FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Kahoot

[7507/9502] Algoritmos y Programación III
Curso 2, grupo 7
Primer cuatrimestre de 2020

Corrector: Diego Sánchez

Fernández Manoukian, Tomás	100773	tfernandez@fi.uba.ar
Mombru, Melanie	103882	mmombru@fi.uba.ar
Mundani Vegega, Ezequiel	102312	emundani@fi.uba.ar
Tardá, Agustín	100932	atarda@fi.uba.ar
Tardá, Rocío	103883	rtarda@fi.uba.ar

Índice

1. Introducción

En este informe se muestra el proceso de desarrollo de una aplicación con su modelo de clases e interfaz gráfica de manera grupal, utilizando al lenguaje de tipado estático Java. Se utiliza con un diseño del modelo orientado a objetos y se trabaja con las técnicas de *Test Driven Development* e Integración Continua. Además se da la documentación en forma de texto, diagramas de clase, diagramas de secuencia, diagramas de paquetes y diagramas de estado.

2. Herramientas de *software* utilizadas

Se han utilizado los siguientes programas para la realización de esta aplicación:

- Java Development Kit 14.0.1
- JavaFX
- Intellij
- Maven 3.6.0
- Git y GitHub
- LaTeX en Overleaf
- Codecov
- Travis CI

3. Supuestos

Se han tomado los siguiente supuestos a partir de situaciones que se consideran no contempladas en la consigna:

- Un `Jugador` puede tener puntos negativos.
- La `Exclusividad` duplica solo cuando un jugador respondió no correctamente.
- Una `Respuesta` vacía no es incorrecta, pero tampoco es correcta.
- Las respuestas parcialmente correctas son consideras correctas al momento de aplicar exclusividad.
- Si un `jugador` no envíe una `respuesta` en las preguntas con penalidad no se suman ni restan puntos.
- No hay un límite de bonus por turno que se puedan utilizar.
- Cuando se acaba el tiempo se debe enviar la respuesta creada hasta el momento.
- No debe haber una `Pregunta` con alguna `Opcion` repetida.

4. Diagrama de clase

Se han realizado varios diagramas de clase en vez de uno grande con el fin de obtener una mayor legibilidad. No se han mostrado métodos como *getters* y *setters* (entre otros) por no ser de interés.

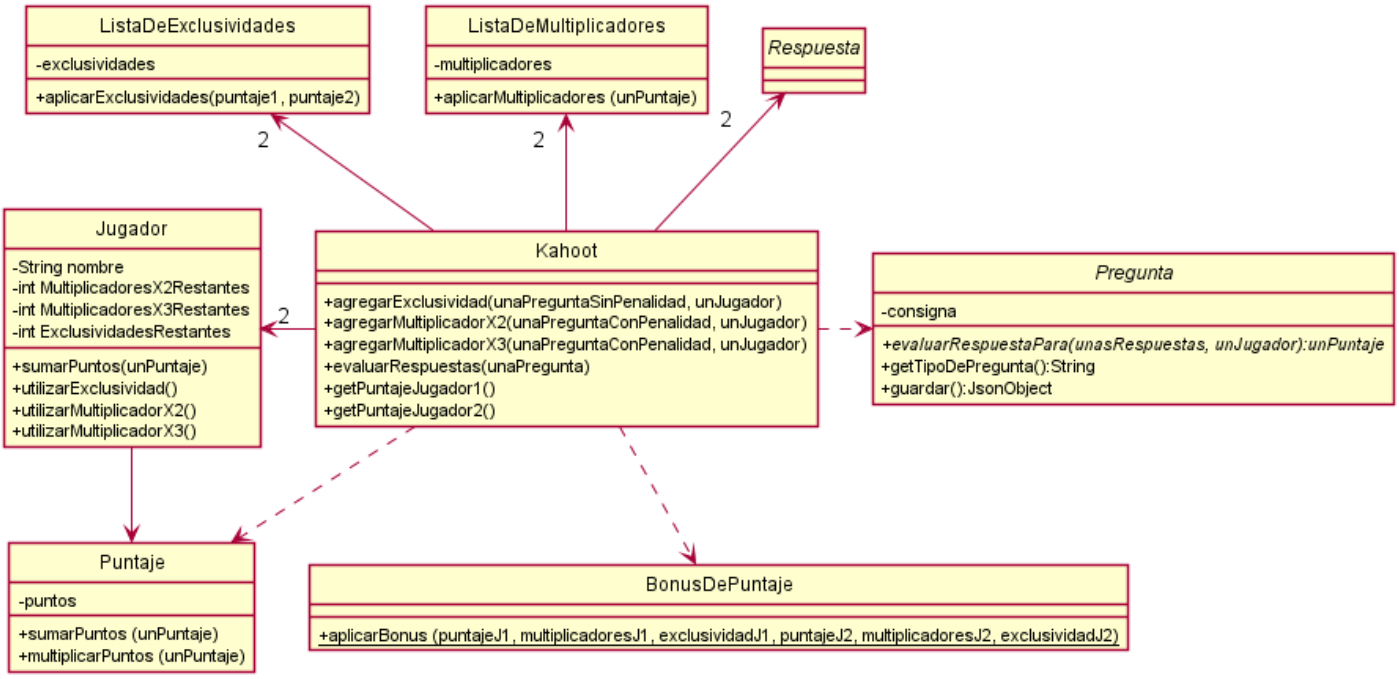

Figura 1: Diagrama de la clase `Kahoot`.

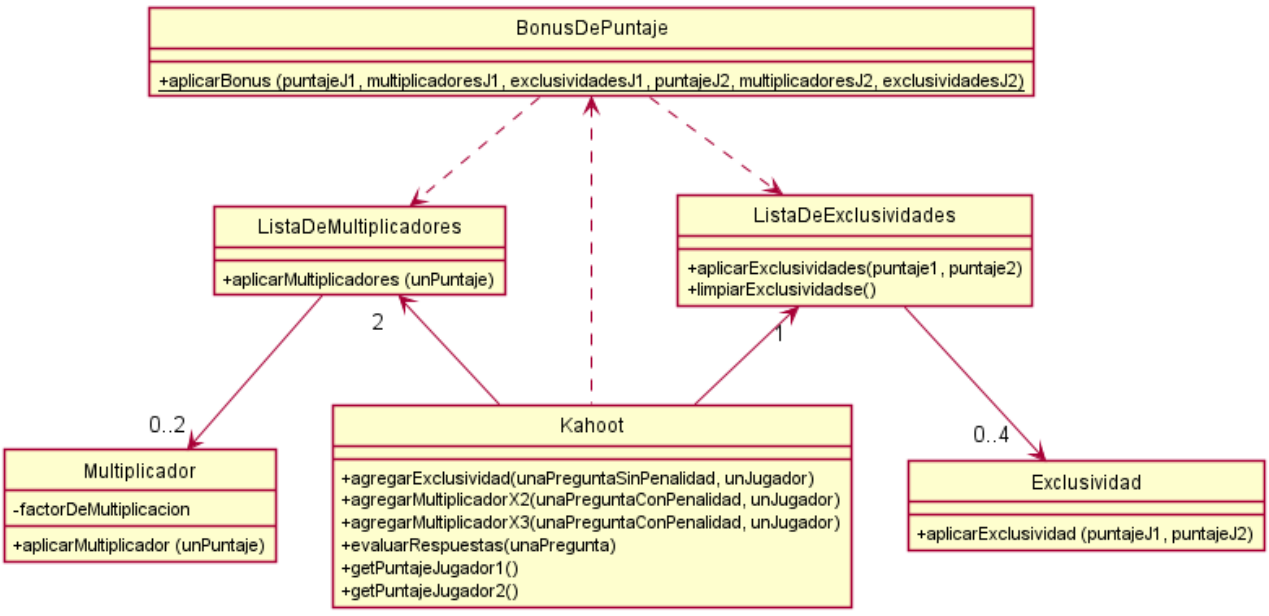

Figura 2: Diagrama de la clase `BonusDePuntaje`.

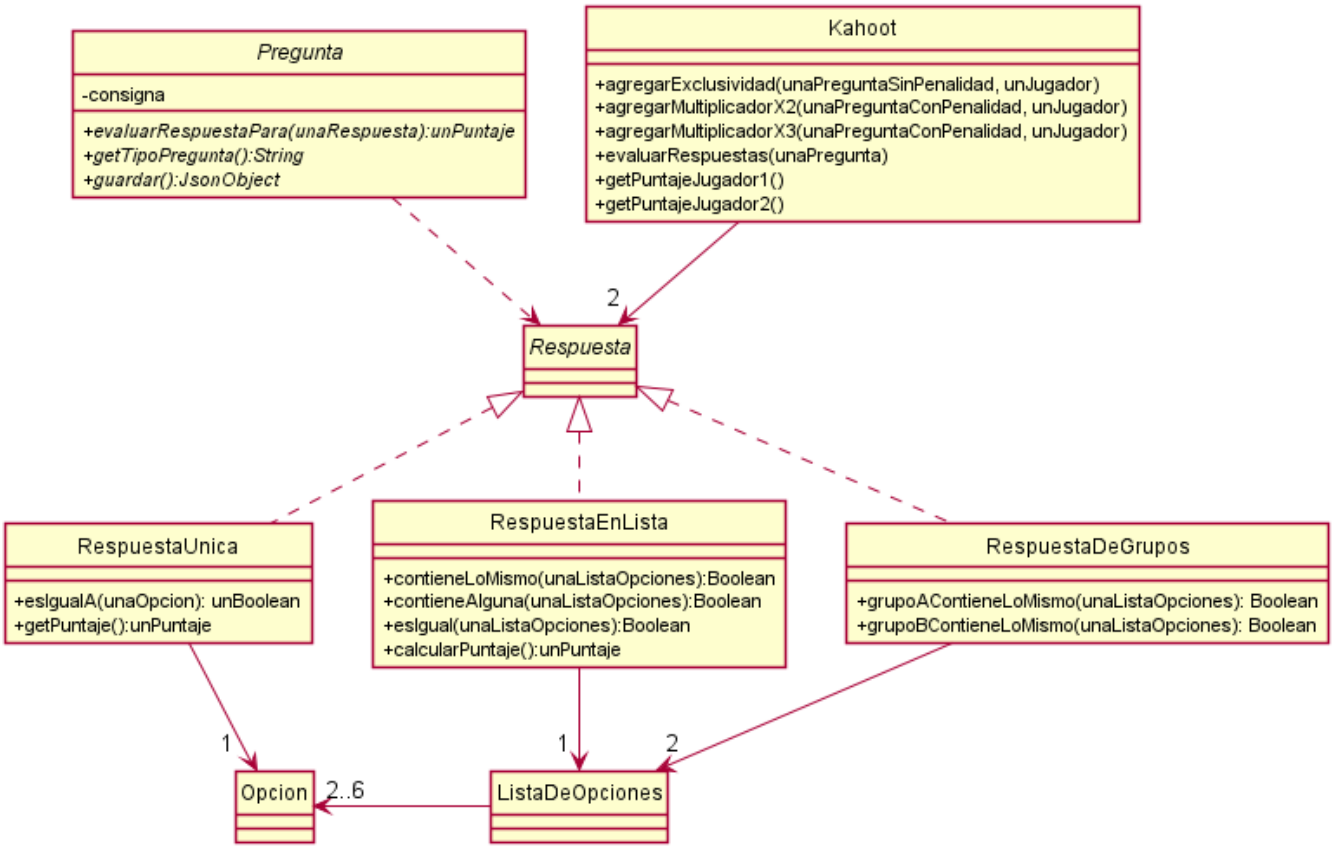

Figura 3: Diagrama de la interfaz `Respuesta`.

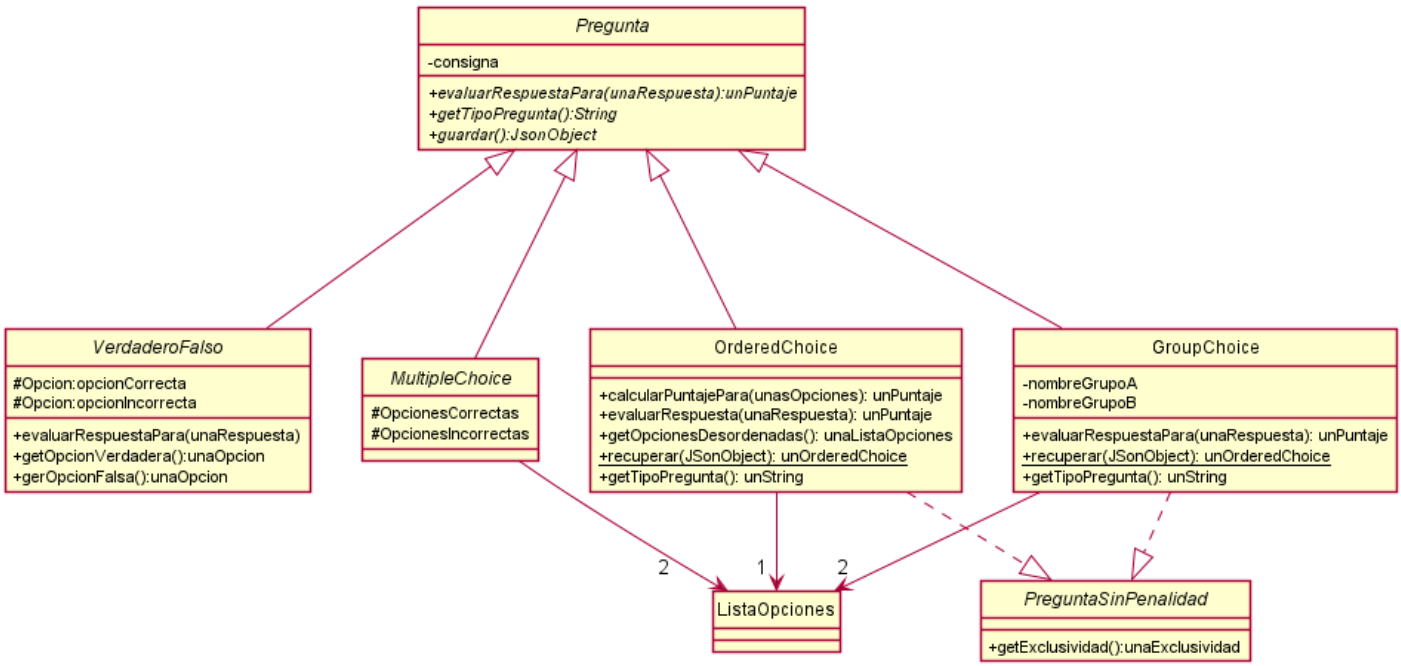

Figura 4: Diagrama de la clase abstracta `Pregunta`.

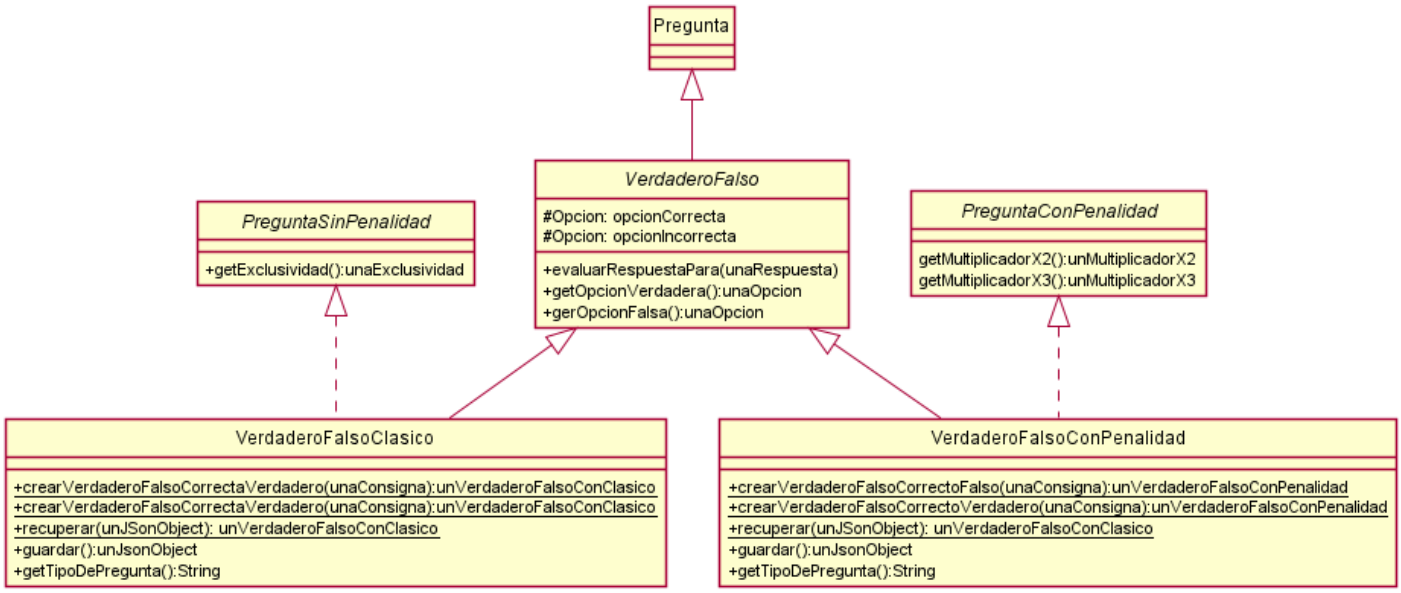

Figura 5: Diagrama de la clase abstracta `VerdaderoFalso`.

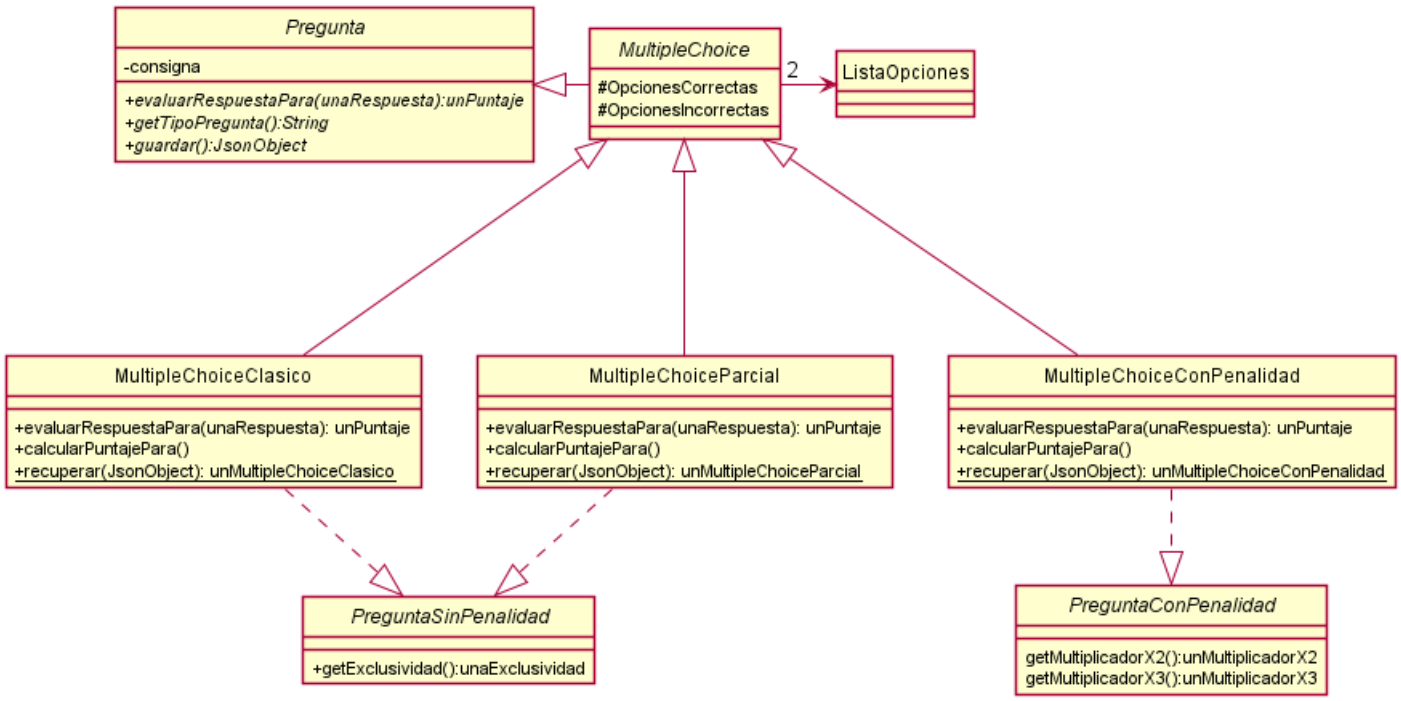

Figura 6: Diagrama de la clase abstracta `MultipleChoice`.

5. Detalles de implementación

Modelo

La clase `Kahoot` tiene almacenados a los dos objetos `Jugador` de la partida, a la `Respuesta` de cada uno, los multiplicadores y exclusividades utilizados en este turno. Se encarga de delegar la aplicación de los multiplicadores y exclusividades a `BonusDePuntaje` basándose en los atributos anteriores. La aplicación de los bonus se hace luego que ambos jugadores hayan respondido.

Cada `Jugador` sabe la cantidad de cada tipo de bonus restante que tiene, y al utilizar uno lo obtiene de la clase `PreguntaSinPenalidad` o `PreguntaConPenalidad` y se lo pasa a Kahoot, quien lo almacena.

La clase `BonusDePuntaje` se encarga únicamente de calcular el puntaje de cada jugador con los bonus aplicados. Los bonus se reciben en una lista, y en caso de haber más de uno se aplicarán de manera multiplicativa: por ejemplo, en caso que los dos jugadores hayan utilizado todas sus exclusividades en el mismo turno, y uno solo haya respondido correctamente, el la cantidad de puntos obtenida será multiplicada $\times 16$.

La clase `Opcion` funciona como contenedor, y cada opción tiene un nombre y un puntaje. Para las preguntas como `MultipleChoiceClasico`, `OrderedChoice` y `GroupChoice` donde el puntaje se calcula a partir de dos conjuntos, el puntaje no se utiliza. Pero para preguntas como `VerdaderoFalso`, `MultipleChoiceParcial` o `MultipleChoiceConPenalidad` donde el puntaje se calcula analizando cada opción en particular, el puntaje vale 1 si se trata de una opción correcta o $0/-1$ si se trata de una incorrecta, dependiendo si se trata de con o sin penalidad.

La interfaz `Respuesta` se creó para poder generalizar la utilización de respuestas en las vistas. No posee ningún método. `RespuestaUnica` la implementa y tiene una sola opción, y se encarga de obtener el puntaje de cada opción, es el tipo de `Repuseta` correspondiente a un `VerdaderoFalso`.
La clase `RespuestaEnLista` es la correspondiente a los `MultipleChoice` y al `OrderedChoice`. Tiene métodos que facilitan saber si la respuesta es correcta o incorrecta, como por ejemplo `ContieneAlguna()`, se utiliza para revisar que no haya ninguna respuesta incorrecta en los `MultipleChoiceParcial` y `MultipleChoiceClasico`; `contieneLoMismo()` se encarga de comparar que dos `ListaDeOpciones` tengan los mismos elementos pero no necesariamente en el mismo orden, y `esIgual()` que dos `ListaDeOpciones` sean iguales.
La clase `RespuestaDeGrupos` contiene dos `ListaDeOpciones` y es la respuesta a preguntas de `GroupChoice`. Dado a que siempre se trabaja con un Grupo A y Grupo B, no hace falta explicitar el nombre de cada grupo.

La clase `Pregunta` es una clase abstracta creada principalmente para poder manejar polimórficamente el método `getTipoPregunta()` y `evaluarRespuesta()`. Se optó por hacerla clase abstracta en vez de interfaz dado a que de esta manera se podía evitar repetir el atributo `consigna` con sus *getters* y *setters*. El método delegado `getTipoPregunta()` se utiliza para poder mostrar un nombre de la pregunta que no necesariamente sea el nombre de la clase, que será mostrado en las vistas.

Las interfaces `PreguntaSinPenalidad` y `PreguntaConPenalidad` son implementadas (una sola a la vez) por todas las clases que heredan de `Pregunta` que no son abstractas. Fueron creadas con el fin de que nuestro modelo muestre que las preguntas tienen `Exclusividad` o `Multiplicador`.

La clase abstracta `VerdaderoFalso` fue creada dado a que sus hijas "son un"`VerdaderoFalso` y además tiene atributos y métodos que son repetidos. La diferencias entre un `VerdaderoFalsoClasico` y `VerdaderoFalsoConPenalidad` son los tipos de bonus que acepta cada una, y y el puntaje correspondiente a la opción falsa (0 cuando es clásico, -1 cuando es con penalidad).

La clase abstracta `MultipleChoice` fue creada para poder manejar polimórficamente la creación de vistas, y se decidió que sea una clase y no una interfaz dado a que sus hijos cumplen con "es un" y además se podían evitar repetir los atributos. La diferencia principal entre cada hijo de `MultipleChoice` es cómo se calcula el puntaje de una respuesta, además de si pueden tener un `Multiplicador` o `Exclusividad`.

La clase `OrderedChoice` tiene un atributo `ListaDeOpciones`, en el cual no solo importa el contenido, sino también el orden. Se le agregó el método `getOpcionesDesordenadas()` para facilitar cómo se muestran las opciones en la vista.

La clase `GroupChoice` posee 2 atributos `ListaDeOpciones` y tiene un título para cada uno. Para evaluar si una respuesta es correcta se recibe una `Respuesta` que contiene dos listas, y se ve que ambas listas coincidan.

Flujo del programa

El programa comienza en el `Main` creando una instancia de `Kahoot`. Luego obtiene del `ManejadorDeArchivos` las preguntas a responder y las almacena en una pila. Después se instancia al `ManejadorDeTurnos`, la clase encargada de manejar el flujo del programa en base a las interacciones del usuario, y por último se muestra la interfaz gráfica. Finalmente el flujo del programa dependerá de las interacciones del usuario, pero será parecido al siguiente:

- Se muestra un menú de bienvenida donde cada jugador elige su nombre (de no hacerlo se le asigna un nombre predeterminado).
- Hasta que se acaben las preguntas:
  - Se muestra una escena donde se muestra de quién será el próximo turno.
  - Se muestra la pregunta al primer jugador.
  - Se muestra una escena donde se muestra de quién será el próximo turno.
  - Se muestra la pregunta al siguiente jugador.
- Se muestra una escena final.
- El usuario elige cuándo cerrar el programa.

Interfaz gráfica y controladores

Se han creado 4 clases de escenas, una para cada tipo genérico de `Pregunta` y no uno para cada `pregunta`, dado a que las diferencias entre estas son cómo se calcula el puntaje, algo ajeno a la interfaz gráfica. Las escenas almacenan propiedades que afectarán a los *layouts* por ejemplo las opciones a mostrar. De esta manera las opciones que ve el usuario son diferentes a las de la pregunta, y puede interactuar con ellas sin problemas. Cuando se reciben las opciones a mostrar por primera vez se las desordenan.

Se crearon contenedores para no tener que programarlos cada vez que se los necesita usar, como por ejemplo los contenedores de los botones y del primer renglón. También se creó una clase abstracta `Constantes`, la cual tiene únicamente atributos `static` y `final` en la cual se almacenaron constantes de colores, estilos y dimensiones para poder estandarizar estos parámetros. El `ContenedorConsigna` además de mostrar la consigna de la pregunta, muestra qué tipo de pregunta es.

Se diferenció entre los layouts del primer renglón con penalidad y sin penalidad, dado a que no tienen que habilitar y dar funcionalidad a los botones de los bonus. A pesar de que para ciertas preguntas algunos bonus nunca serán usados, se decidió deshabilitarlos pero mostrarlos basándonos en cómo maneja IntelliJ al botón "Stop".

Del mismo modo, se diferenció entre los layouts de `VerdaderoFalso` y `MultipleChoice` con y sin penalidad por lo ya dicho. Para evitar tener que repetir largas secciones de código como por ejemplo la obtención del contenedor de opciones se decidió crear las clases abstractas `LayoutVerdaderoFalso` y `LayoutMultiplechoice`.

Se agregó una barra de tareas en la cual se puede salir del programa, maximizar y minimizar tamaño. Además se puede acceder a más información del programa, mostrando la licencia bajo la cual se hizo, cuándo fue creado y sus creadores.

El `LayoutMenuBienvenida` permite elegir a los jugadores sus nombres, mostrando en el fondo el nombre predeterminado en caso de que no pongan ninguno. El `LayoutSiguienteJugador` a pesar de parecer bastante vacío fue creado para evitar que apenas responda un jugador empiece a correr el tiempo del siguiente. Ambos *layouts* activarán sus botones "Siguiente" en caso de presionar Enter.

Para los diferentes *layouts* de preguntas y siguiente jugador se utilizaron paneles vacíos con un tamaño determinado, de tal manea que la estructura de la vista sea consistente, y la posición de los botones no varíe cuando se cambie el tamaño de la ventana. Se consideró usar *BorderPanes*, pero en caso de maximizar los botones quedaban muy lejos de las opciones, esta solución no era cómoda.

El *layout* de `verdaderoFalso` particularmente tiene dos botones, y en base a cuál se seleccione se envía una `RespuestaUnica`. En caos de que se acabe el tiempo y no se selecciones ninguna se envía una respuesta vacía, de tal manera de que si se está en un `VerdaderoFalsoConPenalidad` no se resten puntos.

El *layout* de `MultipleChoice` tiene un `CheckBox` por cada opción y un botón `Enviar`. En caso de acabarse el tiempo, se arma la respuesta en base los `Checkboxes` seleccionados y se envía como una `RespuestaEnLista`; pudiendo restar puntos en caso de tratarse de `MultipleChoiceConPenalidad`.

El *layout* de `OrderedChoice` tiene varias opciones, cada una con dos botones, uno de subir y otro de bajar. Las opciones están guardadas en una lista `OpcionesMostradas`. Los botones de los extremos tienen alguno deshabilitado para evitar que la opción se vaya de rango. Cada vez que se presiona al botón subir de una opción, se la intercambia con la opción anterior, de tal manera que cada vez que se muestre la vista ese sea el orden actual de la lista a mostrar. Cuando se presiona `Enviar` o se acaba el tiempo, se envía la lista ordenada en una `RespuestaEnLista`.

El *layout* de `GroupChoice` tiene varias opciones, cada una con dos `RadioButton`, en el cual se puede elegir el grupo al cual se asigna. A medida que se van usando los `RadioButton`, se agregan o mueven las opciones a alguna de las dos listas con las cuales se enviará la `RespuestaDeGrupos`. En caso de terminarse el tiempo se envía la respuesta en el estado actual.

Por último, el *layoutLayoutFinDelJuego* muestra los puntos con los que terminó cada jugador y al ganador. En caso de no haber ganador se muestra que hubo un empate. Este *layout* no tiene botones, así que lo que le queda al usuario es salir del programa.

6. Excepciones

`CantidadDeOpcionesInvalidaExeption:`
Es lanzada cuando se intenta ingresar una cantidad inválida de opciones.

`ErrorDeEscrituraExeption:`
Es lanzada cuando sucede un error en la escritura del archivo de preguntas.

`ErrorDeLecturaExeption:`
Es lanzada cuando sucede un error en la lectura del archivo de preguntas.

`ErrorSinBonusesExeption:`
Es lanzada cuando se intenta utilizar un bonus y el jugador ya no posee ninguno de ese tipo.

`JugadorNoValidoExeption:`
Es lanzada cuando se intenta utilizar un jugador que no coincide con los dos jugadores originales.

`JugadorSinNombreExeption:`
Es lanzada cuando se intenta crear un jugador con un `String` vacío como nombre.

`OpcionesRepetidasExeption:`
Es lanzada cuando hay una opción repetida en alguna de las preguntas.

`OpcionNoValidaExeption:`
Es lanzada cuando se intenta utilizar una opción que no pertenece a la pregunta con la que se está trabajando.

`PreguntaCorruptaExeption:`
Es lanzada cuando se intenta crear una `Escena` con una pregunta que no coincide con las presentes en el modelo actual.

`RespuestaNoValidaExeption:`
Es lazada cuando se intenta utilizar una respuesta no válida para la pregunta con la que se está trabajando.

7. Diagramas de secuencia

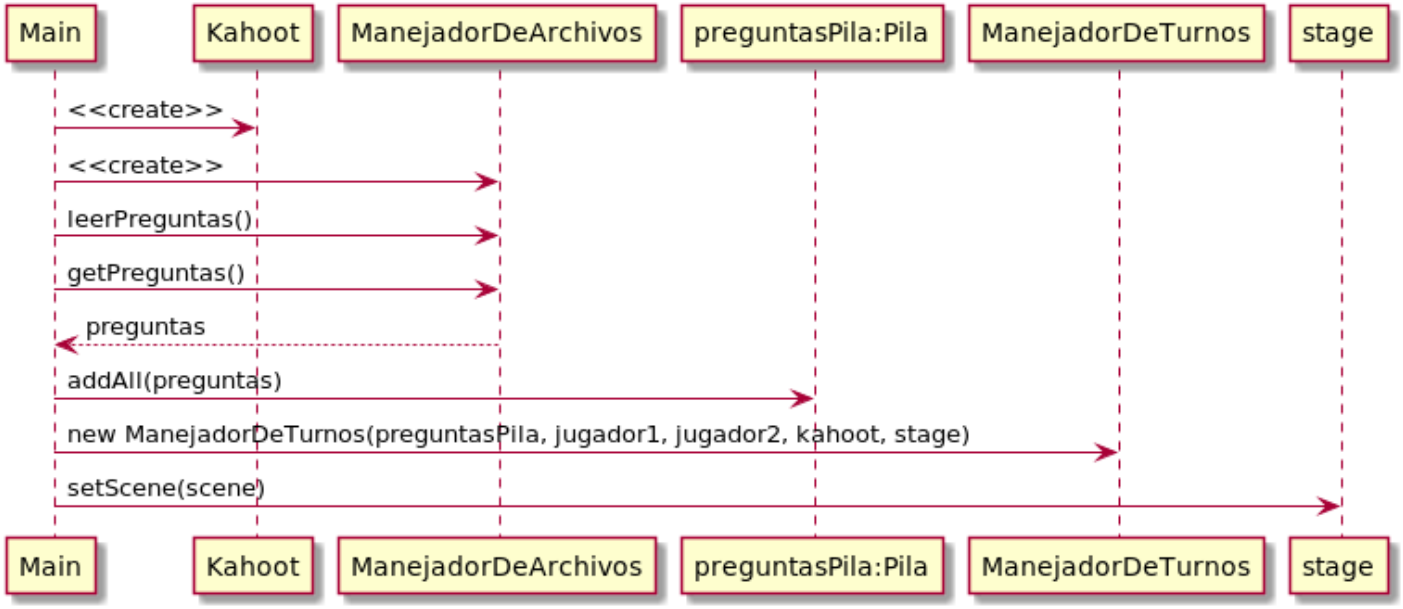

Figura 7: Diagrama de secuencia del funcionamiento del `Main`.

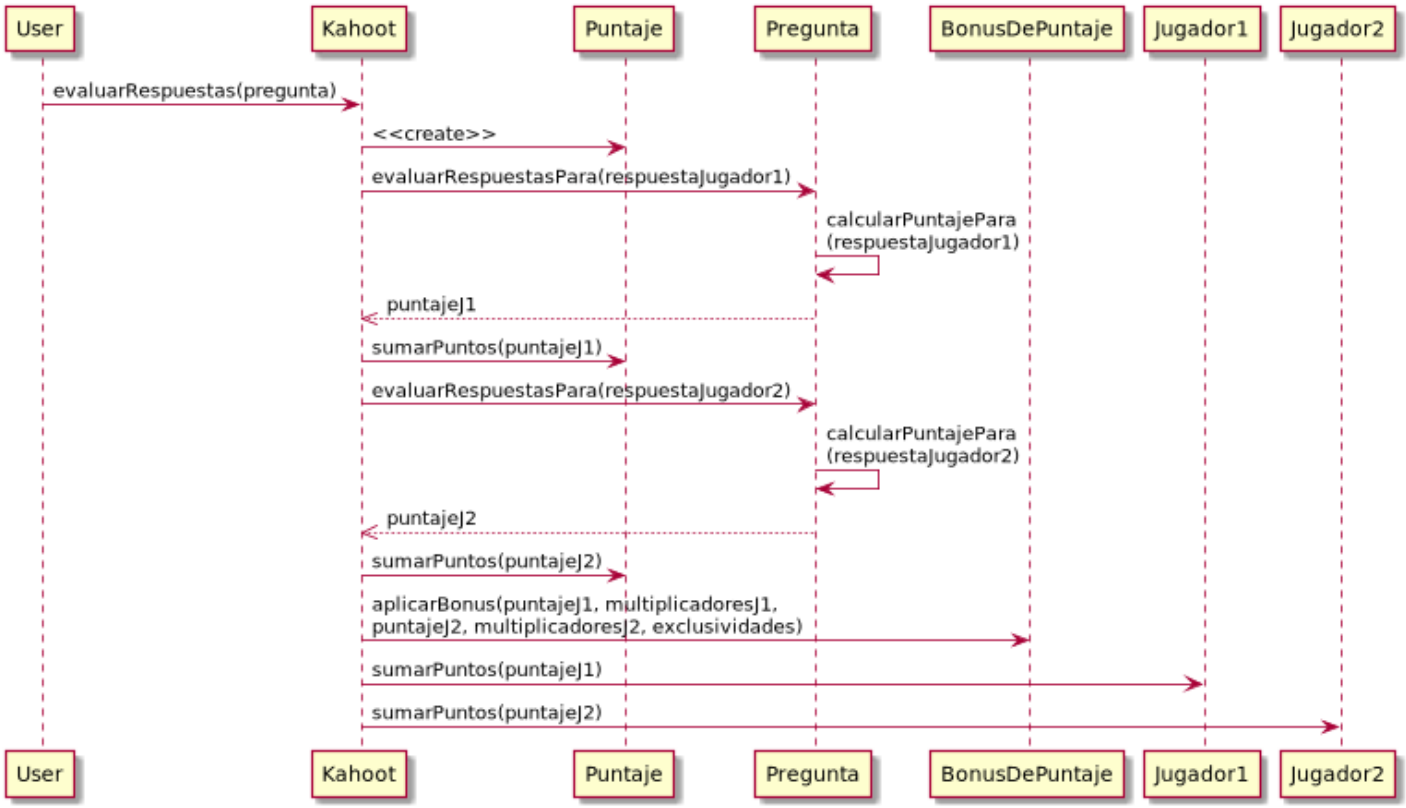

Figura 8: Diagrama de secuencia de como se evalúan las respuestas dadas por los jugadores ante cualquier pregunta.

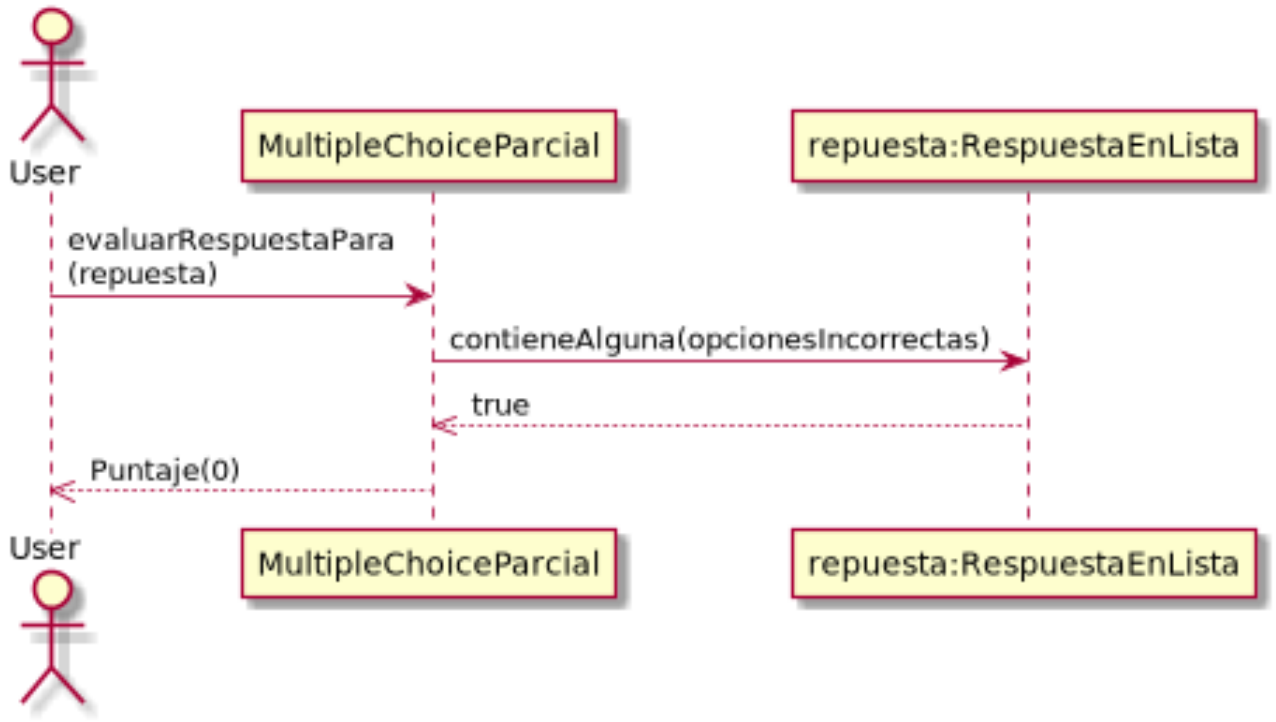

Figura 9: Diagrama de secuencia de como se evalúan las respuestas dadas por los jugadores ante una pregunta de `MultipleChoiceParcial` en un caso `True`.

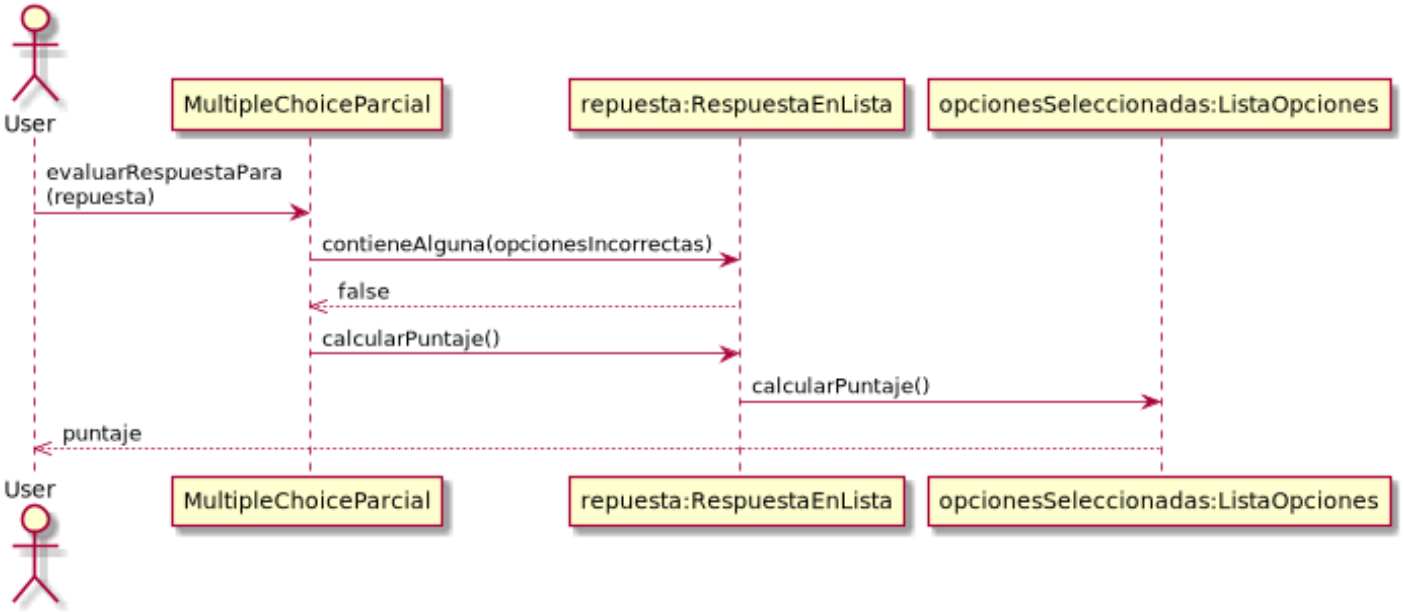

Figura 10: Diagrama de secuencia de como se evalúan las respuestas dadas por los jugadores ante una pregunta de `MultipleChoiceParcial` en un caso `False`.

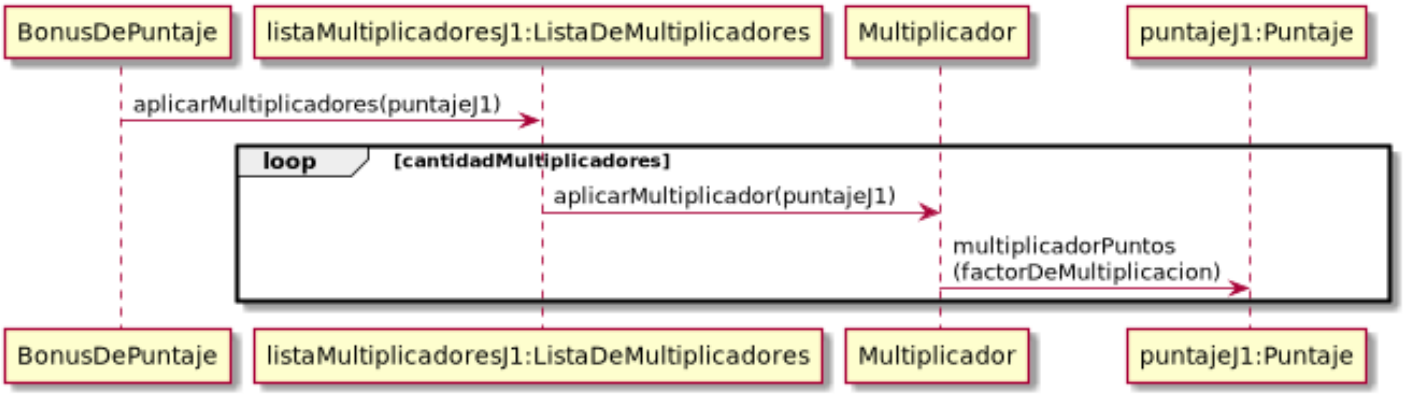

Figura 11: Diagrama de secuencia de la aplicación de `Multiplicadores`.

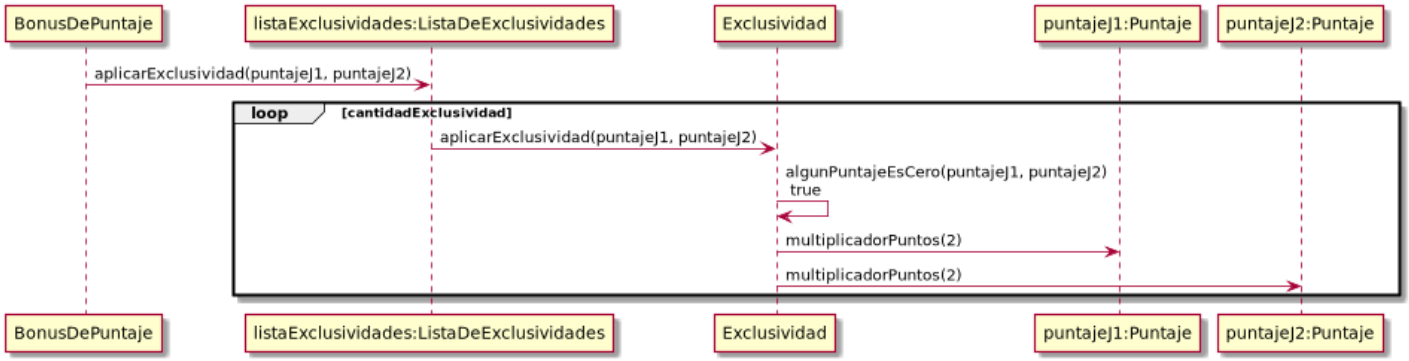

Figura 12: Diagrama de secuencia de la aplicación de `Exclusividades` en una caso `True`.

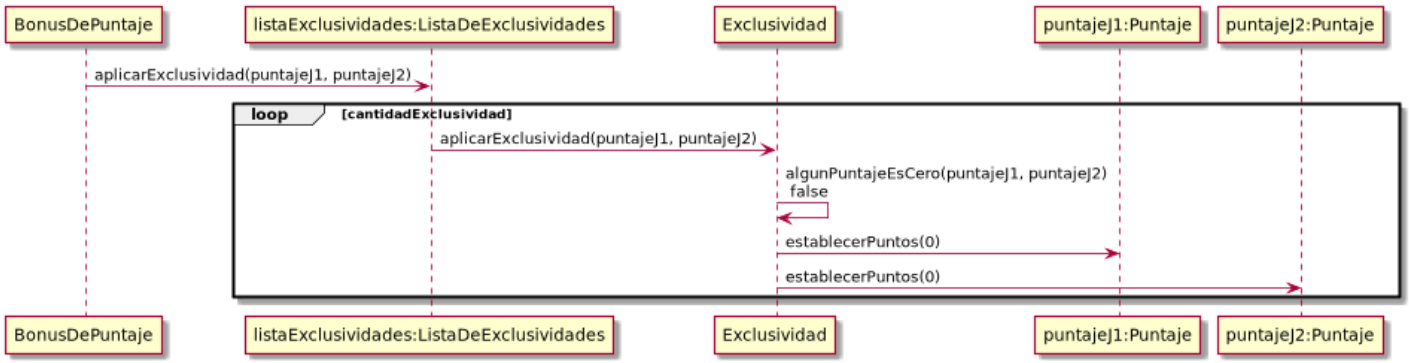

Figura 13: Diagrama de secuencia de la aplicación de `Exclusividades` en una caso `False`.

8. Diagramas de estados

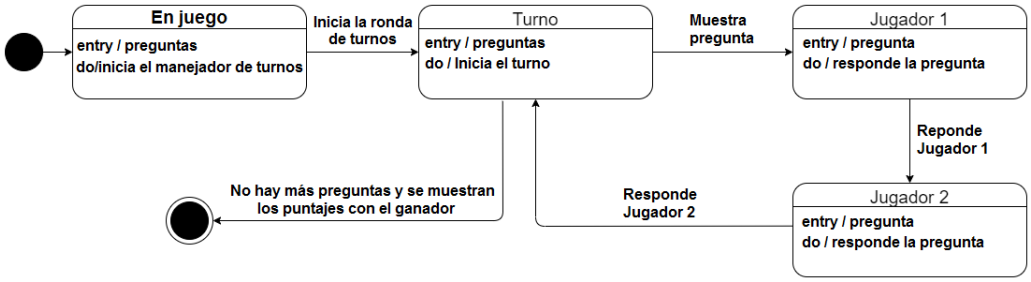

Figura 14: Diagrama de estados para una ejecución del programa

9. Diagramas de paquetes

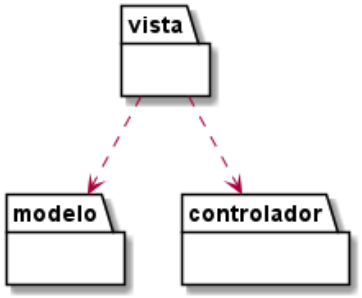

Figura 15: Diagrama de paquetes mostrando la relación entre vista, modelo y controlador

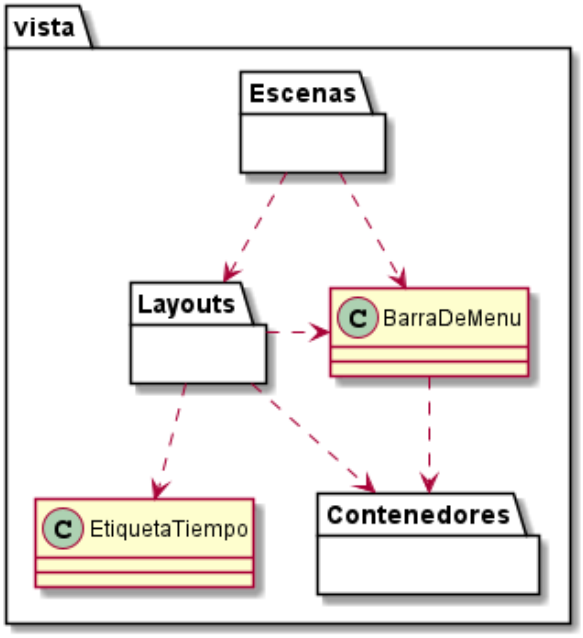

Figura 16: Diagrama de paquetes mostrando las relaciones dentro del paquete vista

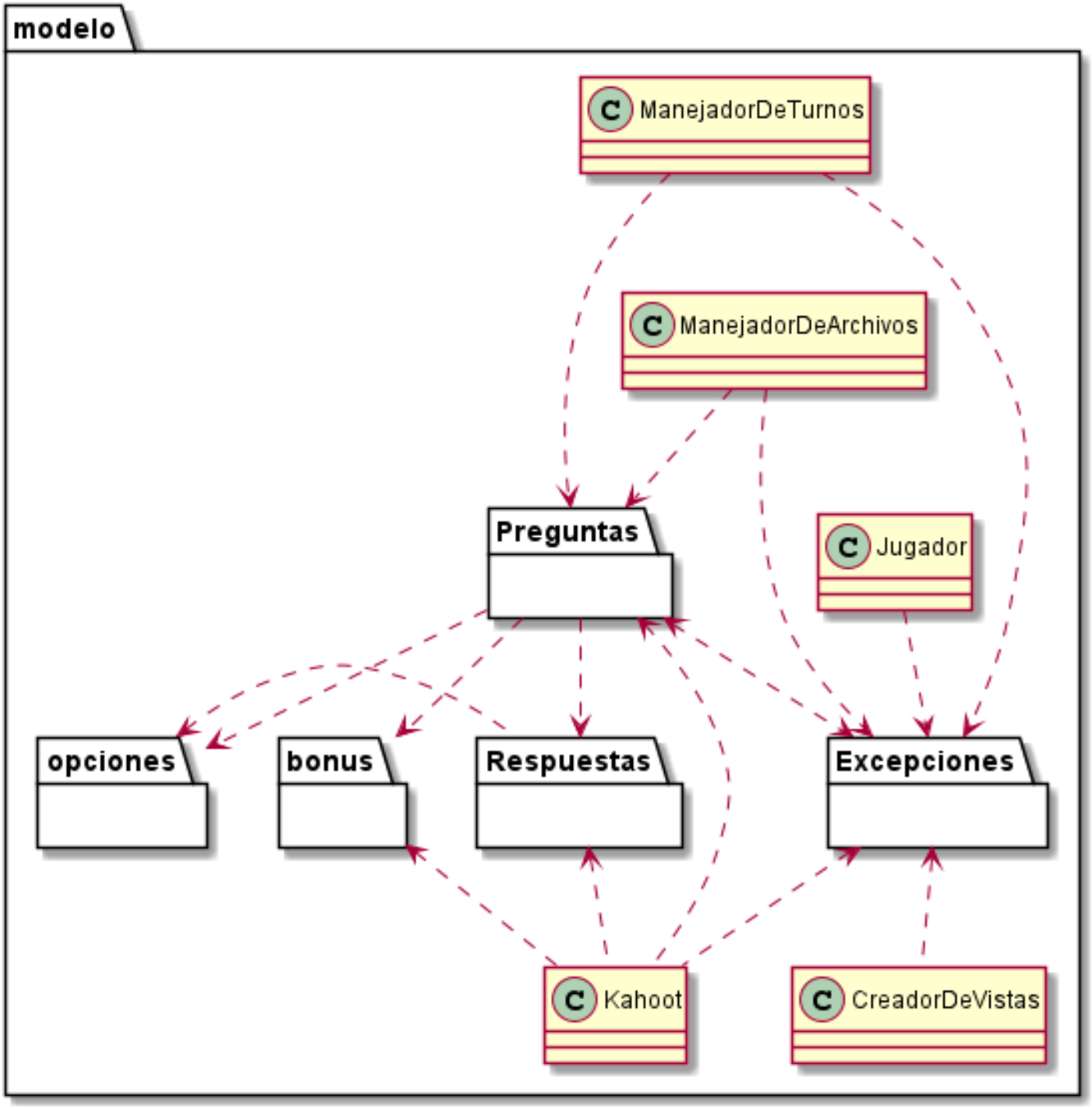

Figura 17: Diagrama de paquetes mostrando las relaciones dentro del paquete modelo