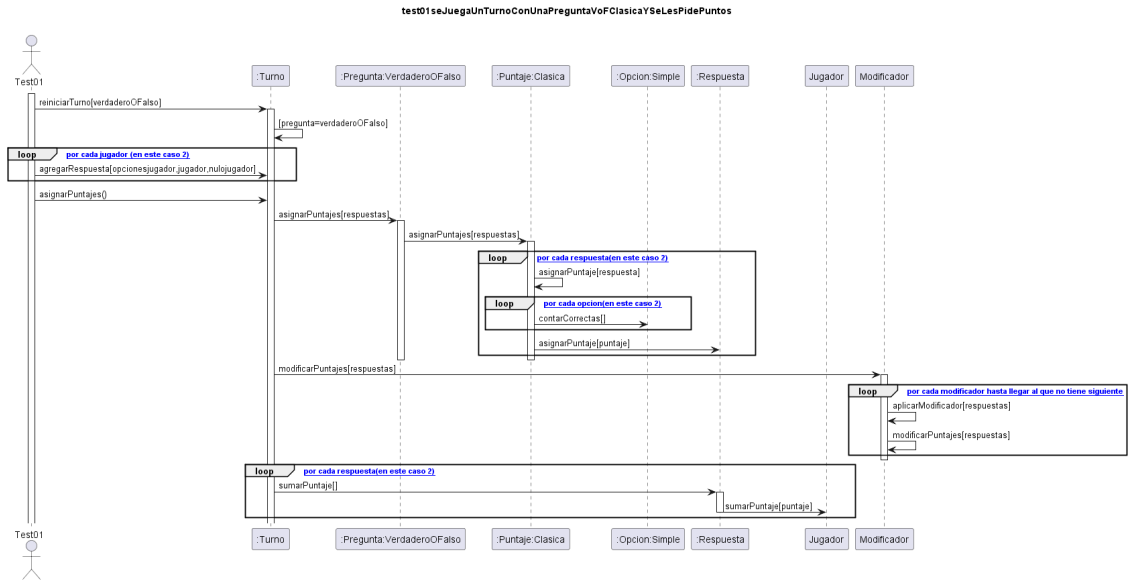

Figura 9: Diagrama de secuencia Test01Turno.

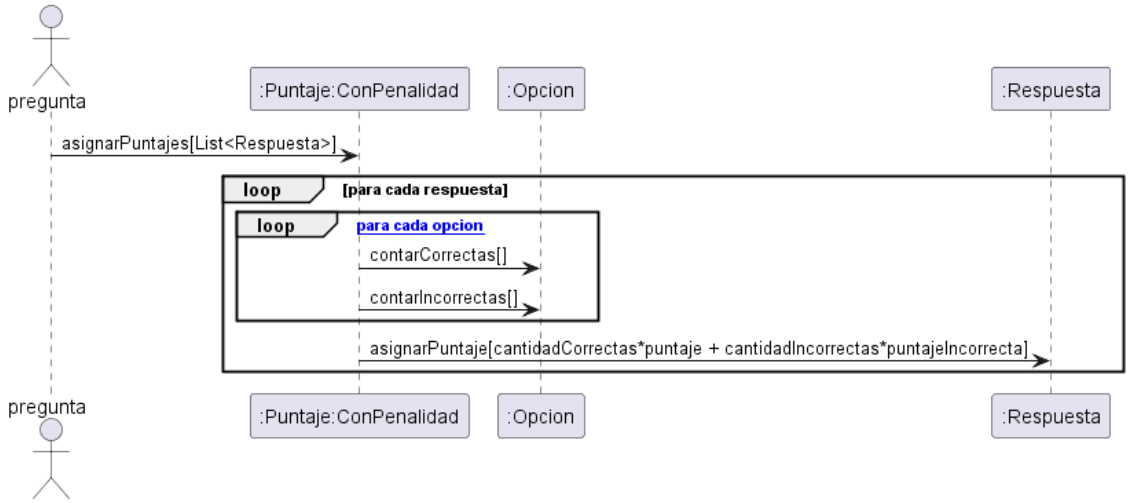

Figura 10: Diagrama de secuencia de uso de ConPenalidad.

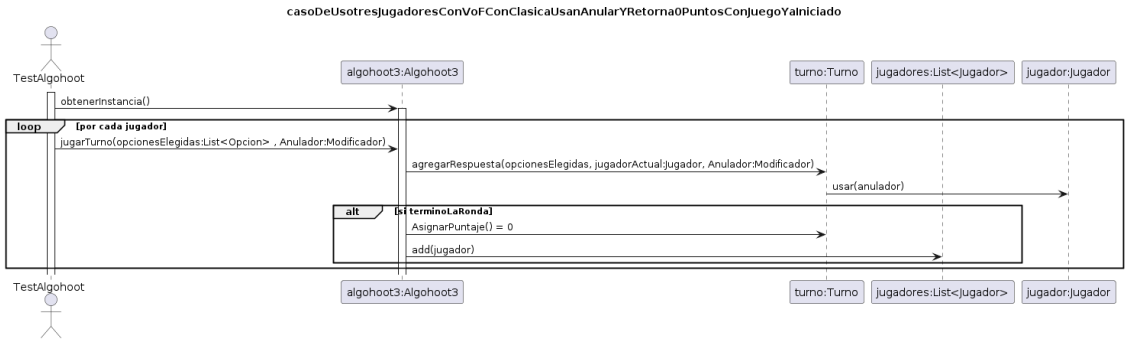

Figura 11: Diagrama de secuencia de Caso de uso algohoot.

5. Diagrama de paquetes

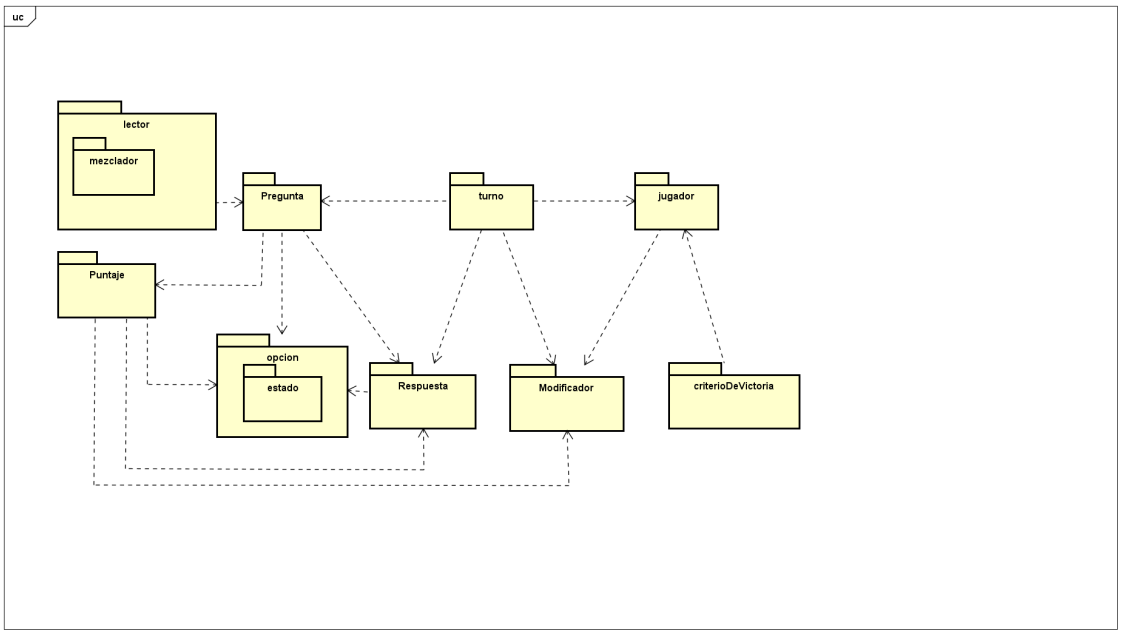

Figura 12: Diagrama de paquetes.

6. Detalles de implementación

Como se puede observar en la figura 1, para modelar las distintas maneras de puntuar las preguntas (de forma clásica, con penalidad, etc) decidimos implementar una clase abstracta Puntaje con distintas subclases. En este caso nos resultó útil usar Herencia, ya que así pudimos implementar todo el comportamiento común a las distintas formas de puntuar en la abstracta Puntaje, y lo particular en sus hijas.

En la figura 2 podemos ver los distintos tipos de modificadores, que heredan de la clase abstracta Modificador. De manera análoga a lo explicado para Puntaje, para modelar los modificadores

fue provechoso usar Herencia para poder unificar el comportamiento y estado compartido entre los distintos modificadores, y delegar en las subclases lo que varía (la manera de alterar a los puntajes). También puede observarse que Modificador contiene otro Modificador, ya que utilizamos el patrón de diseño Decorator. También usamos el patrón Null Object, que nos facilitó modelar el caso de uso en el que los jugadores no seleccionan un Modificador.

En la figura 4 es interesante analizar los atributos de las Preguntas. La abstracta Pregunta contiene un puntaje, en el que delega, naturalmente, la asignación del puntaje. Y las subclases poseen opciones, a las que le delegan la validación. Los distintos tipos de Opción, que realizan un mismo comportamiento de manera distinta, pueden verse en la figura 3

La figura 5 ya nos brinda una perspectiva más global del modelo. Se observa el patrón state en la relación de Turno con Pregunta y Modificador. También se ve la relación de Pregunta y Puntaje, donde la primera delega comportamiento al segundo. Algo similar pasa entre Respuesta y Opción. Observando los atributos quedan *no tan explícitamente* a la vista otros vínculos entre las entidades.

La figura 6 nos muestra la relación de la clase AlgoHoot3 con el resto de las entidades. Para la clase central de la solución, AlgoHoot3, utilizamos el patrón Singleton. Esto nos permite restringir la instanciación de dicha clase y facilitar la relación con las vistas y los controladores. Cabe mencionar que desarrollamos nuestra aplicación siguiendo el patrón MVC (Modelo, Vista, Controlador).

Por último, cabe mencionar que para instanciar las preguntas, modificadores, opciones y jugadores utilizamos fábricas, de acuerdo al patrón Factory Method.

7. Excepciones

ArchivoInexistenteException: Esta excepción se lanza cuando el juego no encuentra el archivo de preguntas necesario.

CantidadDeJugadoresMenorADosException: Esta excepción se lanza cuando se intenta jugar con menos de dos jugadores.

ModificadorInexistenteException: Esta excepción se lanza si el jugador intenta usar un modificador que no posee.

ModificadorInvalidoException: Esta excepción se lanza si se intenta usar un modificador no permitido para la forma de puntuar las preguntas (Penalidad/Sin Penalidad).

OpcionesDeTamanioInvalidoException: Esta excepción se lanza si se intenta enviar una cantidad de opciones mayor a la esperada.

OpcionesIncorrectasException: Esta excepción se lanza si se responde con opciones de un tipo inconsistente con la pregunta que se está respondiendo.