FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Java

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

Alumno	Padrón	E-mail
DÍAZ Juan Ignacio	103488	jidiaz@fi.uba.ar
DI MATTEO Carolina	103963	cdimatteo@fi.uba.ar
VALLCORBA Agustín	103447	avallcorba@fi.uba.ar
DE LA CRUZ Leonardo	80040	ljcruz@fi.uba.ar
ABBATE Mariano	100142	mabbate@fi.uba.ar

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 en Java para dos jugadores similar al famoso juego *Kahoot!* la cual consistirá en una serie de preguntas las cuales los jugadores deben responder como crean correcto, seleccionando una o varias opciones dependiendo del tipo de pregunta.

Se incluyen en la aplicación el modelo de clases y una interfaz gráfica de uso intuitivo para el/los usuario/s. La misma fue desarrollada utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

- Las preguntas de tipo *MultipleChoiceParcial* no permiten que se usen exclusividad de puntaje ó multiplicadores.
- Los jugadores deben seleccionar todas las opciones de una pregunta de tipo *OrderedChoice* antes de enviar su respuesta.
- Los jugadores deben seleccionar al menos una opcion de las preguntas de tipo *MultipleChoice* antes de enviar su respuesta.
- Todas las opciones deben tener un grupo asociado en una respuesta a una pregunta de tipo *GroupChoice*.
- El puntaje de un jugador puede ser negativo.
- Si a un jugador se le acaba el tiempo Se considera que su respuesta es incorrecta (Si la pregunta es de tipo *VerdaderoFalsoPenalidad* se considera que seleccionó la respuesta incorrecta).

3. Modelo de dominio

La idea general al diseñar el modelo fue la de distribuir las responsabilidades equitativamente, de manera que cada clase tuviera un proposito.

Se utiliza el patrón *Facade* en la clase *AlgoKahoot* para minimizar el acoplamiento entre la vista y el modelo. La misma tendrá una ronda por cada pregunta y es responsable de saber de que jugador es el turno.

Se utiliza herencia entre las clases Pregunta *(GroupChoice/MultipleChoice/OrderedChoice/VerdaderoFalso)* y la clase abstracta Pregunta por un lado porque estas cumplen la relación "es un", y por otro lado esta relación ayuda a hacer el código más legible y a evitar repetirlo. Estas preguntas recibiran las respuestas de los jugadores y las evaluarán comparandolas con la respuesta correcta.

Luego la ronda mencionada anteriormente asignará los puntos correspondientes a los respectivos jugadores utilizando un puntuador el cual podría ser un puntuador básico o un puntuador de exclusividad dependiendo de si los jugadores lo utilizan. Entonces comenzará una nueva ronda o terminará el juego.

4. Diagramas de clase

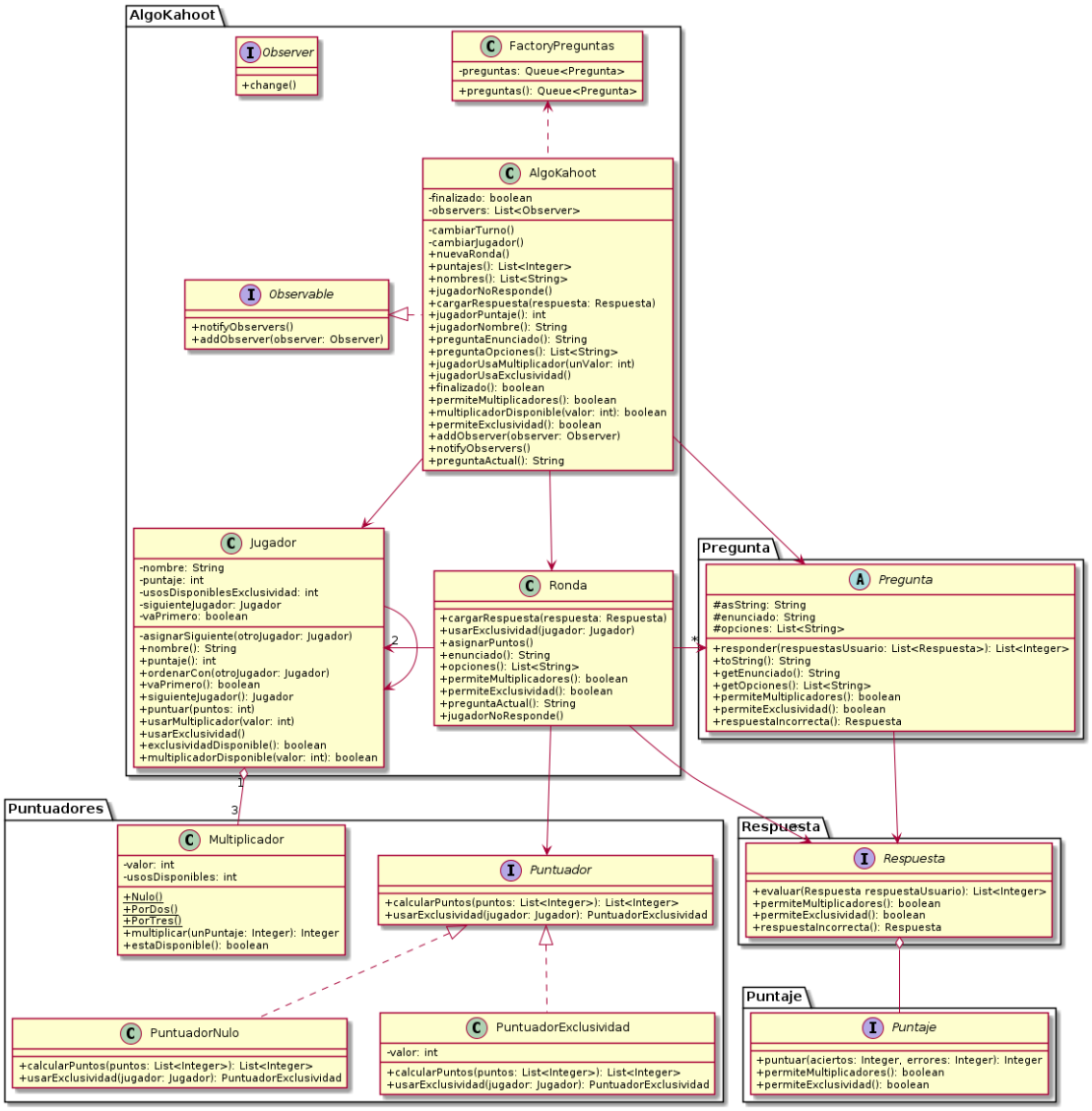

Figura 1: Diagrama de clases del modelo general.

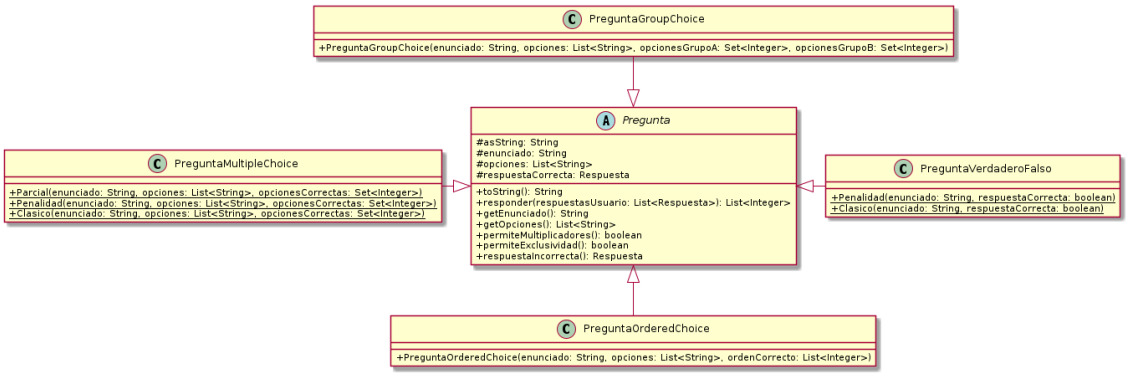

Figura 2: Diagrama de clases de las Preguntas.

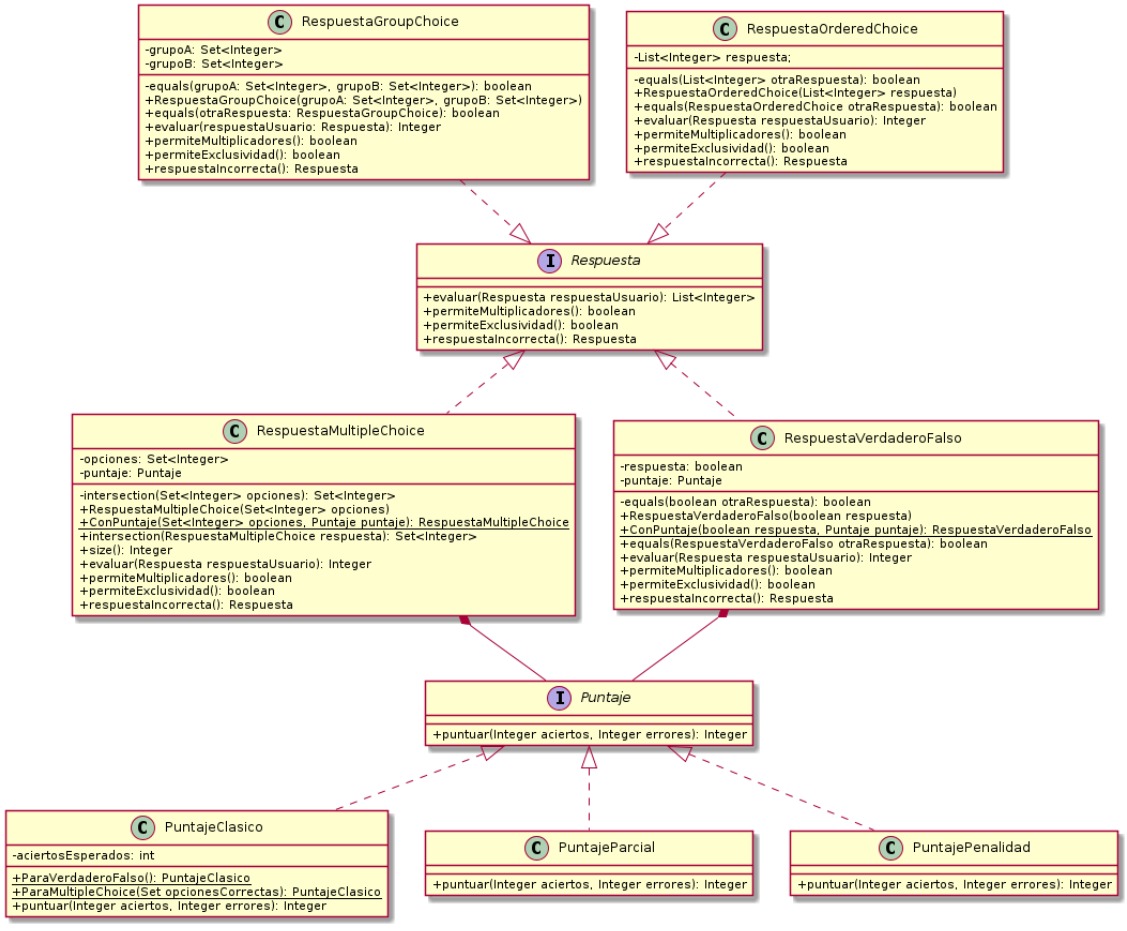

Figura 3: Diagrama de clases de las Respuestas.

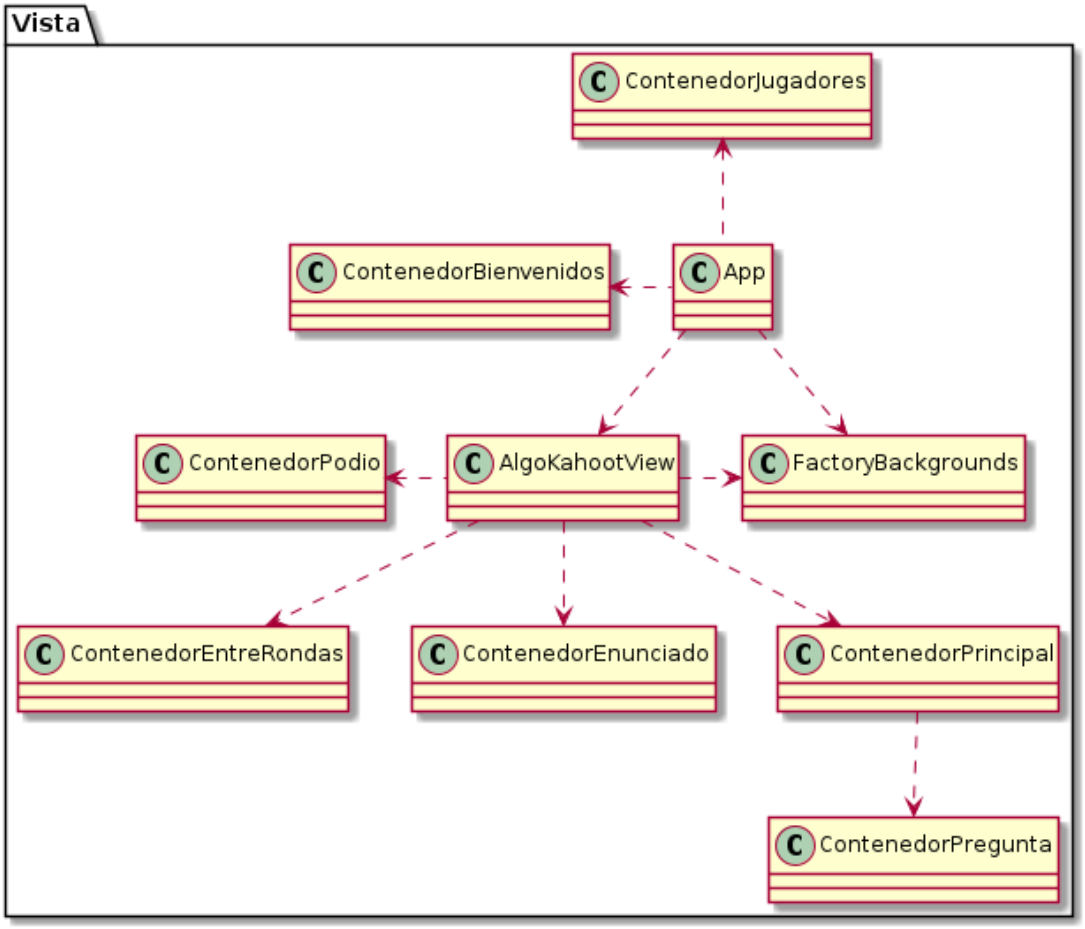

Figura 4: Diagrama de clases de la Vista.

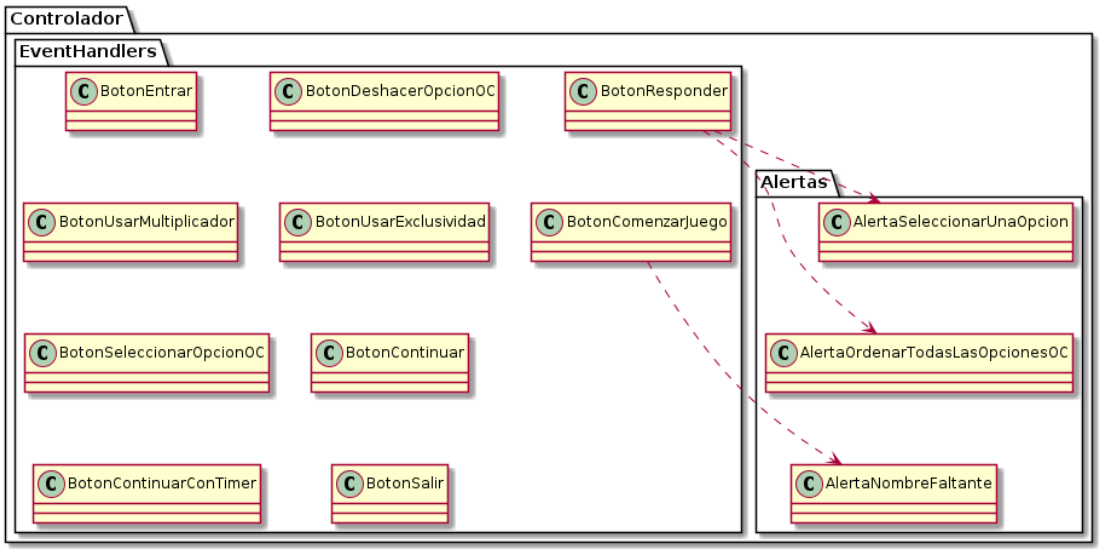

Figura 5: Diagrama de clases de los Controladores.

5. Detalles de implementación

5.1. Sistema de rondas

Para manejar el flujo general de turnos del juego en los que ambos jugadores deben responder a la misma pregunta en diferentes momentos decidimos conveniente implementar una clase *Ronda* que administra la lógica en la que -al jugador actual- le corresponde enviar su respuesta. Para ello *Ronda* conoce a qué *Pregunta* enviar las respuestas, quiénes son los jugadores que participan del juego, y la mecánica necesaria para asignar los puntajes correspondientes según se haya solicitado activar la exclusividad de puntajes, o no. En pocas palabras, cada jugador cargará sus respuestas cuando sea su turno (es decir, cuando sea el jugador actual), una vez que se hayan cargado todas las respuestas la ronda delegará el cálculo de los puntos correspondientes a un objeto de tipo *Puntuador*, y -acto seguido- asignará los puntajes finales de la ronda a quienes corresponda.

5.2. Mecánica de evaluación de respuestas

Cada respuesta correcta es responsable de saber evaluar una respuesta del jugador la cual debe ser instancia de la misma clase que esta. Es por eso que al momento de evaluar la respuesta de un jugador la misma es ”casteada” al tipo de la respuesta correcta para poder operar con ella libremente.

Esto es posible porque la respuesta del usuario a una pregunta y la respuesta correcta asosiada a la misma son del mismo tipo. En caso que no lo fuera, el modelo no estaría siendo usado correctamente, y se lanzaría una excepción de tipo *RespuestaIncompatibleException*.

Con esta solución violamos el principio de *Inversión de Dependencias*, ya que al castear a una clase específica estamos dejando de depender de abstracciones. Sin embargo, la misma no viola el principio de *Segregación de Interfaz*, lo que nos obligaría a implementar métodos que luego no serían utilizados por las clases implementadoras.

5.3. Multiplicadores y Puntuadores

Multiplicadores: Los multiplicadores que solicita utilizar cada jugador en su turno para preguntas con penalidad sólo afectarán a la asignación de su propio puntaje, por lo que, consi-

deramos apropiado que sea el mismo jugador quien administre qué multiplicadores usar al momento de recibir puntos por responder una pregunta. Esto lo implementamos teniendo en cuenta el patrón *State*, siendo que actúa como intermediaro recibiendo un puntaje y multiplicándolo según corresponda (x1, x2 o x3), luego devuelve el puntaje ya multiplicado y el jugador es quien lo suma a su puntaje previo.

Puntuadores: Los puntuadores cumplen el rol de administrar la mecánica de puntuación para la ronda. Esto afecta a ambos jugadores por lo que es necesario que todas las respuestas hayan sido cargadas antes de ser usados. En el caso general, el *PuntuadorNulo* actúa devolviendo los puntajes tal cual los recibe. El caso que resulta más interesante es el del *PuntuadorExclusividad*, que se activa cuando es solicitado por algun jugador en su turno para preguntas sin penalidad. Este puntuador recibe los puntos que hizo cada jugador y los compara para saber quien respondió correctamente y asignarle el doble (o cuádruple) de puntos. En caso de que hubiera un empate, no devuelve puntos para ningún jugador.

6. Excepciones

PowerUpNoDisponibleException Se lanzará si un jugador intentara utilizar un multiplicador o exclusividad de puntaje y no tiene ninguno disponible.

RespuestaIncompatibleException Se lanzará si se intentara contestar a una pregunta con una respuesta que corresponde a una pregunta de otro tipo.

PreguntaDesconocidaException Se lanzará si en el archivo de extensión .json se encontrara una pregunta de un tipo no contemplado en el modelo.

7. Diagramas de secuencia

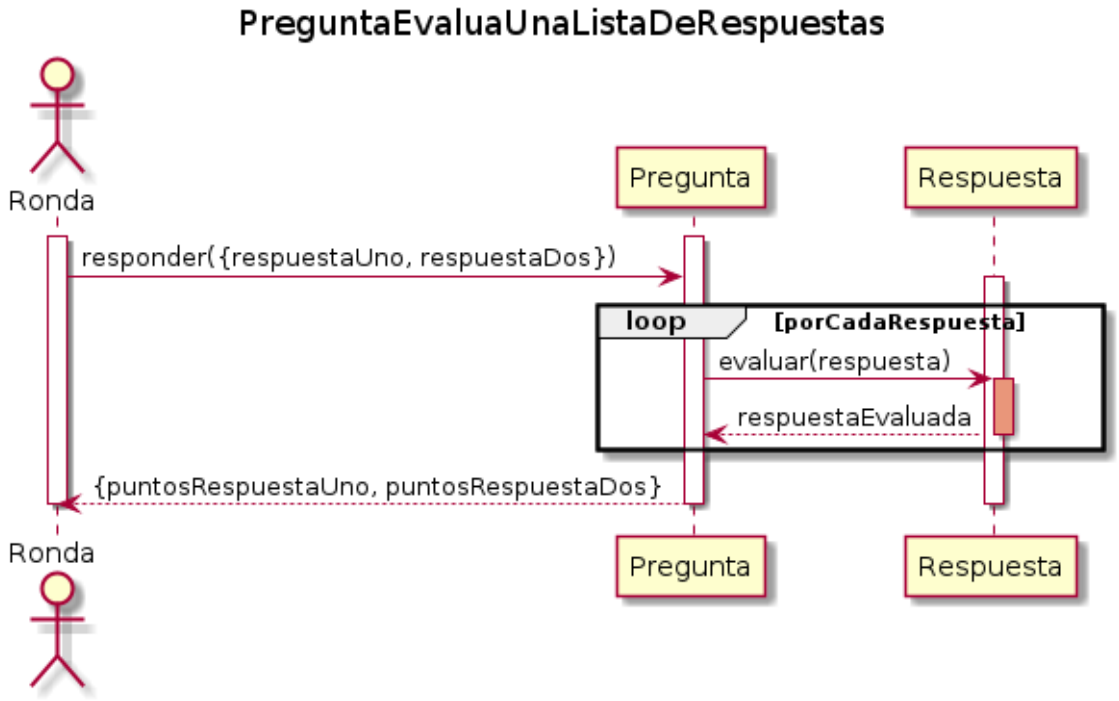

Figura 6: Comportamiento observado al responder una pregunta.

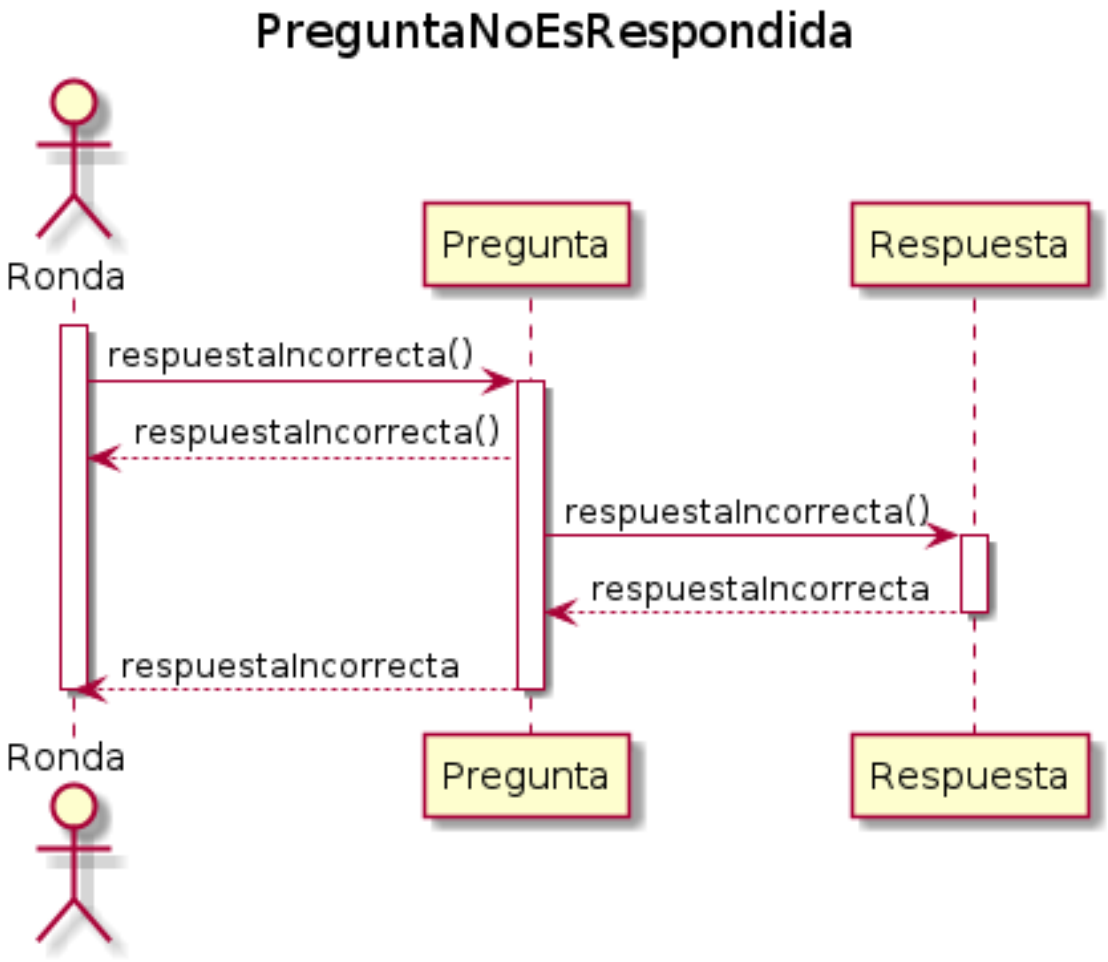

Figura 7: Comportamiento observado al no responder una pregunta.

A continuación se detalla como cada respuesta correcta evalúa una respuesta del usuario.

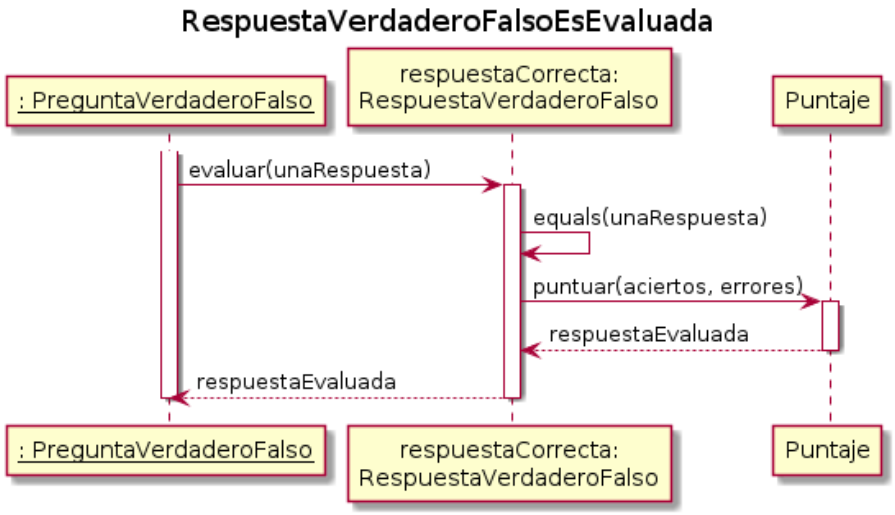

Figura 8: Comportamiento observado al evaluar una respuesta de tipo *VerdaderoFalso*.

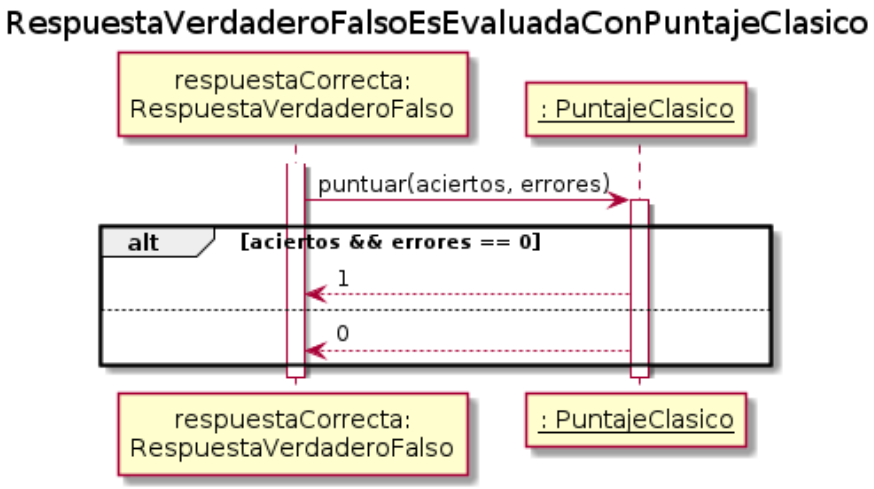

Figura 9: Comportamiento observado al evaluar una respuesta de tipo *VerdaderoFalso* con Puntaje de tipo *Clasico*.

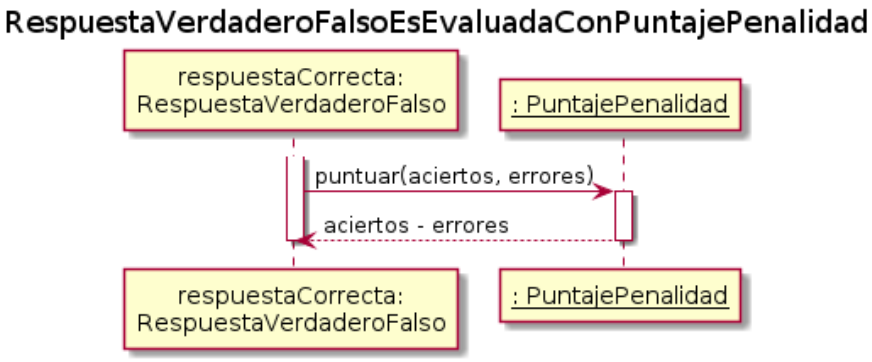

Figura 10: Comportamiento observado al evaluar una respuesta de tipo *VerdaderoFalso* con Puntaje de tipo *Penalidad*.

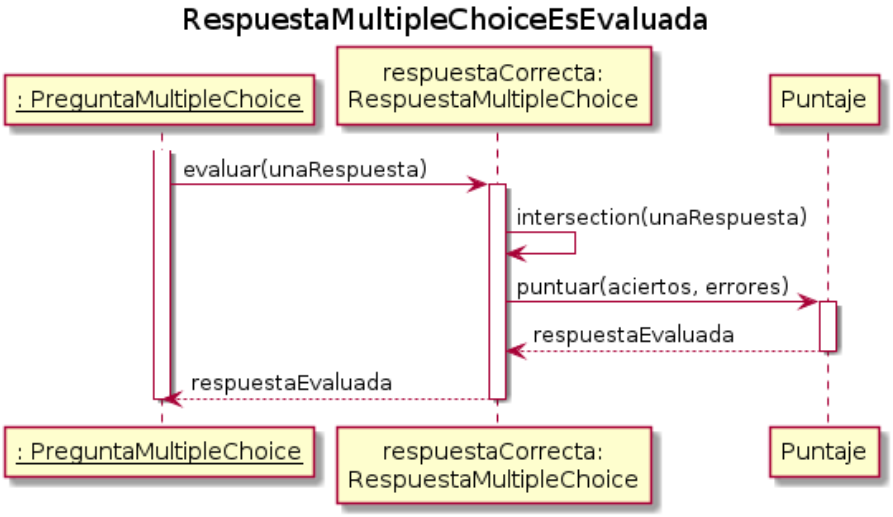

Figura 11: Comportamiento observado al evaluar una respuesta de tipo *MultipleChoice*.

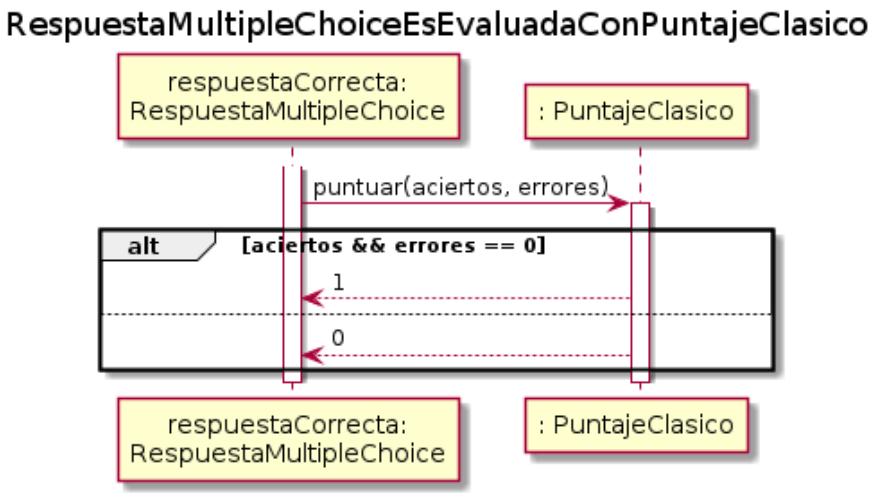

Figura 12: Comportamiento observado al evaluar una respuesta de tipo *MultipleChoice* con Puntaje de tipo *Clasico*.

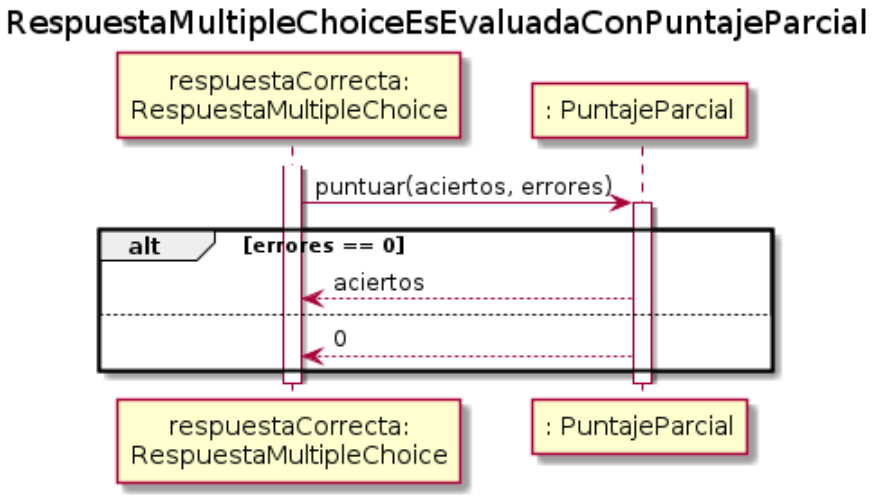

Figura 13: Comportamiento observado al evaluar una respuesta de tipo *MultipleChoice* con Puntaje de tipo *Parcial*.

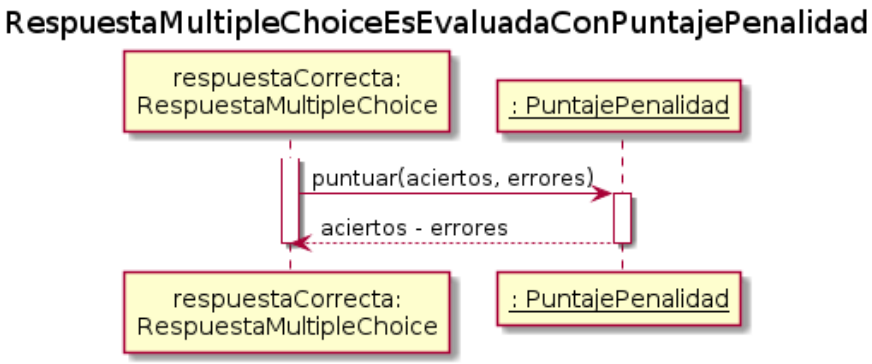

Figura 14: Comportamiento observado al evaluar una respuesta de tipo *MultipleChoice* con Puntaje de tipo *Penalidad*.

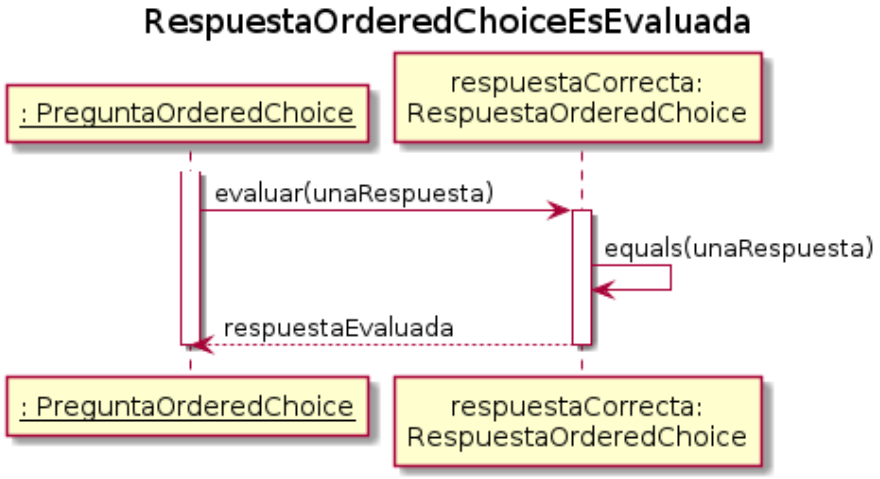

Figura 15: Comportamiento observado al evaluar una respuesta de tipo *OrderedChoice*.

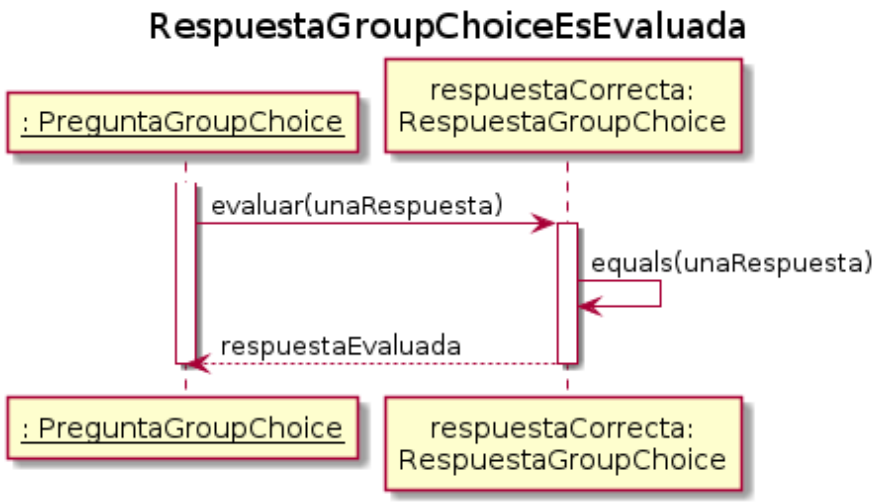

Figura 16: Comportamiento observado al evaluar una respuesta de tipo *GroupChoice*.

8. Diagramas de paquetes

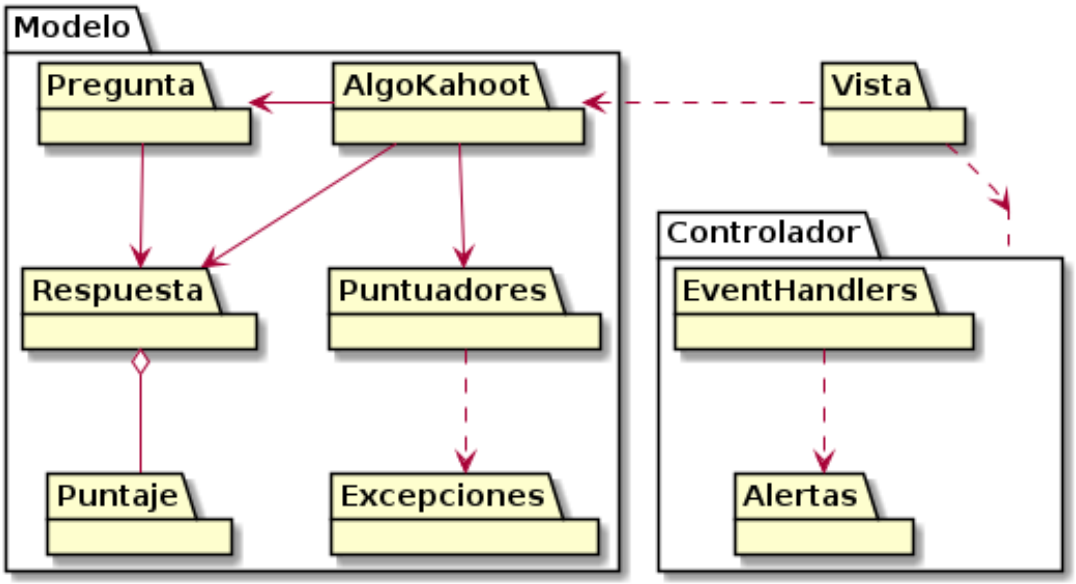

Figura 17: Relación entre paquetes de la aplicación.

9. Diagramas de estado

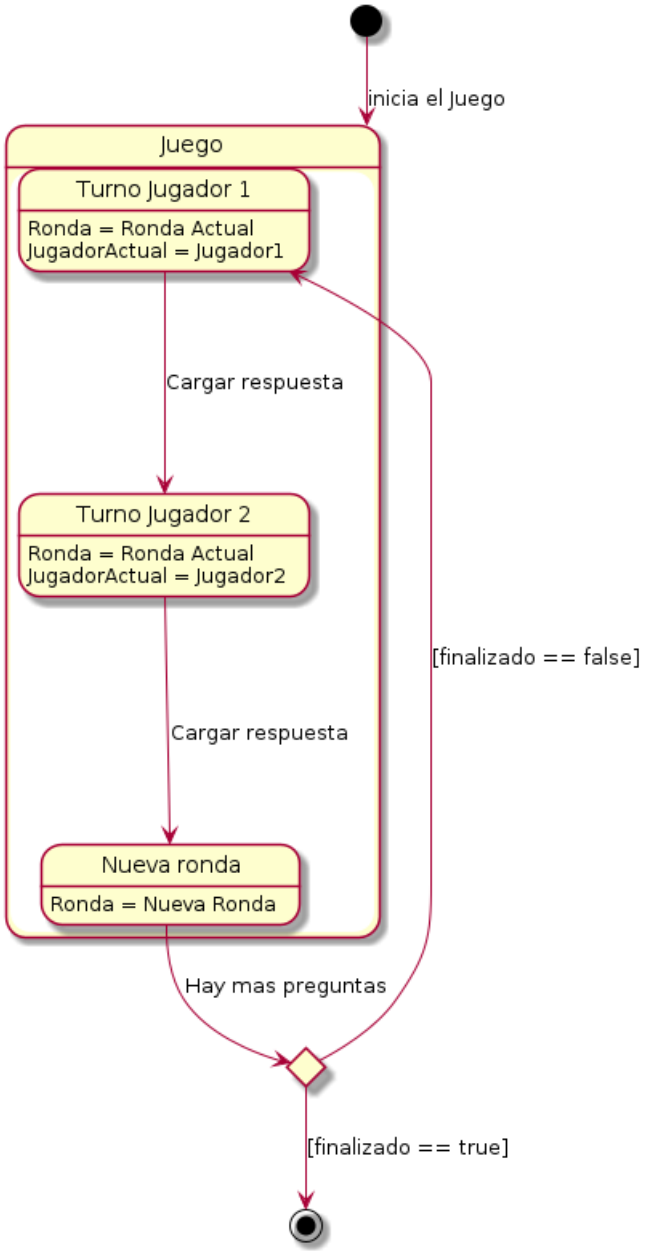

Figura 18: Transición de estados durante el desarrollo del juego.

10. Formato del archivo JSON de preguntas

El archivo JSON consiste un array de preguntas llamado "pregutnasz según el tipo de pregunta tiene un formato particular. Describimos los objetos JSON para cada tipo de pregunta:

10.1. Pregunta VF

```
{
```

```
  "tipo": "verdaderoFalsoPenalidad" ,
  "enunciado": "La manzana es una fruta" ,
  "respuesta": true
}
```

El tipo pude ser verdaderoFalsoClasico o verdaderoFalsoPenalidad. La respuesta true o false.

10.2. Pregunta Multiple Choice

```
{
  "tipo": "multipleChoiceClasico" ,
  "enunciado": "La naranja es..." ,
  "opciones": [ "Una verdura" , "Una fruta" , "Un cítrico" , "Como la banana" ] ,
  "respuestas": [ 1 , 2 ]
}
```

El tipo puede ser: * multipleChoiceClasico * multipleChoiceParcial * multipleChoicePenalidad

En respuesta se va el index (basado en 0) de las repuestas correctas seleccionadas desde las opciones.

10.3. Pregunta Ordered Choice

```
{
  "tipo": "orderedChoice" ,
  "enunciado": "Ordene según el abecedario las siguientes letras" ,
  "opciones": [ "D" , "B" , "C" , "A" ] ,
  "respuestas": [ 3 , 1 , 2 , 0 ]
}
```

En respuestas se indica el array de índices que indican el orden de la secuencia correcta.

10.4. Pregunta Group Choice

```
{
  "tipo": "groupChoice" ,
  "enunciado": "Agrupe según frutas (A) y verduras (B)",
  "opciones": [ "Lechuga" , "Naranja" , "Banana" , "Zanahoria" , "Tomate" ] ,
  "respuestas": { "grupoUno": [ 1 , 2 , 4 ] , "grupoDos": [ 0 , 3 ] }
}
```

En el enunciado por claridad es conveniente indicar claramente el tipo de grupo. Puede ser como en el ejemplo (A) o (B). En las repuestas se dan los índice de las opciones que entran en cada grupo.

10.5. Ejemplo de archivo completo conteniendo dos preguntas

```
{
  "preguntas": [
    {
      "tipo": "verdaderoFalsoPenalidad" ,
      "enunciado": "La manzana es una fruta" ,
      "respuesta": true
    } ,
    {
      "tipo": "multipleChoiceClasico" ,
```

```
        "enunciado": "La naranja es..." ,
        "opciones": [ "Una verdura" , "Una fruta" , "Un cítrico" , "Como la banana" ] ,
        "respuestas": [ 1 , 2 ]
      }
    ]
}
```