.UBAfiuba
FACULTAD DE INGENIERÍA

Trabajo Práctico 2 — Java

KAHOOT

[7507/9502] Algoritmos y Programación III

Curso 2

Grupo 4

Primer cuatrimestre de 2020

Alumnos	Padrón	E-mail
Giardina, Fernando	103732	fgiardina@fi.uba.ar
Panaccio, Guido	103851	gpanaccio@fi.uba.ar
Rotondo, Matías	103852	mrotondo@fi.uba.ar
Stenghele, Juan	104000	jstenghele@fi.uba.ar
Waisten, Lucas	101908	lwaisten@fi.uba.ar

Corrector: Pablo Suárez

Enlace al repositorio: https://github.com/Algoritmos-3-FIUBA/TP2

Informe

1. Introducción

El presente informe reúne la documentación de la solución del segundo trabajo práctico grupal de la materia Algoritmos y Programación III que consiste en desarrollar un juego basado en el Kahoot!, programado en Java y utilizando los conceptos del paradigma de la programación orientada a objetos vistos a lo largo del curso.

2. Supuestos

Debido a que determinadas condiciones del problema a resolver no han sido otorgadas por la cátedra inicialmente, en ciertas ocasiones el criterio al abordar situaciones problemáticas puntuales quedaba a cargo del alumno, se han tenido en cuenta a la hora de modelar la solución los siguientes supuestos.

- El jugador puede tener puntos negativos. Esto ocurre al tener preguntas que penalizan al jugador y sus puntos en ese momento son menores a los que la pregunta penaliza. Si bien a la hora de implementarlo, la diferencia entre suponerlo de esta manera y hacer que el jugador no pueda tener puntos negativos están a un if de distancia.

- Los jugadores pueden no responder ninguna opción, es decir, realizar una respuesta vacía en todos los tipos de pregunta, excepto en pregunta Verdadero Falso Clásico y Verdadero Falso Penalidad. Si el jugador no selecciona ninguna pregunta, recibirá cero puntos.

- Asumimos que toda pregunta puede tener opciones incorrectas. Es decir, todas las preguntas pueden tener opciones que no correspondan a ningún grupo (para group choice) o no formen parte de las opciones a ordenar (ordered choice). Lógicamente, este supuesto solo es para los dos tipos de pregunta antes mencionados porque el resto por el enunciado pueden tener opciones incorrectas. Y si el jugador selecciona una opción incorrecta, aunque todas las demás opciones estén bien, suma cero puntos.

- La pregunta group choice asigna un punto por cada grupo que el jugador realice correctamente. Es decir, si hay dos grupos en los cuales colocar opciones, si ambos son completados correctamente, recibe dos puntos. Si, en

cambio, solamente arma un grupo bien (recordar que puede haber opciones incorrectas) entonces solo recibe un punto.

- En cuanto al uso de la exclusividad decidimos que si solamente un jugador contesta bien y otro u otros usaron exclusividades, aunque el jugador no haya sido el que uso la exclusividad en su respuesta, entonces se le aplica el beneficio.

- Otra suposición que tuvimos que realizar a la hora de implementar la exclusividad para n jugadores, es el hecho de que si se dan las condiciones de uso de la exclusividad (sólo uno contestó bien) y m jugadores utilizaron exclusividad entonces la amplificación de puntos que recibe el jugador que contestó bien es de 2^m. Esto cumple para el caso de dos jugadores que utilizaremos para el Kahoot, pero como si el modelo por open-close es ampliable a n jugadores entonces decidimos ver qué pasaría con n jugadores. Entonces, si hay n jugadores y solo uno contesta bien, si se usó una exclusividad entonces los puntos del jugador se duplican (2^1), si se usaron dos se cuadruplican (2^2) y así para m (2^m).

3. Modelo de dominio

Se procedió a escribir pruebas unitarias propias y, mediante la modalidad TDD pregonada por los docentes, se llevó a cabo el desarrollo sistema. Ante cada funcionalidad agregada, se realizaron pruebas de integración, entre ellas las pedidas por la cátedra, para verificar la integración de las distintas clases que componen el modelo.
A su vez, se presentan en la solución del modelo planteada las siguientes entidades.

3.1 Pregunta y Asociadas

- Pregunta: Clase madre donde heredan todos los tipos de pregunta. Contiene el método abstracto evaluarRespuestas que recibe una linked list de respuestas y en cada clase hija de pregunta se implementará la forma en que se evalúa.

- PreguntaVerdaderoFalso: Clase madre donde heredan las clases PreguntaVerdaderoFalsoPenalidad y PreguntaVerdaderoFalsoClasico. Implementa el evaluarRespuestas, porque en ambos casos la forma de hacerlo es la misma salvo que entrega diferentes puntos. A las preguntas verdadero falso se les debe establecer la opción correcta a través de los setters setVerdaderoOpcionCorrecta y setFalsoOpcionCorrecta.

- PreguntaVerdaderoFalsoPenalidad: Guarda dos opciones con sus respectivos puntajes sabiendo cual de ellas es la correcta. Cuando el jugador responde la opción correcta tendrá un punto, pero si responde la incorrecta se le restará un punto.

- PreguntaVerdaderoFalsoClasico: Guarda dos opciones con sus respectivos puntajes sabiendo cual de ellas es la correcta. Si el jugador responde la opción correcta tendrá un punto, en caso contrario no tendrá puntaje.

- PreguntaMultipleChoiceParcial: La clase conoce sus opciones correctas. Recibe las opciones que seleccionó el jugador y las compara con sus opciones, otorgando los puntos de acuerdo al puntaje de la opción seleccionada. Se considera que una PreguntaMultipleChoiceParcial es correcta cuando se contesta correctamente cuando al menos una de las opciones correctas fue seleccionada.

- PreguntaMultipleChoice: La clase conoce sus opciones correctas. Recibe las opciones que seleccionó el jugador y las compara con sus opciones, si las opciones que seleccionó el jugador coinciden con las opciones correctas de la pregunta se le asigna un punto. En caso contrario, ya sea porque seleccione una opción incorrecta, opciones de menos u opciones de más, el jugador no tendrá puntaje.

- PreguntaOrderedChoice: La clase conoce sus opciones correctas. Recibe las opciones que seleccionó el jugador y las compara con sus opciones, si las opciones que seleccionó el jugador son iguales (están en orden y son las mismas) con las opciones correctas de la pregunta se le asigna un punto. En caso contrario, ya sea porque seleccione una opción incorrecta, opciones de menos, opciones de más o el orden es incorrecto, el jugador no tendrá puntaje.

- PreguntaGroupChoice: Almacena grupos con sus opciones correctas (para el juego son siempre dos). Esta clase recibe una colección de grupos junto con las opciones que el jugador haya colocado en cada grupo. Se encarga de evaluar cada grupo verificando si sus opciones son las mismas que el grupo que posee la pregunta (el grupo correcto).

3.2 Respuesta y Asociadas

- Respuesta: Clase abstracta donde heredan todos los tipos de respuestas. La clase recibe al jugador y puede recibir un multiplicador o una exclusividad (es un o exclusivo), en caso de que no se reciba alguno de estos últimos, se designará uno de cada uno por defecto. Cada Respuesta inicia con un estado incorrecto, y ese estado puede cambiar o no una vez que la respuesta haya sido evaluada por la pregunta. Las clases hijas, se diferencian en cuantas y como guardan las opciones.

- RespuestaUnica: Recibe solo una opción, el jugador y puede recibir el multiplicador o una exclusividad. Le indica a la pregunta la opción seleccionada para que sea evaluada, y así se determine el estado de la respuesta.

- RespuestaMultiple: Recibe una colección de opciones, el jugador y puede recibir el multiplicador o una exclusividad. Le indica a la pregunta la colección de opciones seleccionadas para que sea evaluada, en caso de que sean las opciones correctas se cambia el estado de la respuesta.

- RespuestaGrupos: Recibe una lista de colecciones, el jugador y puede recibir el multiplicador o una exclusividad. PreguntaGroupChoice será la encargada de ir evaluando cada grupo, otorgando el puntaje.

- EstadoRespuesta: Clase abstracta donde heredan los estados que puede tener una respuesta (patrón state). Los estados corresponden a la correctitud de la respuesta y es Pregunta quien le dice que en qué estado debe estar (la corrige).

- EstadoCorrecto: Se encarga de otorgar los puntos a un jugador en caso de que una respuesta sea correcta, además cuando una respuesta cambia a este estado se le notifica a la exclusividad (la exclusividad depende de la cantidad de respuestas correctas).

- EstadoIncorrecto: Cada respuesta inicia con este estado, que puede cambiar o no una vez que la respuesta haya sido evaluada. Se encarga de restarle (o no sumarle nada) puntos al jugador en caso de que la pregunta sea con penalidad.

3.3 Opciones y Asociadas

- Opción: Clase madre donde heredan los tipos de opciones, cada opción puede tener o no un puntaje (hay un constructor para cada caso). Mediante el uso del polimorfismo, se logra que cada tipo de opción entiendan los mismos mensajes pero el comportamiento sea diferente.

- OpcionCorrecta: Sabe agregarse al grupo correspondiente, y para el caso de una respuesta única asigna el estado de una respuesta como correcta.

- OpcionIncorrecta: Sabe agregarse al grupo correspondiente, y para el caso de una respuesta única asigna el estado de una respuesta como incorrecta.

- ColeccionOpciones: Esta clase fue creada para trabajar más ágilmente con conjuntos de opciones. Entre los mensajes más destacables que entiende encontramos la capacidad de determinar si es igual a otra colección (todas sus opciones son iguales y están en el mismo orden), si tienen los mismos elementos (pueden estar en distinto orden) y si comparten al menos un elemento con otra colección. Decidimos implementarlo debido a que ninguna colección primitiva de Java se adapta perfectamente a cada caso.

3.4 Jugador

- Jugador: Es la clase que representa a un jugador real en el juego. Conoce su nombre y puntos. Posee multiplicadores y exclusividades que podrá usar dependiendo del tipo de pregunta. Entiende el método sumarPuntos que es fundamental a la hora de evaluar las respuestas.

3.5 Puntos

- Puntos: Esta clase es utilizada para representar los puntos de cada jugador y los puntos de las opciones. Sabe sumar y multiplicar con otros puntos. Se creó para no tener que trabajar con enteros.

3.6 Amplificador

- Amplificador: Se utiliza para modificar los puntos a otorgar a un jugador en caso de que quiera utilizar un beneficio (multiplicador o exclusividad) al realizar una respuesta. En caso de no utilizar ningún beneficio se tendrá un amplificador por default con factor igual a uno. El Amplificador como su nombre lo indica recibe una cantidad de puntos y los devuelve modificados según un factor ($puntosDevueltos = puntosRecibidos \,.\, Factor$) con el método amplificarPuntos.

3.7 Multiplicador y asociados

- Multiplicador: Clase abstracta donde heredan los tipos de multiplicadores. Cada tipo de multiplicador utiliza su amplificador para modificar los puntos. Una vez usado el multiplicador, este deja de utilizarse cambiando el factor de su amplificador a uno.

- MulplicadorDefault: Cuando el jugador no selecciona ningún multiplicador se utiliza éste por defecto, su amplificador es con factor uno.

- MultiplicadorPorDos: Tiene un factor de amplificación igual a dos.

- MultiplicadorPorTres: Tiene un factor de amplificación igual a tres.

3.8 Exclusividad y asociados

- Exclusividad: Esta clase posee un amplificador que inicialmente tiene un factor igual a dos y su estado en EstadoNoAsigna donde su factor es cero (es decir, no suma puntos). En caso de que haya una respuesta correcta cambia su estado. El estado finalmente decidirá si solamente un jugador contestó bien (ver diagrama de estados para mayor detalle). Delega a la clase EstrategiaMultiplicativa la actualización de su factor. Decidimos extraer en una clase la forma en la que se va aumentando el factor a medida que encuentra otras exclusividades para poder dejarlo abierto a la extensión (principio de open-close) y mantener fuera otra razón de cambio de exclusividad (principio de single responsibility). Si se encuentra una exclusividad diferente a la default en la lista de respuestas a evaluar, entonces se le notifica a respuesta que use el amplificador de Exclusividad.

- Exclusividad Default: En caso de que no se utilice la exclusividad, el jugador tendrá por default una exclusividad donde su amplificador tendrá un factor igual uno con un EstadoNoAsigna.

- EstadoExclusividad: Clase abstracta donde heredan los tipos de uso de la exclusividad (patrón state).

- EstadoAsigna: Indica el uso del beneficio de la exclusividad y establece el amplificador con su factor y puede pasar al estado EstadoNoAsignaMas donde cambia el amplificador.

- EstadoNoAsigna: Se utiliza para no asignar puntaje al jugador, para realizar esto se cambia el factor del amplificador a cero. Puede cambiar de estado a EstadoAsigna cuando se encuentra una respuesta correcta.

- EstadoNoAsignaMas: Una vez asignado el puntaje del jugador por la clase EstadoAsigna, se pasa a este nuevo estado donde el factor del amplificador cambiaría a cero.

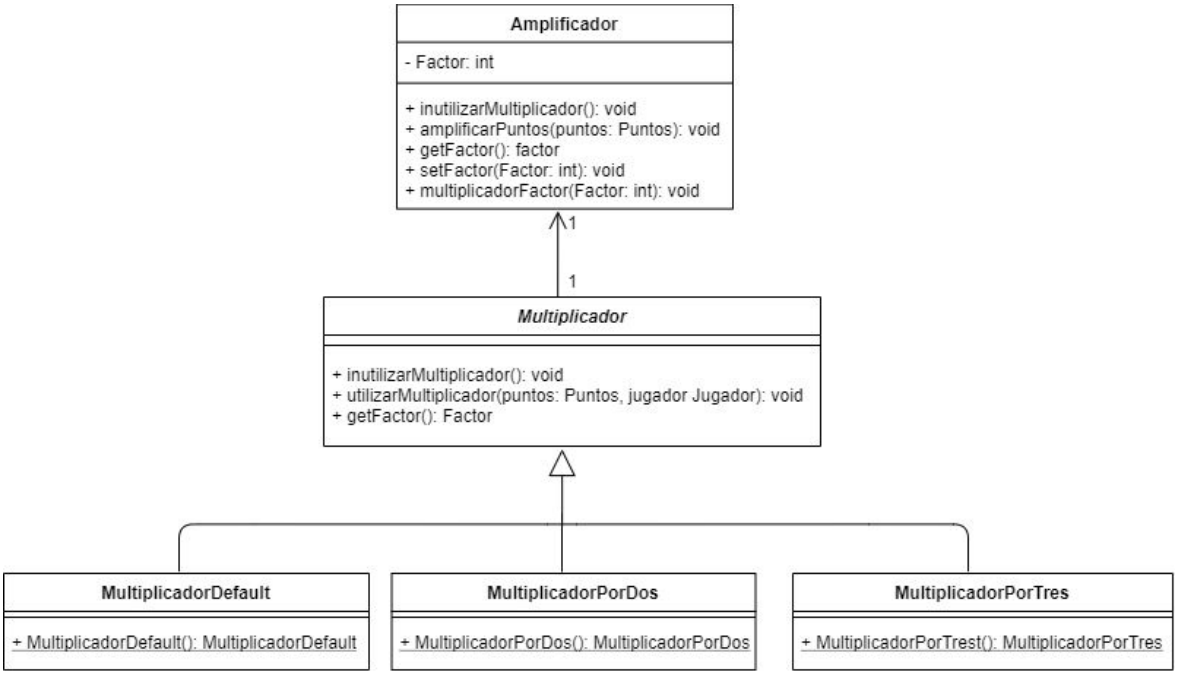
- EstrategiaAumentoFactor: Clase abstracta que recibe la lista de respuestas para actualizarles el factor del amplificador.

- EstrategiaMultiplicativa: Esta clase se encarga de recorrer la lista de respuestas de los jugadores salvo la respuesta actual (la que llama al método), indicando a cada respuesta que actualice su factor de amplificación.

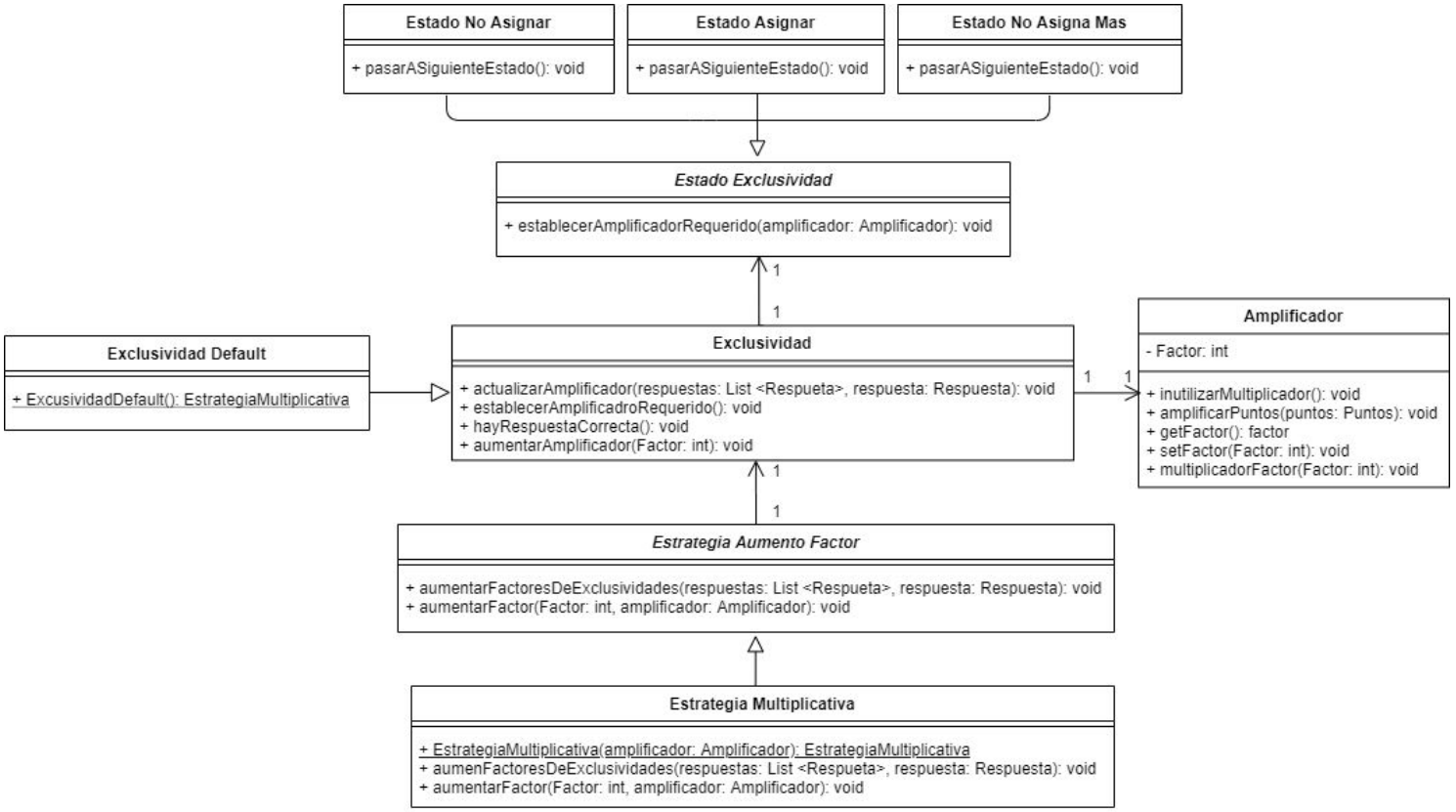
4. Diagrama de clase

A continuación se presenta el diagrama de clases correspondiente al dominio del modelo. Dado al gran volumen del diagrama el mismo será mostrado en partes. Las clases lógicamente corresponden a las explicadas en la sección anterior.

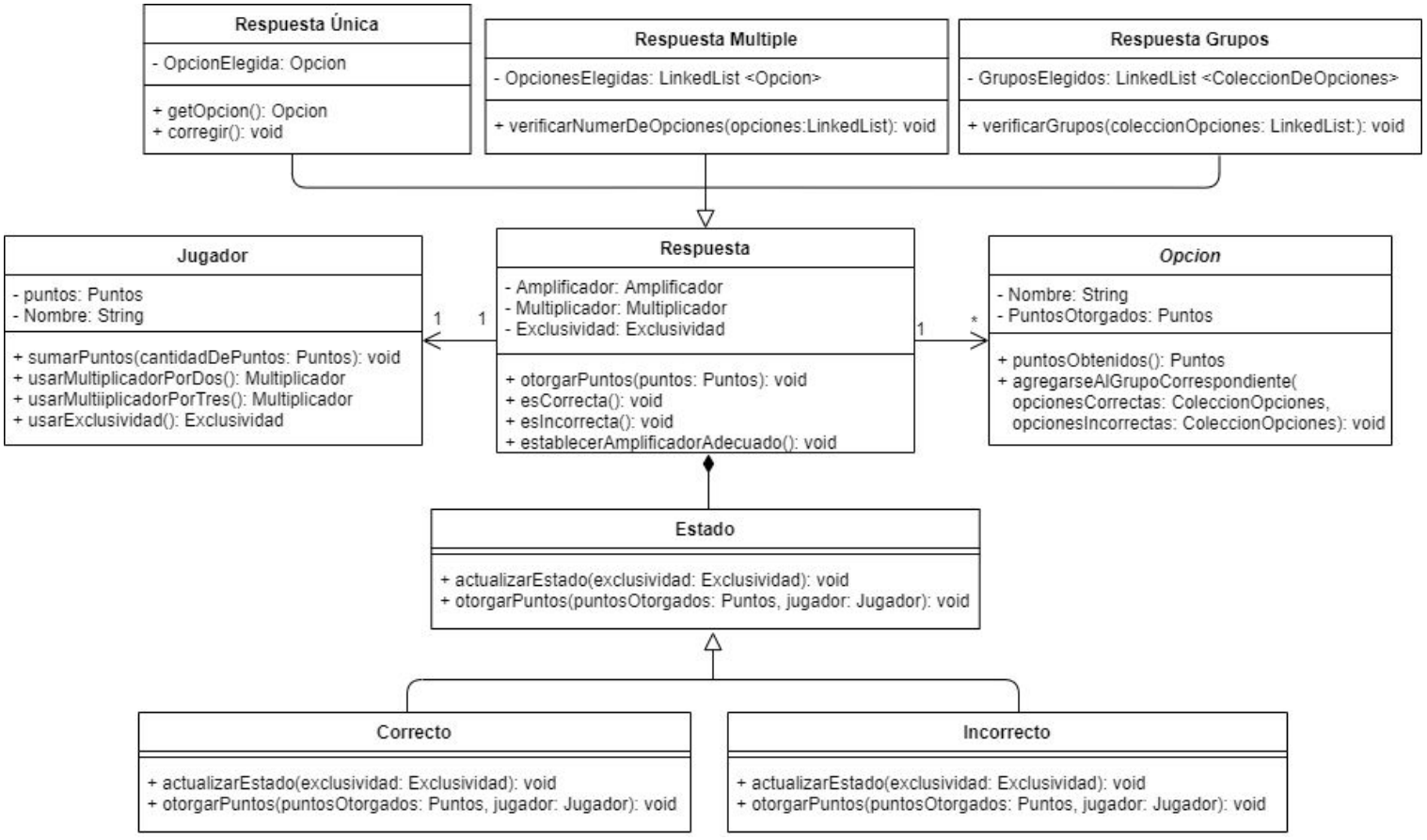
4.1 Diagrama de multiplicadores



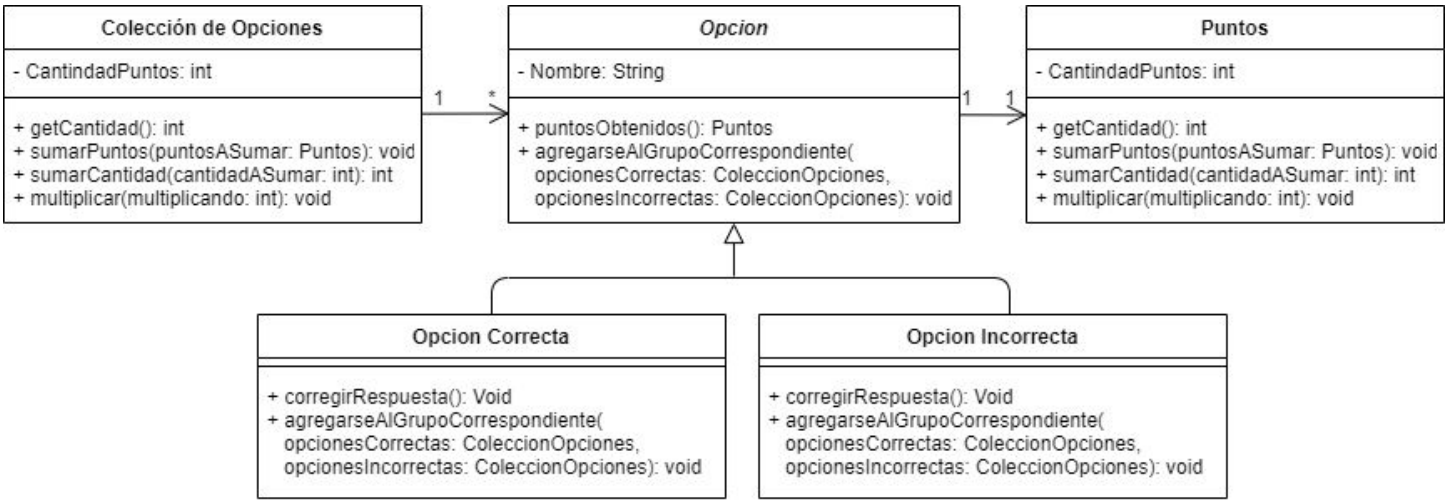
4.2 Diagrama de exclusividad



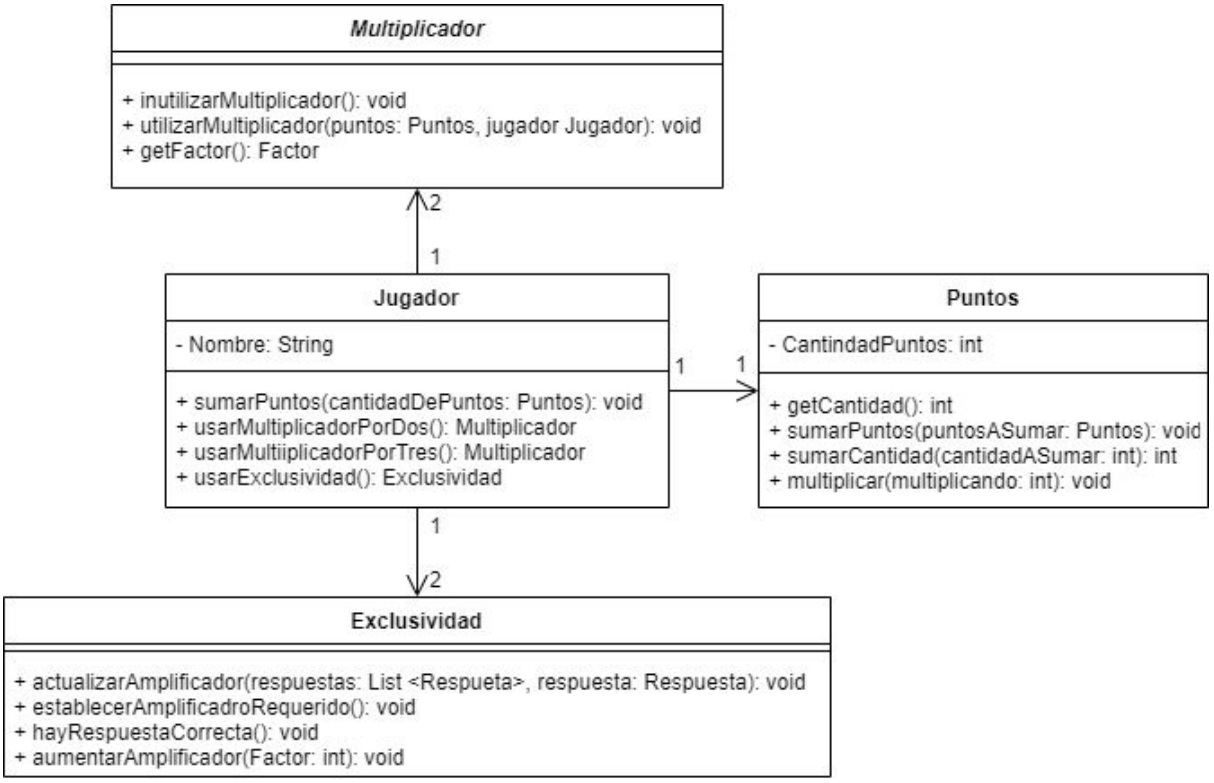
4.3 Diagrama de clases de las respuestas



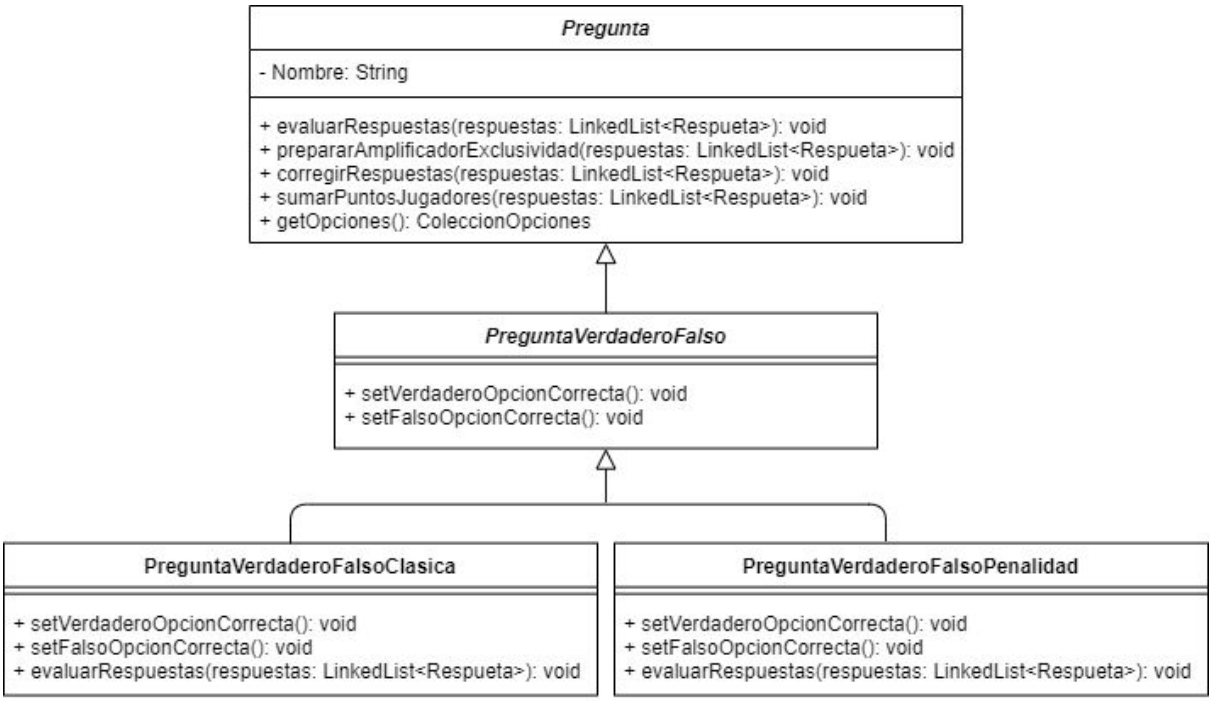
4.4 Diagrama de clases de las opciones

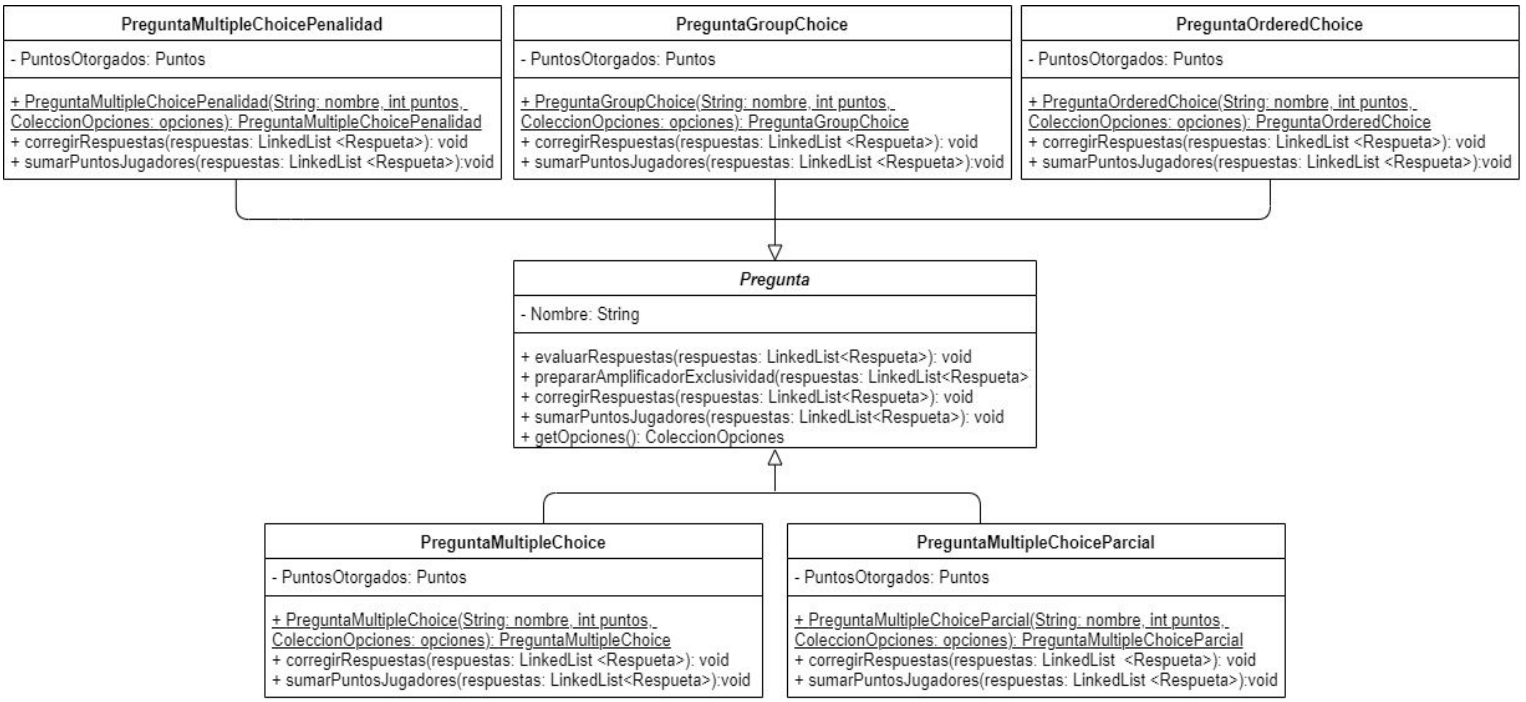


4.5 Diagrama de clases del jugador



4.5 Diagrama de clases de las preguntas




PreguntaMultipleChoicePenalidad
- PuntosOtorgados: Puntos
+ PreguntaMultipleChoicePenalidad(String: nombre, int puntos, ColeccionOpciones: opciones): PreguntaMultipleChoicePenalidad
+ corregirRespuestas(respuestas: LinkedList <Respueta>): void
+ sumarPuntosJugadores(respuestas: LinkedList <Respueta>):void
PreguntaGroupChoice
- PuntosOtorgados: Puntos
+ PreguntaGroupChoice(String: nombre, int puntos, ColeccionOpciones: opciones): PreguntaGroupChoice
+ corregirRespuestas(respuestas: LinkedList <Respueta>): void
+ sumarPuntosJugadores(respuestas: LinkedList <Respueta>):void
PreguntaOrderedChoice
- PuntosOtorgados: Puntos
+ PreguntaOrderedChoice(String: nombre, int puntos, ColeccionOpciones: opciones): PreguntaOrderedChoice
+ corregirRespuestas(respuestas: LinkedList <Respueta>): void
+ sumarPuntosJugadores(respuestas: LinkedList <Respueta>):void
Pregunta
- Nombre: String
+ evaluarRespuestas(respuestas: LinkedList<Respueta>): void
+ prepararAmplificadorExclusividad(respuestas: LinkedList<Respueta>
+ corregirRespuestas(respuestas: LinkedList<Respueta>): void
+ sumarPuntosJugadores(respuestas: LinkedList<Respueta>): void
+ getOpciones(): ColeccionOpciones
PreguntaMultipleChoice
- PuntosOtorgados: Puntos
+ PreguntaMultipleChoice(String: nombre, int puntos, ColeccionOpciones: opciones): PreguntaMultipleChoice
+ corregirRespuestas(respuestas: LinkedList <Respueta>): void
+ sumarPuntosJugadores(respuestas: LinkedList<Respueta>):void
PreguntaMultipleChoiceParcial
- PuntosOtorgados: Puntos
+ PreguntaMultipleChoiceParcial(String: nombre, int puntos, ColeccionOpciones: opciones): PreguntaMultipleChoiceParcial
+ corregirRespuestas(respuestas: LinkedList <Respueta>): void
+ sumarPuntosJugadores(respuestas: LinkedList <Respueta>):void

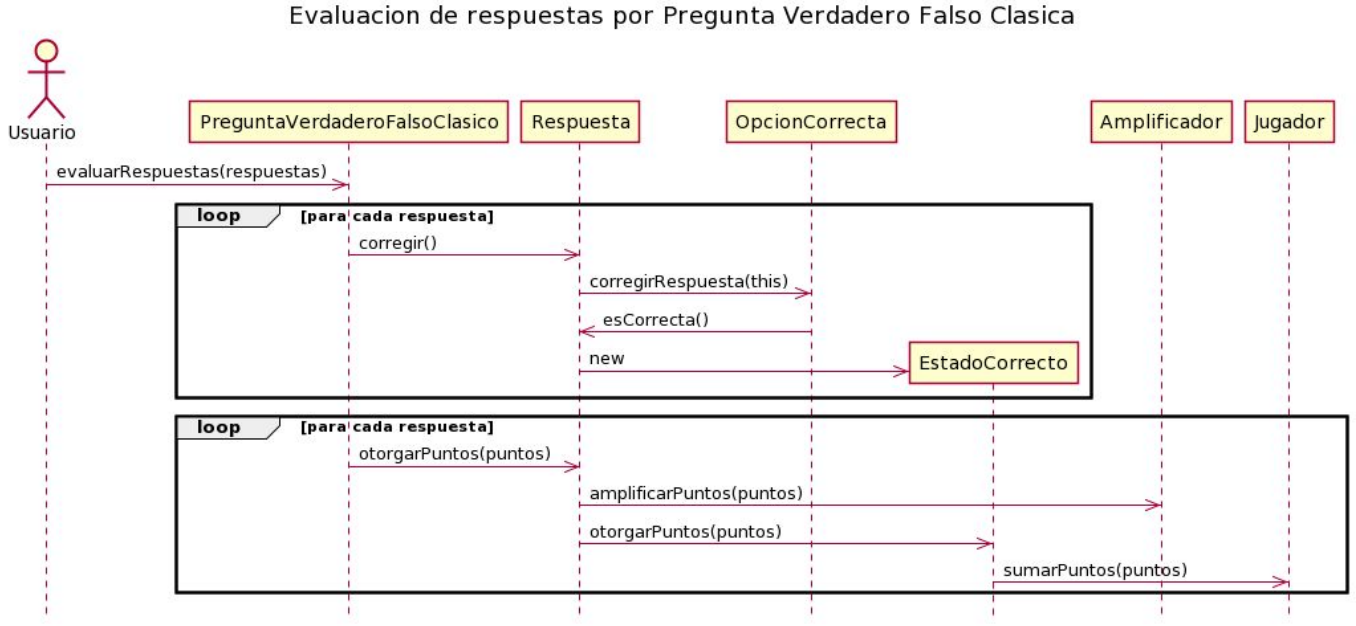
5. Diagramas de secuencias

A continuación presentaremos los casos más interesantes que se encuentran presente en el flujo de ejecución del programa, mediante los diagramas de secuencias.

Los primeros seis diagramas corresponden a la evaluación de respuestas en los diferentes tipos de preguntas.

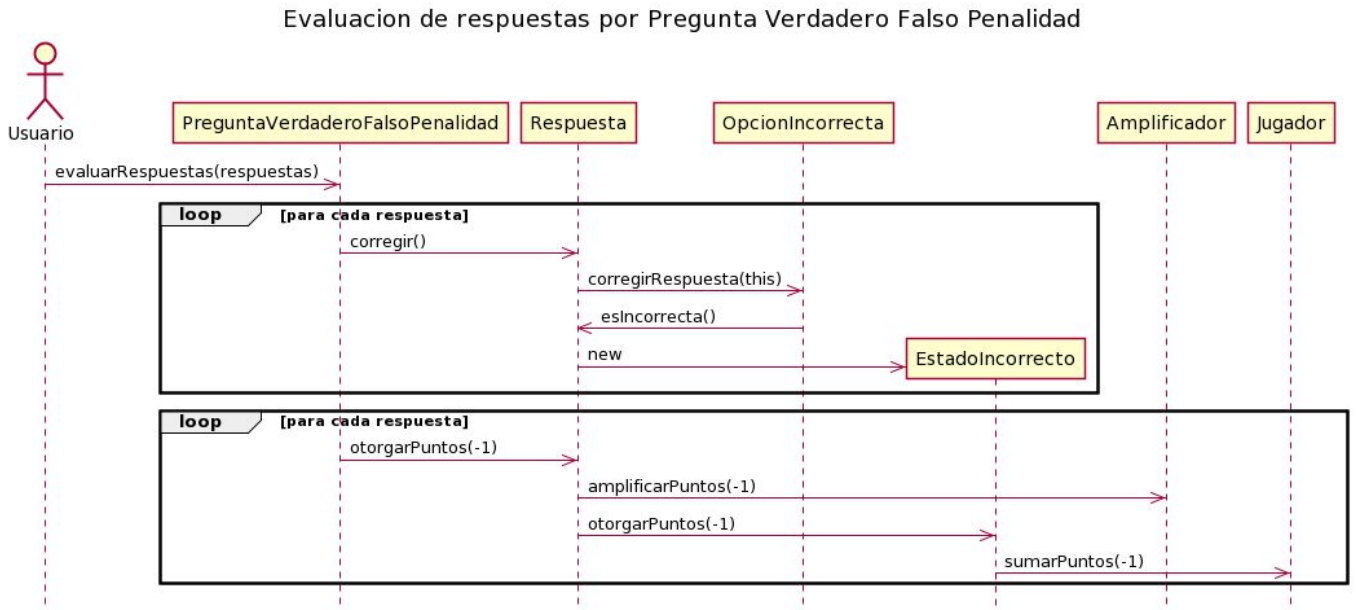
En primer lugar representamos la evaluación de respuesta en verdadero Falso Clásico. El caso representado es un caso genérico en el que todas las respuestas recibidas son correctas. En el primer loop se corrigen las respuestas. Para ello la pregunta se le delega a la respuesta que a su vez se lo pregunta a su opción. Para este tipo de pregunta, si la opción es correcta, la respuesta también lo es y recíprocamente si es incorrecta. Al ser en este caso la opción correcta, entonces también lo es la respuesta e instancia un estado correcto que lo establece como su estado.

En el segundo loop se realiza la entrega de puntos. La pregunta se lo delega a la respuesta, quien los manda a amplificar y posteriormente se los envía al estado. Al ser el estado correcto, le entrega el estado los puntos recibidos al jugador. De esta forma el jugador recibe los puntos.

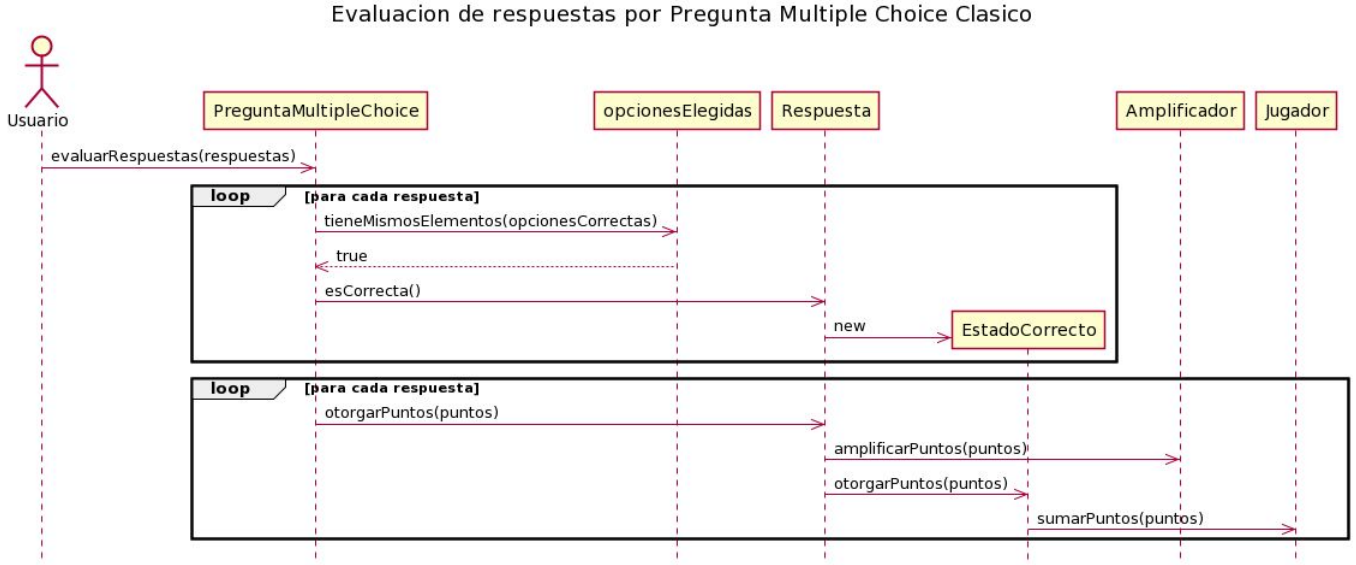


Para el segundo diagrama, mostraremos la evaluación de respuestas en la pregunta verdadero falso penalidad. En este caso, a diferencia del primero la opción contestada es incorrecta, por lo que debe darle puntos negativos (lo penaliza). La

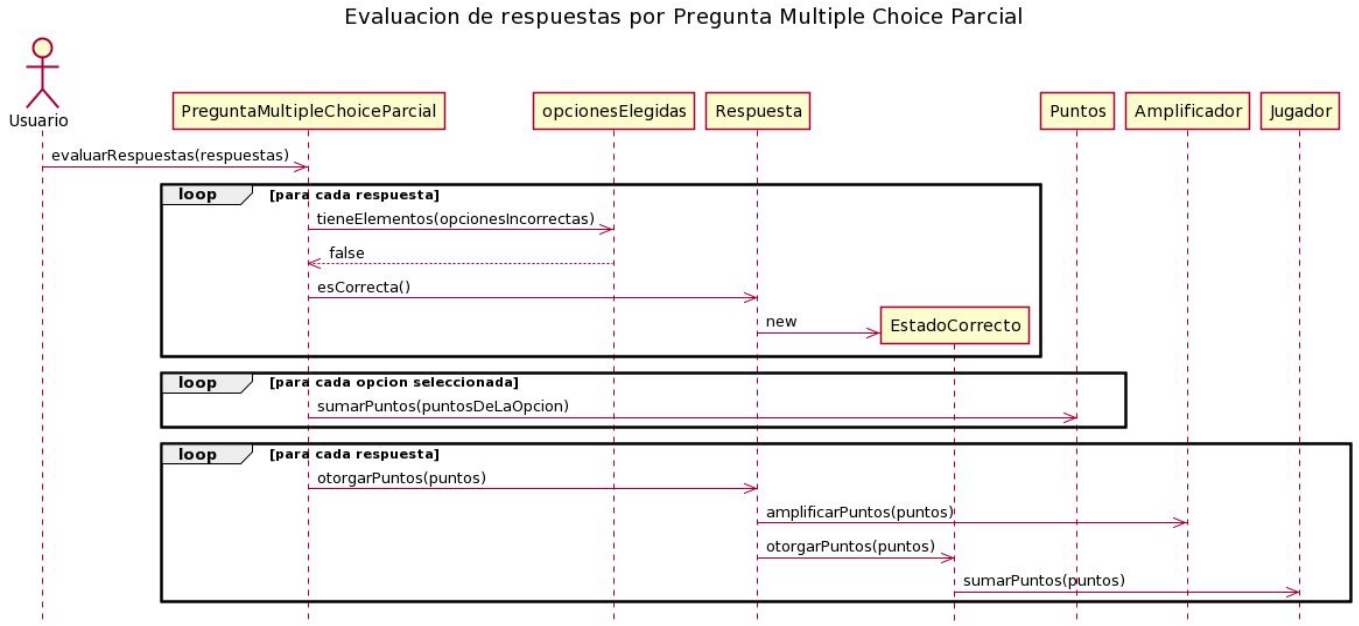
secuencia es similar salvo que se instancia un estado incorrecto en Respuesta al recibir el mensaje esIncorrecta() de la opción. Además plasmamos un caso como los del juego, donde este tipo de pregunta si se responde mal penaliza con un punto (esto se ve en que la hace sumar -1 puntos al jugador).



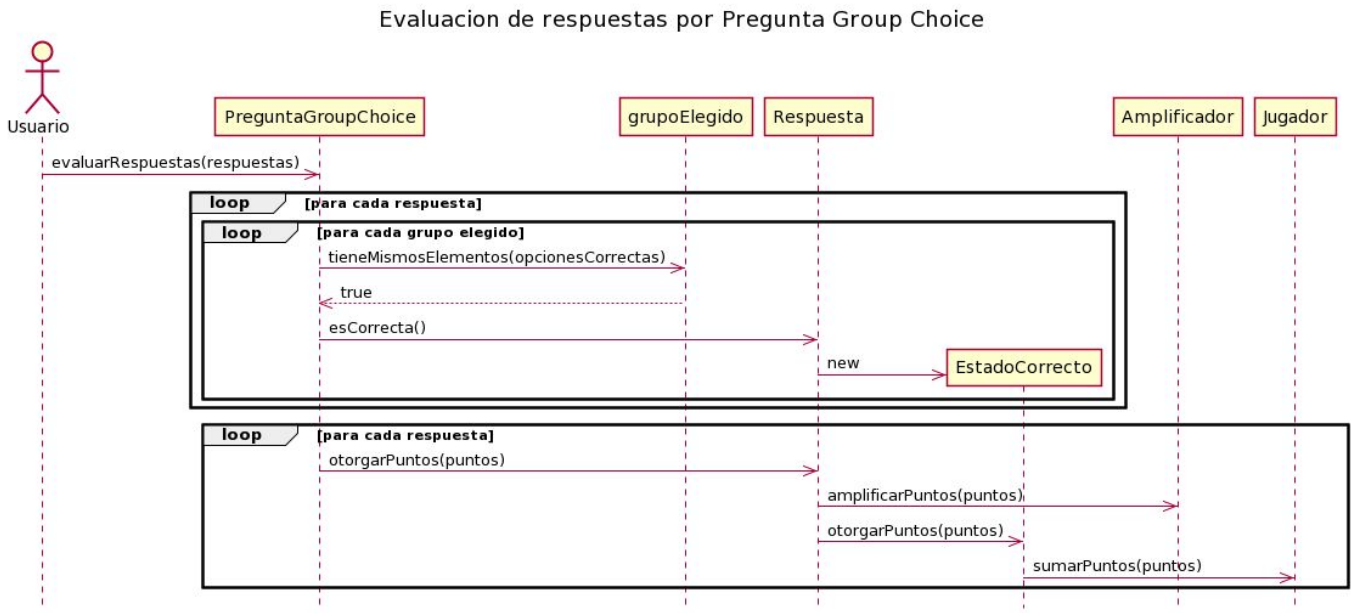
Para el tercer diagrama se muestra la evaluación de respuestas de Multiple choice clásico, también se muestra un caso genérico. Para la corrección de las respuestas la pregunta le envía el mensaje a la colección de opciones elegidas preguntándole si tiene los mismos elementos (aunque puede ser en distinto orden) que sus opciones correctas. Como si en este caso consideramos que las respuestas son correctas, entonces pregunta se notifica la respuesta que establece un estado correcto. Por último, la asignación de puntos es igual a las preguntas anteriores.



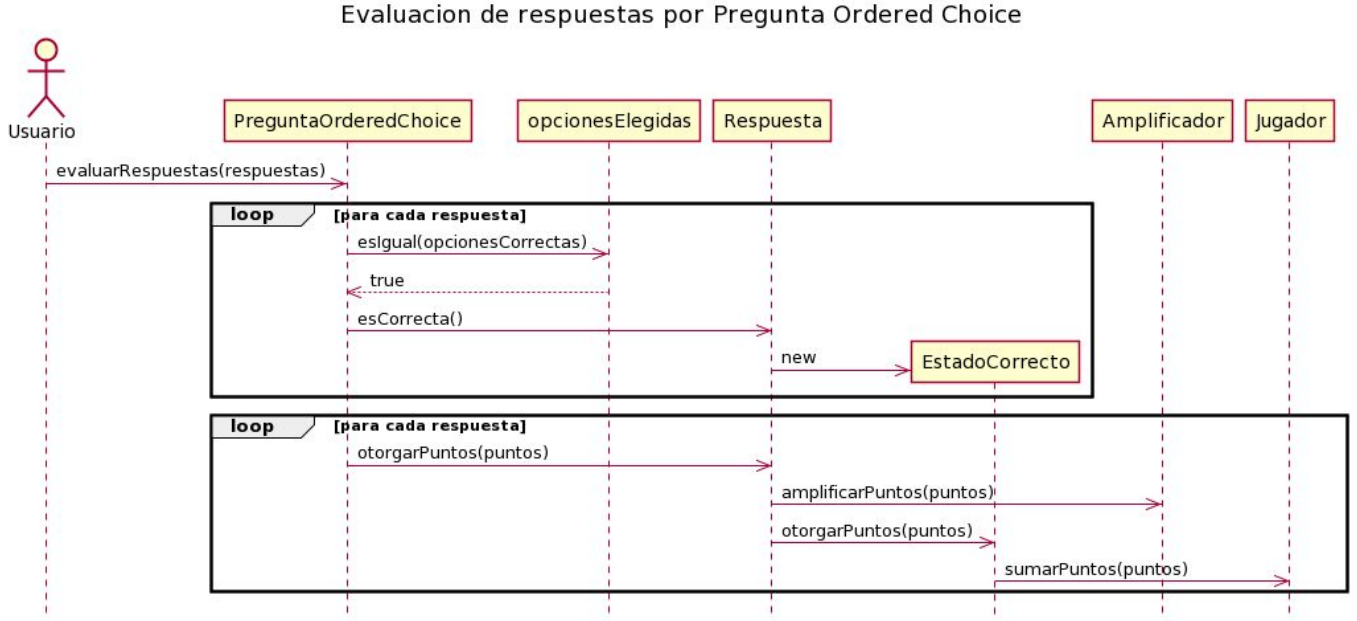
En cuarto lugar mostraremos el caso genérico de evaluación de respuestas multiple choice parcial, mostraremos también un caso en el que todas las respuestas son correctas. La primera parte de correción de las respuestas es muy parecida al diagrama anterior, salvo que Pregunta le pregunta a la colección de opciones recibida como respuesta si tiene elementos de las opciones incorrectas (o sea que si seleccionó alguna incorrecta). Si no los tiene, entonces le notifica a la respuesta que es correcta, y el funcionamiento es similar a la del diagrama anterior. Sin embargo, los puntos a otorgar se calculan sumando los puntos otorgados por cada opción. Una vez calculados se los manda a sumar al jugador de la misma manera que las secuencias anteriores.



En quinto lugar, mostraremos la evaluación de respuestas en Group Choice para un caso genérico donde las respuestas son correctas. En un primer loop se muestra la corrección de cada respuesta, donde para cada grupo (colección de opciones) se pregunta si tiene los mismos elementos que las correctas del grupo preguntado. Si alguno de los grupos es correctos pregunta le notifica a la respuesta que es correcta. Luego el funcionamiento es igual al de las preguntas anteriores, se suman los puntos al jugador.

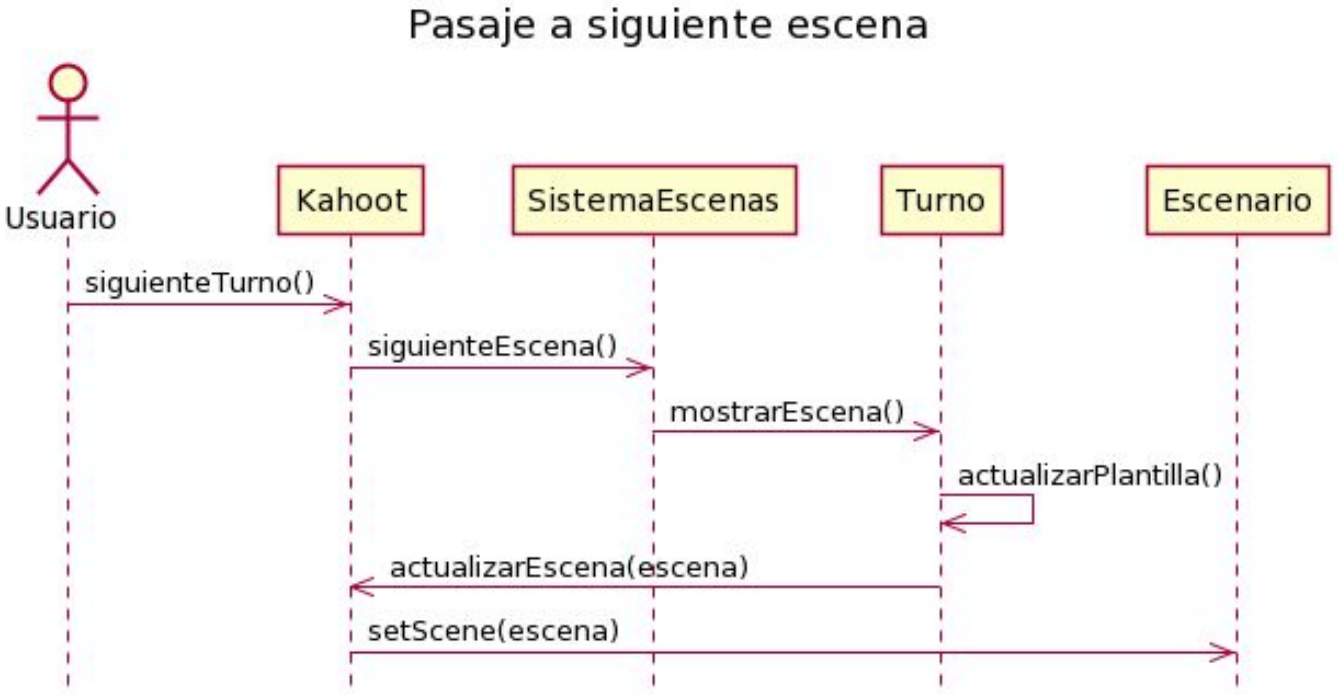


En sexto lugar mostraremos la evaluación en Ordered Choice para un caso genérico donde todas las respuestas son correctas. El funcionamiento es igual al de los diagramas anteriores salvo que la respuesta se considera correcta con el mensaje esIgual que envía la pregunta a la colección recibida. Si la colección recibida tiene las mismas opciones y ordenadas de la misma manera que la colección correcta, entonces la respuesta es correcta. Luego, al igual que las otras preguntas, se asignan los puntos al jugador.



A continuación mostraremos un breve diagrama de cómo funciona el sistema de escenas.

Kahoot entiende el mensaje siguienteTurno(), que lo que hace es establecer cómo escena la siguiente escena según corresponda por el sistema de turnos (o escenas). Para ello el sistema le envía el mensaje mostrarEscena() al siguiente turno que el sistema maneja. La escena (o turno) actualiza la plantilla *.fxml* según corresponda y le envía la escena (o la Scene) a Kahoot quien la establece en el stage.

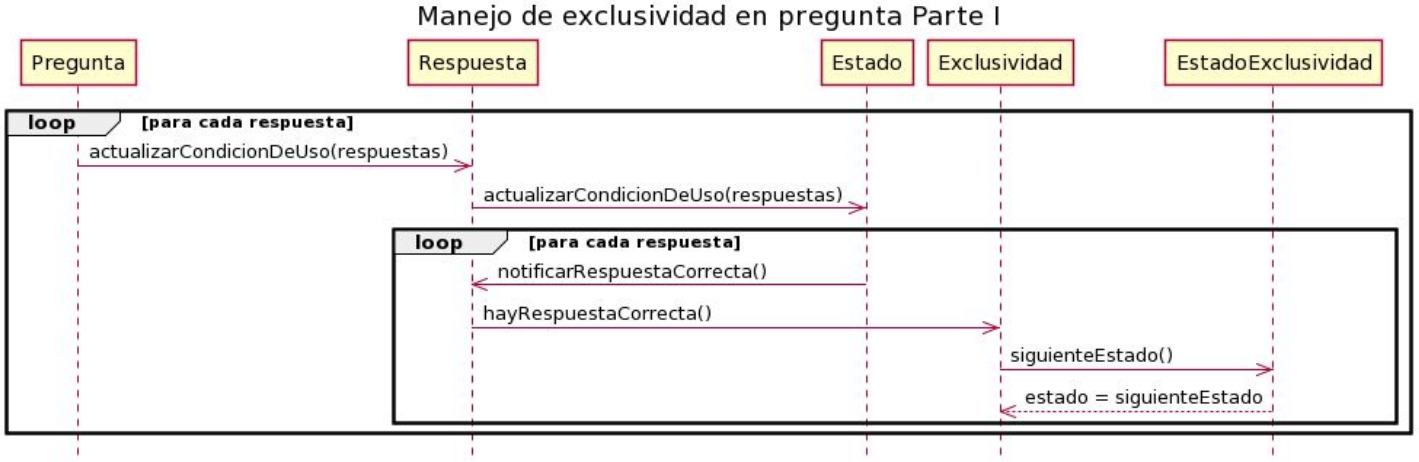


Ahora analizaremos el funcionamiento de la exclusividad, que si bien las preguntas anteriores podrían llegar a usarlas no se mostraron para no complejizar el diagrama. Siempre pregunta realiza la siguiente secuencia salvo que el uso del amplificador nacido depende de si las circunstancias lo ameritan. Los siguientes diagramas muestran el método prepararAmplificadorExclusividad que se llama cuando se evalúa la respuesta. Asumimos (al hacer el diagrama) que todas las respuestas usaron amplificador.

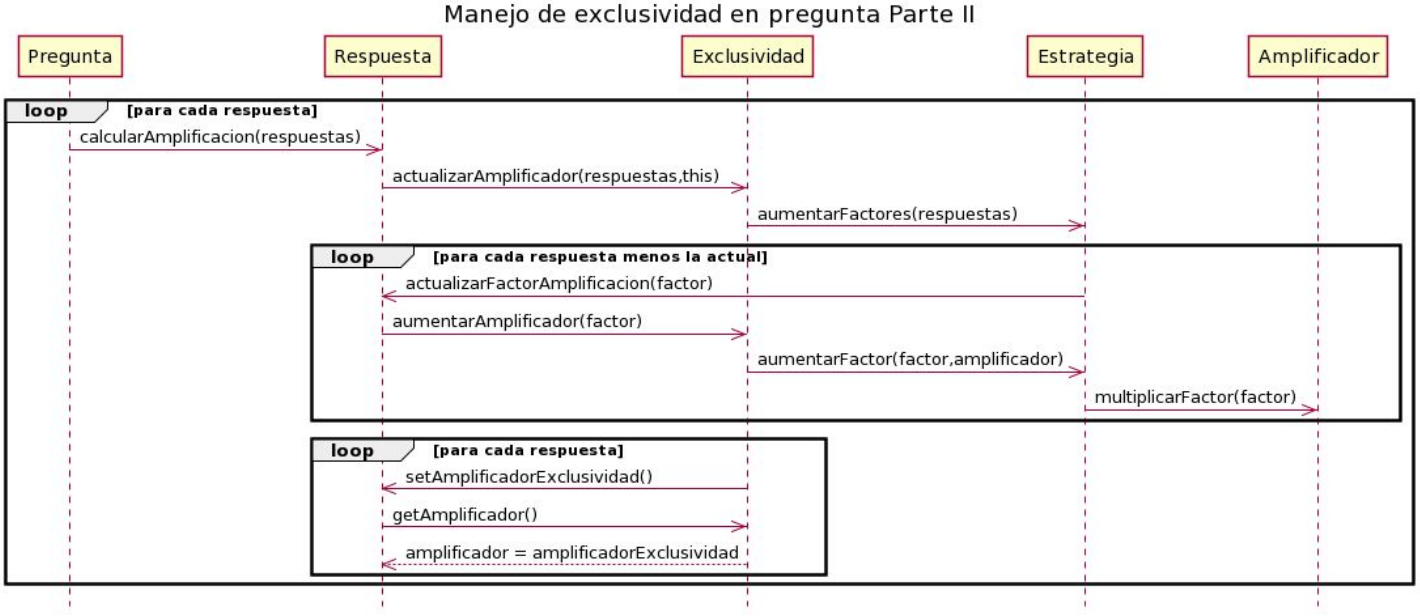
Aclaración: los nombres fueron recortados para ahorrar espacio.
Otra aclaración: el loop del primer diagrama continua al segundo, el del tercero es otro.

El primer diagrama muestra la primera parte de la secuencia luego de que se llama al método prepararAmplificadorExclusividad(respuestas). Para cada respuesta se actualiza la condición de uso, es decir, si hay una respuesta correcta, se le notifica a todas las demás exclusividades que pasan al siguiente estado de asignación. El estado de la respuesta es quien sabe si es correcta o no; si lo es, como en el caso

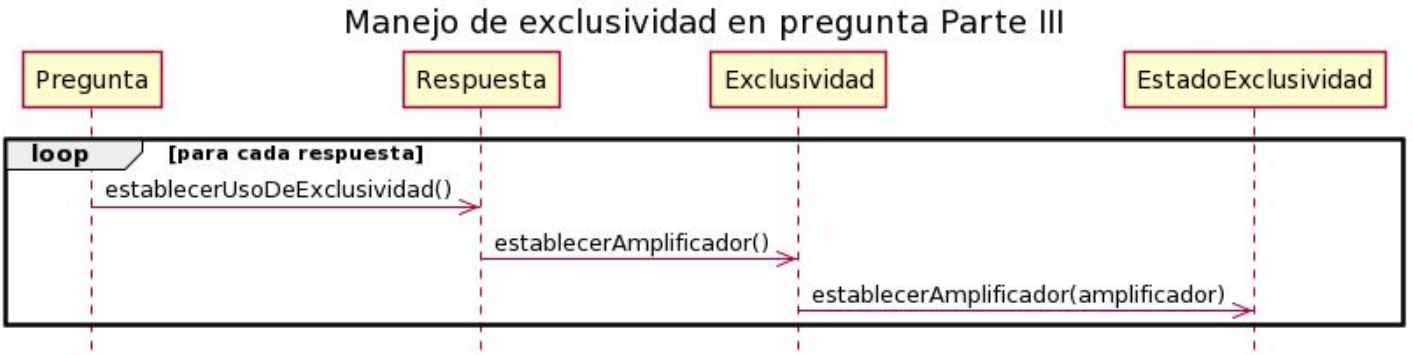
mostrado en el diagrama, entonces el estado le avisa a cada respuesta que es correcta y la exclusividad pasa al siguiente estado en la cadena de estados.



El segundo diagrama muestra cómo continúa el loop anterior. Ahora pregunta llama a cada respuesta para actualizar el factor de amplificador resultante de la exclusividad, que se le delega a la exclusividad y posteriormente a la estrategia quien es quien sabe como aumentar los amplificadores. Luego, a cada respuesta menos la que llamó a la exclusividad se le actualiza el factor de amplificación como la estrategia lo decida, es por esto que respuesta se lo delega. Por último también se le notifica a todas las respuesta que usen el amplificador de la exclusividad, ya que se encontró una que no es default.



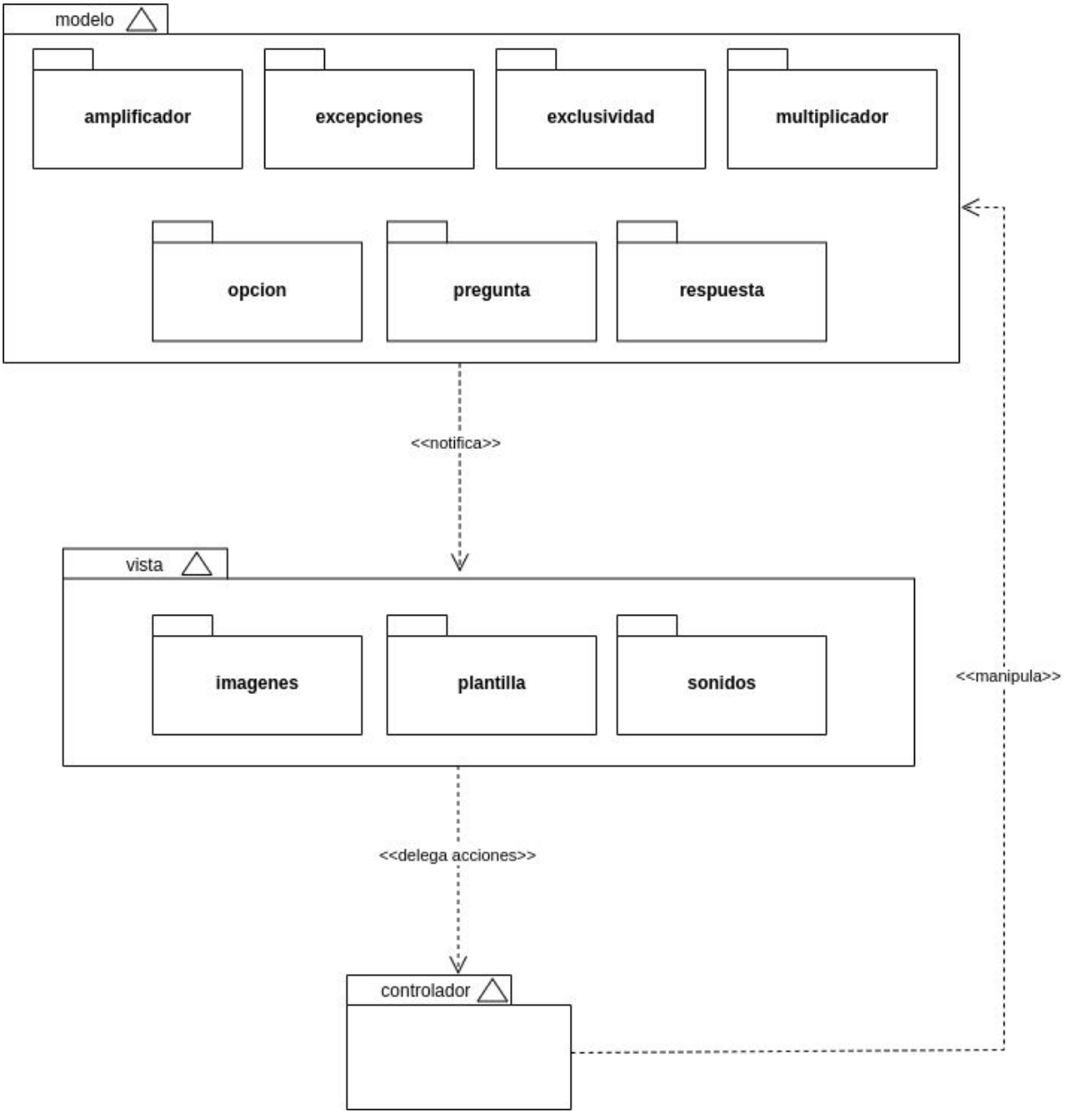
Para la última parte, tenemos un loop que lo que hace es pedirle a cada exclusividad que altere el amplificador según el estado final calculado en el primer diagrama de exclusividad, que depende de la cantidad de respuestas correctas existentes.



Por último, pudiésemos haber mostrado un diagrama para la evaluación de respuesta con un multiplicador, pero no es necesario debido a que es muy similar a la evaluación normal. Al usar un multiplicador, la respuesta usa su amplificador y una vez otorga los puntos multiplicados, lo inutiliza (se establece su factor en uno).

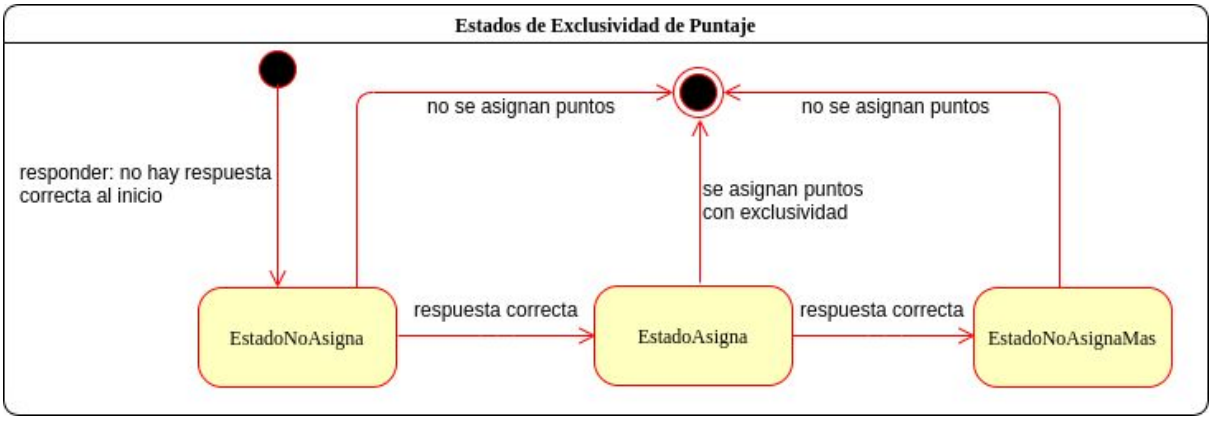
6. Diagrama de paquetes

Este diagrama muestra de qué manera se relacionan los paquetes principales del proyecto, haciéndose énfasis en el patrón MVC que se utilizó para el desarrollo. Además se ilustran los subpaquetes de dichos tres paquetes nodales que hacen a esta arquitectura de software.



7. Diagrama de Estados

Este diagrama ilustra el caso particular del estado en el que una ronda del juego se lleva a cabo, y al momento de responder, debe tenerse en cuenta que la Exclusividad de puntaje se aplicará sí y solo sí, exactamente uno de los dos jugadores responde correctamente, esto significa que, es necesario saber en qué estado se encuentran las respuestas de cada jugador para poder tener en cuenta este beneficio. Por ello, se tiene un estado “No asigna”, que implica que aún no se ha encontrado la respuesta correcta, un estado “Asigna” en donde hay una única respuesta correcta, por lo cual la Exclusividad sería tenida en cuenta. Sin embargo, también está el estado “No asigna más”, que entra en juego cuando más de un jugador respondió correctamente, por lo que los puntos no serán asignados a ninguno de los jugadores durante la ronda, y no hay vuelta atrás hasta no avanzar a la próxima pregunta.



8. Detalles de implementación

Esta sección ha estado y seguirá estando siendo abarcada en distintas secciones a lo largo del presente desarrollo teórico. Sin embargo, se consideró importante remarcar algunos ítems destacados de la implementación del trabajo realizado, haciendo hincapié en los patrones utilizados.

8.1 Exclusividad del Puntaje y el patrón State:

Como se ha explicado e ilustrado con su correspondiente diagrama de Estados en la sección anterior, se tuvo en cuenta que, para tomar una decisión sobre la aplicación o no de la Exclusividad de Puntaje, la colección de respuestas a una pregunta es susceptible a tener distintos estados: en primer lugar, como no se observa ninguna

respuesta correcta (todas son incorrectas hasta que se demuestre lo contrario), la asignación de exclusividad está en estado “No Asigna”, sin embargo, si se encuentra una respuesta correcta, pasa al estado “Asigna”. Se debió tener en cuenta que, este estado puede sufrir otra transición hacia “No Asigna Más”, esto pasa cuando se encuentra, además de la respuesta correcta inicial, otra respuesta, lo que va a implicar la imposibilidad de volver a asignar puntajes con el beneficio de la Exclusividad, por toda la ronda. Estos *estados* junto con sus transiciones, ilustran perfecto cómo se puede aprovechar el paradigma POO en lugar de utilizar booleanos con algún ciclo de búsqueda como se hubiera hecho en la Programación Estructurada.

8.2 Desarrollo basado en MVC.

Si bien es un patrón de arquitectura de software, y no un patrón de diseño, es importante recalcar el uso que se le ha dado. Por un lado, tenemos la Vista que contiene toda la interfaz del usuario, desde imágenes, sonidos hasta los propios archivos *.fxml* mediante los cuales se determina la visualización de las distintas pantallas que se van sucediendo a lo largo del juego. También es el punto de entrada para la ejecución del programa.

Por otro lado, tenemos el paquete Controlador, que, como corresponde, responde a las acciones de los usuarios e invoca peticiones al 'modelo' cuando esto pasa. Más concretamente, cada archivo *.fxml* tiene asociado un controlador, el cual tiene un comportamiento específico que es responder a los distintos eventos que genere el usuario. Además, cada controlador tiene los datos específicos de los jugadores que se cargan en cada pantalla del juego, para que luego la vista pueda mostrarlos. En el Controlador, se maneja tanto el Sistema de Turnos, como el Sistema de Escenas, que permite ir acoplando las distintas situaciones que suceden en el juego.

Tenemos también el modelo, que es donde se contiene toda la información con la cual el juego opera, y se gestionan los Puntos, los Multiplicadores, las Exclusividades, entre otras cosas. Todas estas peticiones las recibe por parte del controlador, para que posteriormente la Vista pueda ser notificada y mostrar las correspondientes actualizaciones.

8.3 Strategy

Para la forma en la que se van modificando los amplificadores de la exclusividad se utilizó un patrón strategy. Para ello creamos una estrategia de aumento de factor que lo que hace es saber cómo aumentar el factor de exclusividad una lista de respuestas que recibe y aumentar el factor de un amplificador según un factor que recibe. Ambos métodos están relacionados ya que cada respuesta cuando la estrategia se lo pide, aumenta su factor de amplificación de exclusividad utilizando el

método propio de la estrategia. Decidimos hacer una estrategia debido al hecho de que deja abierta la posibilidad de en un futuro utilizar una estrategia alternativa si se considera que la actual (decisión que tomamos al extender a n jugadores) no debería utilizarse. Además permite separar el comportamiento en otra clase y no sobrecargar la exclusividad.

8.4 Interfaz gráfica

Para realizar la interfaz gráfica se ha utilizado la aplicación Scene Builder. Mediante el mismo se realizaron las diversas plantillas con extensión FXML. Respetando el patrón MVC cada plantilla tiene asociado su controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos para adaptar los datos a medida que avanza el juego.

En cada plantilla el nombre de las preguntas y opciones se obtienen del archivo cuestionario.json donde la clase LecturaDeArchivo tiene la responsabilidad de realizar la lectura en el json y colocar los nombres en cada plantilla.

8.5. Clase Kahoot y sistema de turnos

La clase Kahoot es la encargada de instanciar a los jugadores con sus nombres, inicia la lectura del archivo delegando esa responsabilidad a la clase LecturaDeArchivo y delega el manejo de turnos a la clase SistemaTurnos.

La responsabilidad de manejar los turnos la tiene la clase SistemaTurnos. Por cada ronda de preguntas, es decir, cuando ambos jugadores contestaron la misma pregunta se mostrará una escena con un intervalo de cuatro segundos indicando el puntaje actual de cada jugador. Al finalizar el juego, cuando los jugadores hayan contestado todas las preguntas, se mostrará una escena indicando el puntaje de cada jugador y el ganador.

9. Excepciones

En caso de que ocurra una situación inesperada es necesario informar donde está ocurriendo esta situación. Las excepciones implementadas en el modelo son:

- MasDeCincoOpcionesException: se utiliza esta excepción para el caso que en que se ingresen más de cinco opciones a las preguntas multiple choice clásico, multiple choice con puntaje parcial, multiple choice con penalidad y ordered choice.

- MasDeSeisOpcionesException: se utiliza esta excepción para el caso que en que se ingresen más de seis opciones por grupo a la pregunta group choice.

- NoHayOpcionesException: se utiliza esta excepción para el caso que en que una pregunta no tenga ninguna opción.

- NoTieneBeneficioException: se utiliza esta excepción para el caso donde el jugador no tiene más beneficios y aún así quiera usar alguno de estos.

- ColeccionDeOpcionesNoTieneNombreException: se utiliza esta excepción cuando se pide el nombre de una colección de opciones la cual se creó sin nombre.