FIUBA - 75.07

Algoritmos y programación III

Trabajo práctico 2: FonTruco

2do cuatrimestre, 2015

(trabajo grupal)

Integrantes:

Nombre	Padrón	Mail
Sebastián Outeiro	92108	sebas_out@hotmail.com
Arnold Colque	94181	arnoldclq3@gmail.com
Andres Cauci	91073	andres.cauci@gmail.com
Juan Manuel Zaragoza	92308	juanmanuelzar@gmail.com

Tutor: Gabriel Falcone

Nota Final:

Correcciones:

Informe

Supuestos

- Cuando se juega con flor y se canta Flor no se puede cantar Envido.
- No se puede cantar contra flor al resto sin que el jugador anterior haya cantado flor.
- No se puede cantar la cantidad de flor que tiene el jugador sin haber aceptado el canto de flor.
- No se puede cantar la cantidad de Envido que tiene el jugador sin haber aceptado el canto de Envido.
- No se puede cantar la cantidad de Envido que tiene el Jugador más de una vez.
- Un jugador no puede cantar un canto más alto sin que el jugador anterior haya cantado un canto de un nivel inferior (ej. no se puede cantar retruco sin que antes algún jugador haya cantado truco).
- No se puede cantar Falta Envido o Contra Flor al Resto si no están instanciados los Equipos.
- No se puede agregar el mismo Jugador a un Equipo dos veces.
- No se puede sumar los puntos de un Jugador o saber qué equipo perdió si no están instanciados los Equipos.
- No se puede iniciar el juego si los Equipos no tienen todos sus jugadores.
- Un juego ya iniciado no se puede volver a configurar.
- No se puede volver a tirar una Carta que ya está jugada.
- Un jugador siempre está asociado a una mesa.
- Un jugador no puede sumar los puntos de la flor si no tiene flor.
- Un jugador no puede tomar más de tres cartas del mazo.
- No se puede obtener un Jugador ganador hasta que no se termine la mano.
- Si hubo empate con las manos gana el equipo que tiene el jugador que es mano dentro de la ronda.
- No se puede jugar una carta dentro de una mano terminada.
- No se puede cantar Flor si se inició el juego con el modo “Sin Flor”.
- Un jugador que “se va al mazo” no puede seguir jugando.
- No puede jugar una carta o cantar un jugador que no sea su turno.
- No se puede obtener un equipo ganador hasta que no se termine la ronda.
- El jugador no puede jugar una carta si la ronda está terminada.
- Solo se puede cantar Envido en la primera mano.
- El que puede efectuar el Canto “Envido” es el jugador que es Pie de la Primer Mano.

Modelo de dominio

A continuación se expone las **entidades principales** del modelo de dominio que pensamos para modelar el juego de cartas Truco Argentino.

Clase	Carta
Responsabilidades:	1º) Almacenar el valor y el palo que representa dicha carta.
	2º) Poder comprarse con otra carta y distinguir cuál de ambas tiene un valor más fuerte en la escala de truco.

Para almacenar el palo se tomó la decisión de utilizar un enumerado, el cual brinda la posibilidad de tener una única instancia en el programa para cada valor del mismo, representando así de manera única los cuatro palos como objetos gracias a los enumerados de Java.

La clase Carta tiene como atributo de clase un objeto del tipo “EscalaDeCartas” al cual le delega la responsabilidad de poder identificar dada dos cartas cuál es la más fuerte en valor para el Truco, o incluso identificar si ambas tienen un mismo valor. Se tomó la decisión de almacenar dicho atributo como atributo de clase ya que la escala de cartas es la misma para todo el juego y no depende de cada instancia de cada carta.

Clase Abstracta	Canto
Responsabilidades:	1º) Conocer los cantos válidos como respuesta a dicho canto.
	2º) Conocer el jugador que realizó el canto.

Las clases que extienden de canto son:

- Flor
- ContraFlor
- ContraFlorAResto
- Envido
- RealEnvido
- FaltaEnvido
- Truco
- ReTruco
- ValeCuatro

Para poder realizar la consulta de si un canto es válido como respuesta solo hace falta utilizar la firma “esUnaRespuestaValidaElCanto”, método el cual retorna un valor booleano.

Clase Abstracta	CantoEnProceso
Responsabilidades:	1º) Encargado de controlar el proceso de un canto guardando o no en un listado los cantos aceptados.
	2º) Conocer el jugador que ganó el proceso.

Las clases que implementan “CantoEnProceso” son:

- CantoEnProcesoParaElTruco
- CantoEnProcesoParaElTanto

Estas clases, no solamente tienen las responsabilidades heredadas, sino que también tienen la de realizar las verificaciones que corresponden a la hora de agregar un nuevo canto al proceso. Todas estas verificaciones, están ligadas con el lanzamiento de excepciones en caso de que se intente realizar un caso no contemplado por las reglas del Truco.

El canto en proceso para el tanto sirve tanto para el envido como para la flor, realizando verificaciones extras de casos particulares donde un canto determinado tenga prioridad por sobre otro.

Clase	Jugador
Responsabilidades:	1º) Debe poder tomar cartas y jugarlas.
	2º) Realizar cantos a la mesa en la que está jugando.
	3º) Calcular sus puntos de envido o de flor.

Los jugadores utilizan un ArrayList para poder almacenar sus cartas, teniendo así la posibilidad de acceso directo para sacar una carta de su mano. Cuando los jugadores se crean, estos no tienen una mesa vinculada inicialmente, sino que es la mesa quien al crearse se vincula con todos los jugadores.

Los jugadores son objetos que extienden de la clase Observable, notificando así a todos sus observadores cuando se realice un cambio en el estado del mismo. Estos cambios estan ligados directamente con la realización de algún tipo de canto.

Clase	Equipo
Responsabilidades:	1º) Contener un conjunto de jugadores e identificar si un jugador pertenece o no al mismo.
	2º) Guardar el puntaje del equipo, conociendo si el equipo ganó o todavia no.
	3º) Proveer la posibilidad de ir recorriendo los jugadores del equipo.

Es la clase que se encarga de agrupar la cantidad de Jugadores que van a crearse para las distintas modalidades del Juego (uno contra uno, dos contra dos o Pica Pica). Sabe a su vez si el equipo es mano debido a un seteo del mismo gracias a la mesa.

Clase	**Mesa**
Responsabilidades:	● Generar Rondas necesarias hasta lograr un Equipo ganador ● Controlar el flujo del juego entre todos los jugadores: cartas , cantos. ● Determinar un Equipo Ganador del Juego ● Reparte cartas a todos los jugadores
Descripción	
Es la clase que se encarga de coordinar el flujo del juego y determina la finalización del Juego. Hace uso del mazo para repartir las cartas a cada jugador. Administra todas las cartas que se juegan, y todos los cantos que se cantan, generando las rondas necesarias, hasta que se logre un Equipo ganador del Juego. Si fuera el caso, (6 jugadores) genera rondas pica pica.	

La mesa es una clase fundamental en nuestro trabajo practico, ya que todos los jugadores poseen una referencia a la misma y cuando estos quieren realizar un canto le pasan dicho mensaje a la Mesa.

La mesa solo realiza una única verificación, y controla para ciertos cantos críticos que pueden implicar la suma de puntos si existe algún equipo ganador. Cuando realiza este control y encuentra que existe un equipo ganador simplemente activa un flag el cual la interfaz visual utilizara para poder determinar que la partida finalizó.

Debido a que la mesa es la que interacciona con el generador de rondas para ir rotando la ronda actual, está también guarda un listado de los jugadores en orden que ira rotando cada vez que deba actualizar la ronda actual, de esta forma logra ir cambiando ronda a ronda el jugador que es mano.

Esta clase también extiende de la clase Observable para poder notificar ante la realización de algún canto lo cual puede desembocar en la finalización de la partida, como así también para informar que se terminó una ronda y se comenzó una nueva. Es importante destacar que cuando se realiza un canto notifica a sus observadores el tipo de canto que se realizao mediante la utilización del argumento del notifiy.

Interfaz	**GeneradorDeRondas**
Responsabilidades:	1º) Clase encargada de generar las rondas del juego.

Interfaz implementada por:

- GeneradorRondasNormales
- GeneradorRondasPicaPica

Es la interfaz que determina qué tipo de Ronda se va a jugar en cada momento del juego. Mediante el **patrón de Strategy**, cuando una mesa se crea se le asigna un generador de ronda correspondiente para que esta cuando necesite una nueva ronda le solicite al generador que realizar dicho trabajo.

Método de la Interfaz:

```
List<Ronda> generar(Equipo equipo1, Equipo equipo2, List<Jugador> jugadoresEnJuego);
```

Como se puede observar la devolución del método es un listado de rondas y esto se debe a que las rondas que se juegan “Pica a Pica” se debe generar 3 rondas de 2 jugadores cada una. De esta forma, en cualquier otro momento en el listado se devuelve 1 sola ronda, y la mesa deberá recorrer este listado removiendo siempre el primer elemento y utilizando como ronda actual. Solo cuando el listado anterior se haya vaciado se le pedirá de nuevo al generador de ronda que le proporcione otro conjunto de rondas.

Clase	**Mano**
Responsabilidades:	● Determinar Jugador ganador de la mano ● Control para que la cantidad de cartas jugadas no supere a la cantidad de jugadores
Descripción	
Esta clase es la que describe la interacción de las cartas entre los jugadores en una mano. Sabe que cartas se Jugaron dentro de esa mano, determina si hubo un empate o un ganador dentro de esa interacción (mano), sabe determinar si la mano finalizó o no y lleva un control de cuantas cartas se pueden jugar dentro de la mano.	

Clase	**Ronda**
Responsabilidades:	● Determinar Equipo ganador de la ronda ● Control de los turnos de los jugadores ● Control sobre los cantos que se pueden realizar, y de las respuestas ● Generar las manos
Descripción	
Esta clase maneja todas las acciones de los Jugadores con sus Cartas y Cantos para un conjunto de manos. Puede identificar qué jugadores están en Juego, cuales se fueron al mazo, cuál de todos los jugadores pueden cantar Envido, sabe que canto está en proceso para una mano, sabe identificar cual es la mano actual y una vez finalizada determina cuál fue el equipo ganador de la Ronda.	

La ronda delega las responsabilidades de las verificaciones de los cantos en las clases CantoEnProceso, teniendo asi un atributo para el CantoEnProcesoDelTruco como así también otro para el CantoEnProcesoDelTanto. Como únicas verificaciones, ante el canto de un tanto, controla que la ronda tenga solo una mano para poder aceptar dicho canto.

Clase	**Mazo**
Responsabilidades:	● Repartir cartas ● Administrar las cartas del juego
Descripción	
Esta clase contiene todas las cartas posibles que se pueden usar dentro del Juego. Su comportamiento es poder repartir cartas y recibir todas las cartas que se repartieron.	

Las interacciones con el Mazo estarán realizadas por la mesa, quien le ira pidiendo cartas para entregarlas a cada uno de los jugadores, como así también al terminar una ronda, la mesa será la encargada de recolectar todas las cartas entregadas al inicio de la ronda a los jugadores para devolverselas al mazo.

Clase	**JuegoTruco**
Responsabilidades:	1º) Configurar y crear el juego en estado válido.

Juego Truco es la primera clase con la que se deberá interactuar para poder crear el juego. A esta clase se le podrán ir pasando Jugadores ya creados para que la misma los asigne en los equipos que correspondan.

La clase proporciona métodos para poder configurar el tipo de juego que se desea jugar, entre ellos la cantidad de jugadores y si se juega con o sin flor.

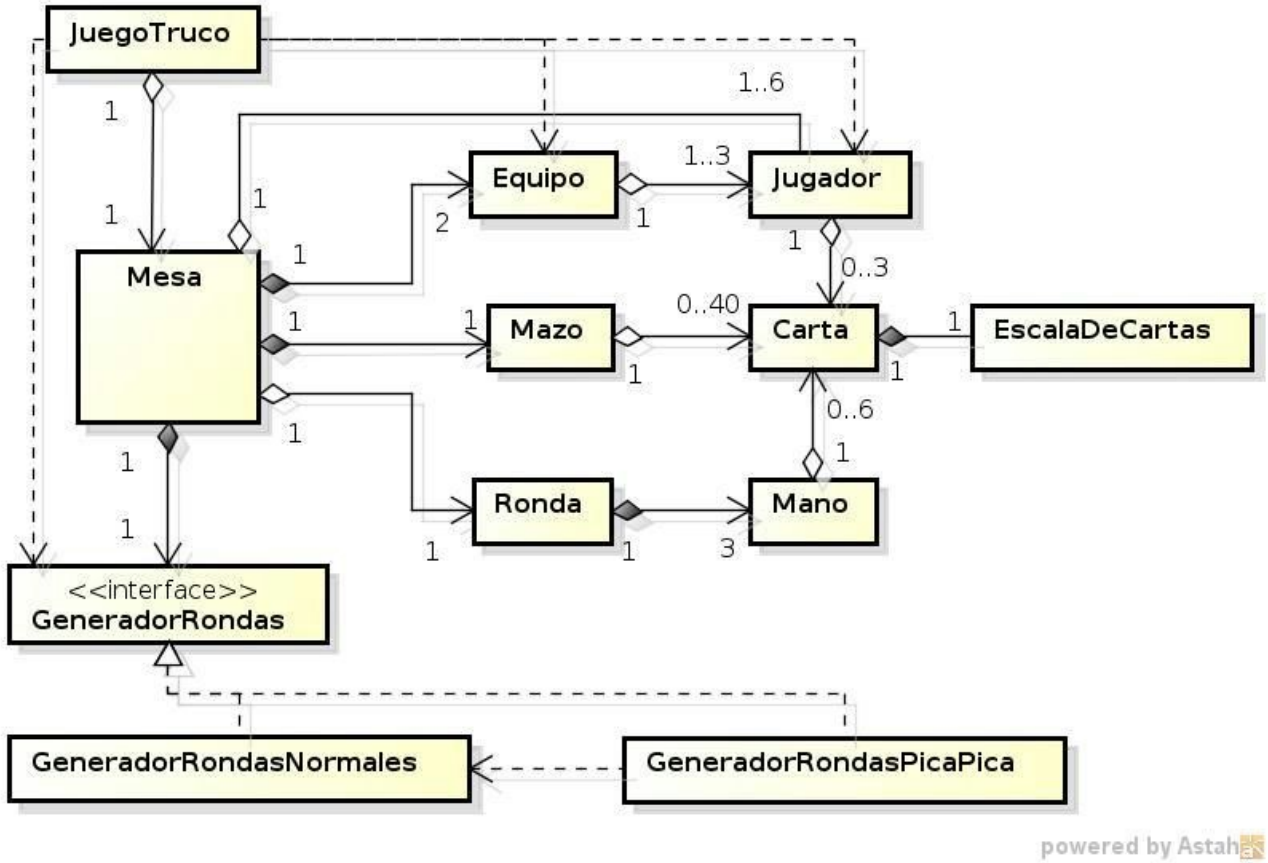
Una vez que se ejecute el método “inicarJuego” el juego se iniciara con todos los parámetros configurados, y el mismo no podrá re-configurarse en un futuro. La devolución de este método será la mesa creada e iniciada correctamente para poder interaccionar con la misma. A su vez, en el caso de que no se ejecute ningún tipo de configuración personalizada, el juego se creará en el modo default el cual es dos jugadores y sin flor.

A continuación se mostrará una tabla de un conjunto de interfaces comunes que se desarrollaron con el propósito de que toda clase que tuviera que pasar un mismo mensaje implementa dicho método bajo el mismo nombre.

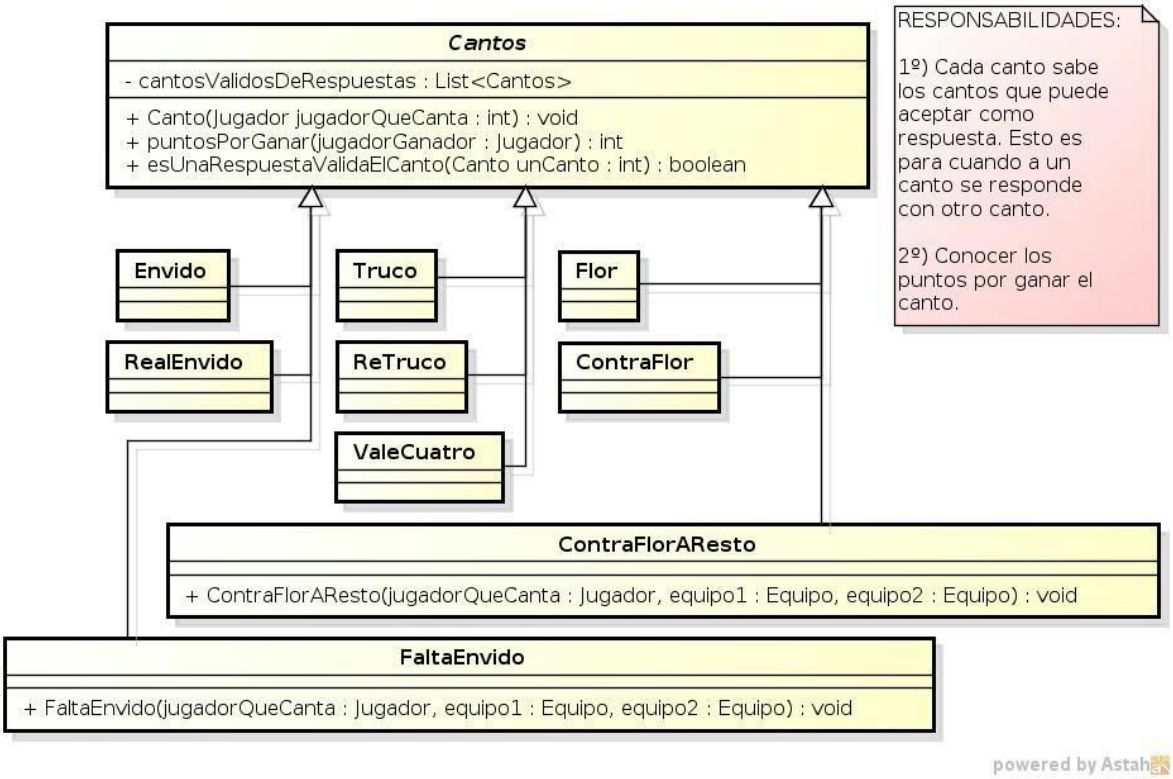
Interfaz	Métodos
Acciones	● jugarCarta (Jugador jugadorQueCanta, Carta carta) ● irseAlMazo (Jugador jugadorQueCanta)
CantosEnvido	● envido (Jugador jugadorQueCanta) ● realEnvido (Jugador jugadorQueCanta) ● faltaEnvido (Jugador jugadorQueCanta) ● cantarTantoDelEnvido (Jugador jugadorQueCanta) ● sonBuenas (Jugador jugadorQueCanta)
CantosFlor	● flor (Jugador jugadorQueCanta) ● contraFor (Jugador jugadorQueCanta) ● contraFlorAResto (Jugador jugadorQueCanta) ● cantarTantosDeLaFlor (Jugador jugadorQueCanta)
CantosGenerales	● quiero (Jugador jugadorQueCanta) ● noQuiero (Jugador jugadorQueCanta)
CantosTruco	● truco (Jugador jugadorQueCanta) ● retruco (Jugador jugadorQueCanta) ● valeCuatro (Jugador jugadorQueCanta)

Diagramas de clases

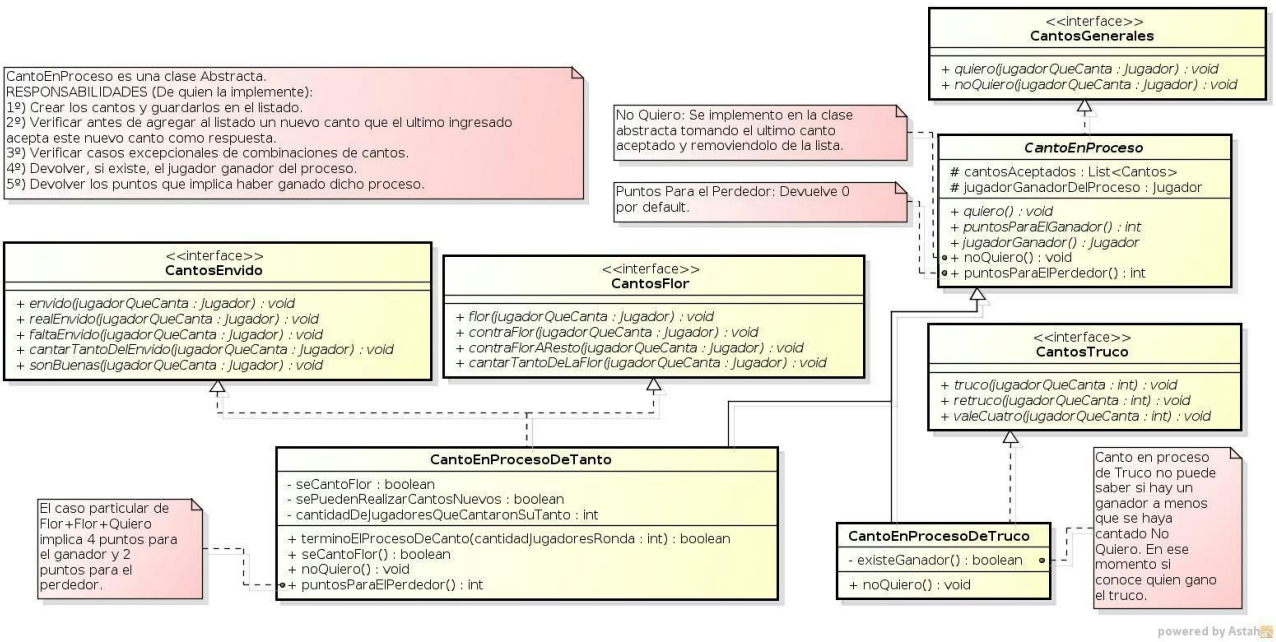
Diagrama de Clases del Modelo de Dominio:



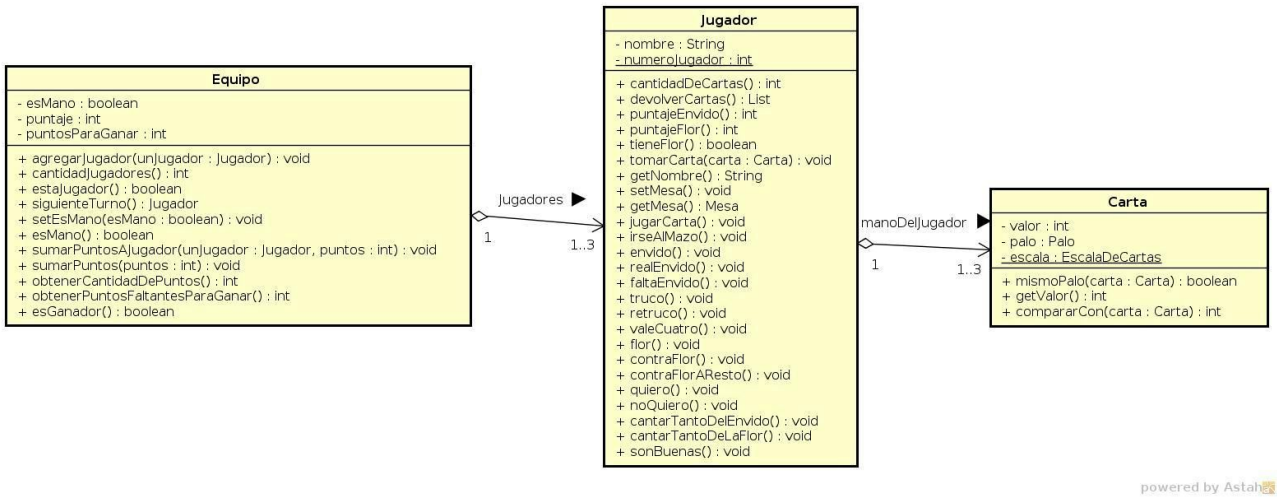
Detalle de la jerarquía y distintos tipos de Cantos:



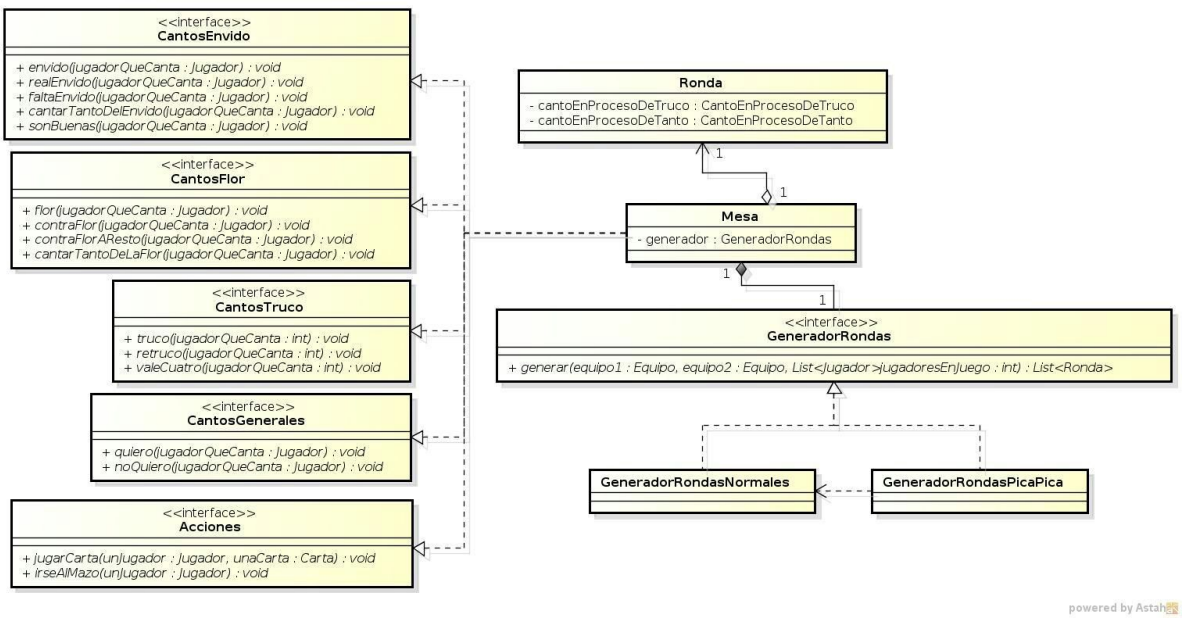
Detalle de la Interfaz de los Cantos:



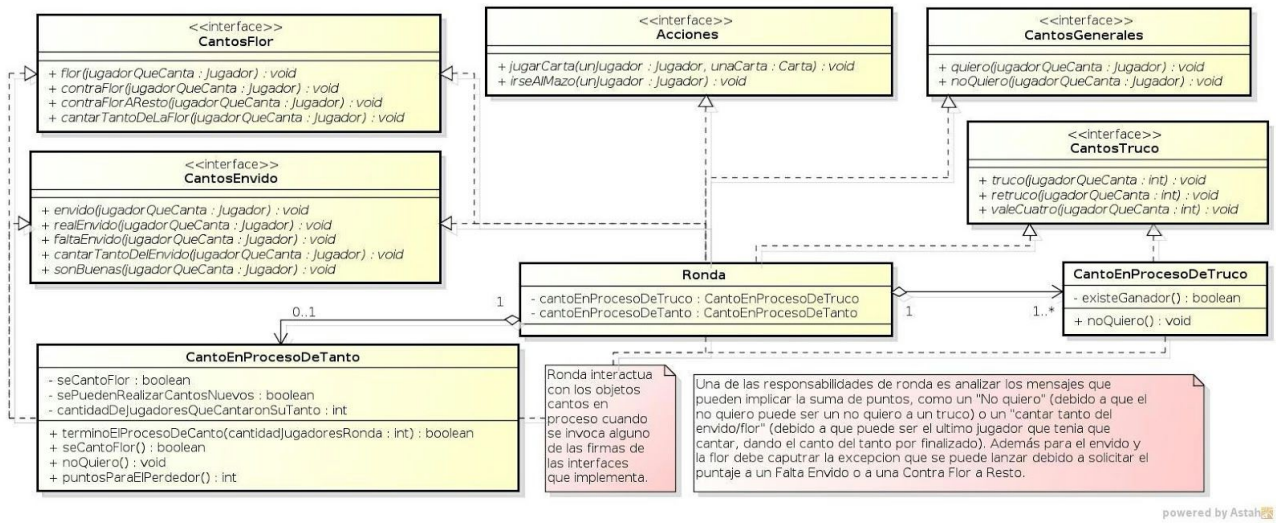
Relación entre las clases Equipo, Jugador y Carta (mano del jugador):



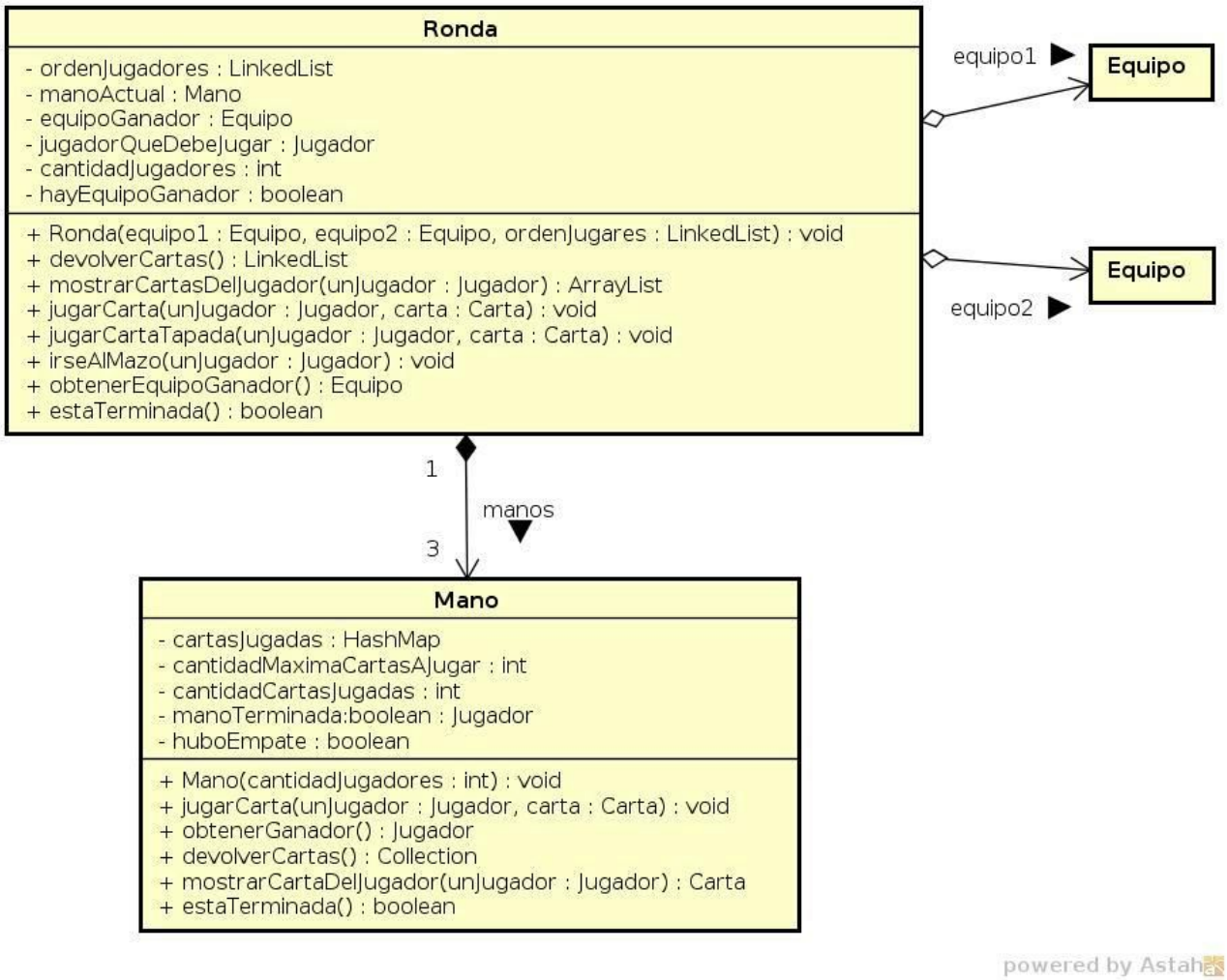
Detalle de relacion entre Mesa y Ronda con el uso de sus respecitvas interfaces:



Interfaces que implementa Ronda:

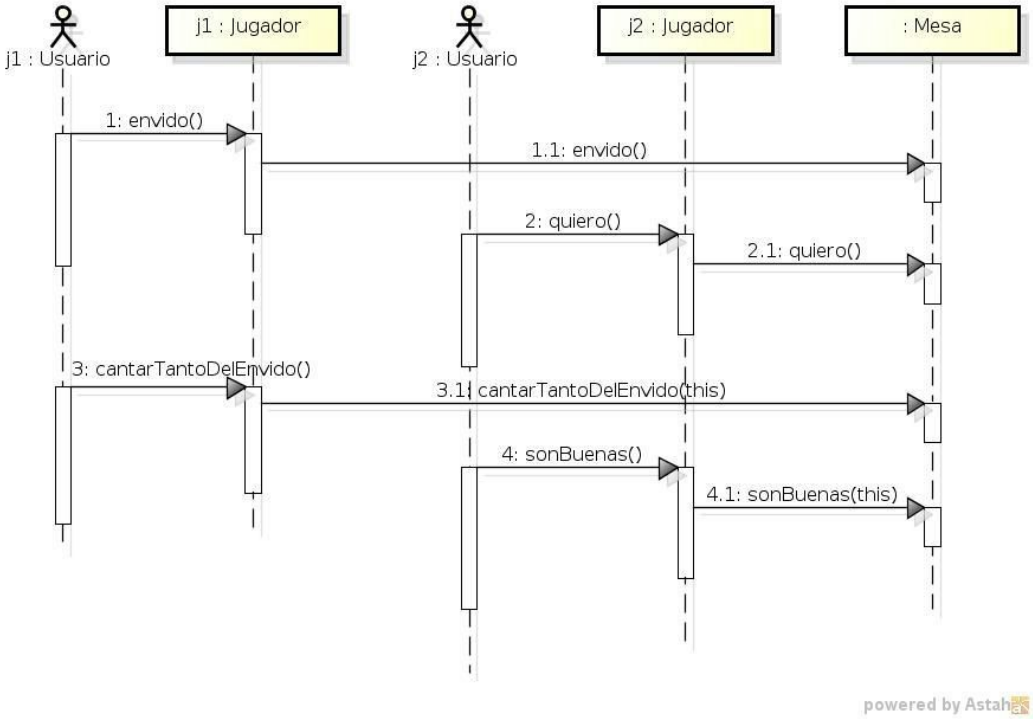


Detalle de cómo está compuesta una Ronda:

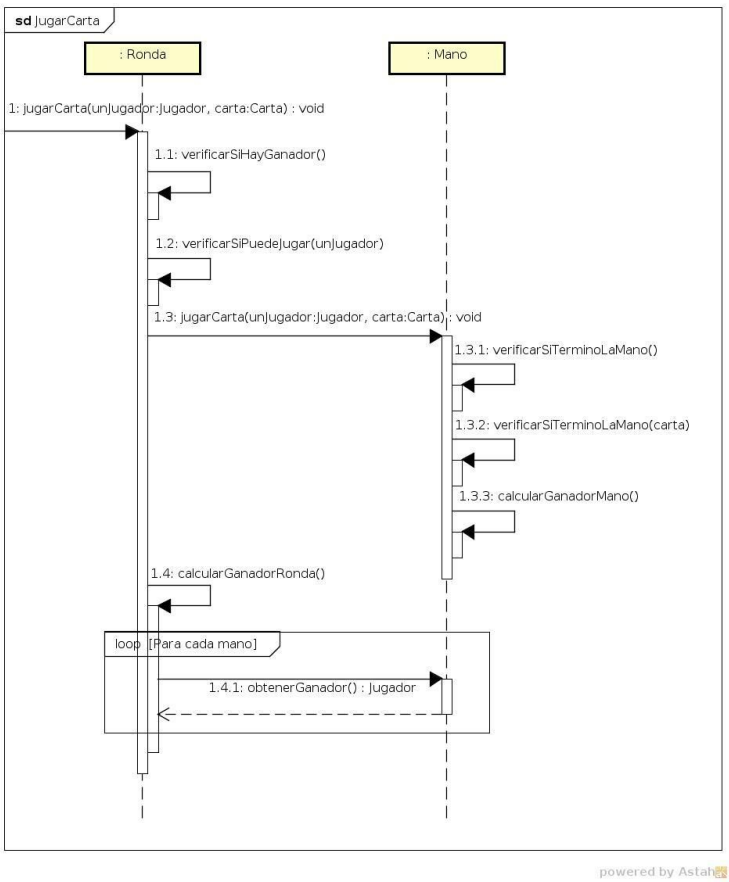


Diagramas de secuencia

Secuencia de un canto entre jugadores y como esto interactúa con la Mesa:



Secuencia de cómo se Juega una carta dentro de una Ronda en una Mano:



Secuencia que describe la interacción entre la Mesa y la Ronda cuando se efectúa un Canto:

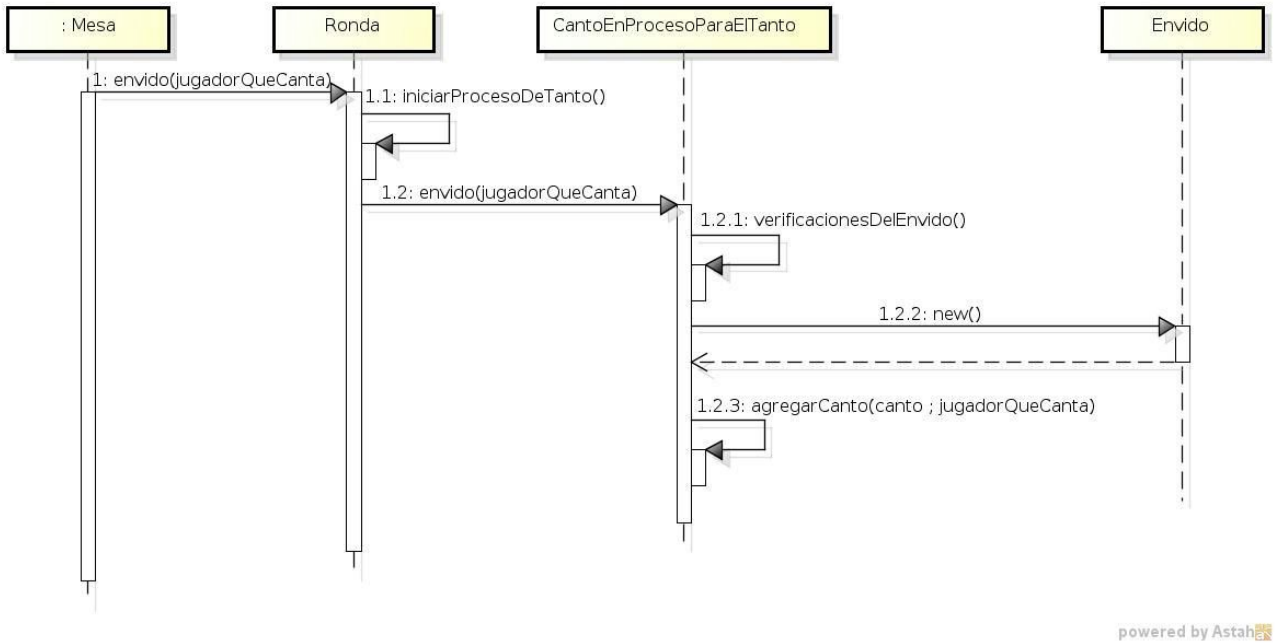
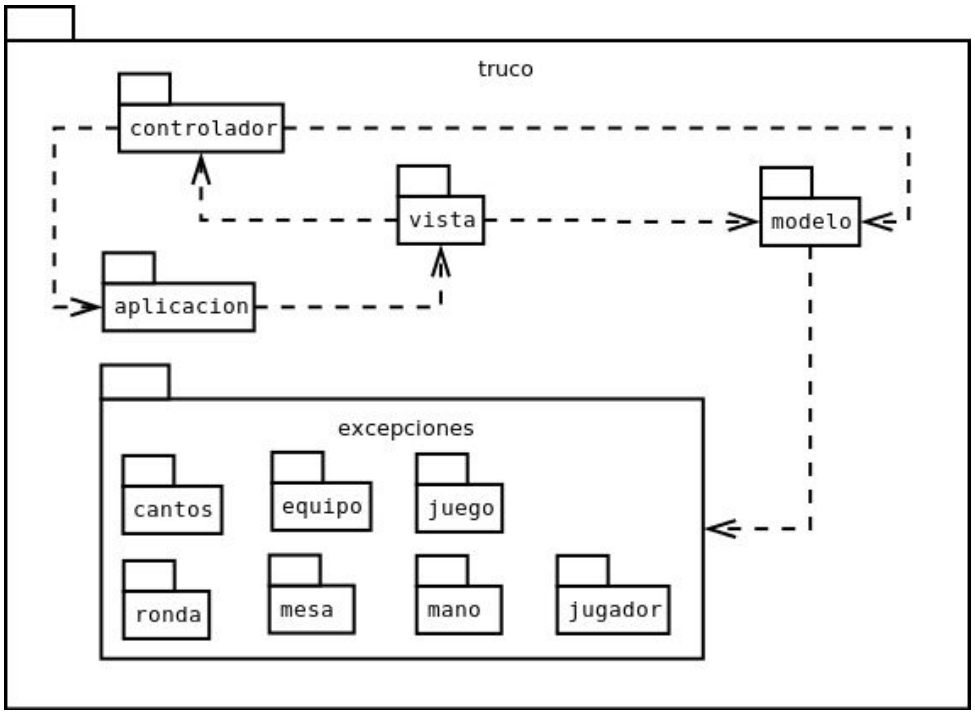


Diagrama de paquetes

Podemos hacer algunas observaciones importantes que se pueden apreciar a simple vista mirando el diagrama de paquetes.



El *modelo* es el único que conoce el paquete *excepciones*. Esto demuestra que la lógica de negocio de la aplicación está completamente controlada por el modelo.

El conjunto de paquetes *aplicacion*, *vista* y *controlador* (junto a *modelo*) son utilizados por la interfaz gráfica de la aplicación.

La relación *controlador*, *vista* y *modelo* guarda mucha relación con el patrón MVC. En este caso, no se ve la relación entre modelo y vista, pero internamente existe, ya que el modelo suele extender de la clase *Observable* que notifica a la vista a través de un mensaje de esta clase.

Clases Observadoras	**PAQUETE**
● Integrador	*CONTROLADOR*
● VentanaJugador ● VistaBotonera ● VistaManoDelJugador ● VistaMesa ● VistaPuntaje ● VistaMensajeDelUltimoCantoRealizado	*VISTA*
Clases Observadas	
● Mesa ● Jugador	*MODELO*

Detalles de implementación

Se utilizaron los siguientes patrones de diseño:

- Strategy: Para los generadores de ronda que utilizara la mesa, de forma tal de pedirle al generador sea quien sea que genere nuevas rondas para que la mesa pueda ir cambiando la ronda actual. (Desarrollado con más detalle en el Modelo del Dominio)
- MVC: Para la separación entre el modelo, la vista y los controladores. Este patrón lo que nos permitió fue reducir al mínimo las interacciones entre la interfaz visual y el modelo, de forma tal que las interacciones al modelo solo se realizan por medio de los controladores.
- Observer y Observable: Para la comunicación entre el modelo y la vista, en complemento con el patrón MVC.

Tanto en el modelo como en la vista, en distintos casos se utilizaron HashMap para mediante un objeto obtener en forma directa otro objeto relacionado. Esto demandó que se sobrescribirá para los objetos que cumplieron el rol de clave, los métodos “equals” y “hashcode”. Entre las clases que se realizó dicha implementación se puede encontrar la clase de “Carta”, “Jugador” y “Canto”. Si bien en el modelo para la clase “Canto” no se tuvo la necesidad de utilizar el método “hashcode” como sí lo fue para la interfaz visual, pudimos observar que es una buena práctica que para todo objeto que se sobreescriba el método “equals” realizar también el desarrollo del método “hashcode”. En la clase “Object” se define como contrato de la función “hashcode” que todo objeto que ante la función “equals” se observa que son iguales, el valor a retornar sea el mismo. Al realizar implementaciones del método “equals” normalmente se rompe con dicho contrato si no se elabora la función “hashcode” en dicha clase.

Es importante también destacar que se utilizaron interfaces con el propósito de poder construir mensajes polimórficos dentro del modelo, de forma tal, que toda clase que tuviera que interaccionar con cierta información, se pasará un único mensaje a través de una misma firma logrando así que cada clase realice las verificaciones que le corresponden, delegando el resto a otros objetos. Para esto, en muchos mensajes se tuvo que pasar como parámetro el objeto que iniciaba el proceso con el fin de no tener que distinguirlo hasta el momento de realizar las verificaciones que correspondan.

Por último, una de las ideas centrales al desarrollar la implementación del trabajo, fue la de no crear ningún tipo de clase que tuviera un exceso de responsabilidades, logrando así crear múltiples objetos y delegando responsabilidades siempre que se pueda. De esta forma, los usuarios pueden jugar interaccionando con la clase Jugador que le corresponde de forma directa, la cual enviará los mensajes a la mesa, evitando así tener una clase superior con la cual todos los usuarios deben interaccionar para poder jugar. A su vez la mesa, clase central en el flujo de la información del modelo, delega muchos de los análisis que se deben realizar ante un canto o una acción, cumpliendo más que nada el rol de distribuir y analizar simplemente los resultados de dichos cantos, controlando el caso eventual de una finalización del juego.

Excepciones

Excepciones relacionados con los Cantos

AlCantarFlorEnUnaRondaNoSePuedeCantarEnvidoException: lanzada por algunos métodos de la clase *CantoEnProcesoParaElTanto* para informar que no se puede cantar envido luego de que se haya cantado flor.

NoSePuedeCantarContraFlorSinHaberCantadoFlorPrimeroException: lanzada por el método “contraFlor” de la clase *CantoEnProcesoParaElTanto* para informar que no se puede cantar flor si antes alguien del otro equipo no canto flor.

NoSePuedeCantarElTantoDeLaFlorAntesDeAceptarUnCantoDeFormaPreviaException: utilizada por los métodos de canto de flor para informar que no se pueden cantar los puntos de la flor si algún jugador no la aceptó antes.

NoSePuedeCantarElTantoDelEnvidoAntesDeAceptarUnCantoDeFormaPreviaException: utilizada por los métodos de canto de envido para informar que no se pueden cantar los puntos del envido si algún jugador no la aceptó antes.

NoSePuedeCantarTantoDosVecesEnUnaRondaException: utilizada por los métodos de canto para informar que no se puede cantar algunos de los tantos más de dos veces en una misma ronda.

RespuestaIncorrectaException: usada por los métodos de canto de la clase *CantoEnProcesoParaElTanto* y *CantoEnProcesoParaElTruco* para informar que la respuesta del jugador es incorrecta y no concuerda con la lógica del juego.

Excepciones de la clase Equipo

ExisteJugadorEnEquipoException: lanzada por el método “agregarJugador(Jugador)” de la clase *Equipo* para informar que el jugador ya se encuentra en el equipo.

JugadorInexistenteException: lanzada por algunos métodos de las clases *ContraFlorAlResto*, *Equipo* y *FaltaEnvido* para informar que el jugador referenciado no existe.

Excepciones de la clase JuegoTruco
ElJuegoConfiguradoNoPoseeTodosLosJugadoresQueDeberiaException: si el juego configurado no terminó de configurar cada uno de los jugadores que deberían participar en la interfaz gráfica.
UnJuegoYaIniciadoNoPuedeConfigurarseException: en caso de quererse reconfigurar la aplicación, ésta lanzará esta excepción.

Excepciones de la clase Jugador
CartaEnManoInexistenteException: lanzada por el método “tirarCarta(Carta)” de la clase *Jugador* para informar que la carta pasado por parámetro no existe.
ElJugadorNoEstaEnNingunaMesaException: si el *Jugador* no se encuentra en la mesa en la que debería estar jugando se lanza esta excepción.
ElJugadorNoTieneFlorException: lanzada por el método “puntajeFlor()” para avisar que el jugador no tiene flor y cantó mal.
JugadorNoPuedeTenerMasDeTresCartasEnManoException: si por error el jugador recibe más de tres cartas, entonces el método lanza esta excepción.

Excepciones de la clase Mano
NoHayGanadorHastaQueLaManoTermineException: lanzada por el método “obtenerGanador()” para informar que no hay ganador de la mano porque todavía no finalizó.
NoHayGanadorHuboEmpateException: en caso de haber un empate en una mano y no hay ningún ganador, se lanza esta excepción.
NoSePuedeJugarMasCartasManoTerminadaException: se lanza esta excepción en caso de que se quiera jugar una carta y la mano ya se encuentre finalizada.

Excepcion de la clase Mesa
NoSeJuegaConFlorException: se lanza en caso de que algún jugador cante flor y no se juegue con ella.

Excepcion de la clase Ronda
EsteJugadorSeFueAlMazoException: lanzada por el método “verificarSiEsteJugadorPuedeJugar(Jugador)” para informar que el jugador se fue al mazo y no puede participar del juego.
NoEsElTurnoDeEsteJugadorException: excepción lanzada cuando en la ronda quiere jugar un jugador que no es su turno.
NoHayEquipoGanadorHastaQueLaRondaTermineException: lanzada para informar que no existe equipo ganador de la rinda hasta que finalice la ronda actual.
NoSePuedeJugarMasCartasRondaTerminadaException: lanzada para informar que la ronda se terminó cuando se quiere jugar una nueva carta.
SoloLosJugadoresPieDeLaRondaPuedenCantarElEnvidoException: lanzada cuando un jugador que no es pie canta para el tanto.
SoloSePuedeCantarElTantoEnLaPrimeraManoException: se lanza si se quiere cantar tanto en otra ronda que no sea la primera.

Interfaz Visual

Clase	Iniciar Aplicación
Responsabilidades:	1º) Inicia la aplicación con la interfaz gráfica.

Posee el main de la interfaz, y arranca ejecutando el Menú Principal. Esta es la clase que se debe ejecutar como “Java Application” para poder jugar.

Clase	MenuPrincipal
Responsabilidades:	1º) Encargada de generar todas las ventanas iniciales de la configuración del juego.

Esta clase interactúa con el modelo, creando el JuegoTruco y configurando dicho juego a medida que se va avanzando por las distintas ventas que muestra el Menú Principal. Una vez que está todo listo para jugar, la última ventana en ser mostrada es la que ejecuta para el “JuegoTruco” creado el mensaje de “iniciarJuego”.

Clase	Integrador (Controlador)
Responsabilidades:	1º) Interaccionar con las distintas interfaces de las Ventanas de los Jugadores.

Integrador es una clase que implementa la interfaz Observer, observando todo el tiempo a la mesa y a su estado. Cada vez que se desarrolle un cambio de turno, este será el encargado de mostrar un telón intermedio, el cual cuando cada jugador interaccione con el único botón se procederá a terminar de mostrar la ventana del jugador que debe jugar en ese turno. El turno del jugador varía también si existe un canto que deba responder.

Clase	VentanaJugador
Responsabilidades:	1º) Mostrar todas las vistas que posee la ventana de un jugador

La clase ventana jugador será la encargada de interaccionar con las distintas clases encargadas de mostrar cada parte de la vista de la ventana de un jugador. Para esto, configurara para cada vista las clases a las cuales estarán suscritas. Las vistas que contiene son:

- Vista Botonera
- Vista Mesa
- Vista Puntaje
- Vista Mano Jugador
- Vista Mensaje Del Último Canto Realizado

Clase	VistaBotonera
Responsabilidades:	1º) Clase encargada de prender y apagar los botones según el turno y el estado del juego

Creará todos los botones de los cantos y acciones (a excepción de jugar carta) que puede realizar cada jugador. Para esto, deberá analizar el estado actual del modelo para poder entender que botones debe habilitar para que puedan ser presionados y cuáles no. Cada botón, tendrá un controlador, y esta clase cuando cree los botones le pasara el controlador correspondiente. En todos los casos, los controladores tienen nada más que una acción directa con el modelo.

Como detalle de implementación se puede comentar que se utilizaron un par de Hashmap con la intención de evitar excesiva cantidad de atributos y acceder a un botón determinado en cualquier momento de forma directa. Dentro de estos HashMap, uno utiliza como clave objetos “Canto”, que permiten para los casos que se está en un proceso de canto obtener el último canto realizado, preguntarle los cantos de respuestas válidas y activar directamente los botones relacionados con dichos cantos.

Clase	VistaMesa
Responsabilidades:	1º) Clase encargada de mostrar las cartas jugadas en la ronda

Cada ronda, la mesa renueva las cartas jugadas, y esta clase se encarga de mostrar las cartas de la ronda, estableciendo prioridad para las cartas del jugador al cual pertenece la ventana, mostrando dichas cartas en la parte inferior de la mesa.

Clase	VistaPuntaje
Responsabilidades:	1º) Muestra el score de cada equipo.

Es importante mencionar que la clase muestra como equipo “Nosotros” al equipo al cual pertenece el Jugador con el cual está relacionado dicha ventana.

Clase	VistaManoJugador
Responsabilidades:	1º) Mostrar las cartas del jugador.
	2º) No permitir jugar cartas cuando hay un canto en proceso.

Las cartas de cada jugador estarán vinculadas con un controlador de jugar carta, el cual se le pasara el jugador al cual pertenece dicha carta y la carta a la cual representa cada botón. Para obtener las imágenes se utiliza una clase llamada “ProveedorDeImagenDeCarta” la cual mediante un HashMap relaciona objetos Carta con su imagen correspondiente.

Clase	VistaMensajeDelUltimoCantoRealizado
Responsabilidades:	1º) Mostrar el último canto realizado

Esta vista va a estar observando todo el tiempo a la clase “Mesa”. De esta forma, cada vez que se realiza un canto, la clase mesa envía como argumento secundario de la notificación una cadena de caracteres que posee la información del canto realizado. Cuando esta cadena es vacía, se asume que se debe retirar de la visibilidad el texto del último canto, y esto suele suceder cuando un jugador juega una carta.