FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 - Balatro

Segundo cuatrimestre de 2024

[7507/9502]
Paradigmas de la Programación

Grupo 7 - integrantes:

Nombre	Padrón	Mail
Felipe Varela Olid	110625	fvarelao@fi.uba.ar
Jonathan Huergo Ramos Maldonado	109220	jhuergo@fi.uba.ar
Federica Mortimer	108034	fmortimer@fi.uba.ar
Manuel Francisco Ruiz	111096	mafruiz@fi.uba.ar

Índice:

Introducción

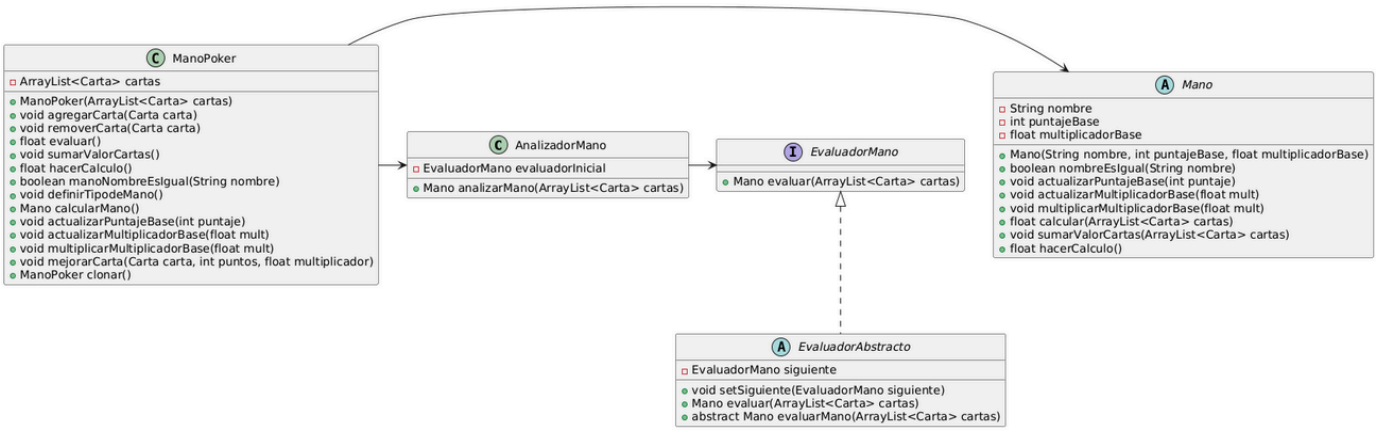
En este informe se presenta la documentación de la resolución propuesta para programar el juego ‘Balatro’ en java y se explicará a grandes rasgos el modelo.

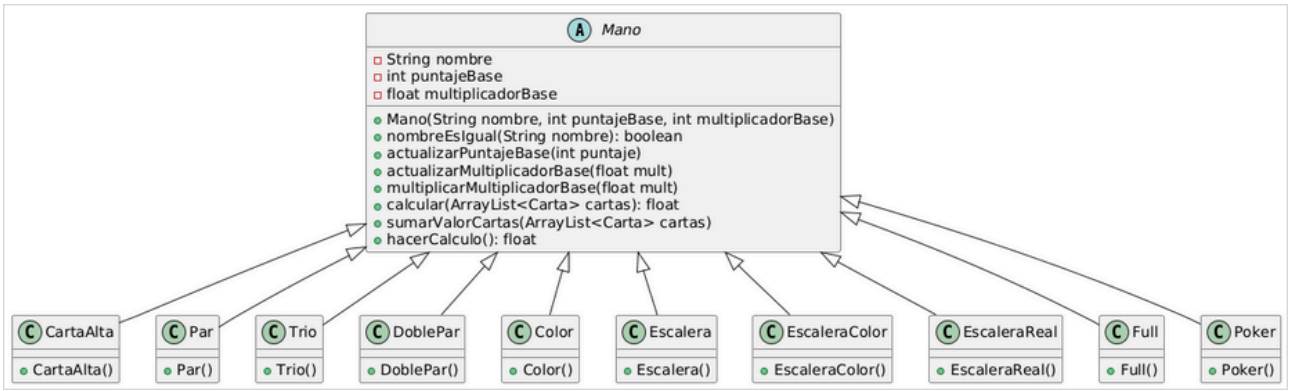
1. Supuestos

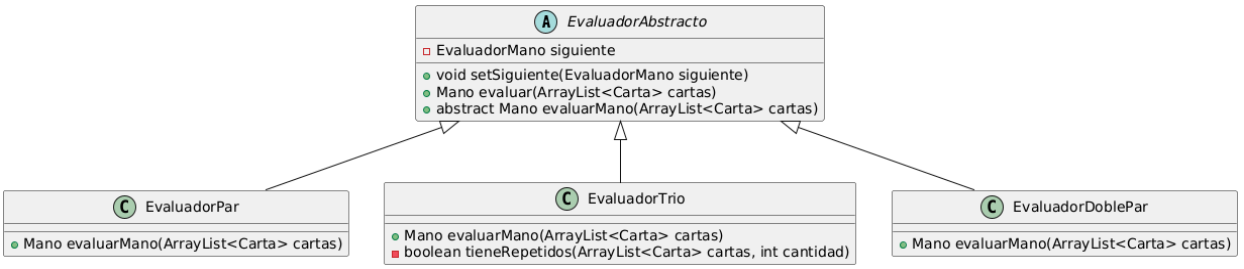
- En el comienzo de cada ronda, se muestra una tienda. Se tomó como supuesto que el jugador debe elegir una de las posibles cartas mostradas en la tienda.
- Los tarots se usan automáticamente al comprarse, no puede decidirlo el jugador.

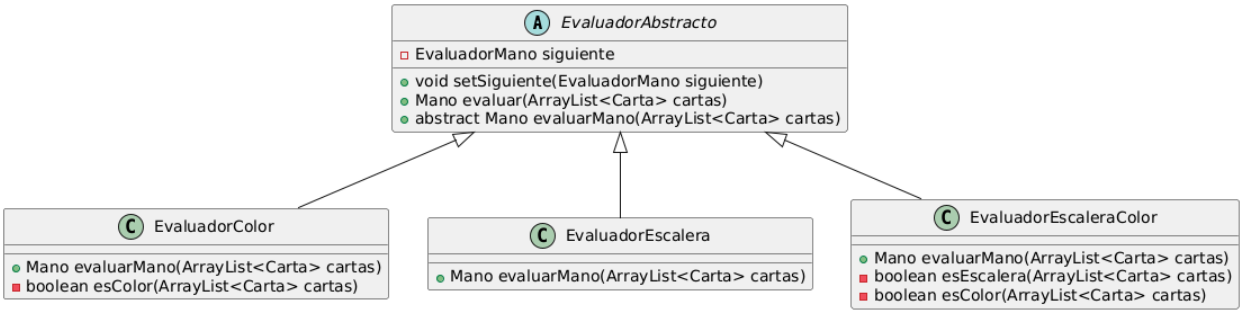
2. Diagramas de clases

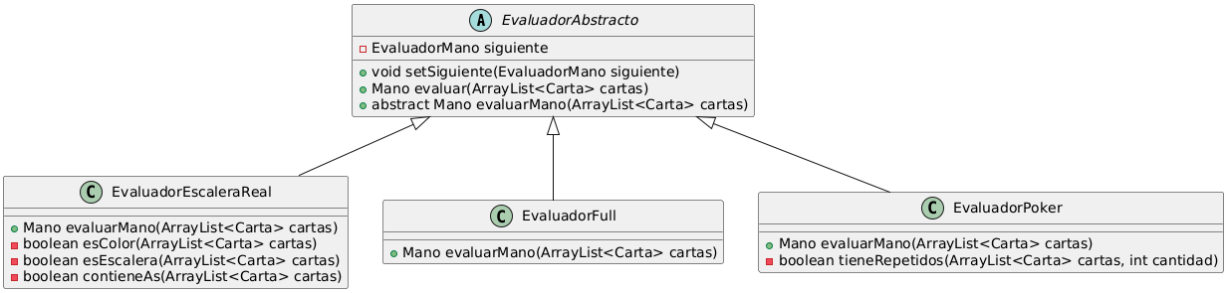

Jugador
Mazo mazo
ArrayList<Carta> cartasDisponibles
int limiteCartas
ManoPoker manoPoker
ArrayList<Comodin> comodines
ArrayList<Tarot> tarotsUsados
int descartes
int jugadas
Jugada jugadaActual
boolean perdio
Jugador()
Jugador(Mazo mazo)
Jugador(Mazo mazo, ArrayList<Carta> mano, ManoPoker manoPoker, ArrayList<Comodin> comodines)
actualizarPuntajeBase(int puntajeBase)
actualizarMult(float mult)
multiplicarMult(float mult)
agregarComodin(Comodin comodin)
agregarTarot(Tarot tarot)
repartirCartas()
descartarCartas()
reiniciarMano()
calcularPuntosPorDescarte(int puntosPorDescarte): int
calcularPuntosPorDescarte(float puntosPorDescarte): float
jugar(int numeroRonda): float
crearJugada()
crearJugada(int numeroRonda)
evaluarJugadaActual(): float
evaluarJugadas(int numeroRonda): float
mostrarAlerta(String mensaje)
tieneCarta(Carta carta): boolean
agregarCarta(Carta carta)
removerCarta(Carta carta)
superaLimite(): boolean
setCantidadDeDescartes(int cantDeDescartes)
setCantidadDeManos(int cantDeManos)
perdio(): boolean
agregarCartaMazo(Carta carta)
*
Jugada
ManoPoker manoPoker
ArrayList<Comodin> comodines
int descartes
int numeroRonda
ArrayList<Tarot> tarotsUsados
Jugada(ManoPoker manoPoker, ArrayList<Comodin> comodines, int descartes)
Jugada(ManoPoker manoPoker, ArrayList<Comodin> comodines, int descartes, ArrayList<Tarot> tarotsUsados)
Jugada(ManoPoker manoPoker, ArrayList<Comodin> comodines, int descartes, ArrayList<Tarot> tarotsUsados, int numeroRonda)
actualizarPuntajeBase(int puntajeBase)
actualizarMult(float mult)
multiplicarMult(float mult)
tieneDescartes(): boolean
tieneManoDeTipo(String manoEsperada): boolean
calcularPuntosPorDescarte(int puntosPorDescarte): int
calcularPuntosPorDescarte(float puntosPorDescarte): float
evaluarJugada(): float
tieneNumeroRonda(int numeroOtraRonda): boolean

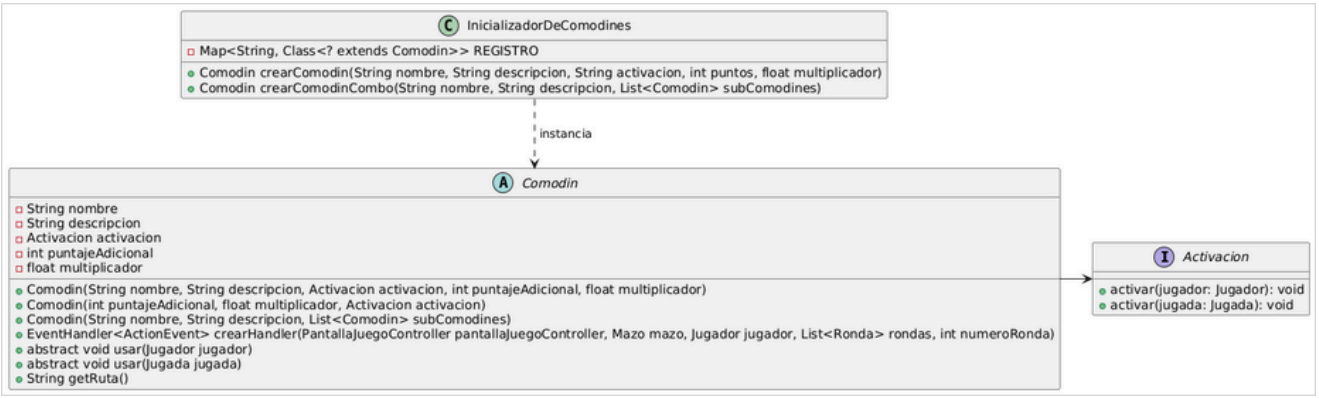

ManoPoker
- ArrayList<Carta> cartas
+ ManoPoker(ArrayList<Carta> cartas)
+ void agregarCarta(Carta carta)
+ void removerCarta(Carta carta)
+ float evaluar()
+ void sumarValorCartas()
+ float hacerCalculo()
+ boolean manoNombreEsIgual(String nombre)
+ void definirTipodeMano()
+ Mano calcularMano()
+ void actualizarPuntajeBase(int puntaje)
+ void actualizarMultiplicadorBase(float mult)
+ void multiplicarMultiplicadorBase(float mult)
+ void mejorarCarta(Carta carta, int puntos, float multiplicador)
+ ManoPoker clonar()
AnalizadorMano
- EvaluadorMano evaluadorInicial
+ Mano analizarMano(ArrayList<Carta> cartas)
EvaluadorMano
+ Mano evaluar(ArrayList<Carta> cartas)
Mano
- String nombre
- int puntajeBase
- float multiplicadorBase
+ Mano(String nombre, int puntajeBase, float multiplicadorBase)
+ boolean nombreEsIgual(String nombre)
+ void actualizarPuntajeBase(int puntaje)
+ void actualizarMultiplicadorBase(float mult)
+ void multiplicarMultiplicadorBase(float mult)
+ float calcular(ArrayList<Carta> cartas)
+ void sumarValorCartas(ArrayList<Carta> cartas)
+ float hacerCalculo()
EvaluadorAbstracto
- EvaluadorMano siguiente
+ void setSiguiente(EvaluadorMano siguiente)
+ Mano evaluar(ArrayList<Carta> cartas)
+ abstract Mano evaluarMano(ArrayList<Carta> cartas)

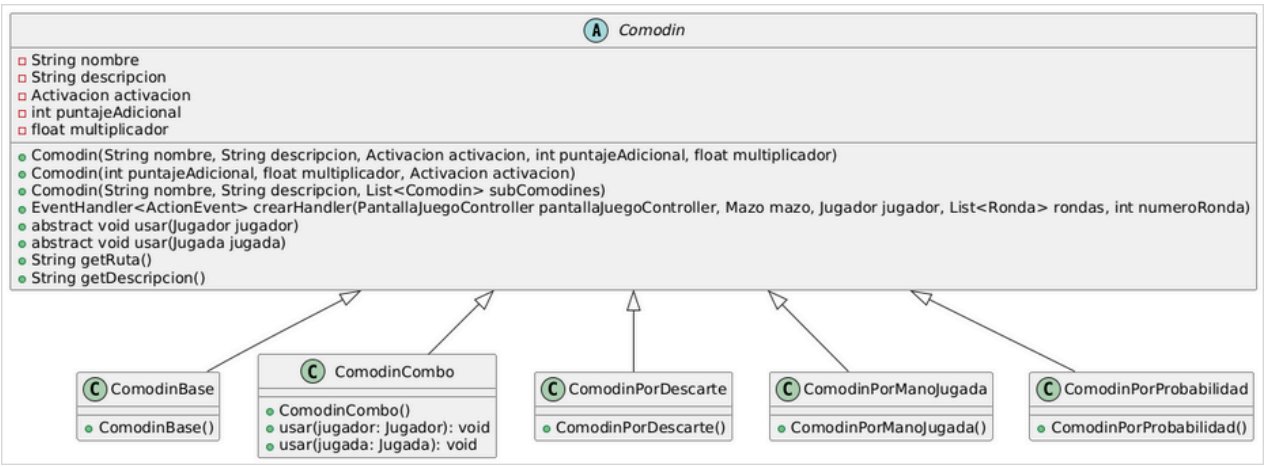

Mano
- String nombre
- int puntajeBase
- float multiplicadorBase
+ Mano(String nombre, int puntajeBase, int multiplicadorBase)
+ nombreEsIgual(String nombre): boolean
+ actualizarPuntajeBase(int puntaje)
+ actualizarMultiplicadorBase(float mult)
+ multiplicarMultiplicadorBase(float mult)
+ calcular(ArrayList<Carta> cartas): float
+ sumarValorCartas(ArrayList<Carta> cartas)
+ hacerCalculo(): float
CartaAlta
+ CartaAlta()
Par
+ Par()
Trio
+ Trio()
DoblePar
+ DoblePar()
Color
+ Color()
Escalera
+ Escalera()
EscaleraColor
+ EscaleraColor()
EscaleraReal
+ EscaleraReal()
Full
+ Full()
Poker
+ Poker()

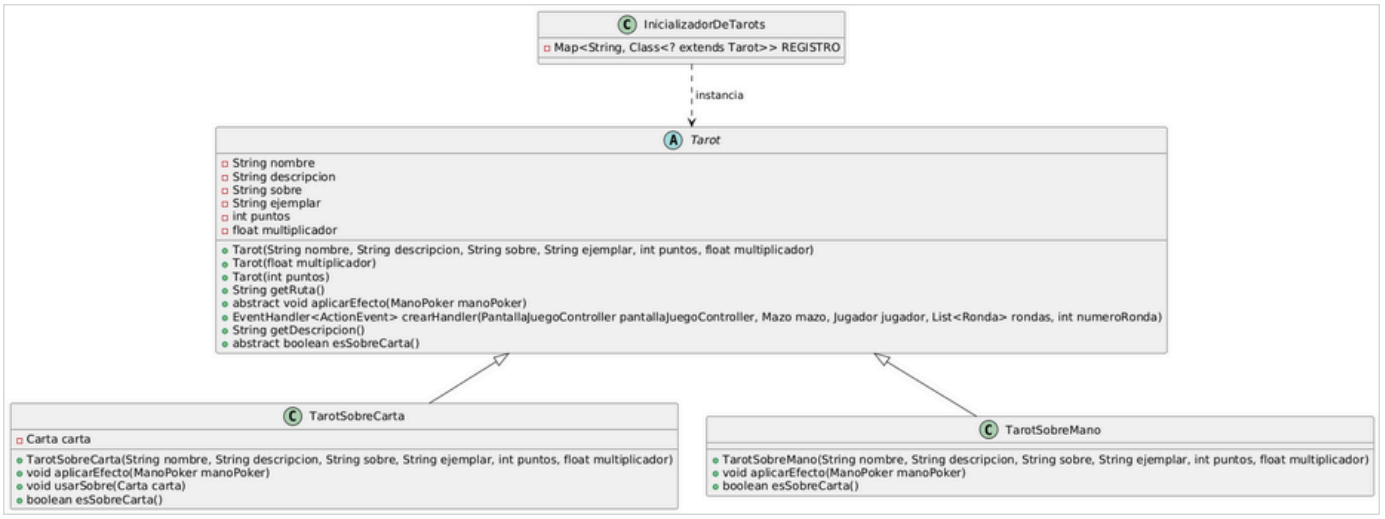

EvaluadorAbstracto
- EvaluadorMano siguiente
+ void setSiguiente(EvaluadorMano siguiente)
+ Mano evaluar(ArrayList<Carta> cartas)
+ abstract Mano evaluarMano(ArrayList<Carta> cartas)
EvaluadorPar
+ Mano evaluarMano(ArrayList<Carta> cartas)
EvaluadorTrio
+ Mano evaluarMano(ArrayList<Carta> cartas)
- boolean tieneRepetidos(ArrayList<Carta> cartas, int cantidad)
EvaluadorDoblePar
+ Mano evaluarMano(ArrayList<Carta> cartas)


EvaluadorAbstracto
- EvaluadorMano siguiente
+ void setSiguiente(EvaluadorMano siguiente)
+ Mano evaluar(ArrayList<Carta> cartas)
+ abstract Mano evaluarMano(ArrayList<Carta> cartas)
EvaluadorColor
+ Mano evaluarMano(ArrayList<Carta> cartas)
- boolean esColor(ArrayList<Carta> cartas)
EvaluadorEscalera
+ Mano evaluarMano(ArrayList<Carta> cartas)
EvaluadorEscaleraColor
+ Mano evaluarMano(ArrayList<Carta> cartas)
- boolean esEscalera(ArrayList<Carta> cartas)
- boolean esColor(ArrayList<Carta> cartas)

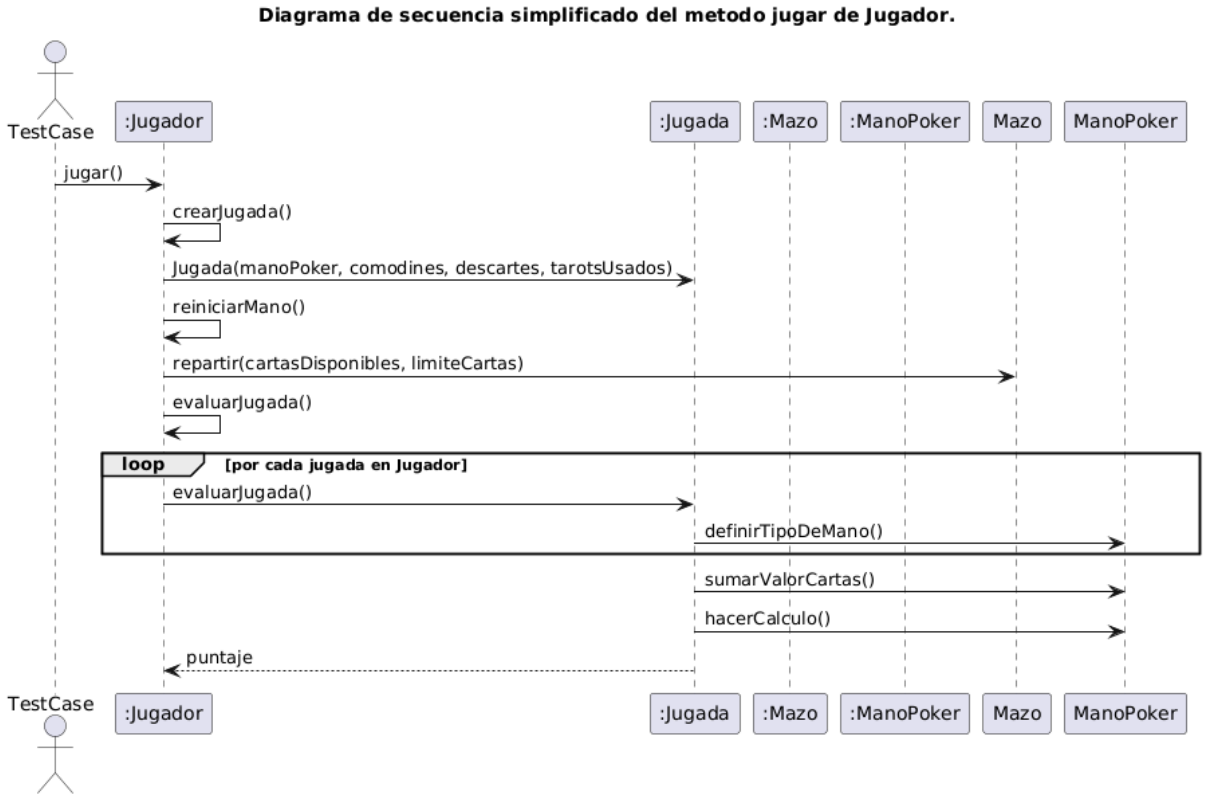

EvaluadorAbstracto
- EvaluadorMano siguiente
+ void setSiguiente(EvaluadorMano siguiente)
+ Mano evaluar(ArrayList<Carta> cartas)
+ abstract Mano evaluarMano(ArrayList<Carta> cartas)
EvaluadorEscaleraReal
+ Mano evaluarMano(ArrayList<Carta> cartas)
- boolean esColor(ArrayList<Carta> cartas)
- boolean esEscalera(ArrayList<Carta> cartas)
- boolean contieneAs(ArrayList<Carta> cartas)
EvaluadorFull
+ Mano evaluarMano(ArrayList<Carta> cartas)
EvaluadorPoker
+ Mano evaluarMano(ArrayList<Carta> cartas)
- boolean tieneRepetidos(ArrayList<Carta> cartas, int cantidad)

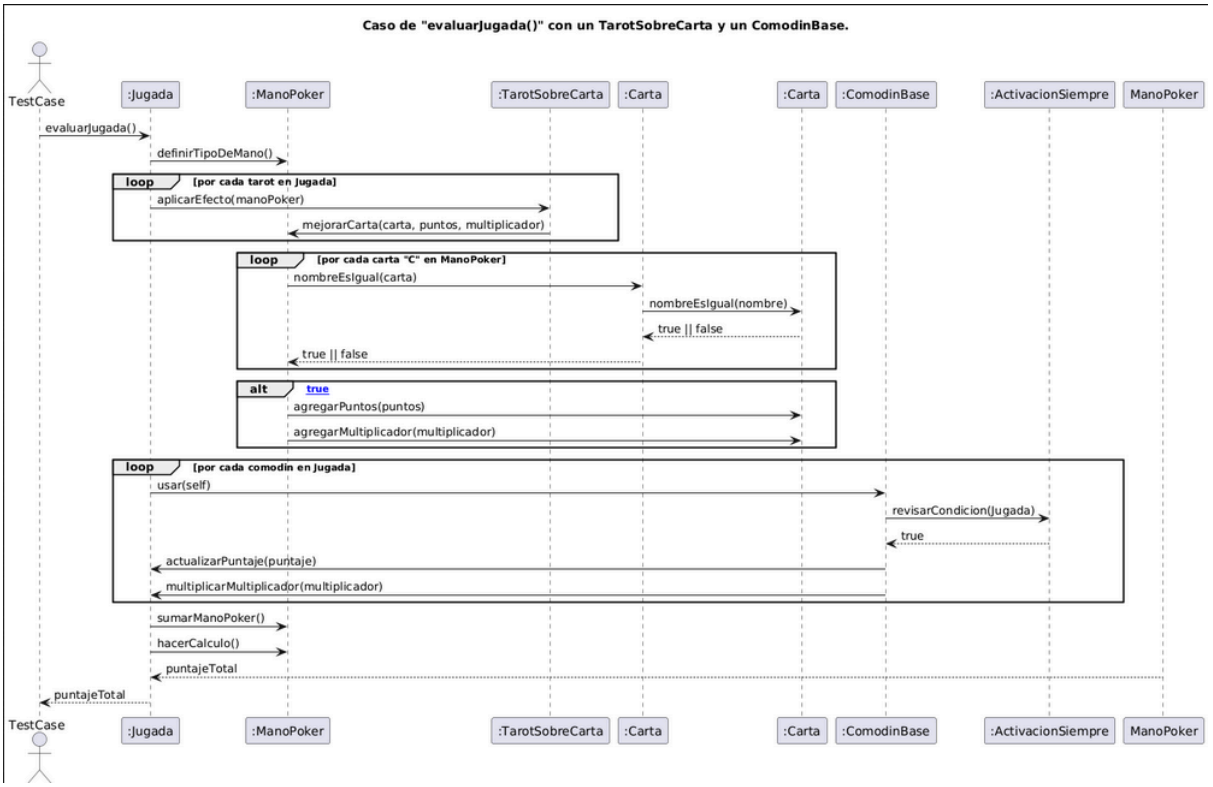

InicializadorDeComodines
- Map<String, Class<? extends Comodin>> REGISTRO
+ Comodin crearComodin(String nombre, String descripcion, String activacion, int puntos, float multiplicador)
+ Comodin crearComodinCombo(String nombre, String descripcion, List<Comodin> subComodines)
instancia
Comodin
- String nombre
- String descripcion
- Activacion activacion
- int puntajeAdicional
- float multiplicador
+ Comodin(String nombre, String descripcion, Activacion activacion, int puntajeAdicional, float multiplicador)
+ Comodin(int puntajeAdicional, float multiplicador, Activacion activacion)
+ Comodin(String nombre, String descripcion, List<Comodin> subComodines)
+ EventHandler<ActionEvent> crearHandler(PantallaJuegoController pantallaJuegoController, Mazo mazo, Jugador jugador, List<Ronda> rondas, int numeroRonda)
+ abstract void usar(Jugador jugador)
+ abstract void usar(Jugada jugada)
+ String getRuta()
Activacion
+ activar(jugador: Jugador): void
+ activar(jugada: Jugada): void


Comodin
- String nombre
- String descripcion
- Activacion activacion
- int puntajeAdicional
- float multiplicador
+ Comodin(String nombre, String descripcion, Activacion activacion, int puntajeAdicional, float multiplicador)
+ Comodin(int puntajeAdicional, float multiplicador, Activacion activacion)
+ Comodin(String nombre, String descripcion, List<Comodin> subComodines)
+ EventHandler<ActionEvent> crearHandler(PantallaJuegoController pantallaJuegoController, Mazo mazo, Jugador jugador, List<Ronda> rondas, int numeroRonda)
+ abstract void usar(Jugador jugador)
+ abstract void usar(Jugada jugada)
+ String getRuta()
+ String getDescripcion()
ComodinBase
+ ComodinBase()
ComodinCombo
+ ComodinCombo()
+ usar(jugador: Jugador): void
+ usar(jugada: Jugada): void
ComodinPorDescarte
+ ComodinPorDescarte()
ComodinPorManoJugada
+ ComodinPorManoJugada()
ComodinPorProbabilidad
+ ComodinPorProbabilidad()

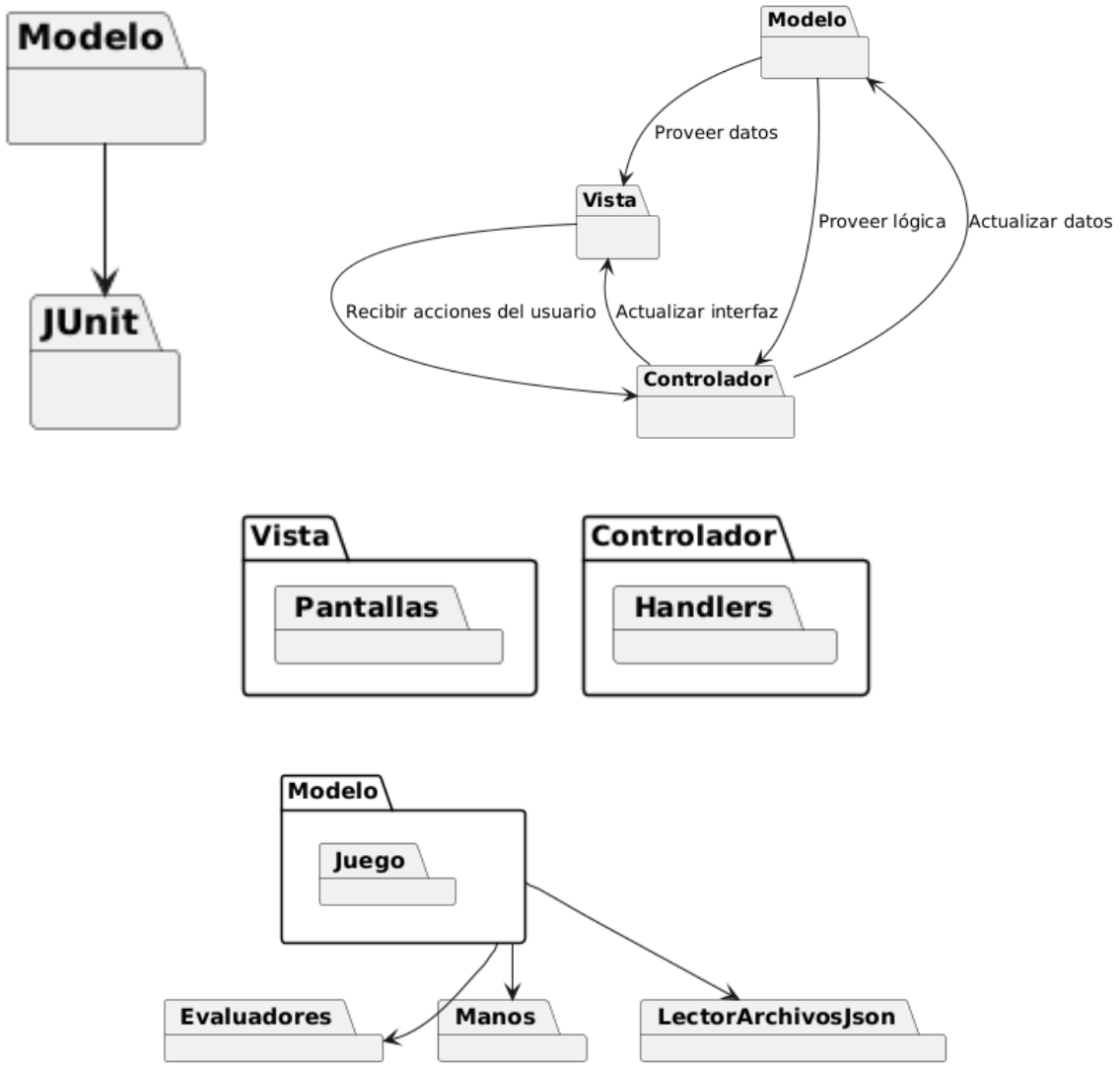


3. Diagramas de secuencia




Caso de "evaluarJugada()" con un TarotSobreCarta y un ComodinBase.
TestCase
:Jugada
:ManoPoker
:TarotSobreCarta
:Carta
:Carta
:ComodinBase
:ActivacionSiempre
ManoPoker
evaluarJugada()
definirTipoDeMano()
loop
[por cada tarot en Jugada]
aplicarEfecto(manoPoker)
mejorarCarta(carta, puntos, multiplicador)
loop
[por cada carta "C" en ManoPoker]
nombreEsIgual(carta)
nombreEsIgual(nombre)
true || false
true || false
alt
true
agregarPuntos(puntos)
agregarMultiplicador(multiplicador)
loop
[por cada comodin en Jugada]
usar(self)
revisarCondicion(Jugada)
true
actualizarPuntaje(puntaje)
multiplicarMultiplicador(multiplicador)
sumarManoPoker()
hacerCalculo()
puntajeTotal
puntajeTotal
TestCase
:Jugada
:ManoPoker
:TarotSobreCarta
:Carta
:Carta
:ComodinBase
:ActivacionSiempre
ManoPoker

4. Diagramas de paquete



5. Detalles de implementación

5.1 - Patrones utilizados

- Chain of responsibility:

Al momento de determinar a qué tipo corresponde una mano de poker jugada se utilizó el patrón chain of responsibility. Esto nos permitió ir comprobando que la mano jugada corresponda a una mano poker determinada desde la más a la menos probable.
Se utilizaron la clase EvaluadorAbstracto, que implementa la lógica base para delegar la responsabilidad a las clases “evaluadoras” concretas de evaluar la mano, y la interfaz EvaluadorMano, que define el método genérico de evaluación de manos. Este patrón tiene como beneficios la flexibilidad las clases concretas (EvaluadorPar, EvaluadorTrio etc.) pueden reorganizarse o extenderse sin modificar el código base. Además promueve la

escalabilidad, si se necesita agregar nuevas categorías de evaluación, esto puede hacerse implementando una nueva clase concreta y añadiendola a la cadena.

- Prototype:

La clase Jugador guarda instancias de la clase Jugada que tienen una copia de todos los elementos requeridos para el cálculo del puntaje de la respectiva jugada. Una vez creada la jugada su estado interno será inmutable. Al requerir el puntaje total se itera por todas las jugadas calculando su puntaje. Este patrón nos permite calcular puntajes sin usar un acumulador, ni depender de cambios en el estado actual del juego como lo son tarots o nuevos comodines.

- Modelo-Vista-Controlador:

En el proyecto, el patrón MVC se implementó de la siguiente manera:

Modelo: Las clases relacionadas con la lógica del juego y los datos, como Juego, Carta y otras clases que representan las reglas y estados del juego.

Vista: Las clases que extienden VBox, como EndGameScreen, CollectionScreen y JuegoScreen, que manejan las interfaces gráficas y muestran datos al usuario.

Controlador: Clases como FlujoJuegoController actúan como intermediarios entre el modelo y las vistas, procesando eventos como clics en botones y actualizando las pantallas de manera dinámica.

Este patrón nos permite separar las responsabilidades entre el modelo, la vista y el controlador. Si se desea agregar nuevas interfaces visuales o (por ejemplo, otra pantalla en el juego), esto puede hacerse sin afectar la lógica central del modelo o los controladores existentes.

5.2 - Uso de herencia vs. delegación

Por un lado, se ha utilizado herencia en el proyecto para modelar relaciones jerárquicas entre clases, permitiendo la reutilización de código común. Por ejemplo, las clases ComodinBase y ComodinCombo heredan de la clase base Comodin, lo que permite aprovechar métodos comunes como usar(). Además, se utiliza polimorfismo para manejar diferentes tipos de comodines de manera uniforme.

Por otro lado, si en lugar de heredar, se delega el comportamiento a otro objeto, esto brinda mayor flexibilidad, ya que no crea una jerarquía rígida. Se ha utilizado por ejemplo en la clase Jugador al delegar actualizarPuntajeBase() a la clase ManoPoker, que a su vez delega actualizarPuntajeBase() a la clase TipoDeMano respectiva de la ManoPoker.

5.3 - Principios de diseño utilizados

Se han tenido en cuenta los principios SOLID.

- Principio de segregación de la interfaz: se aplicó este principio, por ejemplo, en la interfaz ActivacionPorJugada y ActivacionPorJugador que declaran los métodos

revisarCondicion(Jugador jugador) y revisarCondicion(Jugada jugada) respectivamente.

- Principio Open/closed: este principio se puede ver claramente en los tarots y comodines ya que si quisiéramos crear nuevos comodines o tarots no sería necesario modificar el código existente.

- Principio Single Responsibility: se aplica al tener por ejemplo al separar las distintas responsabilidades de los controladores. diferentes controladores, como PantallaJuegoController y SeleccionarCartaController, etc., que se encargan de responsabilidades específicas:

  ➔ PantallaJuegoController: Maneja la lógica de la pantalla principal del juego.
  ➔ SeleccionarCartaController: Se encarga de gestionar la selección de una carta.

  Esto asegura que cada controlador tiene una responsabilidad clara, lo que facilita su mantenimiento y extensión.

- Principio de Sustitución de Liskov: este principio también se ve en los comodines y los tarots donde los comodines y tarots pueden ser utilizados a través de su clase madre sin la necesidad que desde fuera se sepa la distinguir la clase.

- Principio de Inversión de Dependencia: por ejemplo la implementación del patrón “Chain of Responsibility” cumple este principio dado que en este caso, la clase AnalizadorMano(módulo de alto nivel) depende de la interfaz EvaluadorMano para configurar la cadena de evaluadores(módulos de bajo nivel). Tanto los módulos de alto y bajo nivel dependen de la abstracción EvaluadorMano. Esto elimina la dependencia directa entre los módulos de alto nivel y los detalles de implementación, cumpliendo con el principio.