FIUBA - 75.07

Algoritmos y programación III

Trabajo práctico 2: FonTruco

2do cuatrimestre, 2015

(trabajo grupal)

Alumnos:

Nombre	Padrón	PuertoEstelarTerranMail
Ezequiel Martin Ortega Mateo	95126	ezequiel_ortegamateo@hotmail.com
Micaela Lean Cole	96364	mleancole@icloud.com
Marcos Pozzo	96309	mpozzo94@hotmail.com

Fecha de entrega final: Miércoles 2/12/2015 - Jueves 3/12/2015

Tutor: Nicolás Paez

Nota Final:

Introducción

Objetivo del trabajo

Aplicar los conceptos enseñados en la materia a la resolución de un problema, trabajando en forma grupal y utilizando un lenguaje de tipado estático (Java)

Consigna general

Desarrollar la aplicación completa, incluyendo el modelo de clases e interfaz gráfica. La aplicación deberá ser acompañada por prueba unitarias e integrales y documentación de diseño. En la siguiente sección se describe la aplicación a desarrollar.

Descripción de la aplicación a desarrollar

Se deberá desarrollar una aplicación que implemente el juego de cartas popular Truco (https://es.wikipedia.org/wiki/Truco_(juego_de_naipes)) .

Las características del mismo serán:

- Se podrá seleccionar la variante con/sin Flor.
- El juego será entre jugadores humanos.
- Se podrá jugar en las variantes 2 jugadores, 4 jugadores y pica-pica.
- En la variante de 2 jugadores, se podrá jugar contra la computadora, con una inteligencia mínima y determinística (esto se comentará en clase).

Se desarrollará la interfaz visual para la interacción entre los jugadores.

Entregables

- Código fuente de la aplicación completa, incluyendo también: código de la pruebas, archivos de recursos
- Script para compilación y ejecución (ant)
- Informe, acorde a lo especificado en este documento

Formas de entrega

Habrá **4 entregas formales.** Las mismas tendrán una calificación de **APROBADO o NO APROBADO** en el momento de la entrega.

Aquél grupo que acumule 3 no aprobados, quedará automáticamente desaprobado con la consiguiente pérdida de regularidad en la materia. En cada entrega se debe traer el informe actualizado.

1er Entrega: *Mínimamente* (se aconseja avanzar lo más posible) pruebas de integración y unitarias funcionando que contemplan:

- Modelado de las cartas y los valores entre ellas
- Modelado de 'manos' y 'mesa'
- Modelado de cálculo envido y flor para una 'mesa' dada

2da Entrega: **Modelo del Juego Completo** con todas las pruebas unitarias y de integración que contemplen todos los casos del enunciado, simulando partidas completas en todas sus variantes.

3er Entrega: Interfaz gráfica Parcial. A determinar por el ayudante.

4ta y última Entrega: Trabajo Práctico completo funcionando, con interfaz gráfica final, sonidos e informe completo.

Fechas de entrega programadas

1er Entrega: *Miércoles 11 / 11 / 2015 - Jueves 12 / 11 / 2015*

2da Entrega: *Miércoles 18 / 11 / 2015 - Jueves 19 / 11 / 2015*

3er Entrega: *Miércoles 25 / 11 / 2015 - Jueves 26 / 11 / 2015*

4ta y última Entrega: *Miércoles 2 / 12 / 2015 - Jueves 3 / 12 / 2015*

Informe

Supuestos

1. **Flor**
   a. Para cantarla se debe tener tres cartas del mismo palo;
   b. Otorga 3 puntos al que la cantó y no puede ser rechazada;
   c. No se puede cantar dos veces flor en la misma ronda*;
   d. Se puede redoblar con contraflor o contraflor al resto;
   e. Contraflor otorga 6 puntos al ganador;
   f. Si no es querida, son 4 puntos para el que la cantó;
   g. Contraflor al resto otorga la cantidad de puntos faltantes para ganar;
   h. Si no es querida, otorga la cantidad de puntos de la instancia anterior (flor o contraflor).
2. **Envido**
   a. Se puede redoblar siempre y cuando suba la jerarquía del envido
   envido < real envido < falta envido
   Excepto en el envido-envido;
   b. Envido otorga 2 puntos al ganador y uno al que lo cantó si no es querido;
   c. Real envido otorga 3 puntos al ganador y uno al que lo cantó si no es querido;
   d. Falta envido otorga al ganador los puntos que le faltan al contrario para ganar y uno al que lo cantó si no es querido.
3. **Truco**
   a. Se puede redoblar siempre y cuando suba la jerarquía del truco
   truco < retruco < vale cuatro;
   b. Truco otorga 2 puntos al ganador y 1 al que lo cantó si no es querido;
   c. ReTruco otorga 3 puntos al ganador y 2 al que lo cantó si no es querido;
   d. Vale cuatro otorga 4 puntos al ganador y 3 al que lo cantó si no es querido.
4. **Partida**
   a. Sólo se puede cantar flor en primera (si se juega con flor);
   b. Sólo se puede cantar envido en primera y si no se cantó flor;
   c. No se puede cantar flor o envido luego del truco;
   d. En caso de cantarse truco en primera, se podrá cantar envido o flor;
   e. Flor, envido y truco deben responderse, no se puede jugar una carta en caso contrario;
   f. El jugador podrá irse al mazo en cualquier momento (se da como concluida la ronda);
   g. Si se abandona en primera son 2 puntos para el contrario, en cualquier otro caso se le otorga al contrario la cantidad de puntos de la instancia actual de truco;
   h. En caso de empatar en tantos (flor o envido) ganará el equipo que es mano***;
   i. En caso de empatar las tres manos** de la ronda, ganará el equipo que es mano;
   j. El jugador ganador de una mano arranca jugando la mano siguiente, en caso de empate, arrancará el jugador que es mano;
   k. El jugador mano va cambiando cada ronda.

*una ronda son tres manos o menos.
**se considera una mano cuando cada jugador puso en juego una carta (primera, segunda o tercera).
***se considera mano al primer jugador en recibir las cartas.

Modelo de dominio

Elegimos representar estados, con clases:
[Ver los diagramas de clases de los estados.]

Para determinar (por ejemplo) cuántos puntos sumarle al ganador del envido, se produce un intercambio de mensajes entre las clases del envido que fueron llamadas. Es decir, si se cantó envido, luego real envido y luego quiero; el estado real envido sabe que antes de ser llamada, se cantó envido. Entonces, le otorga al ganador sus puntos de estado, 3; más los puntos de los estados anteriores, 2; un total de 5. [Ver diagrama del envido compuesto.]

Esta misma lógica llevamos a cabo para calcular los puntos del truco, la flor, y las rondas.
En el caso de las rondas, la situación a determinar, es si la mano terminó o no: emparda la primera, gana la segunda; emparda la tercera, gana la primera; y todas los casos posibles.

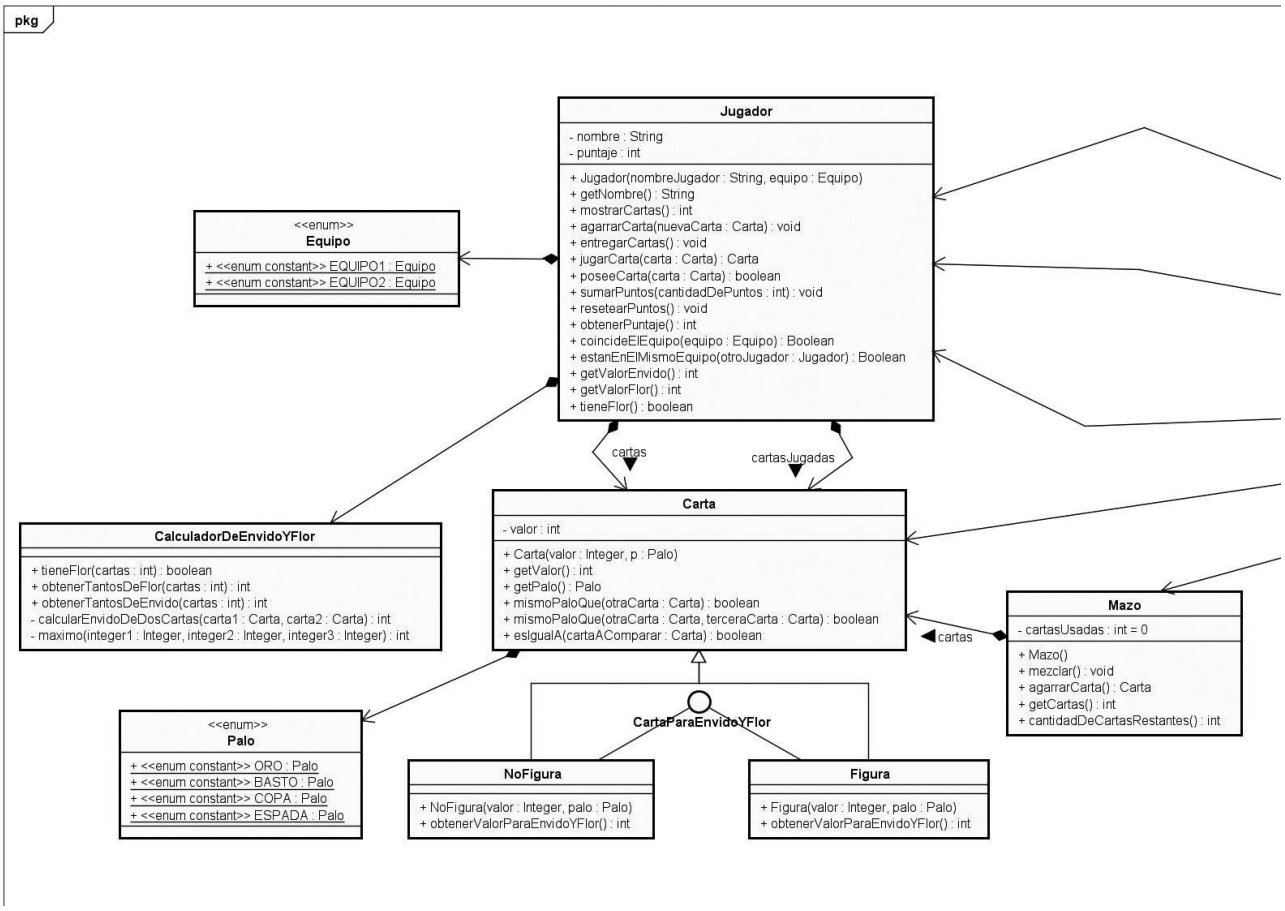
El **CalculadorDeEnvidoYFlor** tiene como responsabilidad calcular los tantos del envido y de la flor del jugador, basándose en las cartas que el mismo posee. No conoce al jugador, sólo sus cartas. Cada jugador para calcular sus tantos le pasa sus cartas al calculador, y éste, sin guardar ni modificarlas, devuelve el tanto que posee.

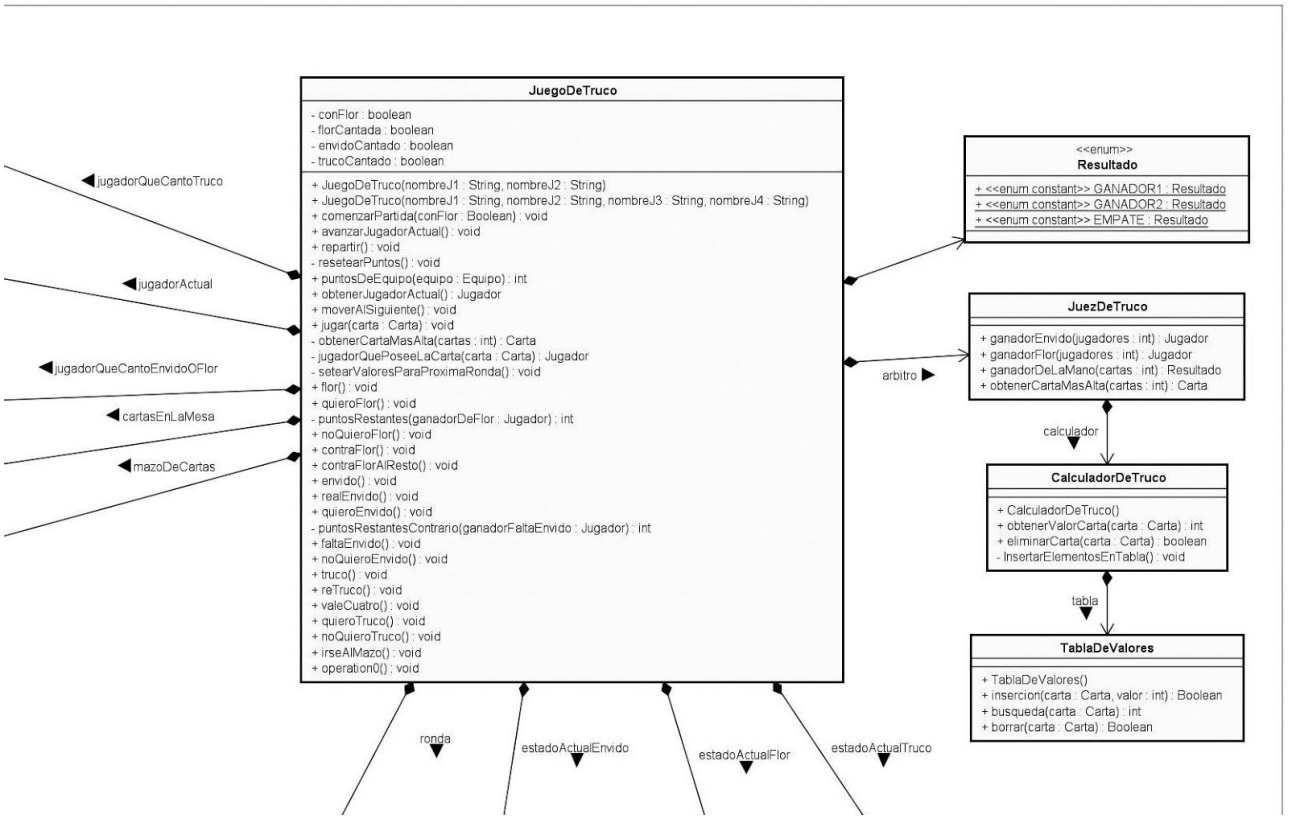
El **CalculadorDeTruco** se encarga de devolver el valor relativo de las cartas. El valor relativo es utilizado para saber qué carta gana.

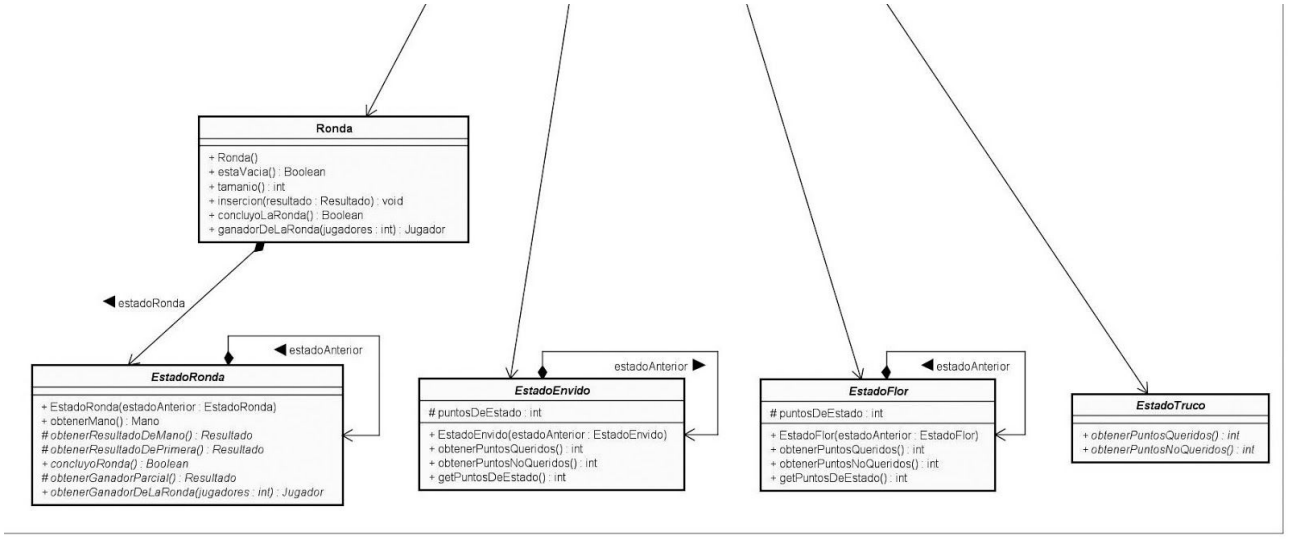
El **JuezDeTruco** se encarga de calcular el ganador de cada mano, del envido y de la flor. En el caso del envido y flor devuelve al ganador para qué **JuegoDeTruco** le asigne los puntos al ganador. En el caso del ganador de la mano devuelve un enumerado que indica que equipo ganó y dicho resultado se le pasa a **Ronda**.

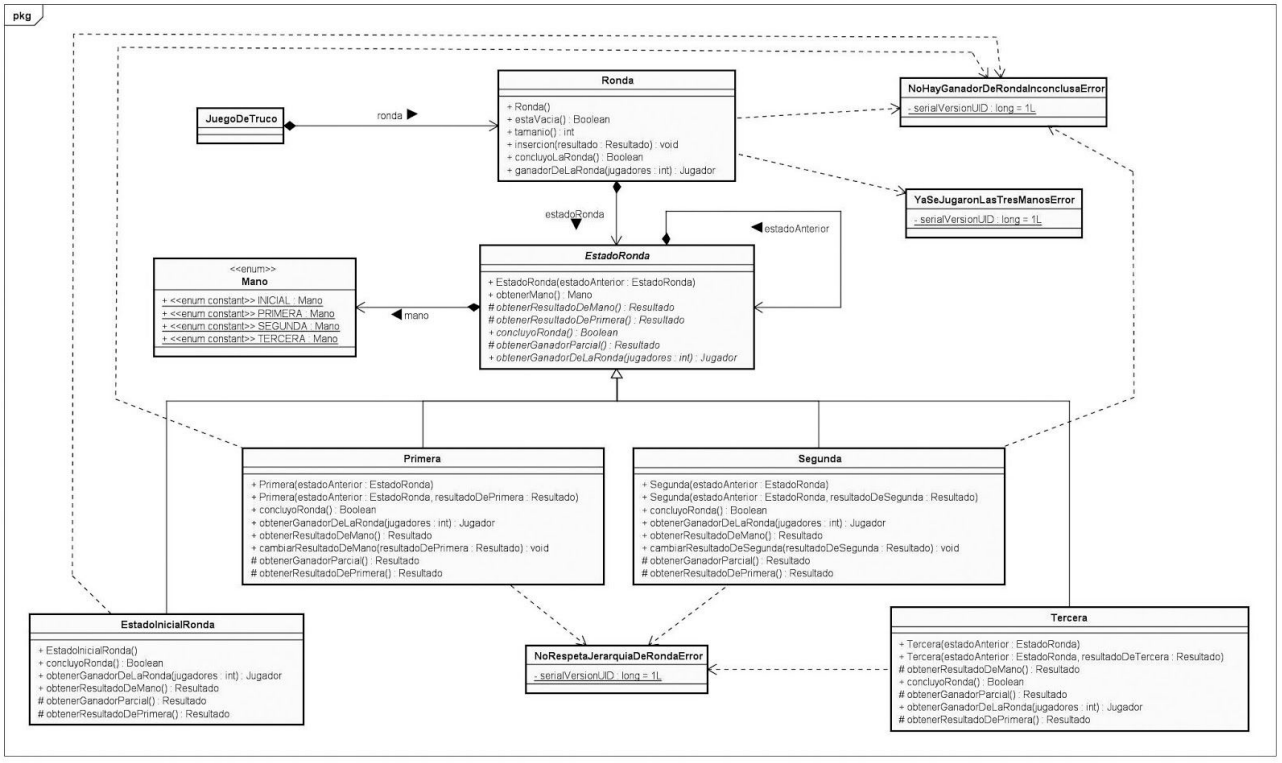
La **ListaCircular** se encarga de tener siempre en la primer posición al jugador que es mano.

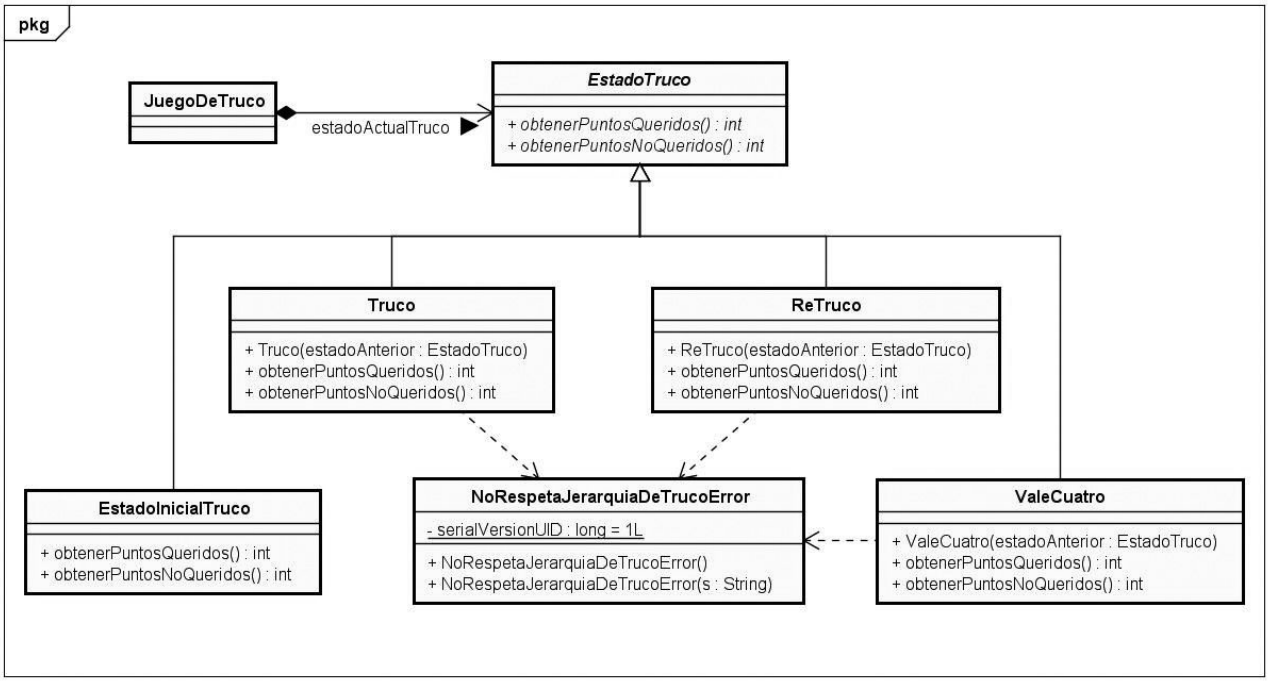
Diagramas de clases

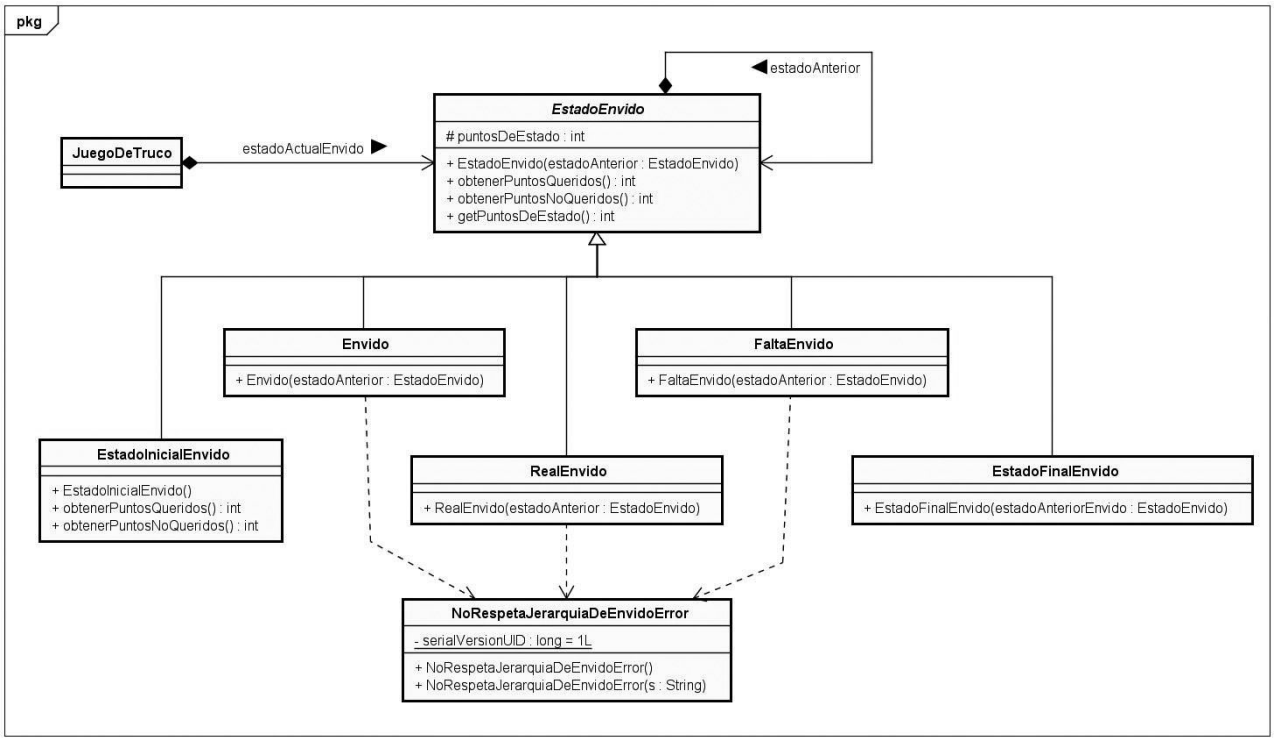


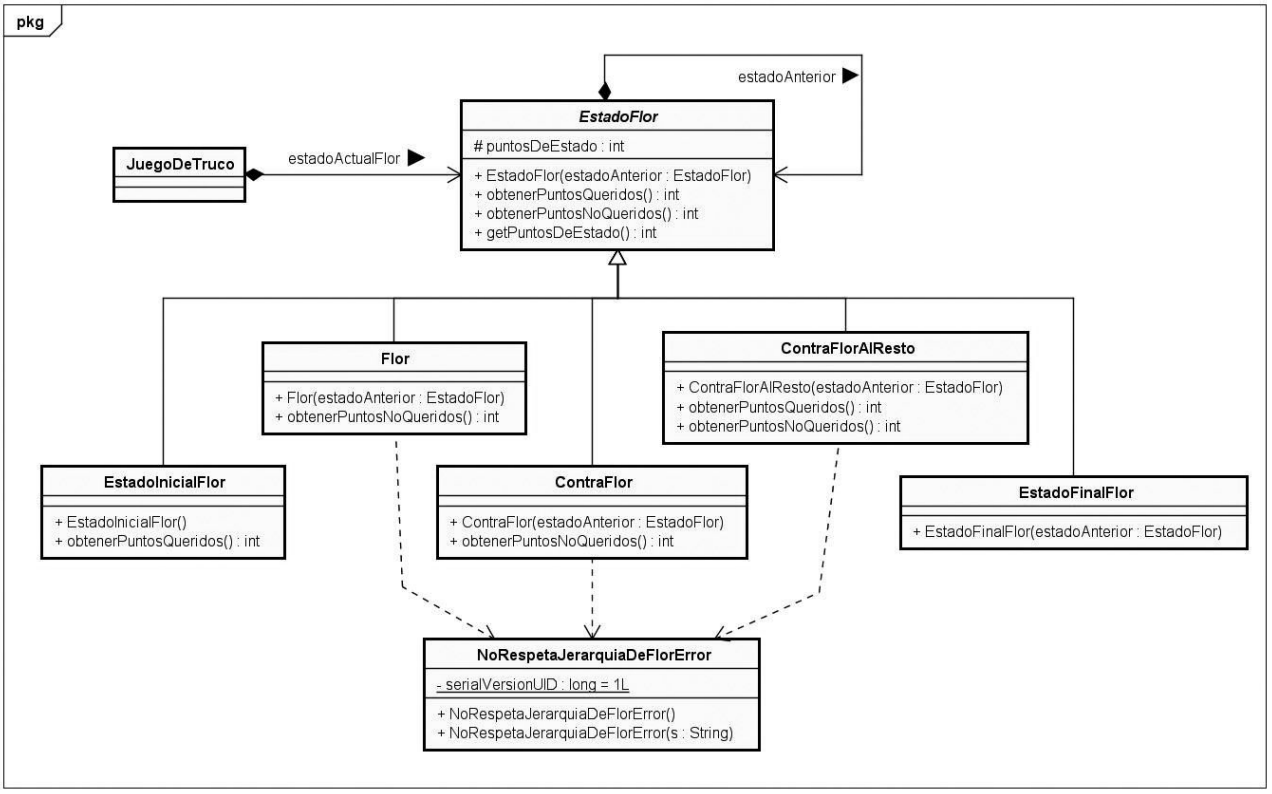



Ronda
+ Ronda()
+ estaVacia() : Boolean
+ tamanio() : int
+ insercion(resultado : Resultado) : void
+ concluyoLaRonda() : Boolean
+ ganadorDeLaRonda(jugadores : int) : Jugador
estadoRonda
estadoAnterior
EstadoRonda
+ EstadoRonda(estadoAnterior : EstadoRonda)
+ obtenerMano() : Mano
# obtenerResultadoDeMano() : Resultado
# obtenerResultadoDePrimera() : Resultado
+ concluyoRonda() : Boolean
# obtenerGanadorParcial() : Resultado
+ obtenerGanadorDeLaRonda(jugadores : int) : Jugador
estadoAnterior
EstadoEnvido
puntosDeEstado : int
+ EstadoEnvido(estadoAnterior : EstadoEnvido)
+ obtenerPuntosQueridos() : int
+ obtenerPuntosNoQueridos() : int
+ getPuntosDeEstado() : int
estadoAnterior
EstadoFlor
puntosDeEstado : int
+ EstadoFlor(estadoAnterior : EstadoFlor)
+ obtenerPuntosQueridos() : int
+ obtenerPuntosNoQueridos() : int
+ getPuntosDeEstado() : int
EstadoTruco
+ obtenerPuntosQueridos() : int
+ obtenerPuntosNoQueridos() : int

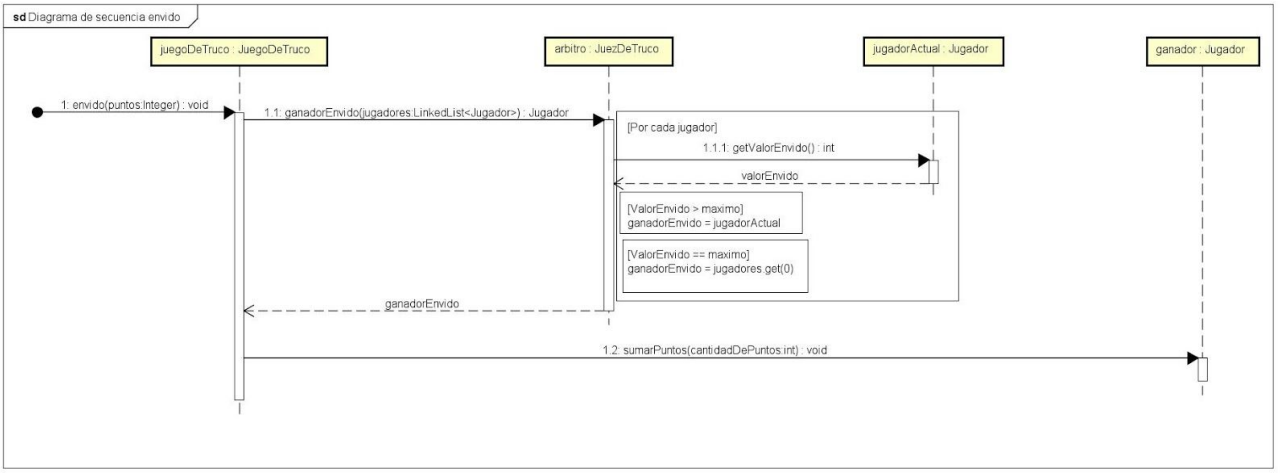

pkg
JuegoDeTruco
ronda
Ronda
+ Ronda()
+ estaVacia() : Boolean
+ tamanio() : int
+ insercion(resultado : Resultado) : void
+ concluyoLaRonda() : Boolean
+ ganadorDeLaRonda(jugadores : int) : Jugador
NoHayGanadorDeRondaInconclusaError
- serialVersionUID : long = 1L
YaSeJugaronLasTresManosError
- serialVersionUID : long = 1L
estadoRonda
estadoAnterior
<<enum>>
Mano
+ <<enum constant>> INICIAL : Mano
+ <<enum constant>> PRIMERA : Mano
+ <<enum constant>> SEGUNDA : Mano
+ <<enum constant>> TERCERA : Mano
mano
EstadoRonda
+ EstadoRonda(estadoAnterior : EstadoRonda)
+ obtenerMano() : Mano
# obtenerResultadoDeMano() : Resultado
# obtenerResultadoDePrimera() : Resultado
+ concluyoRonda() : Boolean
# obtenerGanadorParcial() : Resultado
+ obtenerGanadorDeLaRonda(jugadores : int) : Jugador
Primera
+ Primera(estadoAnterior : EstadoRonda)
+ Primera(estadoAnterior : EstadoRonda, resultadoDePrimera : Resultado)
+ concluyoRonda() : Boolean
+ obtenerGanadorDeLaRonda(jugadores : int) : Jugador
+ obtenerResultadoDeMano() : Resultado
+ cambiarResultadoDeMano(resultadoDePrimera : Resultado) : void
# obtenerGanadorParcial() : Resultado
# obtenerResultadoDePrimera() : Resultado
Segunda
+ Segunda(estadoAnterior : EstadoRonda)
+ Segunda(estadoAnterior : EstadoRonda, resultadoDeSegunda : Resultado)
+ concluyoRonda() : Boolean
+ obtenerGanadorDeLaRonda(jugadores : int) : Jugador
+ obtenerResultadoDeMano() : Resultado
+ cambiarResultadoDeSegunda(resultadoDeSegunda : Resultado) : void
# obtenerGanadorParcial() : Resultado
# obtenerResultadoDePrimera() : Resultado
EstadoInicialRonda
+ EstadoInicialRonda()
+ concluyoRonda() : Boolean
+ obtenerGanadorDeLaRonda(jugadores : int) : Jugador
+ obtenerResultadoDeMano() : Resultado
# obtenerGanadorParcial() : Resultado
# obtenerResultadoDePrimera() : Resultado
NoRespetaJerarquiaDeRondaError
- serialVersionUID : long = 1L
Tercera
+ Tercera(estadoAnterior : EstadoRonda)
+ Tercera(estadoAnterior : EstadoRonda, resultadoDeTercera : Resultado)
# obtenerResultadoDeMano() : Resultado
+ concluyoRonda() : Boolean
# obtenerGanadorParcial() : Resultado
+ obtenerGanadorDeLaRonda(jugadores : int) : Jugador
# obtenerResultadoDePrimera() : Resultado

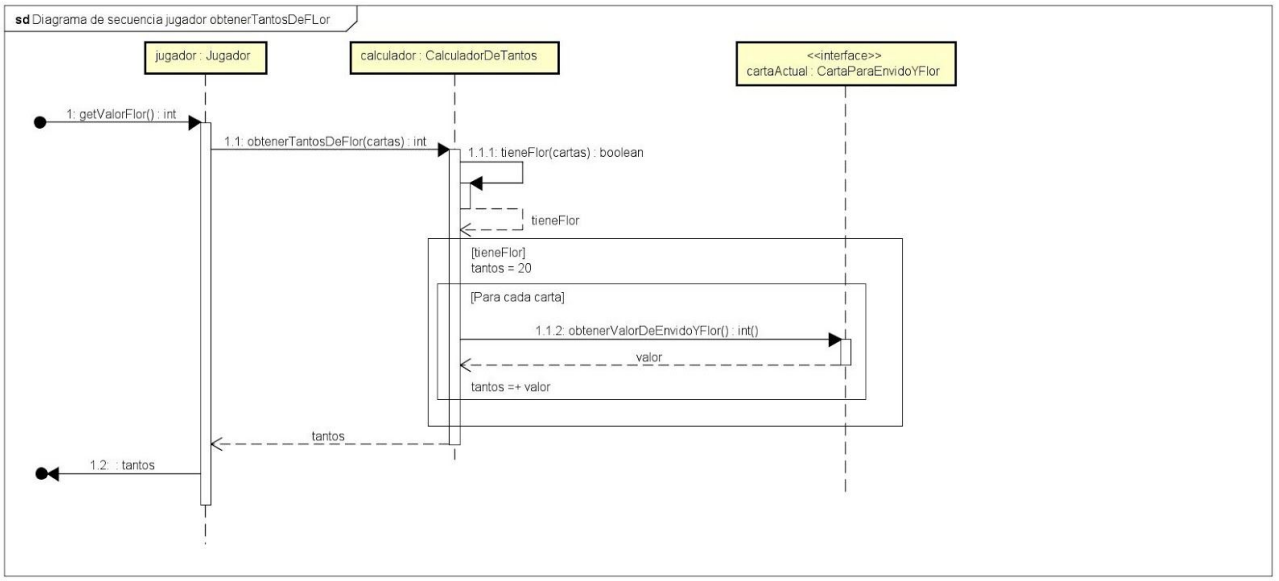

pkg
JuegoDeTruco
estadoActualTruco
EstadoTruco
+ obtenerPuntosQueridos() : int
+ obtenerPuntosNoQueridos() : int
Truco
+ Truco(estadoAnterior : EstadoTruco)
+ obtenerPuntosQueridos() : int
+ obtenerPuntosNoQueridos() : int
ReTruco
+ ReTruco(estadoAnterior : EstadoTruco)
+ obtenerPuntosQueridos() : int
+ obtenerPuntosNoQueridos() : int
EstadoInicialTruco
+ obtenerPuntosQueridos() : int
+ obtenerPuntosNoQueridos() : int
NoRespetaJerarquiaDeTrucoError
- serialVersionUID : long = 1L
+ NoRespetaJerarquiaDeTrucoError()
+ NoRespetaJerarquiaDeTrucoError(s : String)
ValeCuatro
+ ValeCuatro(estadoAnterior : EstadoTruco)
+ obtenerPuntosQueridos() : int
+ obtenerPuntosNoQueridos() : int

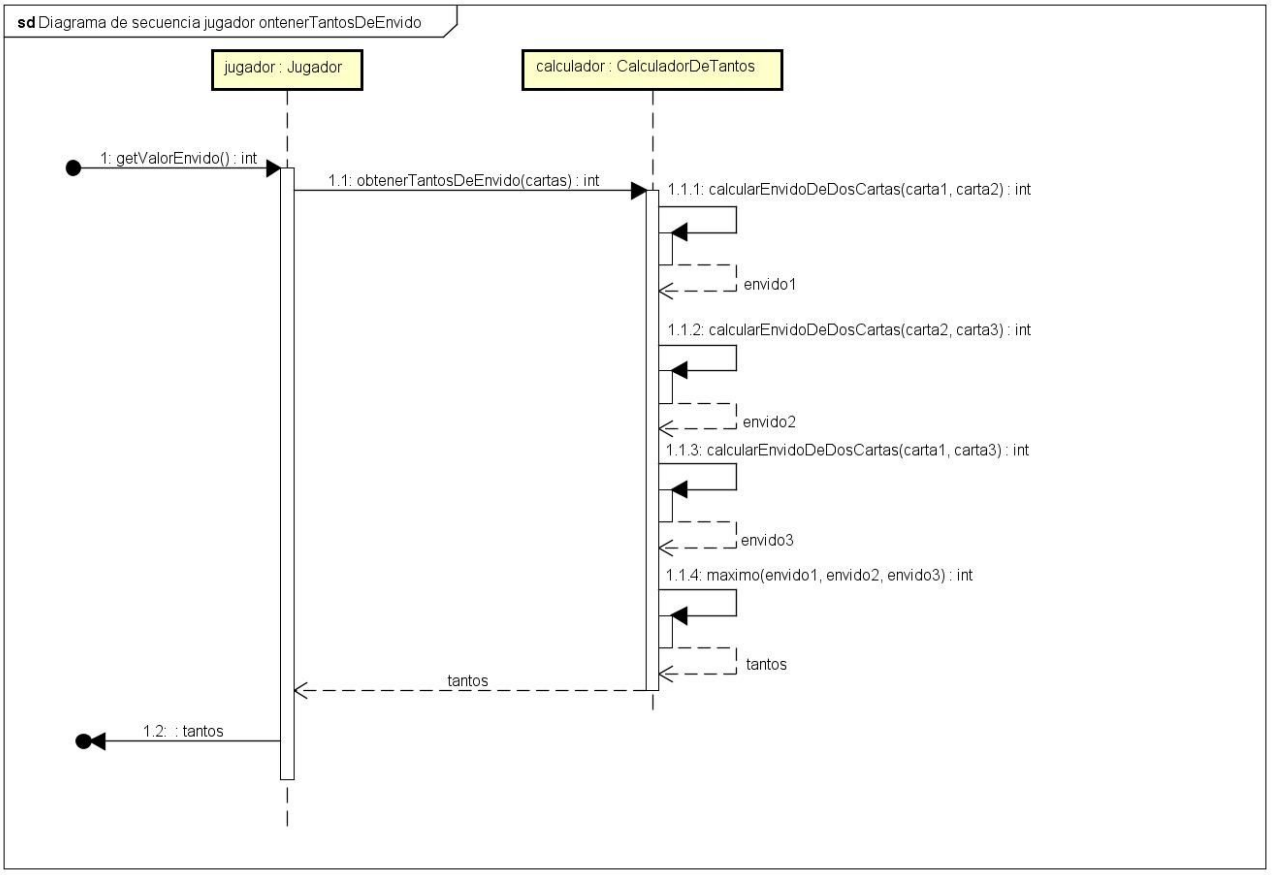

pkg
estadoAnterior
JuegoDeTruco
estadoActualEnvido
EstadoEnvido
puntosDeEstado : int
+ EstadoEnvido(estadoAnterior : EstadoEnvido)
+ obtenerPuntosQueridos() : int
+ obtenerPuntosNoQueridos() : int
+ getPuntosDeEstado() : int
Envido
+ Envido(estadoAnterior : EstadoEnvido)
FaltaEnvido
+ FaltaEnvido(estadoAnterior : EstadoEnvido)
EstadoInicialEnvido
+ EstadoInicialEnvido()
+ obtenerPuntosQueridos() : int
+ obtenerPuntosNoQueridos() : int
RealEnvido
+ RealEnvido(estadoAnterior : EstadoEnvido)
EstadoFinalEnvido
+ EstadoFinalEnvido(estadoAnteriorEnvido : EstadoEnvido)
NoRespetaJerarquiaDeEnvidoError
- serialVersionUID : long = 1L
+ NoRespetaJerarquiaDeEnvidoError()
+ NoRespetaJerarquiaDeEnvidoError(s : String)

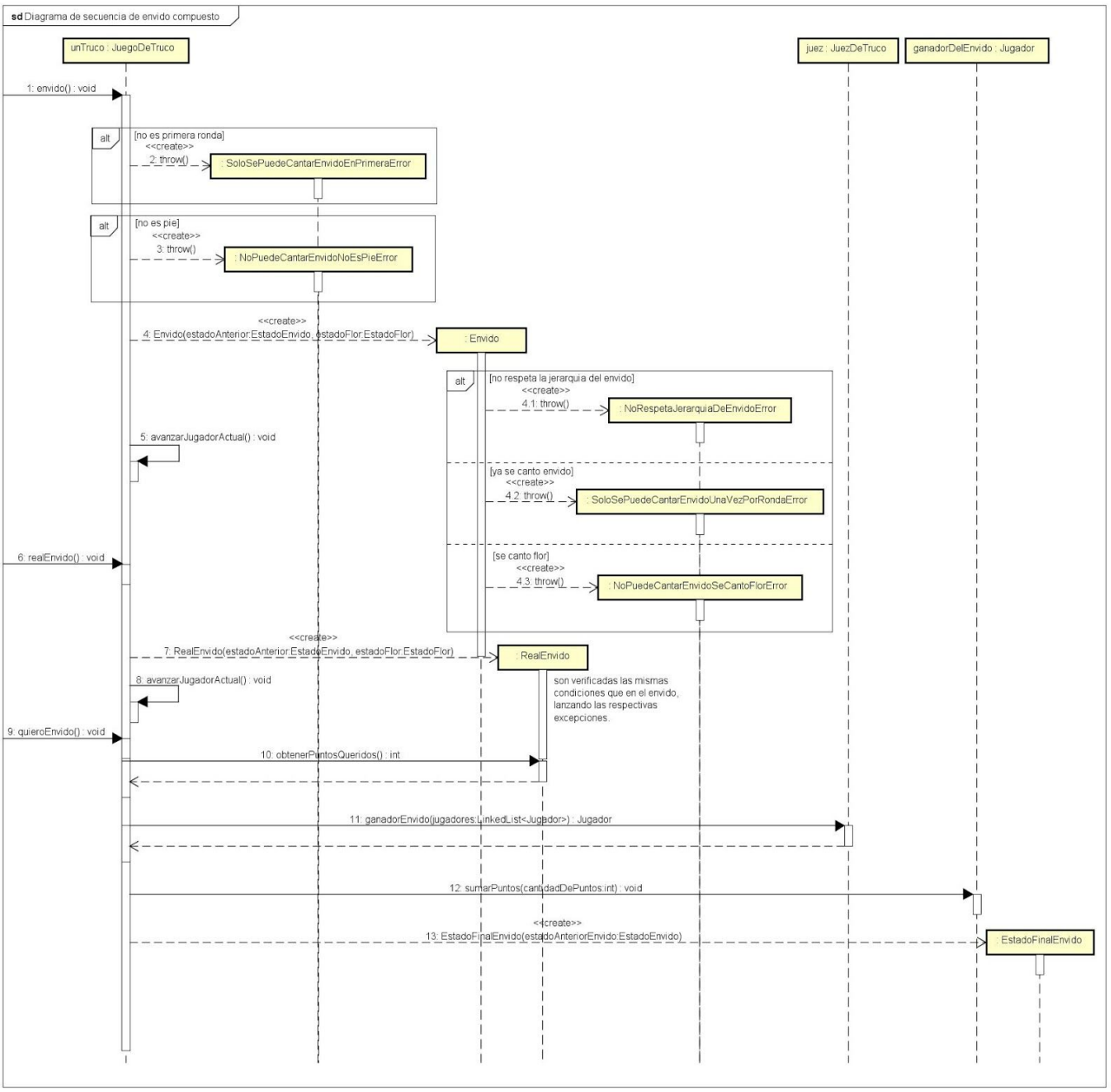


Diagramas de secuencia




sd Diagrama de secuencia jugador obtenerTantosDeFLor
jugador : Jugador
calculador : CalculadorDeTantos
<<interface>>
cartaActual : CartaParaEnvidoYFlor
1: getValorFlor() : int
1.1: obtenerTantosDeFlor(cartas) : int
1.1.1: tieneFlor(cartas) : boolean
tieneFlor
[tieneFlor]
tantos = 20
[Para cada carta]
1.1.2: obtenerValorDeEnvidoYFlor() : int()
valor
tantos =+ valor
tantos
1.2: : tantos


sd Diagrama de secuencia jugador ontenerTantosDeEnvido
jugador : Jugador
calculador : CalculadorDeTantos
1: getValorEnvido() : int
1.1: obtenerTantosDeEnvido(cartas) : int
1.1.1: calcularEnvidoDeDosCartas(carta1, carta2) : int
envido1
1.1.2: calcularEnvidoDeDosCartas(carta2, carta3) : int
envido2
1.1.3: calcularEnvidoDeDosCartas(carta1, carta3) : int
envido3
1.1.4: maximo(envido1, envido2, envido3) : int
tantos
tantos
1.2: : tantos



Nota: utilizamos este esquema como mecanismo para el intercambio de mensajes entre estados de truco, flor, y ronda.

Detalles de implementación

Implementamos una lista circular para saber qué jugador es mano, y avanza al siguiente luego de cada ronda. Esto simula la manera en que se juega al truco en la realidad, ya que todos los jugadores se sientan en forma circular, siempre con alguien a su izquierda y alguien a su derecha.

Para sumar correctamente los tantos del envido y flor, creamos las clases **Figura** y **NoFigura**, que heredan de **Carta**. De esta forma, las figuras suman cero en el envido y flor; y el resto suma su valor de la carta.

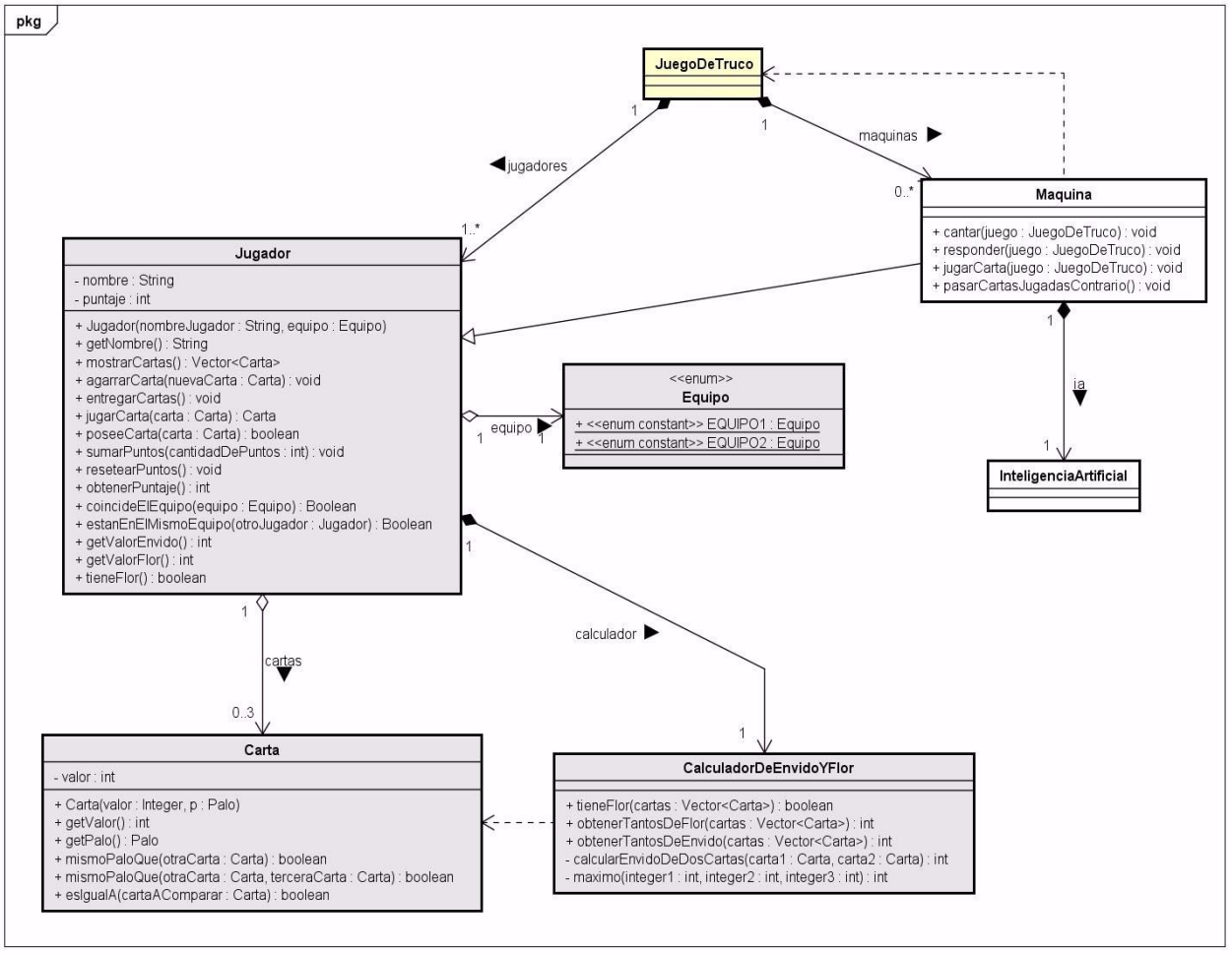
Para saber qué carta es más alta que las demás, creamos la clase **TablaDeValores**, en la cual se le pueden agregar las cartas junto con un valor relativo para cada una. Al pedirle que busque una carta devuelve el valor relativo asignado.

Para el cálculo de puntos que se le deben dar al jugador ganador resultante de un enfrentamiento parcial de envido o truco, usamos las clases para representar los estados correspondientes. [Ver en los diagramas de clase].

Para saber si la ronda ya había concluído y el ganador de la misma, creamos **Ronda**. En la cual se almacenan los resultados de las diferentes manos jugadas. En el caso de que la ronda haya concluido, se le puede pedir el ganador de la misma.

Las clases Estado**Truco**, Estado**Flor** y Estado**Envido** tienen, entres sus clases hijas, un estado inicial. El cual se utiliza cuando no se ha cantado nada. El mismo se utiliza para inicializar los estados de truco, envido y flor. A su vez, las clases EstadoFlor y EstadoEnvido tienen estados finales, ya que luego de verificar quién es el ganador de los mismos, la partida sigue su curso (a diferencia del truco). Una vez finalizada la ronda, los estados se setean para la próxima ronda (estados iniciales).

Inteligencia artificial



La clase **Maquina** es la que tiene un tributo con la inteligencia artificial. Hereda de **Jugador**, así que se puede manipular como un jugador más.

Cada vez que sea el turno de la máquina, ya sea para responder, cantar o jugar, se le pasarán las cartas jugadas por el contrario. invocando el método **pasarCartasJugadasContrario(cartas)**.

Cuando un jugador canta, ya sea truco, envido o flor, se le dirá a la máquina que responda a través del método **responder(JuegoDeTruco)**. En este método, usando la IA y las cartas jugadas por el contrario, la máquina decidirá si quiere, no quiere o redobla la apuesta.

Cuando sea el turno de la máquina, se le dirá que cante y que juegue. Va a cantar, si es que lo cree conveniente, cuando se invoca el método **cantar(JuegoDeTruco)**. Y va a jugar cuando se invoque el método **jugar(JuegoDeTruco)**.

En todos los métodos de **Maquina** se utiliza la **IA**, salvo cuando se le pasan las cartas jugadas por el contrario.

Para implementar la IA de la máquina se nos ocurrió que se podría utilizar un diseño de redes neuronales, de tipo Hopfield para ser más específico. Este tipo de redes neuronales implementa una memoria asociativa. Se puede entrenar con una serie de patrones. Luego de entrenarla, se les puede mostrar un patrón diferente y ésta buscará al patrón más parecido con los cuales se la entrenó para saber cómo accionar. *(Practical AI programing with java 3rd edition - Mark Watson)*

Excepciones

Las excepciones que creamos son lanzadas desde muchas de las clases modeladas. Éstas representan situaciones que violan las reglas del juego de truco.

Todas las excepciones son capturadas en el control de la interfaz gráfica. Cada vez que el usuario intente realizar una acción no permitida en el juego de truco, una nueva ventana de alerta es mostrada por pantalla; informándole al usuario lo sucedido.

Algunas de las excepciones creadas:

CantidadDeCartasInvalidaError: es lanzada cuando un jugador ya tiene 3 cartas y pide otra.
JugadorNoTieneFlorError: es lanzada cuando se canta flor sin tener 3 cartas del mismo palo.
NoHayMasCartasError: es lanzada si un jugador pide una carta y no quedan más en el mazo.
SoloSePuedeCantarEnPrimeraError: es lanzada si un jugador pretende cantar envido o flor en una fase donde no está permitido.
ValorDeCartaInvalidoError: es lanzada si la carta a crear no es válida en el truco.
NoRespetaJerarquiaDeEnvidoError: es lanzada si se violan las reglas del diálogo del envido.
NoRespetaJerarquiaDeTrucoError: es lanzada si se violan las reglas del diálogo del truco.
NoRespetaJerarquiaDeFlorError: es lanzada si se violan las reglas del diálogo del envido.
NoRespetaJerarquiaDeRondaError: es lanzada si se violan las reglas del diálogo del envido.
NoPuedeCantarTrucoSeCantoEnvidoError: es lanzada cuando se canta truco pero previamente se cantó envido y no fue respondido.
NoPuedeJugarSeCantoEnvidoError: es lanzada cuando un jugador quiere poner en juego una carta sin responder el envido.

NoPuedeJugarSeCantoTrucoError: es lanzada cuando un jugador quiere poner en juego una carta sin responder el truco.
YaSeJugaronLasTresManosError: es lanzada cuando se le quiere agregar a **Ronda** un resultado de una mano y ya tiene 3 resultados de diferentes manos.
NoHayGanadorDeRondaInconclusaError: es lanzada cuando la ronda no concluyó y se le pide a **Ronda** que entregue al ganador de la misma.
SoloSePuedeCantarEnvidoUnaVezPorRondaError: es lanzada cuando una vez finalizado el envido, un jugador lo vuelve a cantar en la misma ronda.

Checklist de corrección

Esta sección es para uso exclusivo de los docentes, por favor no modificar.

Carpeta

Generalidades

- ¿Son correctos los supuestos y extensiones?
- ¿Es prolija la presentación? (hojas del mismo tamaño, numeradas y con tipografía uniforme)

Modelo

- ¿Está completo?¿Contempla la totalidad del problema?
- ¿Respeta encapsulamiento?
- ¿Hace un buen uso de excepciones?
- ¿Utiliza polimorfismo en las situaciones esperadas?

Diagramas

Diagrama de clases

- ¿Está completo?
- ¿Está bien utilizada la notación?

Diagramas de secuencia

- ¿Está completo?
- ¿Es consistente con el diagrama de clases?
- ¿Está bien utilizada la notación?

Diagrama de estados

- ¿Está completo?
- ¿Está bien utilizada la notación?

Código

Generalidades

- ¿Respeta estándares de codificación?
- ¿Está correctamente documentado?