FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Algohoot

[7507/9502] Algoritmos y Programación III
Curso 1
Grupo T8
Primer cuatrimestre de 2020

Repositorio: https://github.com/erick12m/algo3_tp2

Alumno:	TORRESETTI, Lisandro
Número de padrón:	99846
Email:	ltorresetti@fi.uba.ar
Alumno:	SOSA, Martín
Número de padrón:	98741
Email:	csosa@fi.uba.ar
Alumno:	RODRÍGUEZ, Alejo
Número de padrón:	99869
Email:	atrodriguez@fi.uba.ar
Alumno:	MARTÍNEZ QUINTERO, Erick
Número de padrón:	103745
Email:	egmartinez@fi.uba.ar
Alumno:	LOVERA, Daniel
Número de padrón:	103442
Email:	dlovera@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 desarrollar una reimaginación de Kahoot, en Java utilizando los conceptos del paradigma de la orientación a objetos vistos en el curso.

2. Supuestos

Los supuestos realizados fueron:

- El mínimo puntaje posible para un jugador es cero.
- Para las preguntas del estilo group o las del estilo ordered, una respuesta es válida solo si se usan todas las opciones, caso contrario, se toma como respuesta incorrecta.
- Solo se puede utilizar un multiplicador de cada tipo y una exclusividad de puntaje en toda la partida. En caso que el jugador intente activarlo de nuevo, saldrá una alerta.
- Cada jugador tiene 25 segundos para responder a la pregunta, en caso de que se acabe el tiempo se mostrará un mensaje en la pantalla del juego, y la respuesta guardada será lo último que haya seleccionado el jugador (si es que llegó a hacerlo).
- El juego consta de 10 preguntas de diferentes tipos, el mismo finaliza una vez que se hayan respondido las 10 preguntas.
- Las preguntas serán tomadas de manera aleatoria de entre las preguntas cargadas en los archivos JSON que acompañan al juego.

3. Modelo de dominio

Se utiliza una clase Kahoot, la cual controla los aspectos más importantes del juego, posee una lista de jugadores, y una de preguntas. La clase jugador está pensado para manejar su puntaje, y las preguntas están representadas por una clase abstracta "Pregunta", cuyas clases herederas poseen el comportamiento de cada tipo de preguntas en el juego (verdadero y falso, multiple choice, group choice y ordered choice).
Dependiendo del tipo de pregunta el jugador puede activar (o no), algunas funcionalidades, tales como multiplicadores del puntaje (usado en preguntas con "penalidad") y una exclusividad del puntaje (para preguntas sin "penalidad"), las cuales pueden multiplicar el puntaje del jugador por cada pregunta bien respondida.
Las preguntas se generan por medio de un archivo JSON, el cual es leído por el programa y genera las clases pregunta que serán utilizadas por la clase Kahoot. Estos archivos poseen aproximadamente 50 preguntas de distintos tipos, de las cuales se seleccionan 10 de forma aleatoria para el juego.
Una vez son respondidas todas las preguntas presentadas, el juego finaliza y se muestra el jugador ganador (en el caso de que un jugador posea más puntos que el otro) o se muestra que hubo un empate.
Para visualizar el juego se desarrolló una interfaz (utilizando java fx).

4. Diagramas de clase

Dado a la complejidad del modelo, hemos separado el diagrama principal en el diagrama principal, el diagrama de la clase Pregunta y el diagrama de la interfaz Corrector

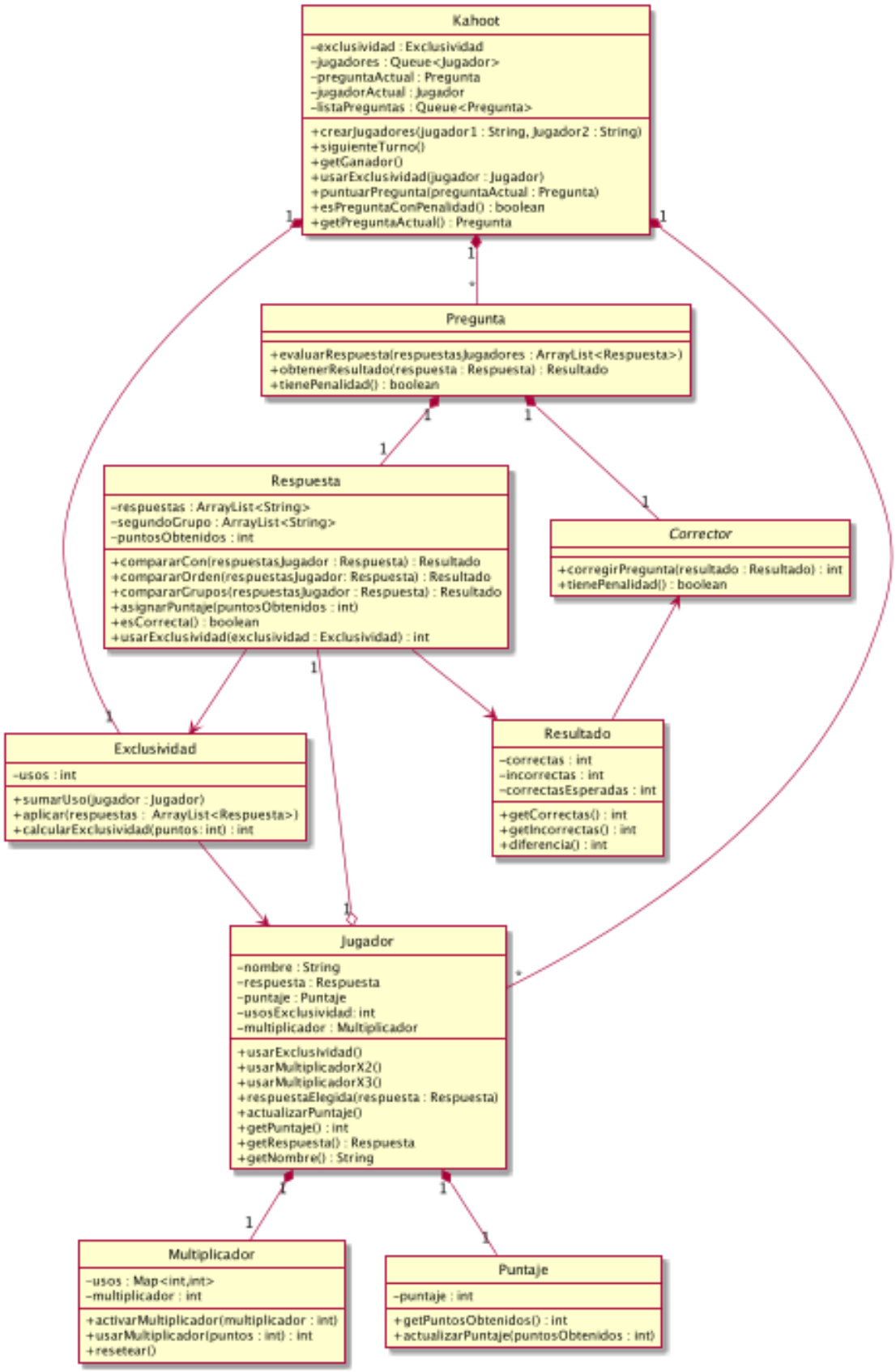

Figura 1: Diagrama de clase de AlgoHoot.

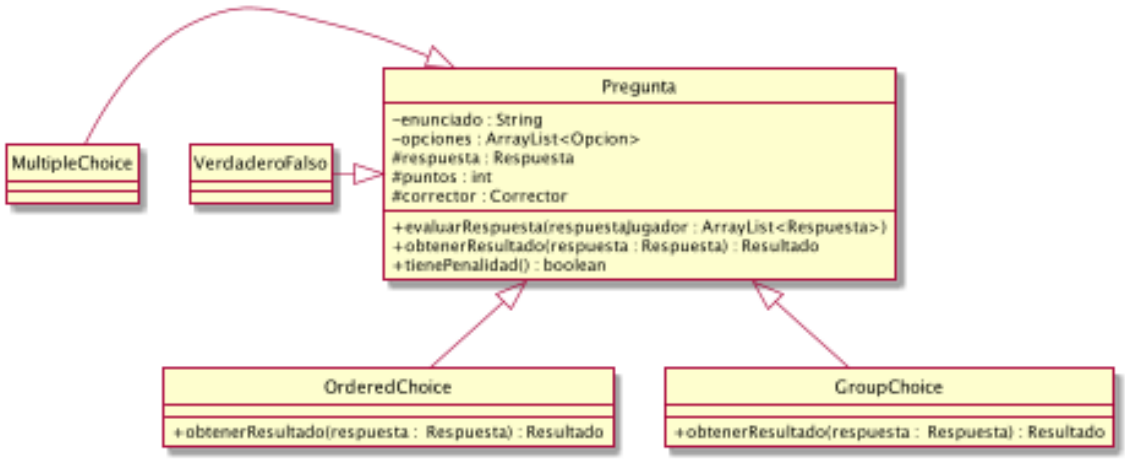

Figura 2: Diagrama de clase de Pregunta.

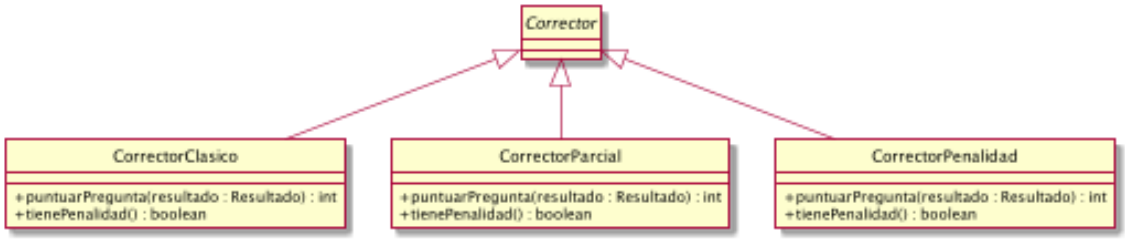

Figura 3: Diagrama de la interfaz Corrector.

5. Detalles de implementación

5.1. Interfaz corrector

La interfaz Corrector es la que nos permite distinguir que tipo de puntaje otorga cada pregunta, los puntajes pueden ser: con penalidad, con puntaje parcial o con puntaje clásico.
La misma toma las respuestas dadas por los jugadores y determina el comportamiento dado por el método 'corregirPregunta' utilizada por el método 'evaluarRespuesta' de la clase pregunta.
Esta interfaz se creó con el fin de utilizar el patrón State y asignarle al momento de la creación un comportamiento definido a la pregunta, así esta conoce utilizando al corrector como evaluarse en cada caso, además de que evita la creación de distintos tipos de clases Pregunta.

5.2. Persistencia de Archivos

Para la persistencia de las preguntas se utilizaron archivos de tipo json y se creo la clase AdaptadorCorrector que es la responsable de deserializar los objetos en este formato. Para evitar agregar más responsabilidades sobre el AdaptadorCorrector se creó la clase GestorObjetosJson que se encarga finalmente de recuperar los objetos que fueron serializados.

5.3. Clase Kahoot

Es la encargada de manejar los aspectos principales del programa, la misma posee como atributos la lista de preguntas, las cuales se crean una vez se construye la clase (mediante la fábri-

caDePreguntas) y la lista de jugadores, que se crean una vez que el controladorJuego recibe los nombres de los jugadores.
La clase Kahoot maneja la distribución de los jugadores y las preguntas a lo largo de las rondas de la partida, enviándole mensajes a las respectivas instancias para que se calculen y corrijan sus puntajes, además se encarga de determinar el fin del juego como también el respectivo ganador.

5.4. Clase controladorJuego

Es la clase encargada de conectar a la clase Kahoot con la interfaz. Crea los jugadores que formarán parte de la clase Kahoot. Y ejecuta acciones tales como la transición entre jugadores y preguntas durante las rondas, el manejo de ventanas emergentes para alertar sobre errores o sobre el ganador del juego y también de refrescar las pantallas. También se encarga de setear el timer para cada una de las preguntas y de manejar el reproductor de música. Posee un método que es el encargado de manejar gran parte del flujo del juego, el mismo se muestra a continuación:

```
public void siguienteTurno(StackPane stackPregunta) {
        timer.stop();
        try {
            Respuesta respuestaJugador = controladorPregunta.getRespuestaJugador();
            kahoot.getJugadorActual().respuestaElegida(respuestaJugador);
            kahoot.siguienteTurno();
            this.actualizarTablero();
            timer.play();
        } catch (RondaFinalizadaException e) {
            this.corregirRespuestas();
            this.preguntaActual = kahoot.getPreguntaActual();
            this.controladorPregunta = this.getControladorCorrecto();
            this.mostrarRespuestaCorrecta();
            this.respuestaCorrecta = kahoot.getPreguntaActual().getRespuesta();

            stackPregunta.getChildren().clear();
            stackPregunta.getChildren().add(new VistaPregunta(controladorPregunta));
            this.actualizarTablero();
        }
        catch (GameOverException gameOverException) {
                this.corregirRespuestas();
                this.gameOver();
        }
    }
```

Como se puede observar, lo primero que realiza este método es detener el timer ya que pueden darse diferentes situaciones las cuales afectarían el tiempo que posee el siguiente jugador para responder la pregunta. Este método se ejecuta cada vez que el jugador clickea el botón ”continuar”, que figura en la parte inferior del tablero del juego, para indicar que terminó de responder la pregunta. Una vez que se detiene el timer se obtiene la respuesta que ha introducido el jugador y se la setea a la clase Jugador para que luego pueda corregirse y el jugador puede actualizar el puntaje. Una vez seteada la pregunta, se llama al método siguienteTurno() de la clase *Kahoot*. El método implementado en esta clase es el siguiente:

```
    public void siguientePregunta() throws GameOverException {
        try{
            preguntaActual = listaDePreguntas.remove();
            contador = 0;
            this.siguienteJugador();
        } catch(NoSuchElementException e){
```

```
            throw new GameOverException();
        } catch (RondaFinalizadaException e) {
        }
    }

    public void siguienteJugador() throws RondaFinalizadaException {
        if (contador == CANTIDAD_JUGADORES){
            throw new RondaFinalizadaException();
        }
        jugadorActual = jugadores.remove();
        jugadores.offer(jugadorActual);
        contador++;
    }

    public void siguienteTurno() throws GameOverException, RondaFinalizadaException {
        try{
            this.siguienteJugador();
        } catch (RondaFinalizadaException e) {
            this.siguientePregunta();
            throw new RondaFinalizadaException();
        }
    }
```

Este método puede lanzar dos excepciones: *GameOverException* que se da cuando se han mostrado las 10 preguntas o *RondaFinalizadaException*. En caso de que no se lance ninguna excepción significa que solo un jugador a respondido a la pregunta mostrada en pantalla, por lo que se cambia al siguiente jugador para que pueda responderla. Dependiendo de la excepción lanzada, el *ControladorJuego* realiza distintas acciones.

En caso de lanzarse la excepción de que la ronda ha finalizado, se corrigen las respuestas de ambos jugadores y se muestra una ventana que indica cual era la respuesta correcta a la pregunta mostrada y además indica el puntaje de cada jugador. Si la excepción lanzada es GameOver, el comportamiento es similar al de la excepción anterior, con la diferencia que en este caso se calcula quién es el ganador del juego, el cual luego se lo muestra en una nueva ventana. Esta ventana posee el botón "finalizar", con el cual se le da un fin al juego y se cierra el programa.

5.5. Clase ControladorPregunta

Esta clase es abstracta y fue creada con el fin de manejar la interacción con el usuario dependiendo el tipo de pregunta. Las clases que heredan de esta son: *ControladorVerdaderoFalso, ControladorGroupChoice y ControladorMultipleOrderedChoice*. Cada una posee un comportamiento distinto al guardar la respuesta del jugador.

En el caso de *ControladorVerdaderoFalso*, dado que solo hay dos opciones, lo que se hace es guardar la respuesta siempre en la posición inicial de un arreglo, de esta forma si el jugador se arrepiente de su respuesta, al hacer click en la opción contraria esta será la nueva respuesta y la que se pasará a corregir en caso de que el jugador haga click en el botón Continuar.

Para el *ControladorMultipleOrderedChoice* el comportamiento es distinto al controlador anterior, ya que en estos casos la lista puede poseer mas de un elemento. Lo que se realiza para guardar la respuesta del jugador esta implementado en la clase madre, ya que las preguntas de tipo *Multiple Choice y Group Choice* poseen el mismo comportamiento, por lo que este controlador no sobrescribe al método de la clase madre, el mismo se muestra a continuación:

```
public void guardarRespuesta(Button botonOpcion){
    String textoOpcion = botonOpcion.getText();
    if (respuestaJugador.contains(textoOpcion)) {
        respuestaJugador.remove(textoOpcion);
        botonOpcion.setStyle("");
    }
    else {
        respuestaJugador.add(textoOpcion);
        this.activarBoton(botonOpcion);
    }
}
```

Como se puede observar en el código, se obtiene el texto del botón presionado, ya que esta es la respuesta a la pregunta, si el texto no se encuentra en la lista se lo guarda, en caso contrario se interpreta como que el jugador se arrepiente de su elección y desea borrar la opción elegida.

Por último, el *ControladorGroupChoice* posee un funcionamiento similar al controlador anterior, con la diferencia que para guardar la respuesta el jugador debe hacer click en el botón *Grupo 1* o *Grupo 2*. Al hacer esto sucede lo siguiente:

```
@Override
public void setGrupo(Button botonGrupo){
    if (this.botonesActivos.contains(botonGrupo)){
        botonGrupo.setStyle("");
        this.botonesActivos.remove(botonGrupo);
    }else{
        this.activarBoton(botonGrupo);
    }
    this.establecerGrupo(botonGrupo.getText());
    this.refrescarOpciones();
}

private void establecerGrupo(String textGrupo){
    switch (textGrupo){
        case GRUPO_1:
            this.grupo1 = new ArrayList<>(this.respuestaJugador);
            this.respuestaJugador.clear();
            break;
        case GRUPO_2:
            this.grupo2 = new ArrayList<>(this.respuestaJugador);
            this.respuestaJugador.clear();
            break;
    }
}
```

La primera parte del código es la encargada de marcar los botones presionados con un borde verde para que el jugador sepa que cosas fueron seleccionadas. El segundo método, *establecerGrupo*, es el encargado de guardar la respuesta, una vez guardada la respuesta se limpia la lista, de esta forma si el jugador quiere borrar su opción bastará con volver a presionar el botón del respectivo grupo, o puede volver a generar una respuesta y guardarla presionando el botón.

6. Excepciones

NoTieneMultiplicadorException Dado a que solo es posible utilizar multiplicadores en preguntas con penalidad y solo una vez cada uno en el juego, es necesario avisar en las clases

que pretendían utilizar dicha funcionalidad. Se utiliza ésta excepción para identificar que los usos del multiplicador ya fueron utilizados y que por lo tanto no puede usarlo.

NoTieneExclusividadException Similar a la excepción anterior, la exclusividad de puntaje solo es funcional en las preguntas que no tengan penalidad. Se la utiliza para identificar que los usos de dicha funcionalidad ya se han terminado.

RondaFinalizadaException Se utiliza para identificar que la pregunta ya fue mostrada a los 2 jugadores y que las mismas también fueron (o no) respondidas y que se pueda mostrar la siguiente pregunta o que finalice el juego.

GameOverException Se utiliza para identificar que todas las preguntas ya fueron mostradas y respondidas y que, por lo tanto, no hay mas rondas en el juego.

7. Diagramas de secuencia

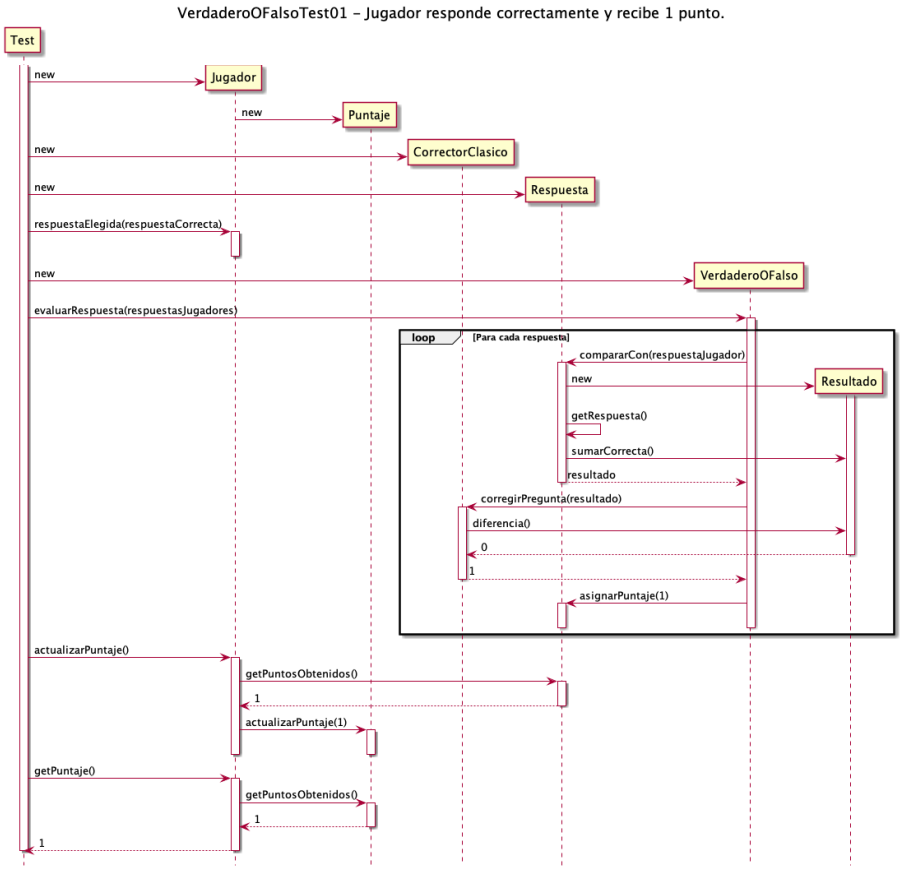

Figura 4: Diagrama de secuencia, Test de pregunta Verdadero o falso

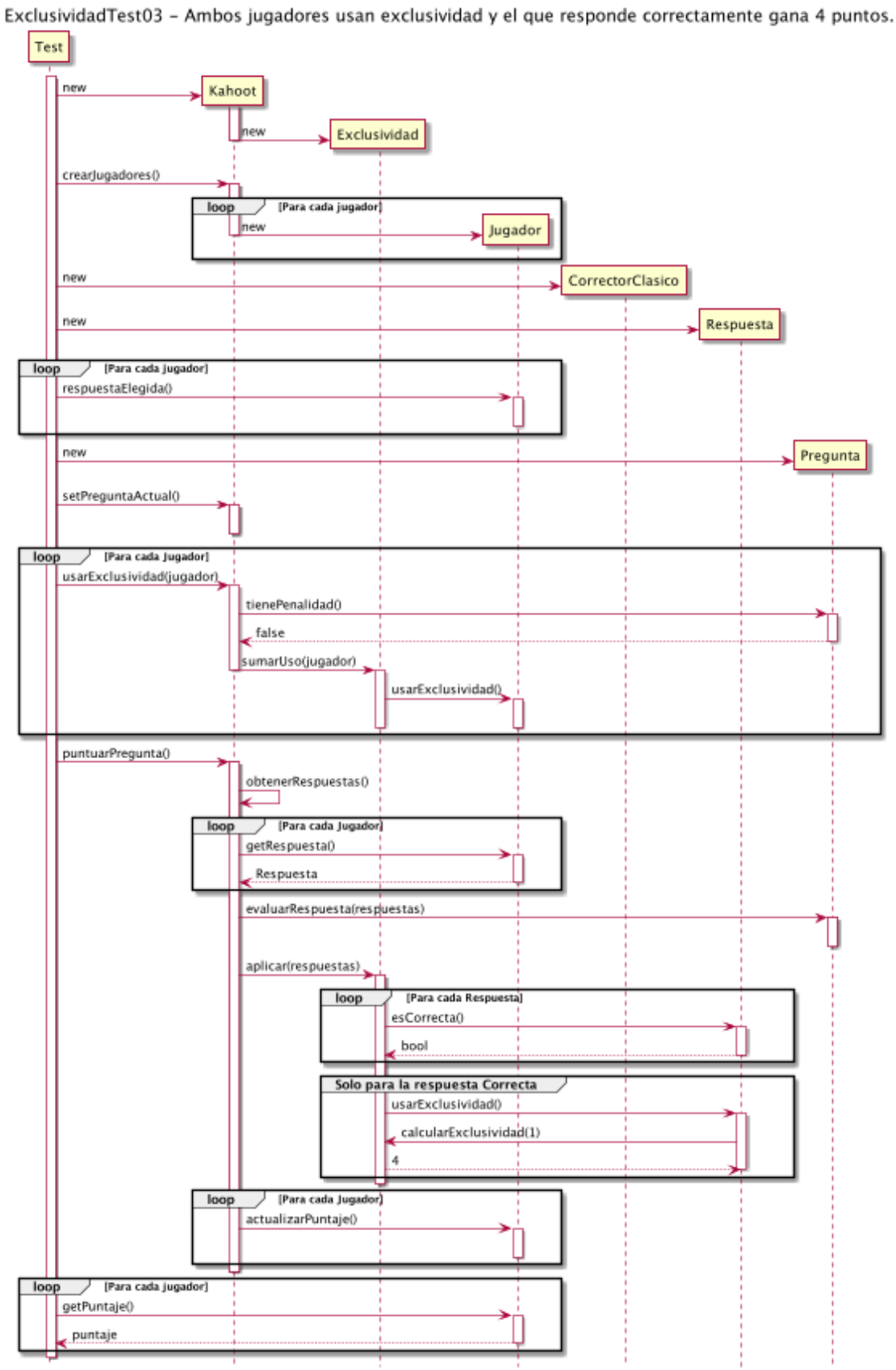

Figura 5: Diagrama de secuencia, test de exclusividad de puntaje.

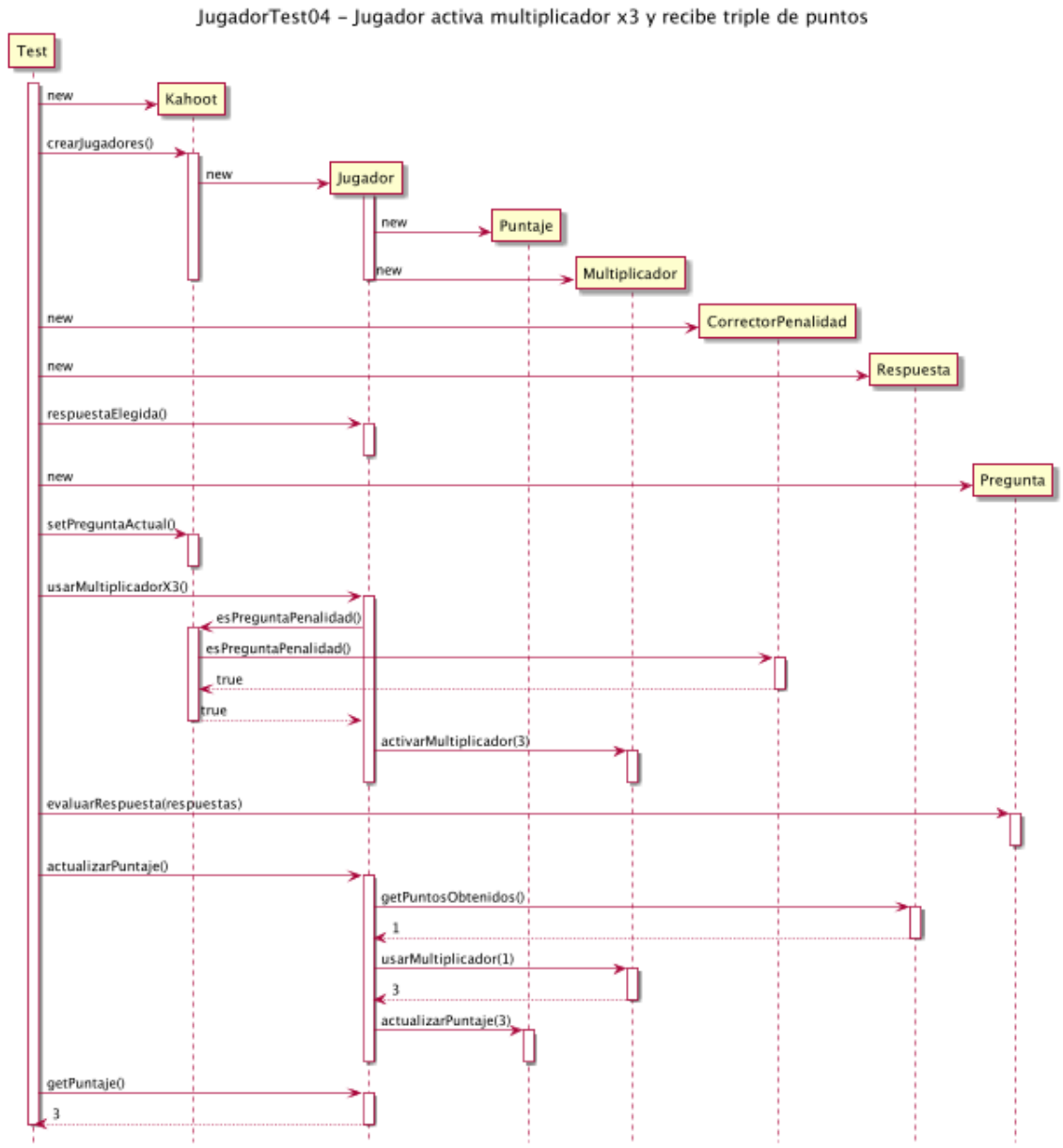

Figura 6: Diagrama de secuencia, test de multiplicador.

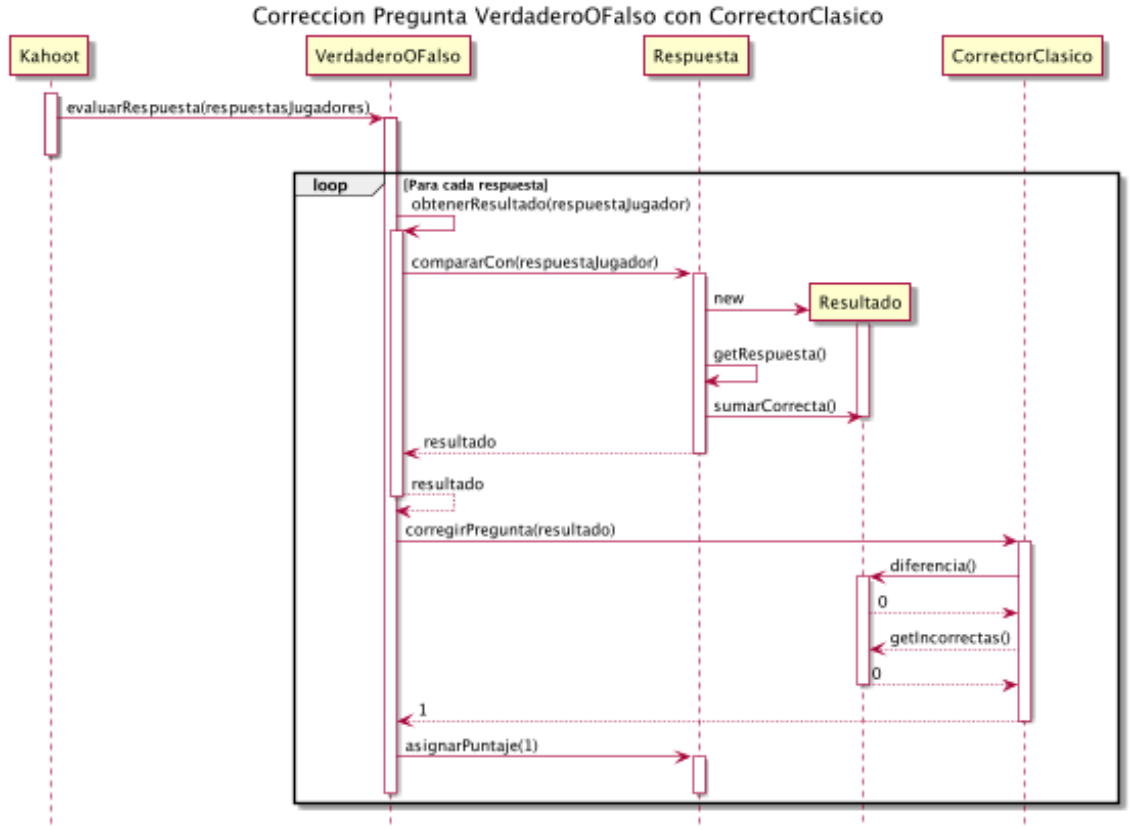

Figura 7: Diagrama de secuencia, funcionamiento de CorrectorClasico en pregunta VerdaderoFalso.

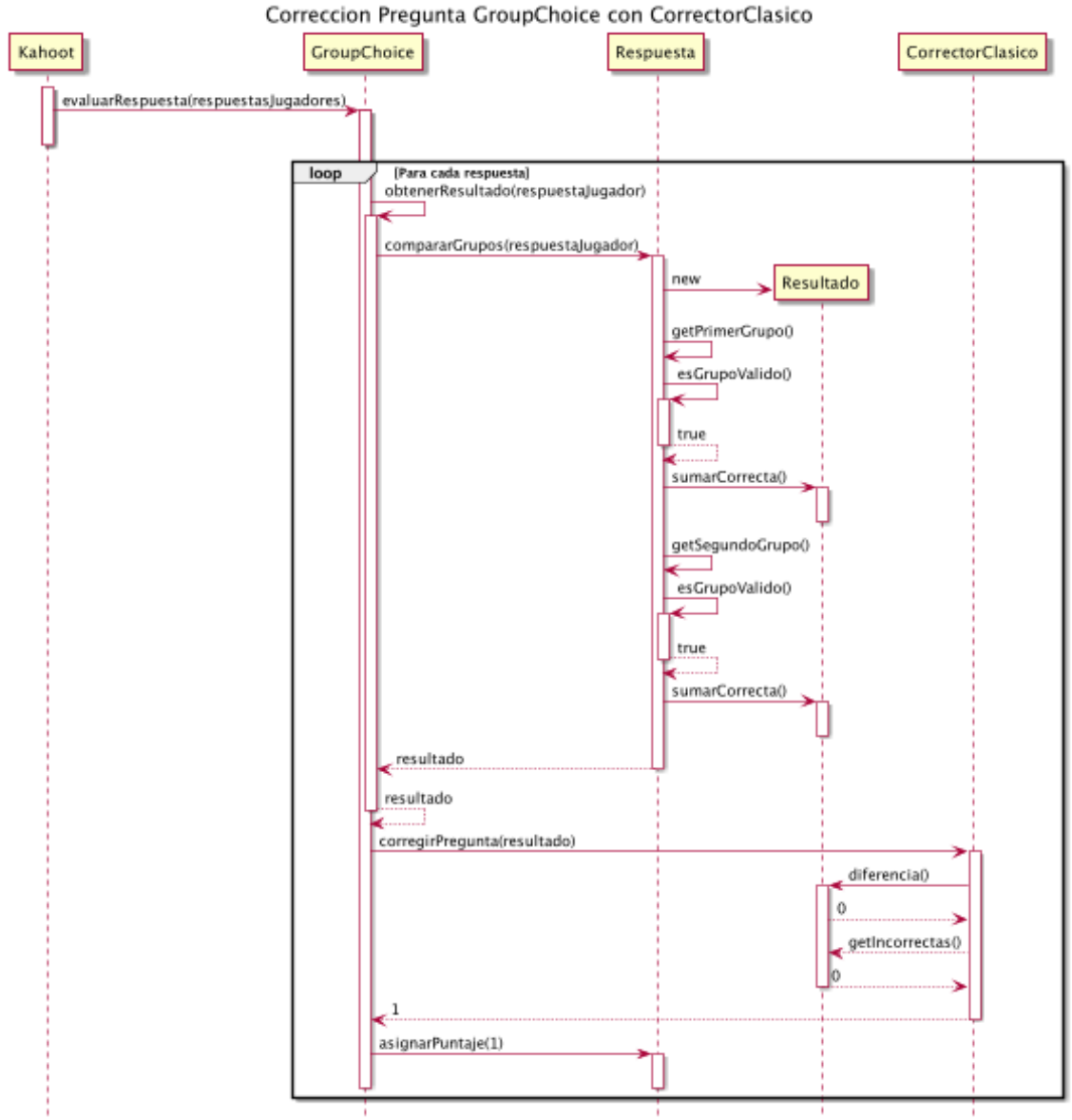

Figura 8: Diagrama de secuencia, funcionamiento de CorrectorClasico en pregunta GroupChoice.

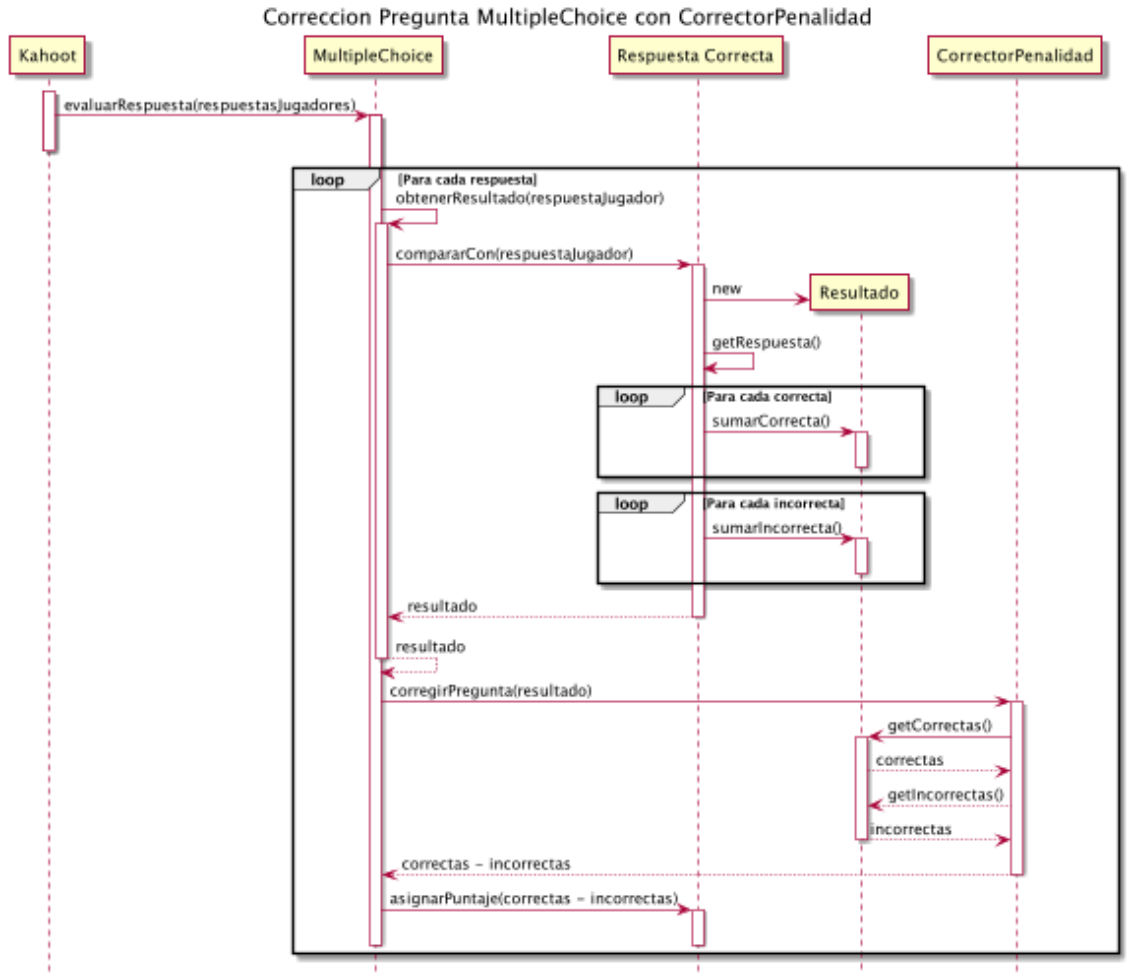

Figura 9: Diagrama de secuencia, funcionamiento de CorrectorPenalidad en pregunta Multiple-Choice.

8. Diagramas de Paquetes

A continuación se muestran los paquetes creados en el diseño del juego, para esto se utilizo el modelo vista controlador (MVC).

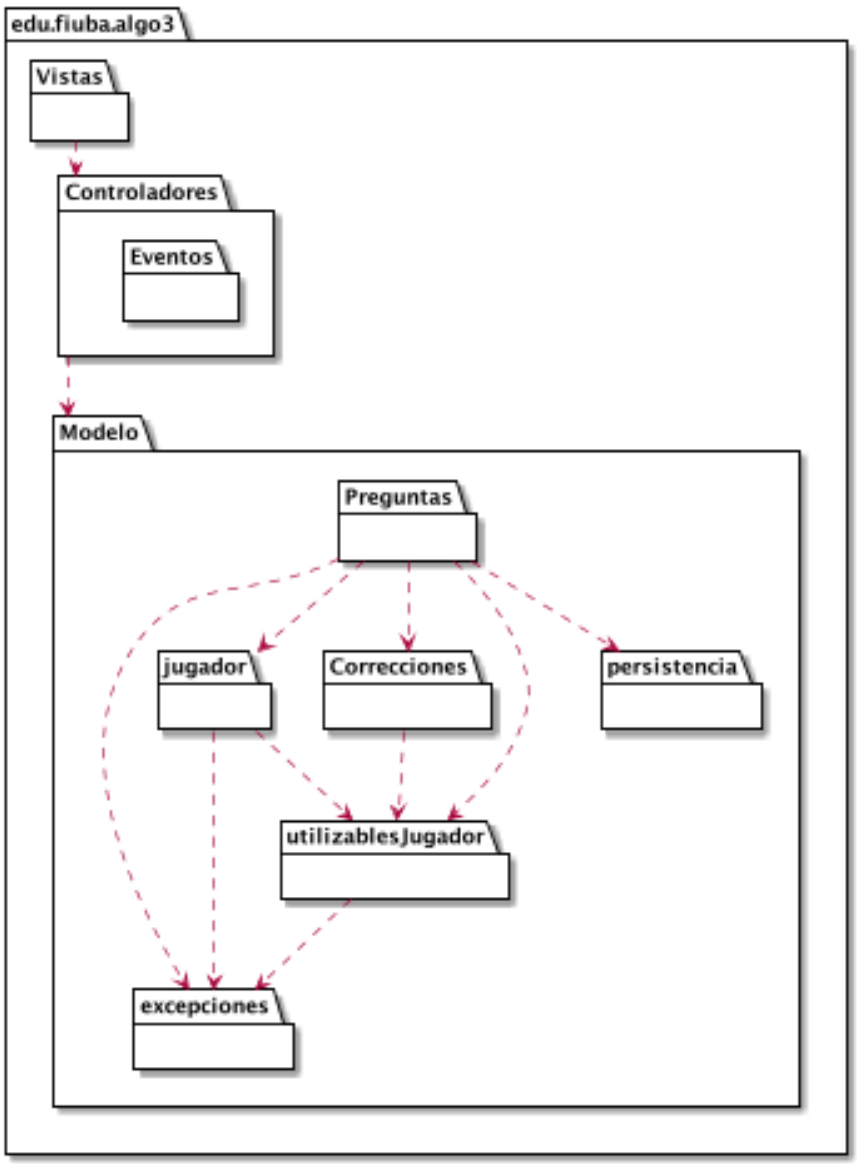

Figura 10: Diagrama de paquetes

9. Diagrama de Estados

Se muestra el diagrama de estado del juego, desde el comienzo, hasta la finalización del mismo, bien sea por que se terminan las preguntas y se obtiene un ganador o porque se presiona el botón salir en algún momento del mismo.

Figura 11: Diagrama de estados