**FIUBA - 75.07**

**Algoritmos y programación III**

*Trabajo práctico 2: FonTruco*

2do cuatrimestre, 2015

(trabajo grupal)

Alumnos:

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Padrón** | **PuertoEstelarTerranMail** |
| Bourbon Navarro, Rodrigo | 96961 | rodrigobourbon@gmail.com |
| Gomez Peter, Federico Manuel | 96091 | fedemgp@gmail.com |
| Suppes, Maximiliano | 96938 | maxisuppes@gmail.com |
|  |  |  |

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

***Tutor***:

***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

Para el desarrollo del trabajo se supuso que:

* El pica pica es una forma de juego que se da siempre en un juego de 3 personas por equipo.
* De las varias variantes que tiene el truco, se decidió que no se contempla el maldón. La falta envido ganada le otorga al ganador 30 puntos, si ambos están en las malas, o la menor distancia en puntos de ambos para llegar a 30 si están en buenas.

## Modelo de dominio

Las principales clases que se desarrollaron en el modelo son *Mesa*, *Carta*, *Equipo*, *Jugador*, *Mano*, *CalculadorTanto*, *ValoresTruco*, *Mazo*, *Vuelta*, *Puntaje* con sus posibles estados, *Buenas* o *Malas*, los distintos palos según la interfaz *Palo*, y todos los estados posibles que contempla el juego como implementaciones de *EstadoJuego* (*NadaCantado*, *TantoJugado*, *TrucoCantadoTantoNoJugado*, *TrucoCantadoTantoJugado*, *TrucoQuerido*, etc.) que se mostrarán en la secciones siguientes en los distintos diagramas.

La mesa es quien maneja por completo el flujo del juego, todas las acciones son referidas a la mesa, cada una de las cuales es delegada en mayor parte en las distintas instancias de *Vuelta*, que son *PrimeraVuelta*, *SegundaVuelta* y *TerceraVuelta*. Estas últimas son las encargadas de determinar el ganador de cada ronda y el final de la misma, sumándole los puntos correspondientes a cada equipo.

Los distintos estados del juego son los encargados de determinar qué se puede o no cantar, y cuántos puntos deben ser asignados a un equipo en caso de que el estado sea querido o no querido.

El mazo es conocido por la mesa y se le da la responsabilidad de repartir las cartas a todos los jugadores. Cada jugador recibe una instancia de *Mano* con tres cartas.

Aparte del modelo se encuentra la inteligencia artificial, *JugadorIA*, que hereda de *Jugador*, y posee como atributo una instancia de la mesa en la cual está jugando de manera de poder comunicarle su accionar.

Diagramas de clases

Los diagramas de clases se encuentran en la carpeta “*Diagramas de Clases*”, bajo los nombres “*Diagrama de Clases - Mesa*”, “*Diagrama de Clases - Carta*”, “*Diagrama de clases - Vuelta, Jugador y Equipo*”, “*Diagrama de Clases - EstadoJuego y excepciones I*”, “*Diagrama de Clases - EstadoJuego y excepciones parte II*”, “*Diagrama de Clases - MesaPicaPica*”, para una mejor visualización.

Las excepciones de *NoSePuedeCantarX* que se describirán en la sección destinada a las excepciones, donde *X* representa los estados del juego, no se realizan en el diagrama de clases ya que se aprecia mejor en el diagrama de estados que se presenta a continuación.

Diagramas de secuencia

Los diagramas de secuencia se encuentran en la carpeta “*Diagramas de Secuencia*”, bajo los nombres “*Diagrama de Secuencia - Envido cantado, querido y no querido*”,

Diagrama de paquetes

## El diagrama de paquetes se adjunta junto a este documento bajo el nombre “*Diagrama de Paquetes – Modelo*” para una mejor visualización.

Diagrama de estado

## El diagrama de estados se adjunta junto a este documento bajo el nombre “*Diagrama de Estados - Cantos*” para una mejor visualización.

Detalles de implementación

El patrón de diseño *State* fue utilizado para resolver uno de los temas más sensibles del trabajo, como lo es los posibles cantos que tiene cada jugador en un momento determinado de la ronda. Se creó la clase *Vuelta* como clase abstracta de la cual heredan *PrimeraVuelta*, *SegundaVuelta* y *TerceraVuelta*, las cuales se encargan de decidir el ganador de cada vuelta y el ganador de la ronda completa, determinan el final de la ronda y la mesa delega en ellas todos los cantos de los jugadores. A su vez, las instancias de estas clases poseen una referencia al estado del juego, que es el que sabe qué se cantó hasta el momento y qué se puede cantar, siendo estos estados por ejemplo *NadaCantado*, *TantoJugado*, *TrucoCantadoTantoNoJugado*, *TrucoCantadoTantoJugado*, *TrucoQuerido*, etc.

Para el puntaje de los equipos también se utilizó el patrón  *State*, ya que el equipo puede estar en las *buenas*, o en las *malas*, y según en cuál se encuentre, responderá de manera distinta cuando se quiera determinar los puntos que el falta envido o la contra flor al resto debe entregar al ganador.

Excepciones

# JugadorNoPuedeCantarTantoNoEsPrimeraVuelta: se lanza cuando se está jugando la segunda o tercer vuelta de la ronda y un jugador intenta cantar tanto.

Las siguientes excepciones se lanzan dentro del estado asociado al canto que validan cuando se le pregunta al mismo si está en un estado válido y previenen la realización de cualquier acción, como hacer una jugada o hacer otro canto, que no sea responder al canto actual:

# EnvidoNoQueridoNoSePuedeJugarException: se lanza en *EnvidoCantado*.

# EnvidoEnvidoNoQueridoNoSePuedeJugarException: se lanza en *EnvidoEnvidoCantado*.

# RealEnvidoNoQueridoNoSePuedeJugarException: se lanza en *RealEnvidoCantado*.

# FaltaEnvidoNoQueridoNoSePuedeJugarException: se lanza en *FaltaEnvidoCantado*.

* FlorNoAceptadaNoSePuedeJugarException: se lanza en *FlorCantada*.
* FlorFlorNoQueridaNoSePuedeJugarException: se lanza en *FlorFlorCantada*.
* ContraFlorAlRestoNoQueridaNoSePuedeJugarException: se lanza en *ContraFlorAlRestoCantada*.
* ContraFlorAlPartidoNoQueridaNoSePuedeJugarException: se lanza en *ContraFlorAlPartidoCantado*.
* TrucoNoQueridoNoSePuedeJugarException: se lanza en *TrucoCantado*.
* RetrucoNoQueridoNoSePuedeJugarException: se lanza en *RetrucoCantado*.
* ValeCuatroNoQueridoNoSePuedeJugarException: se lanza en *ValeCuatroCantado*.

Las excepciones a continuación se lanzan dentro de los distintos estados cuando se le pide al *EstadoJuego* realizar un canto que este no permite. Para ver la relación entre los cantos y cuáles son las posibilidades permitidas ver la sección *Diagrama de estados*.

# NoSePuedeCantarEnvido

# NoSePuedeCantarEnvidoEnvido

# NoSePuedeCantarRealEnvido

# NoSePuedeCantarFaltaEnvido

* NoSePuedeCantarFlorException
* NoSePuedeCantarContraFlorException
* NoSePuedeCantarTrucoException
* NoSePuedeCantarRetrucoException
* NoSePuedeCantarValeCuatroException

La próxima excepción es lanzada en los métodos de *PrimeraVuelta* si el juego fue iniciado en la modalidad sin flor y se intentó cantarla.

* JuegoSinFlorException

Las siguientes dos excepciones se lanzan en el estado de juego *FlorCantada*, al invocar el método *quiero()* y *noQuiero()* respectivamente. Cuando la mesa atrapa la primer excepción, interpreta que la flor no tuvo respuesta por parte del equipo contrario y que debe asignarle los puntos al equipo que la cantó.

* NoSePuedeQuererFlorException
* NoSePuedeNoQuererFlorException

Cuando un estado de juego correspondiente al truco es querido y se intenta querer nuevamente se lanzan estas excepciones. Esto no es necesario realizarlo para los estados relacionados con el tanto (envido y flor) ya que al decir *quiero()* se devuelve un nuevo estado (de acuerdo al *Diagrama de estados*), y se obtiene el ganador, asignándole los puntos correspondientes al equipo.

* TrucoYaFueQueridoException: se lanza en *TrucoQuerido*.
* RetrucoYaFueQueridoException: se lanza en *RetrucoQuerido*.
* ValeCuatroYaFueQueridoException: se lanza en *ValeCuatroQuerido*.

Las próximas excepciones son lanzadas cuando se intenta cantar retruco o vale cuatro y el equipo que lo hace fue el último en cantar, situación inválida en el juego.

* EquipoQueCantoTrucoNoPuedeCantarRetrucoException: se lanza en *TrucoQuerido*.
* EquipoQueCantoRetrucoNoPuedeCantarValeCuatroException: se lanza en *RetrucoQuerido*.

Cuando se intenta querer o no querer cuando no se cantó nada aún se lanza esta excepción.

* NadaCantadoException: se lanza en *NadaCantado*.

Se lanza en el constructor de Equipo cuando no recibe ningún jugador.

* EquipoSinIntegrantesException

Se lanza cuando un jugador canta tanto y su lugar en la mesa no se lo permite.

* JugadorNoPieNoPuedeCantarEnvido

Se lanza cuando se intenta calcular la flor de un jugador, el equipo del jugador la recibe, y si ninguno tiene, se lanza a un nivel superior, indicando que el canto de flor fue inválido.

* JugadorNoTieneFlorException

Esta excepción es lanzada cuando un jugador realiza una jugada, pero la carta que se intenta jugar no está en la mano del jugador.

* LaCartaNoSeEncuentraEnLaManoDelJugadorException

Es lanzada si un jugador realiza una jugada sin una carta, es decir con un objeto *null*.

* NoHayCartasParaJugar

Esta excepción se lanza cuando al sumarle los puntos del tanto o del truco a un equipo, los puntos del mismo superan los treinta puntos necesarios para ser consagrados ganadores.

* JuegoTerminadoException

# 

# **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?