Universidad de Buenos Aires
Facultad De Ingeniería
Año 2020- 1er Cuatrimestre

Algoritmos y programación III (95.02)

(Curso 2)

Trabajo práctico N°2
Tema: Cajú
Fecha: 20 de agosto de 2020
Alumnos:

Chaparro, Raúl Antonio <rchaparro@fi.uba.ar>	- #96222
De Vita, Stefano Mauro <sdevita@fi.uba.ar>	- #104462
Grzegorczyk, Iván <igrzegorczyk@fi.uba.ar>	- #104084
Tarzia, Luciana Vanina < ltarzia@fi.uba.ar>	- #100662
Vazquez, Francisco Manuel <igrzegorczyk@fi.uba.ar>	- #104128

Índice

1. Introducción

El siguiente trabajo tiene como objetivo desarrollar en Java un juego tipo *quiz* para dos jugadores, con seis tipos de preguntas. El mismo deberá implementar una interfaz gráfica intuitiva que le permita a cada jugador seleccionar la opción u opciones que crea correctas, poder elegir el orden de las opciones o agruparlas en grupos según corresponda. El objetivo que se busca es poder aplicar los conceptos vistos sobre el paradigma de objetos, UX y calidad de *software*.

2. Supuestos

Para el desarrollo de la aplicación se tuvieron que analizar distintos casos borde y a partir de eso poder definir algunos supuestos en el flujo de la aplicación:

- En la clase Panel, para usar los métodos `usarTriplicador` y `usarDuplicador` se debe haber creado anteriormente una instancia de Jugador y una instancia de Pregunta.
- En el caso en que el jugador no responda una vez acabado el tiempo en el cual debía responder la pregunta, su respuesta será un conjunto vacío.
- En la pregunta del tipo `Multiple Choice con Puntaje Parcial` y en el caso que se utilice la exclusividad, se multiplicará el puntaje del jugador únicamente si uno de los jugadores responde mal. De lo contrario, si responde parcialmente bien o completamente bien, no se contarán los puntos para ninguno de los dos jugadores.
- En todos los tipos de pregunta debe existir por lo menos una respuesta correcta.

3. Diagramas de clases

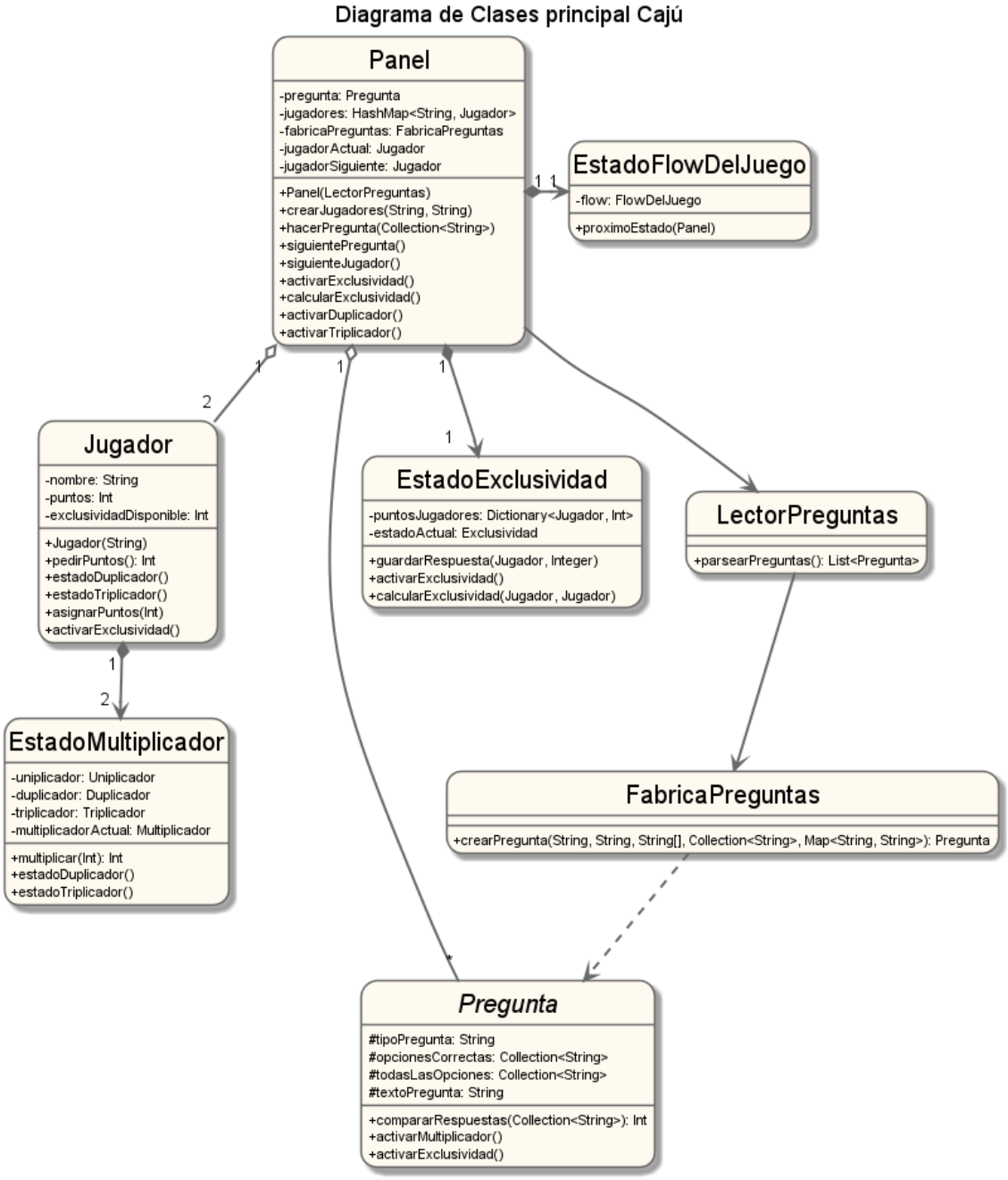

Figura 1: Diagrama de clase.

Como se puede ver en el diagrama de la figura 1, hay una clase donde se canaliza el flujo del programa. Esta clase es `Panel`. En ella se crean los jugadores y se mantiene una referencia al jugador actual y al siguiente. Se le indica a los jugadores contestar la pregunta y asignarse los puntos. Además se guarda una lista de todas las preguntas del tipo Pregunta, que son creadas

mediante una fábrica de preguntas.

Para la creación de las preguntas se recurrió a la clase `FabricaDePreguntas` que de acuerdo al `String` que recibe como parámetro sabe que tipo de pregunta crear. De allí se implementa una clase por tipo de pregunta, las cuales heredan de la clase abstracta `Pregunta`. Al mismo tiempo, las preguntas implementan un comportamiento en específico, según que tipo de pregunta sea. Los comportamientos están implementados mediante una interfaz. Los mismos son: `Comportamiento-Clasico`, `ComportamientoConPenalidad` y `ComportamientoConPuntajeParcial`.

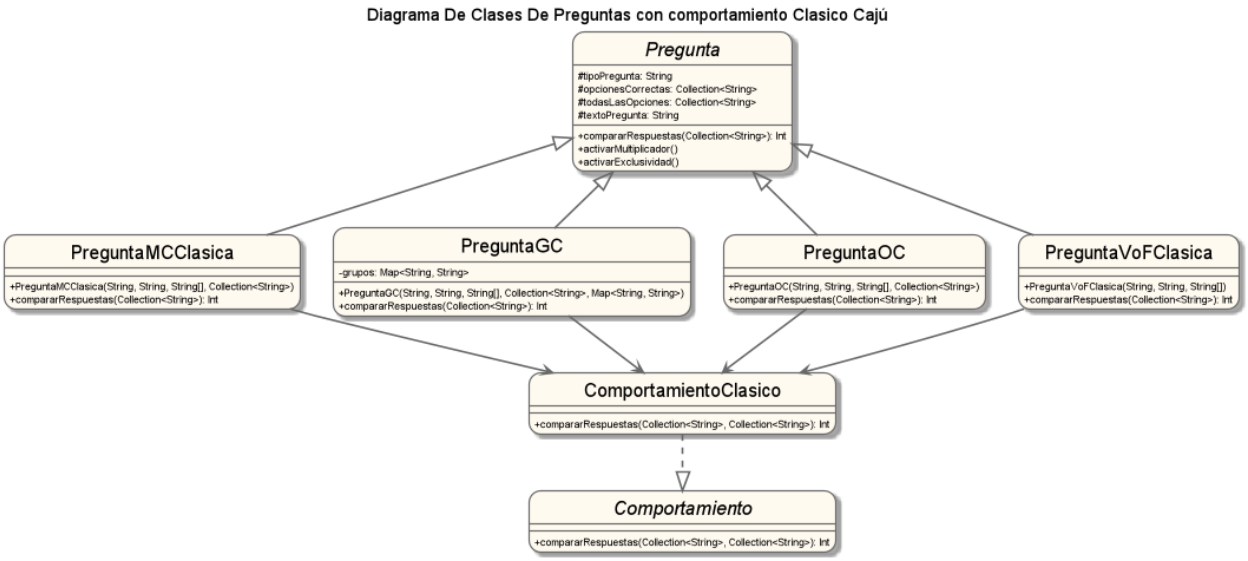

Figura 2: Diagrama de clases de Preguntas con comportamiento Clásico.

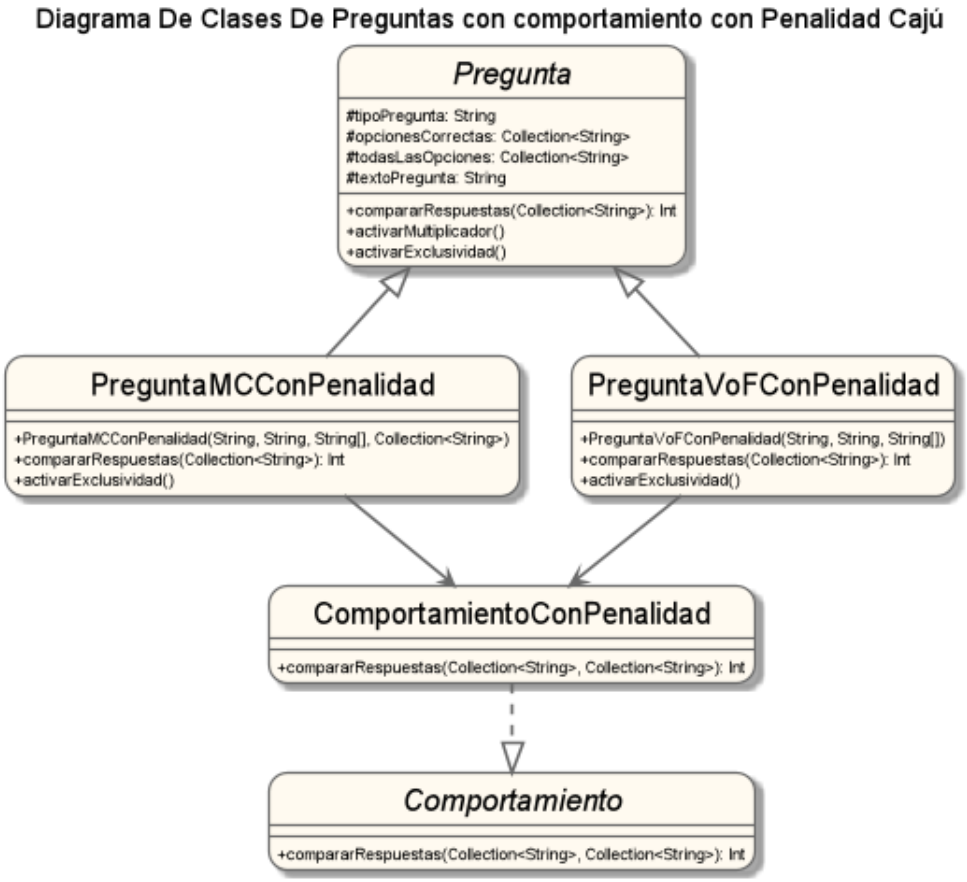

Figura 3: Diagrama de clases de Preguntas con comportamiento con Penalidad.

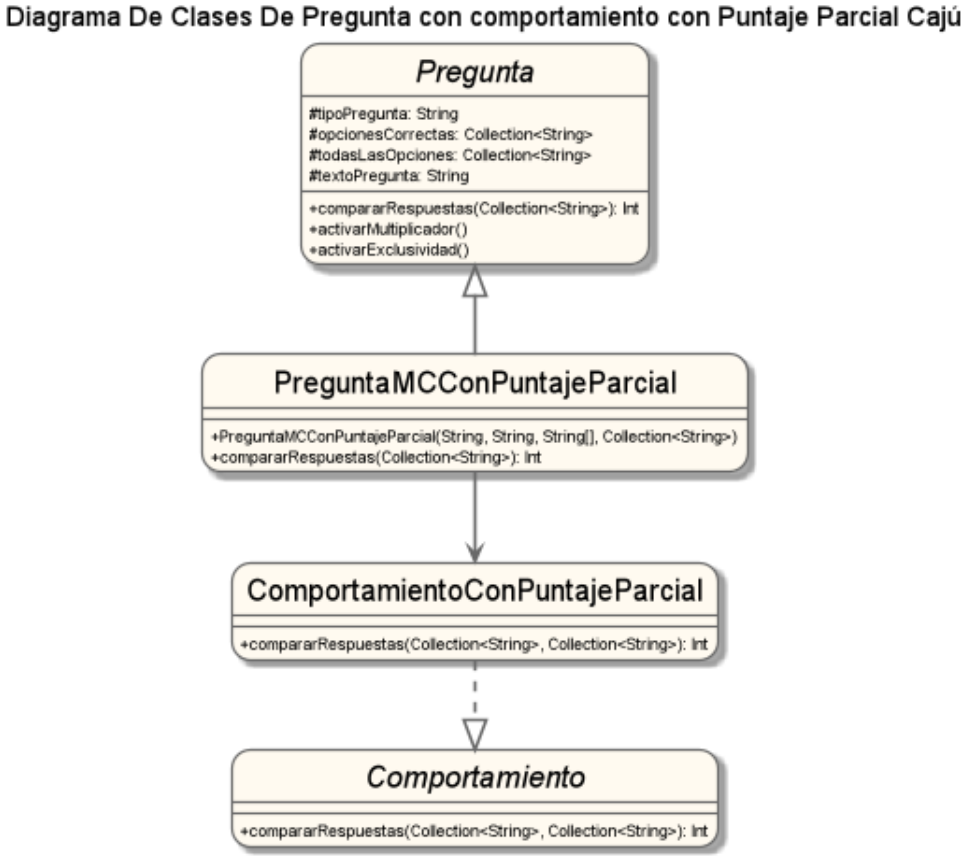

Figura 4: Diagrama de clases de Preguntas con comportamiento con Puntaje Parcial.

La responsabilidad más importante que tiene la clase `Pregunta` es la comparación entre las respuestas elegidas por el jugador y las respuestas correctas. El `Comportamiento` se encarga de calcular los puntos según como se haya respondido.

La clase `Jugador` tiene la responsabilidad de saber si tiene disponibles sus exclusividades y de asignarse los puntos calculados por `Pregunta`.

Se tienen tres máquinas de estados para implementar las *exclusividades*, los *multiplicadores* y el *flujo del juego* (turnos y rondas). Por un lado, se creó la interfaz `Exclusividad`. Por el otro, la interfaz `Multiplicador` y por último, la interfaz `FlowDelJuego`.

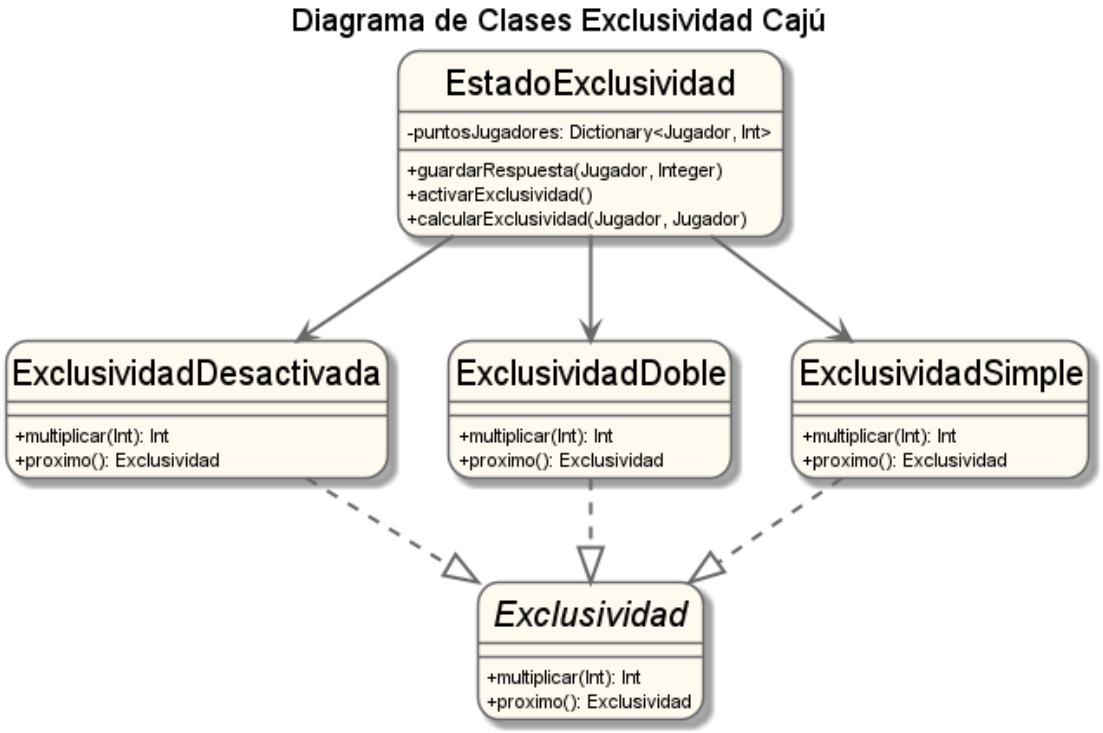

Figura 5: Diagrama de clases de la máquina de estados de las Exclusividades.

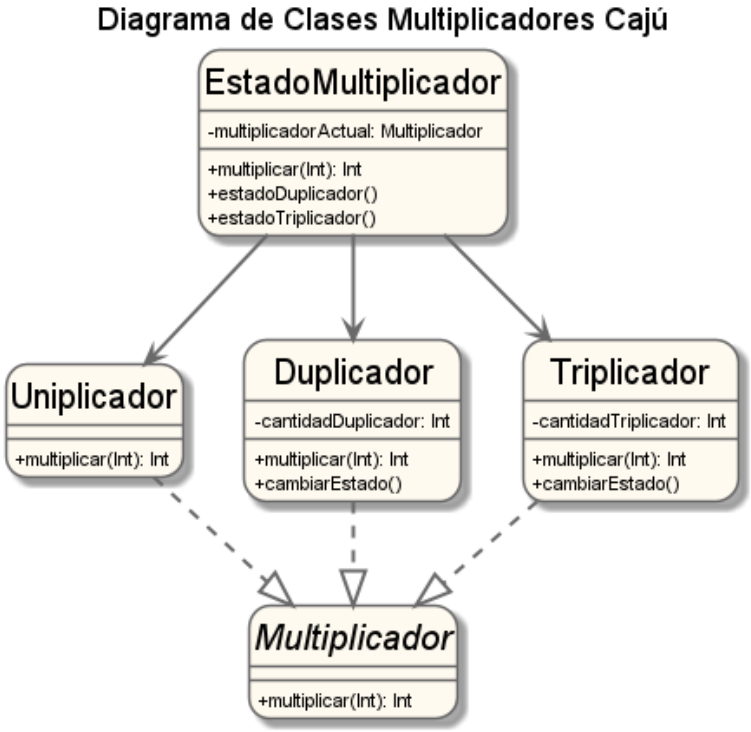

Figura 6: Diagrama de clases de la máquina de estados de los Multiplicadores.

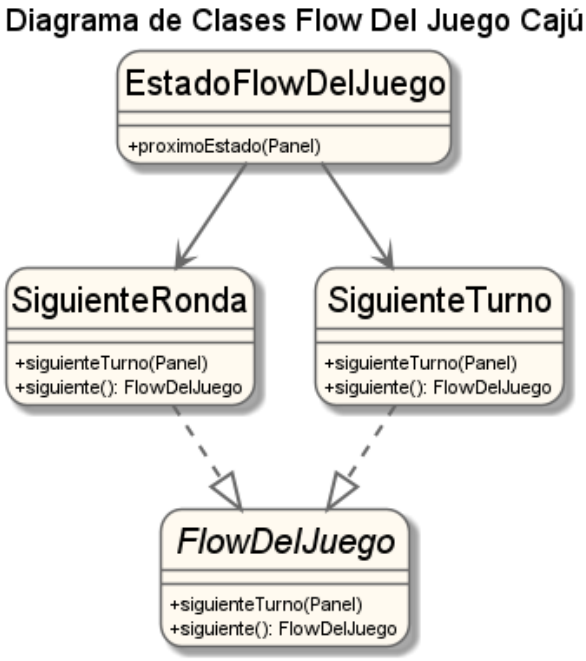

Figura 7: Diagrama de clases de la máquina de estados del flujo del juego.

4. Diagramas de secuencia

En las próximas figuras se muestran diagramas de secuencia generales:

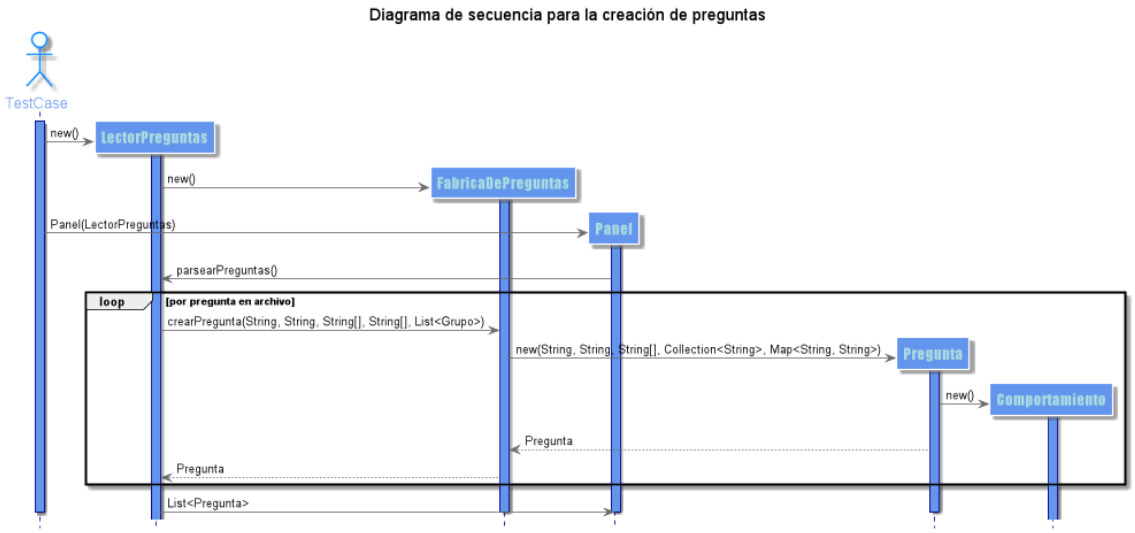

Figura 8: Diagrama de secuencia para la creación general de preguntas.

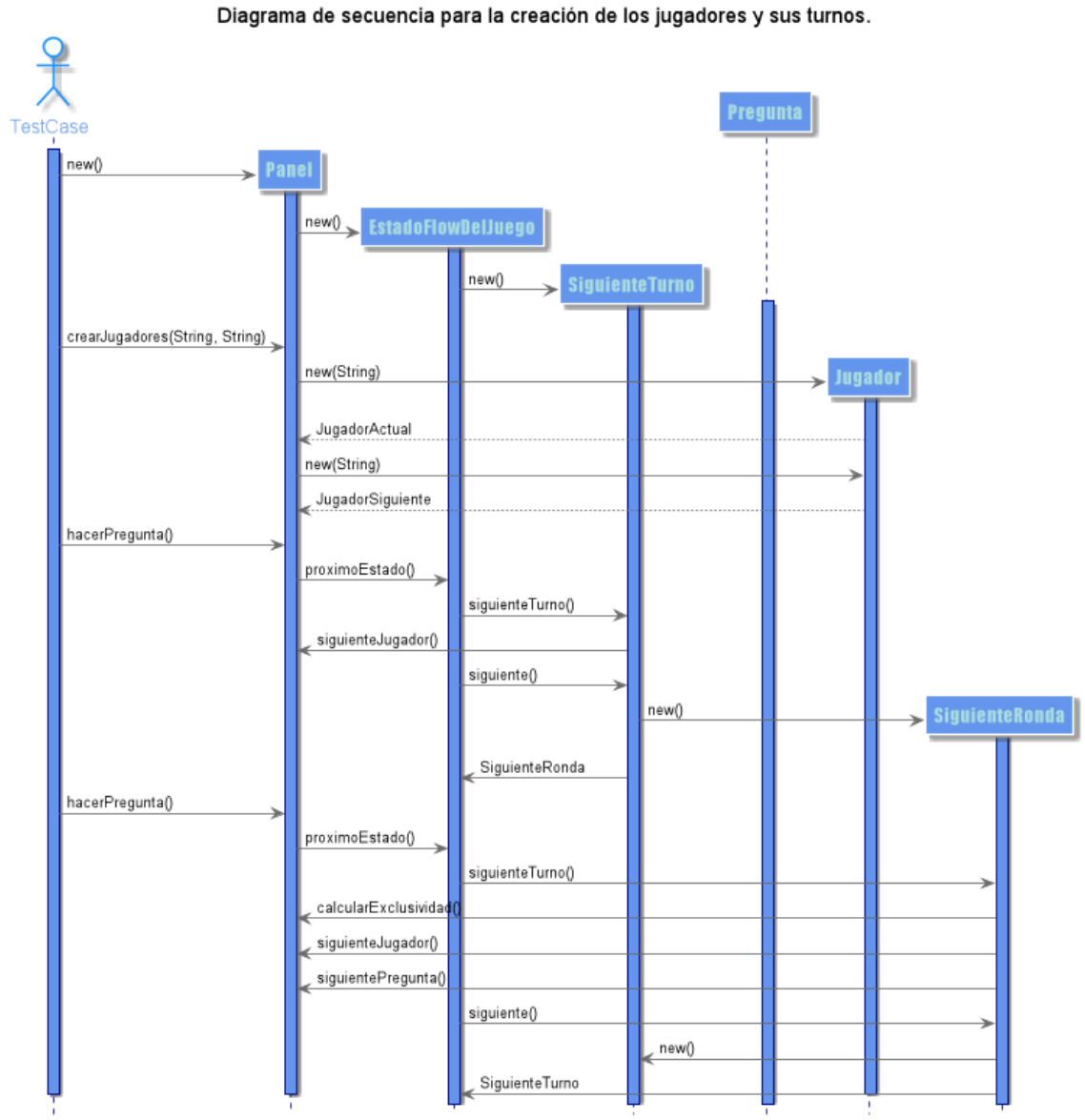

Figura 9: Diagrama de secuencia para la creación de Jugadores y la modalidad de turnos.

A continuación se mostrarán ejemplos de lanzamientos de excepciones para los casos en que no se pueda usar `Multiplicadores` o `Exclusividad`:

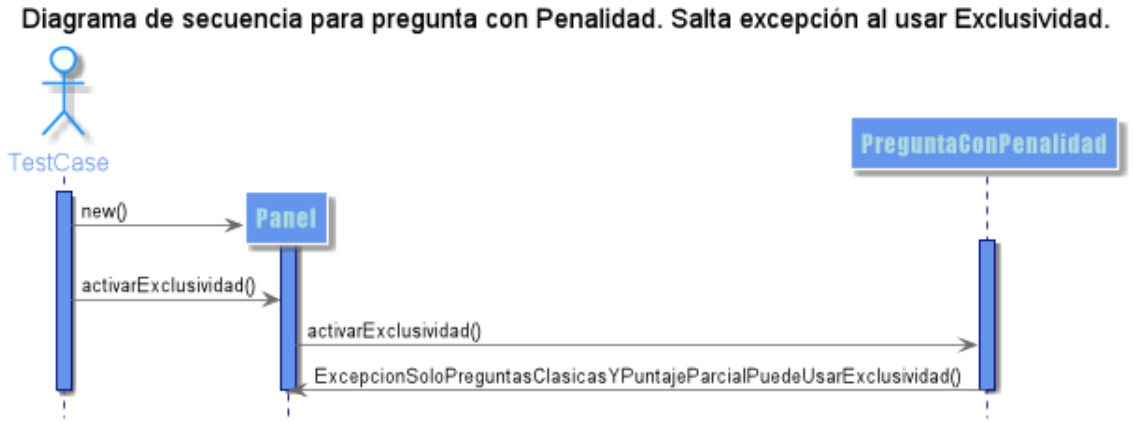

Figura 10: Diagrama de secuencia que lanza una excepción cuando se quiere utilizar Exclusividad en preguntas con Penalidad.

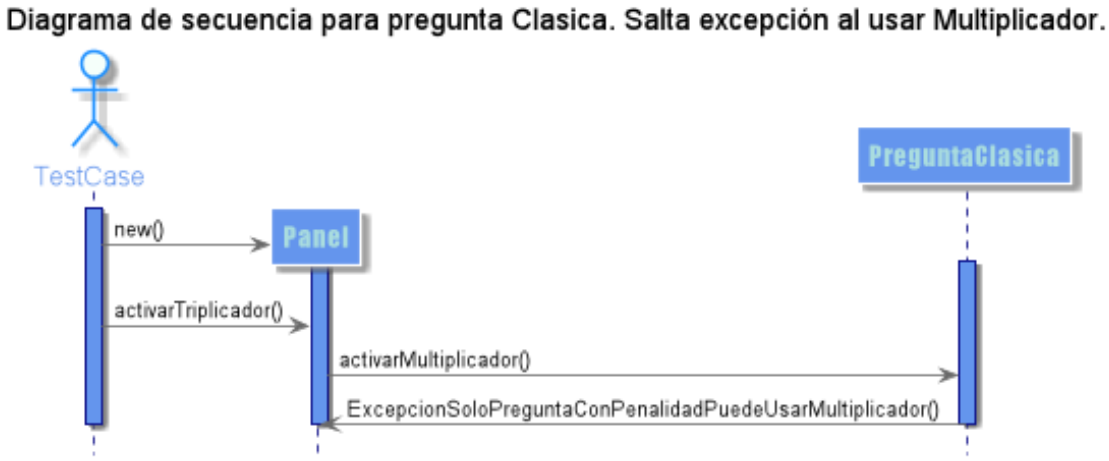

Figura 11: Diagrama de secuencia que lanza una excepción cuando se quiere utilizar Multiplicadores en preguntas sin Penalidad.

Por otra parte, en la imagen de la figura 12 se muestra el flujo del juego para una pregunta `Multiple Choice` utilizando `Exclusividad`. A su vez, la imagen de la figura 13 muestra el flujo del juego para una pregunta `Verdadero o Falso con Penalidad` utilizando un `Duplicador`.

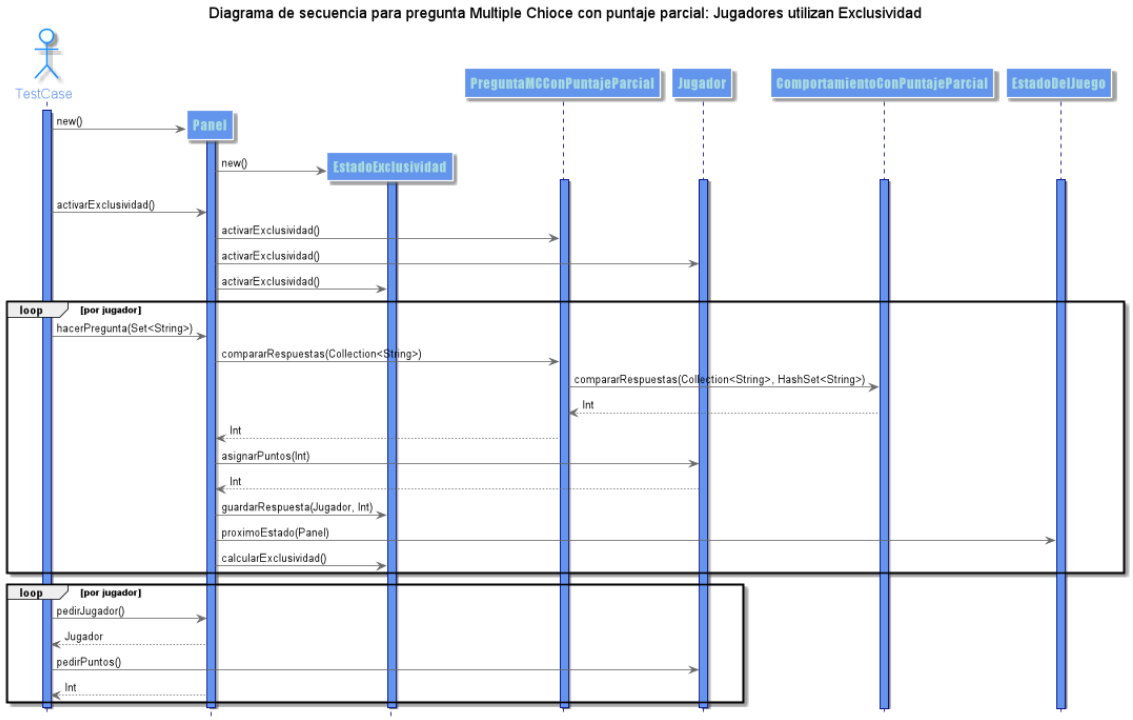

Figura 12: Diagrama de secuencia del flujo de una pregunta Multiple Choice con Exclusividad.

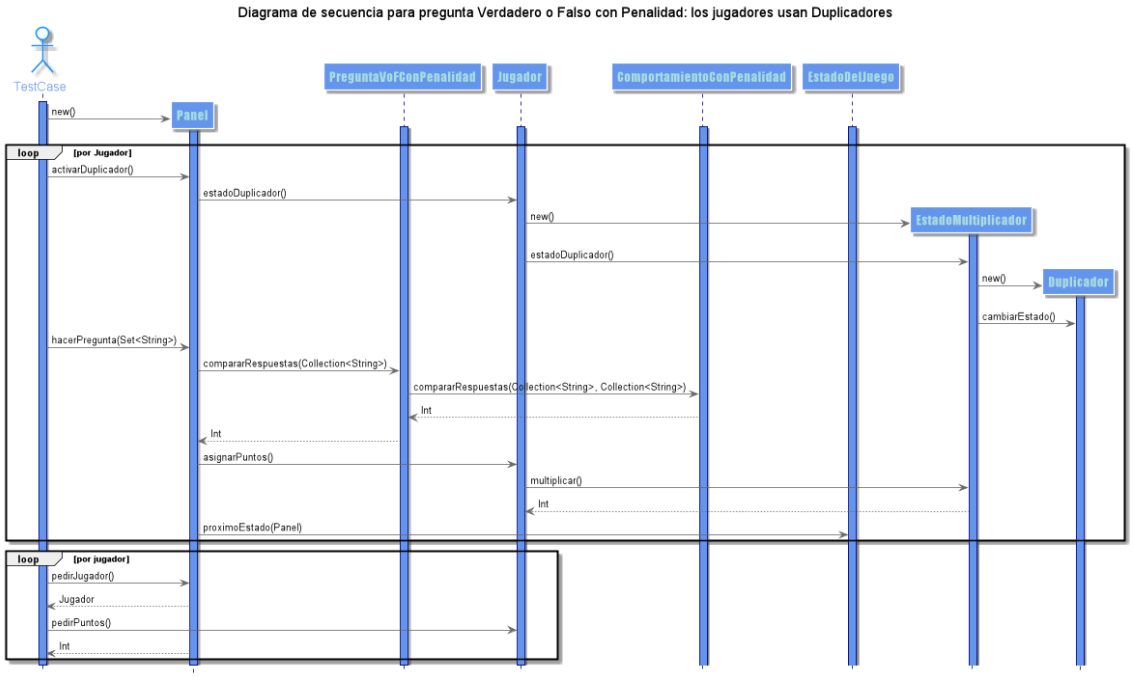

Figura 13: Diagrama de secuencia del flujo de una pregunta Verdadero o Falso con Penalidad con Duplicador.

5. Diagramas de paquetes

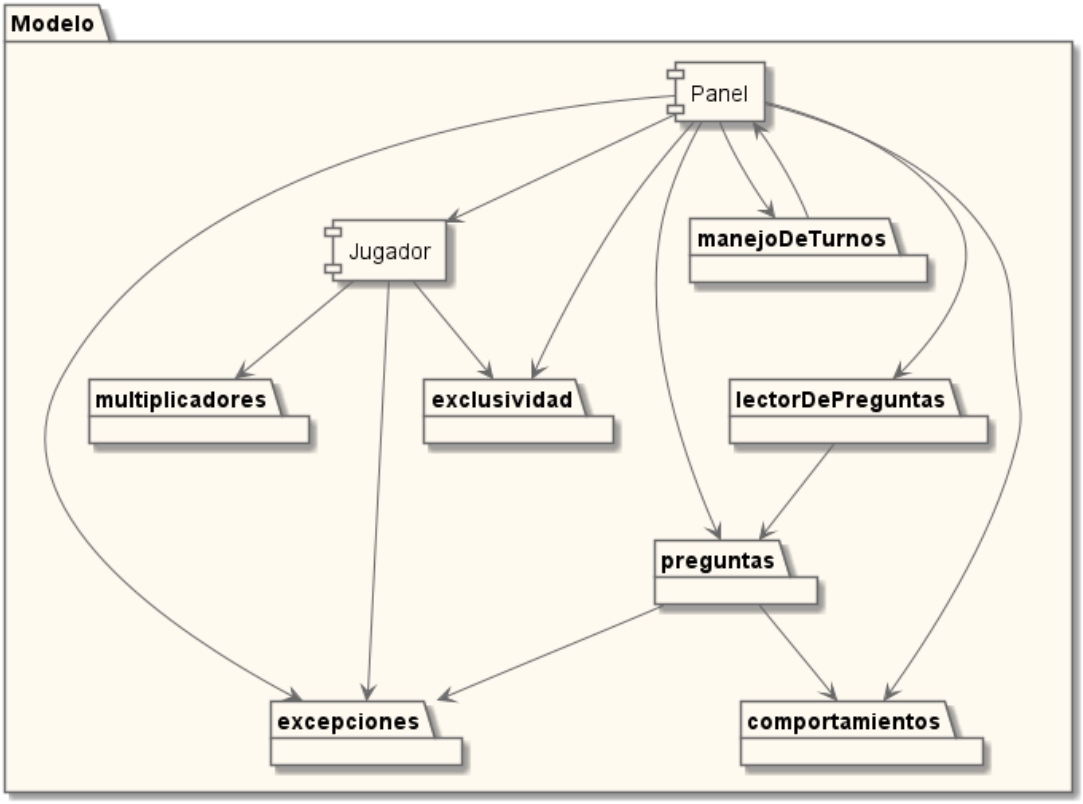

Figura 14: Diagrama de paquetes para el modelo del programa.

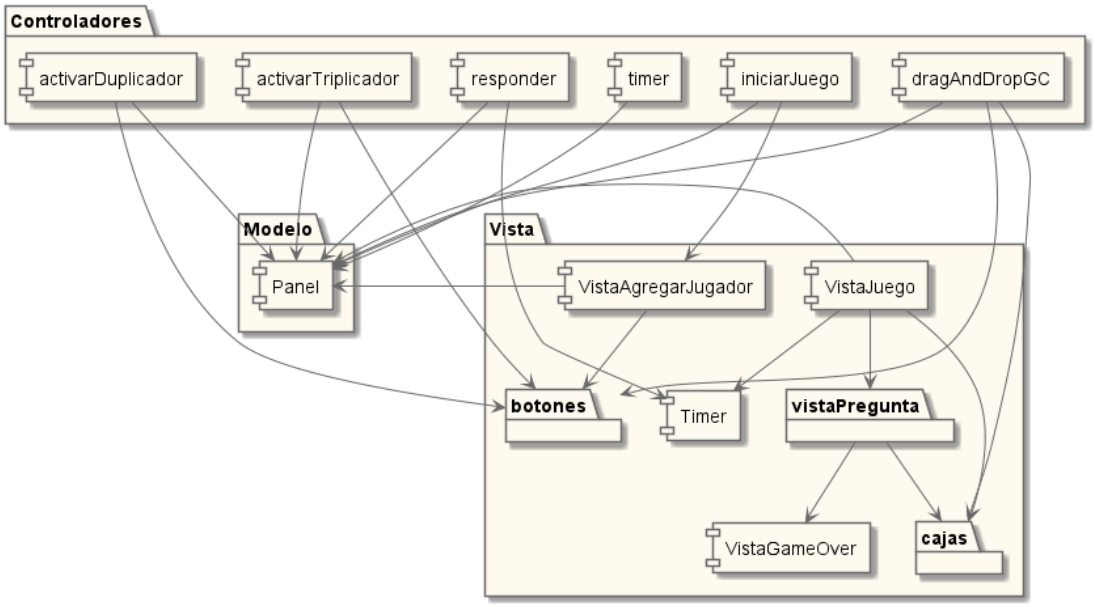

Figura 15: Diagrama de paquetes para la interfaz gráfica del programa.

6. Diagramas de estados

A continuación se exponen los diagramas de estado

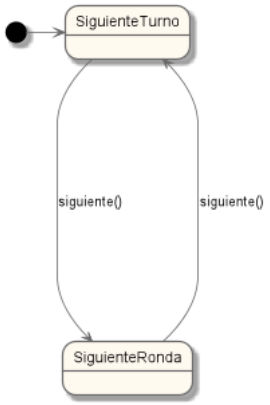

Figura 16: Diagrama de Estado para los turnos de los jugadores.

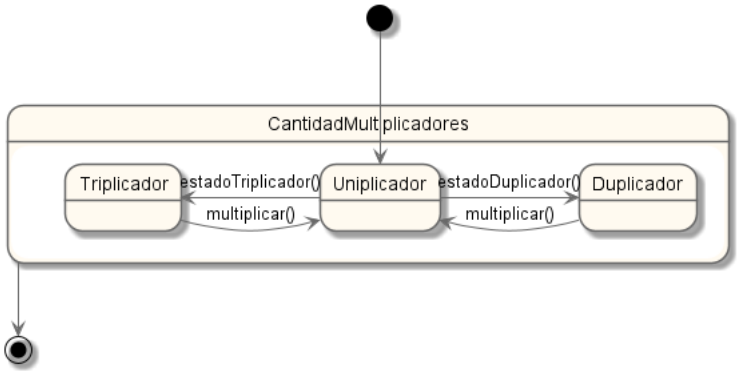

Figura 17: Diagrama de Estado para el uso de Multiplicadores.

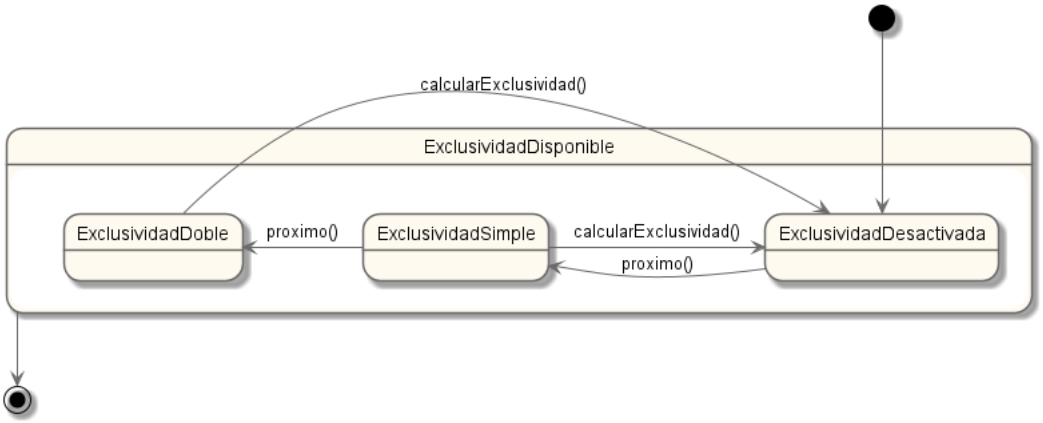

Figura 18: Diagrama de Estado para el uso de Exclusividad.

7. Detalles de implementación

A continuación se detalla un poco más exhaustivamente ciertos detalles que se usaron para el desarrollo del juego.

- Para la creación de las preguntas se utilizó el **patrón de diseño *Factory***. Se decidió implementarlo con el objetivo de liberarle a Panel la responsabilidad de la creación de las preguntas. De este modo, se encapsula la creación de las mismas en una clase externa. Existe una desventaja, que consiste en que la fábrica se implementa con un `switch case` que habría que modificar si se quisiera agregar otro tipo de pregunta en un futuro. Lo mencionado anteriormente rompe con el principio SOLID Open-Closed. Sin embargo, se concluyó que las ventajas que provee el patrón son mayores a las desventajas.

- Se implementaron tres tipos de máquinas de estados. Por un lado, se creó una para la implementación de los multiplicadores. Existe la interfaz *Multiplicador*; las clases que la im-

plementan son **Duplicador**, **Triplicador** y **Uniplicador**. Esta última se utiliza en el caso en que no se haga uso de los multiplicadores. La clase **EstadoMultiplicador** se encarga de los cambios de estado. Por otro lado, existe otra máquina para las exclusividades. Las clases que implementan la interfaz *Exclusividad* son: **ExclusividadSimple**, **ExclusividadDoble** y **ExclusividadDescativada**. La clase **EstadoExclusividad** tiene el propósito del cambio de estados. La tercera máquina de estados es utilizada para el manejo del flujo de los turnos y las rondas del juego, en donde los turnos se definen por jugador y las rondas por pregunta. Para el cambio de estados se utiliza la clase **EstadoFlowDelJuego**. La interfaz utilizada en este caso es *FlowDelJuego* y las clases que la implementan son **SiguienteTurno** y **SiguienteRonda**.

8. Excepciones

1. ExcepcionPreguntaGCInvalida: se lanza cuando la cantidad de opciones recibidas en la pregunta de tipo Group Choice es menor a dos o es mayor a seis. También cuando no se reciben los grupos que contendrán las respectivas respuestas.

2. ExcepcionPreguntaMCInvalida: se lanza cuando la cantidad de opciones recibidas en la pregunta de tipo Multiple Choice es menor a dos o es mayor a cinco.

3. ExcepcionPreguntaOCInvalida: se lanza cuando la cantidad de opciones recibidas en la pregunta de tipo Ordered Choice es menor a dos o es mayor a cinco.

4. ExcepcionPreguntaVOFInvalida: se lanza cuando la cantidad de opciones correctas recibidas en la pregunta de tipo Verdadero o Falso es distinta de uno.

5. ExcepcionSoloPreguntaConPenalidadPuedeUsarMultiplicador: se lanza cuando se quiere usar un Multiplicador en preguntas de los tipos Clasicas o con Puntaje Parcial.

6. ExcepcionSoloPreguntasClasicasYPuntajeParcialPuedeUsarExclusividad: se lanza cuando se quiere usar un Multiplicador en preguntas del tipo con Penalidad.

7. ExcepcionTipoPreguntaInvalida: se lanza cuando se recibe un tipo de pregunta inexistente.

8. ExcepcionYaUsasteTuDuplicador: se lanza cuando el jugador quiere usar de nuevo el Multiplicador x2.

9. ExcepcionYaUsasteTuTriplicador: se lanza cuando el jugador quiere usar de nuevo el Multiplicador x3.

10. ExcepcionYaActivasteTuExclusividad: se lanza cuando el jugador quiere usar nuevamente su exclusividad en su turno.

11. ExcepcionYaUsasteLasExclusividades: se lanza cuando el jugador quiere usar su exclusividad cuando ya no tiene más.

12. ExcepcionYaNoHayPreguntasParaHacer: se lanza cuando ya no hay preguntas disponibles.

9. Conclusión

Viendo en retrospectiva, podemos afirmar que se hicieron muchos cambios drásticos en el modelo del programa. Por ejemplo, en un principio se decidió implementar una clase por por tipo de pregunta. Luego, se notó que era innecesario tener tantas clases y se decidió hacer clases de pregunta por tipo de comportamiento. Con este razonamiento, se vio que no se podía generalizar el comportamiento para el tipo de pregunta Group Choice. Como resultado y finalmente, se decidió implementar una clase por tipo de pregunta y a la vez, definir una interfaz de comportamiento, en la cual, cada tipo de pregunta implementa un comportamiento. También el trabajo se vio afectado por los tiempos. Más que nada en la fase de implementación de la interfaz gráfica. Esto ocurrió a pesar de tener una planificación estricta del desarrollo del trabajo práctico. Como consecuencia, se comprendió que no se debía planear a futuro, es decir, no pensar en nuevos requerimientos si no es lo solicitado. Además, se puede decir que tal vez hubo falta de tiempo por la inexperiencia en cuanto al manejo de la herramienta JavaFx.