FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — JAVA

[7507/9502] Paradigmas de Programación
Curso Suárez
Segundo cuatrimestre de 2024

Alumno:	KRESTA, Facundo Ariel
Número de padrón:	110857
Email:	fkresta@fi.uba.ar

Alumno:	PAGANI, Lucas Nicolas
Número de padrón:	110777
Email:	lpagani@fi.uba.ar

Alumno:	ZANONI MUTTI, Ignacio
Número de padrón:	110884
Email:	izanoni@fi.uba.ar

Índice

1. Introducción

El presente informe reúne la documentación de la solución del segundo trabajo práctico de la materia Paradigmas de Programación, que consiste en desarrollar un juego en JAVA utilizando los conceptos del paradigma de la orientación a objetos vistos en el curso.

2. Supuestos

Seleccionada: Se toma que una carta que se seleccione, si se la vuelve a seleccionar, la misma se deseleccionará.

Puntaje: Toda modificación de puntaje y/o multiplicador resultará en enteros positivos.

EscaleraReal: Aunque tengan el mismo puntaje base y multiplicador que Escalera de Color, serán clases distintas ya que se podrán modificar aisladamente.

Carta: El valor del AS, J, Q, K; sera de 10 (a diferencia del juego original donde AS vale 11 y las figuras 10).

Tarot: Los tarots para juegos especificos de puntos y multiplicador, suman estos valores en vez de modificarlos.

Tienda: Al no haber dinero en el juego, la tienda habilitará una compra por turno.

Tienda: Una vez poseidos 5 comodines no se podrán ni agregar ni quitar los mismos, así mismo al intentar comprar un nuevo comodín de la tienda, esto no será posible. De forma similar ocurrirá al tener dos tarots para usar en las cartas e intentar comprar uno nuevo.

3. Diagramas de clase

3.1. Relación entre Ronda, Mano, Seleccionadas y Mazo

En la figura 1, se muestra la relación entre Ronda, Mano, Seleccionadas y Mazo. Notemos como Ronda implementa Observado, lo cual servirá para el patrón Observer[1]

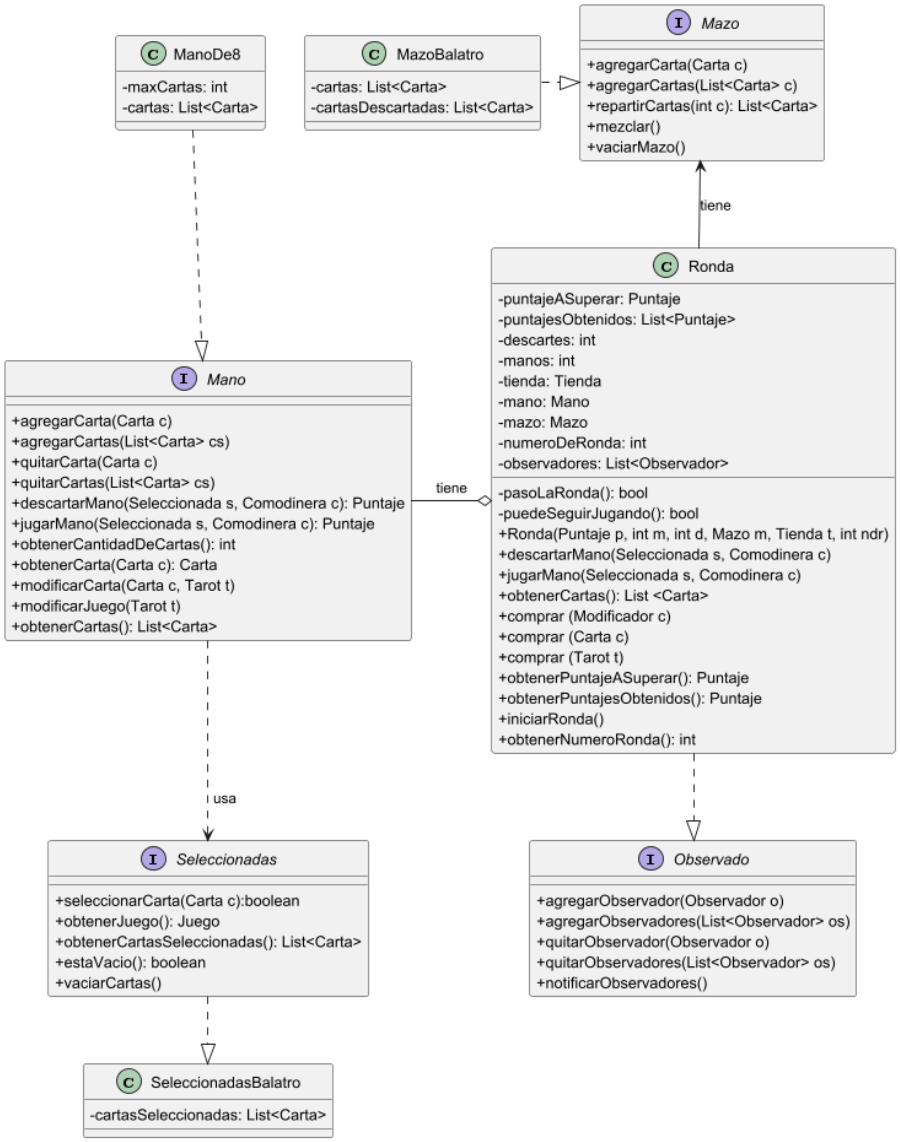

Figura 1: Diagrama de la relación entre Ronda, Mano, Seleccionadas y Mazo.

[1] explicado en detalles de implementación 4.3.

3.2. Carta

En este diagrama de clases podemos ver como la clase Carta tiene como atributos al Palo y al Tarot, siendo estas dos, una interfaz y una clase abstracta respectivamente.

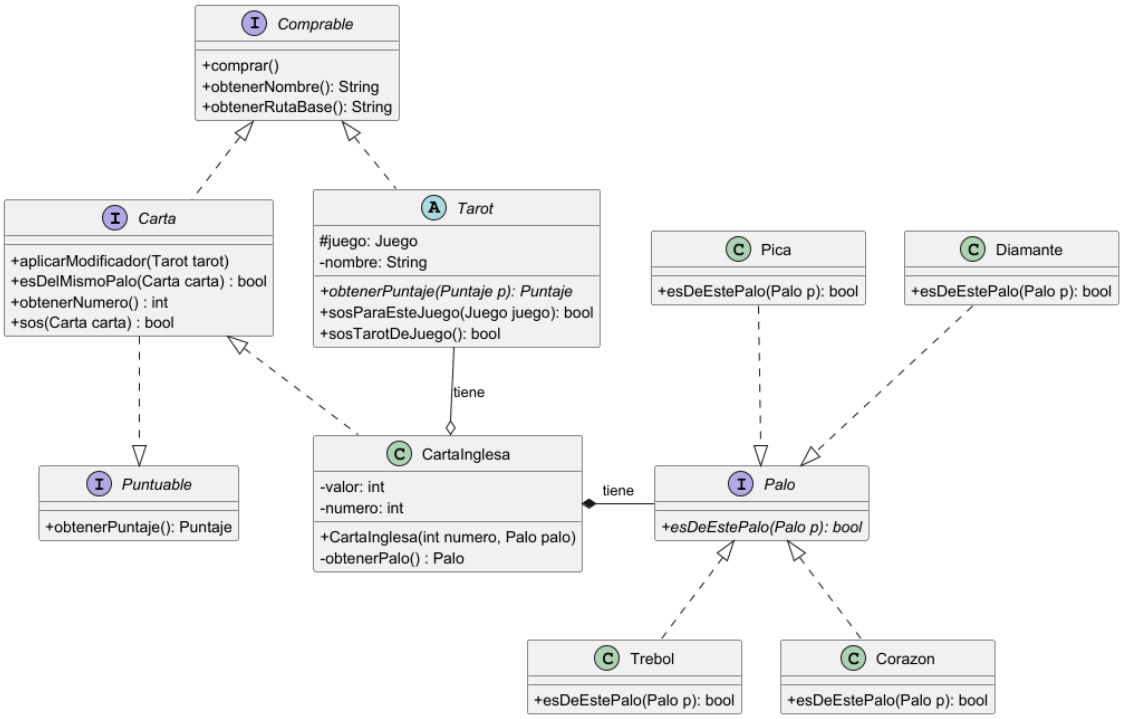

Figura 2: Carta.

3.3. Tarot

Al igual que con los otros dos diagramas vistos, aquí se ve Tator más a detalle, con sus respectivas clases las cuales heredan del mismo y las interfaces que implementa.

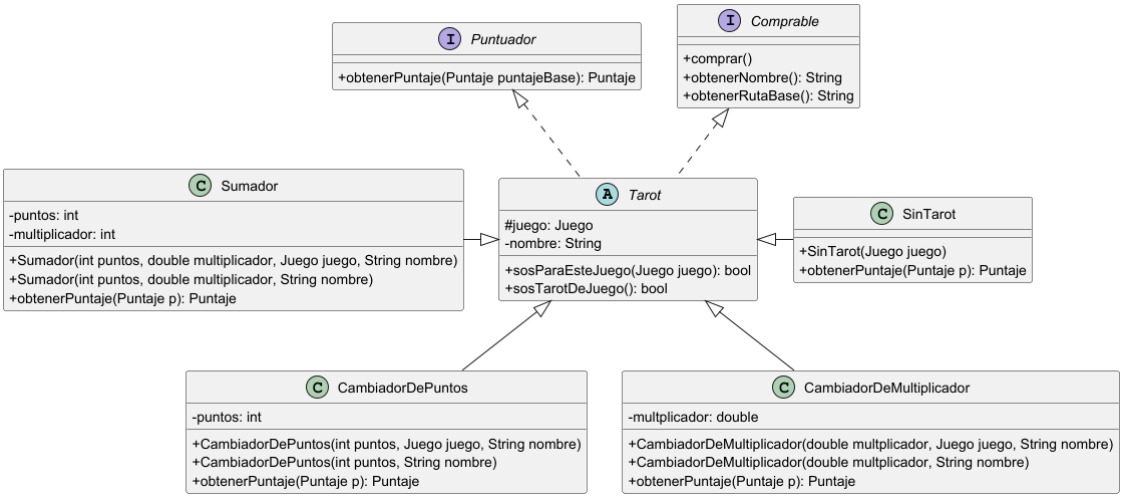

Figura 3: Tarot.

3.4. Tarotera

En este diagrama podemos observar como Tarotera implementa Observado y ademas tiene una lista de Tarots.

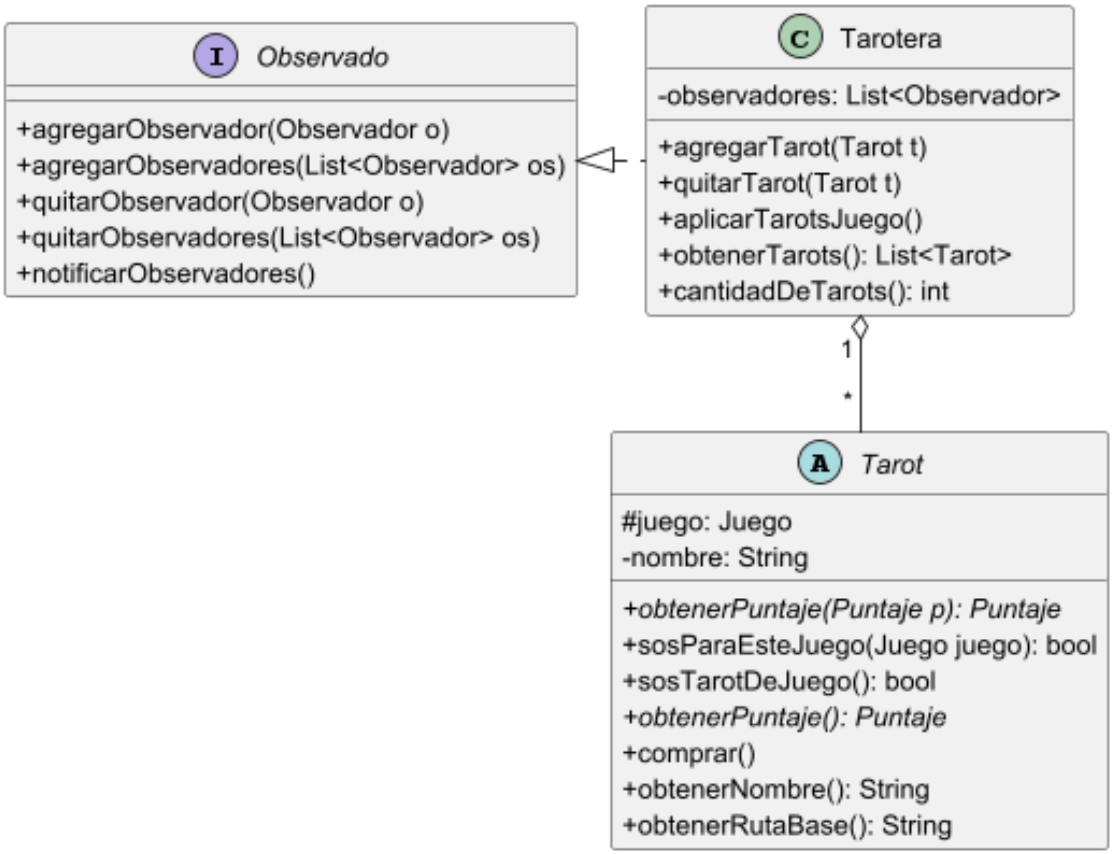

Figura 4: Tarotera.

3.5. Comodinera

Aqui se puede ver como Comodinera, al igual que Tarotera, implementa Observado y tiene una lista de Modificadores.

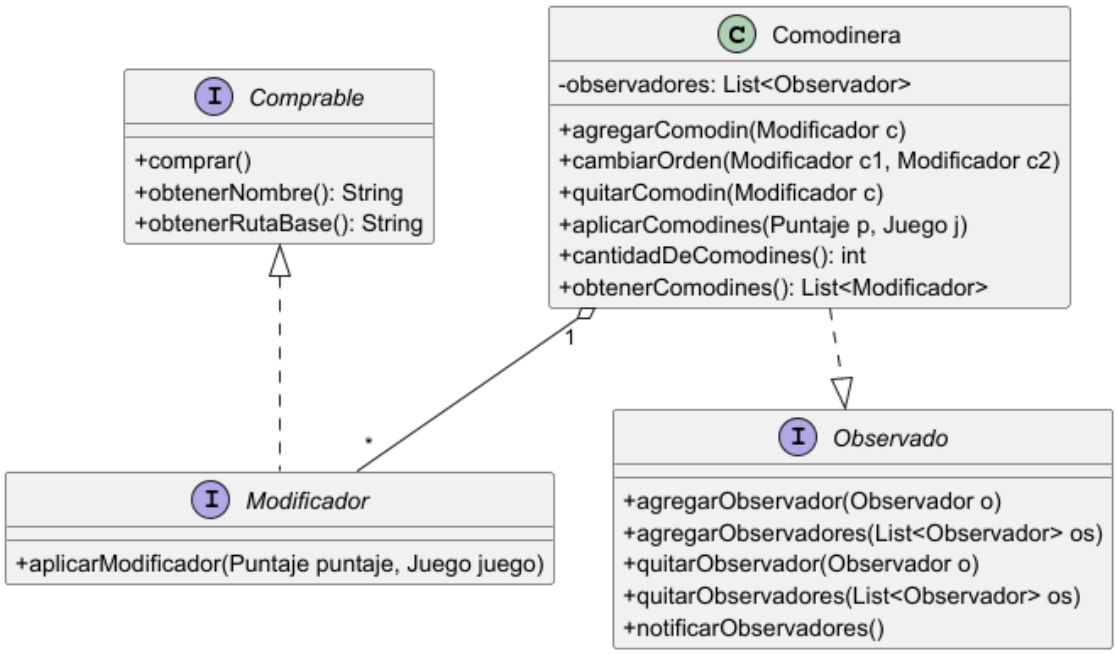

Figura 5: Comodinera.

3.6. TiendaYComprable

En este diagrama podemos ver como Tienda Balatro tiene una lista de comprables y se especifica quienes son realmente los comprables.

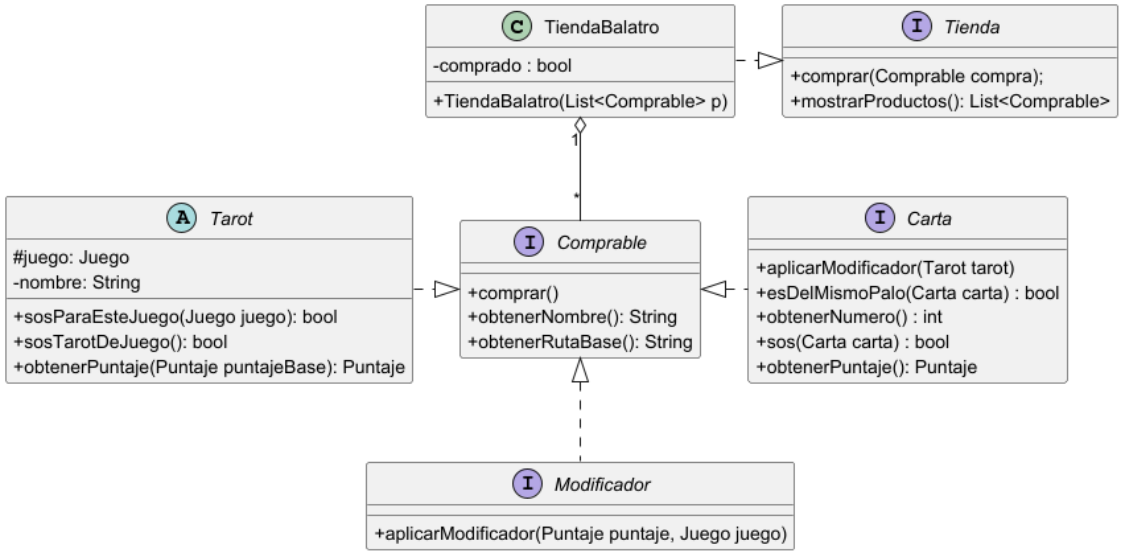

Figura 6: TiendaYComprable.

3.7. Juego

En este diagrama (figura 7) se explicita Juego con todas las clases herederas del mismo. Notar que Juego tiene un método de clase que será el encargado de crear la instancia de Juego correcta al momento de jugar las Seleccionadas

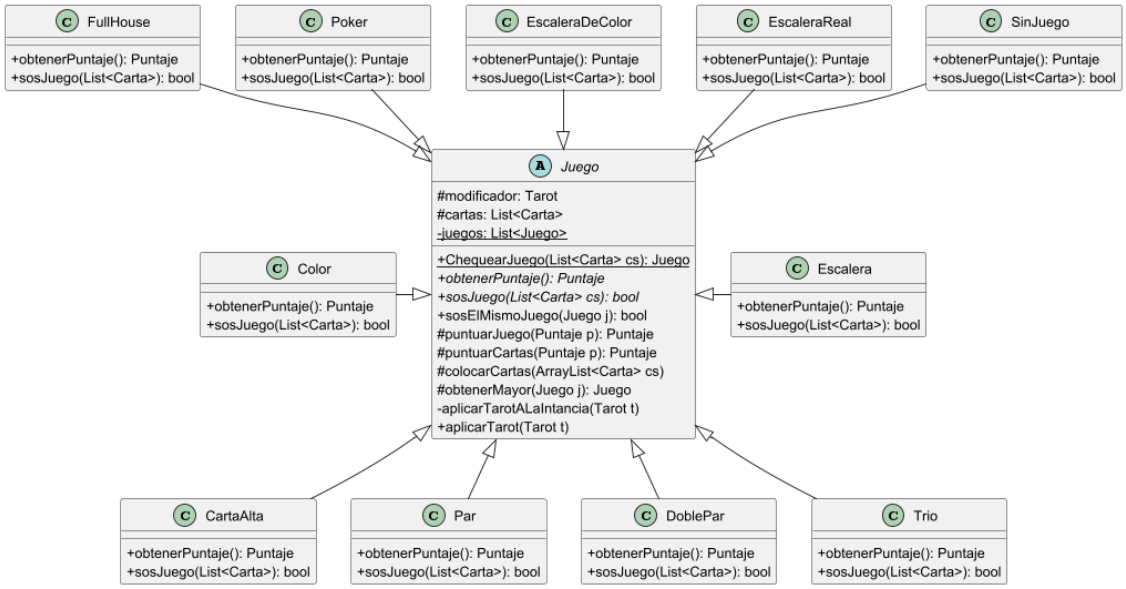

Figura 7: Juego.

3.8. Puntaje

En este diagrama (figura 8) se explicita las interfaces que hacen uso de Puntaje.

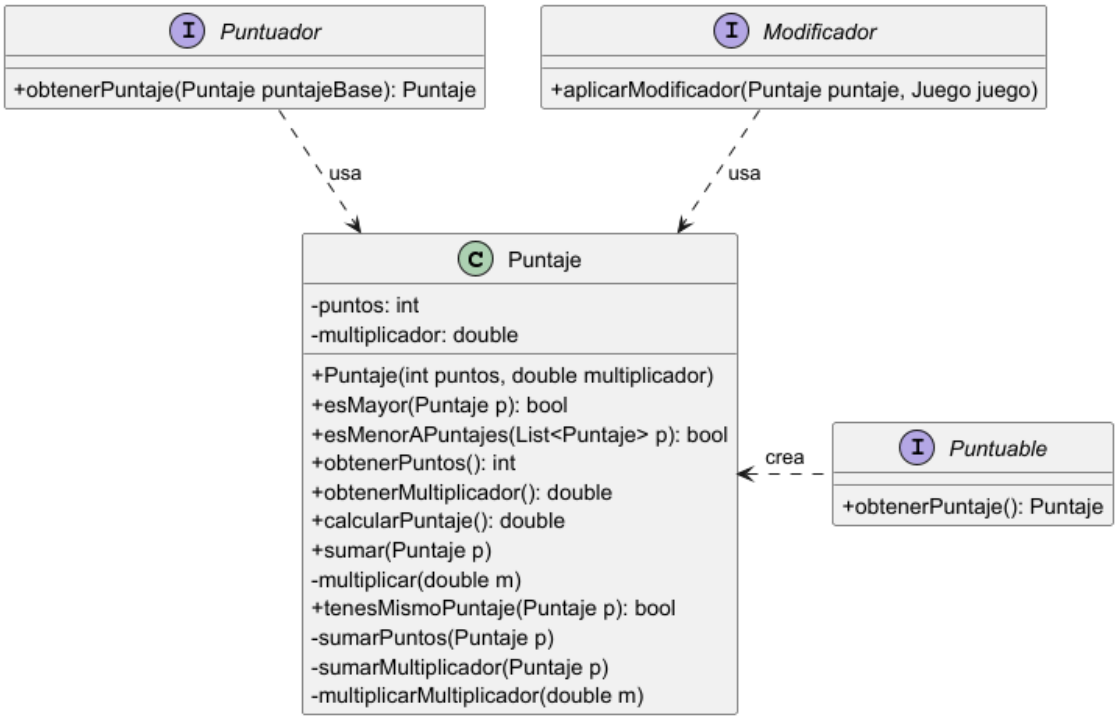

Figura 8: Puntaje.

3.9. Comodin

Aquí se ven tanto los Comodines, y sus respectivos tipos, siendo uno de ellos ComodínCombiación que tal cual se ve, es tan solo varios comodines aplicados a la vez.

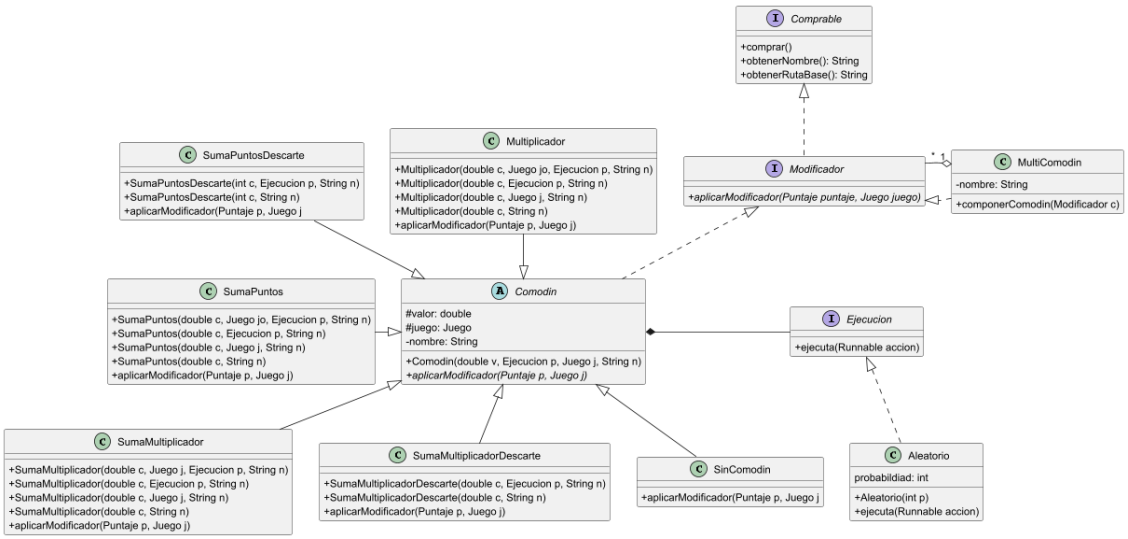

Figura 9: Comodin.

4. Detalles de implementación

4.1. Reconocer el juego óptimo

Un detalle interesante de implementación se da en la clase Juego, particularmente en el método de clase chequearJuego(). Allí se recibe una lista de cartas y a partir de ellas, comprobando cada juego, se devuelve una instancia del juego que maximice el puntaje, inyectando la dependencia. Como minimo, el juego sera SinJuego donde no habría ninguna carta añadida, y ya si hay al menos una, como mínimo, cartaAlta.

```
public static Juego chequearJuego(ArrayList<Carta> cartas) {
    Juego juegoSeleccionado = new SinJuego();
    ...
    return juegoSeleccionado;
}
```

Se ve como queda la variable juegoSeleccionado se inicializa con una instancia de SinJuego, y que esta solo cambia si se encuentra un Juego mejor que este; tomando en cuanta que mejor significa que da un mayor puntaje, ya sea porque de base es mejor juego, o porque al ser afectado por algún modificador este cambia. Luego para recorrer los juegos solo se recorren los juegos posibles, comprobando lo antes dicho:

```
public static Juego chequearJuego(ArrayList<Carta> cartas) {
    ...
    for (Juego juegoActual : juegos) {
        juegoActual.colocarCartas(cartas);
        juegoSeleccionado = juegoActual.obtenerMayor(juegoSeleccionado);
    }
    ...
}
```

4.2. Tarot

Para resolver el problema de los tarots, usamos el patrón Strategy, donde delegamos el comportamiento en clases concretas: CambiadorDeMultiplicador, CambiadorDePuntos, Sumador y SinTarot las cuales heredan de la clase abstracta Tarot y se inyectan a la carta y al juego a traves de un método. Por defecto, ambos van a tener SinTarot el cual tambien es un Tarot. En carta se ve así:

```
public class CartaInglesa {
    Tarot modificador;
    ...
    public CartaInglesa(int numero, Palo palo) {
        ...
        this.modificador = new SinTarot();
        ...
    }
}
```

Y a la hora de modificar el estado del Tarot de la carta, a través del método aplicarModificiador:

```
public class CartaInglesa {
    Tarot modificador;
    ...
    public void aplicarModificador(Tarot tarot) {
        this.modificador = tarot;
    }
}
```

4.3. Implementación comodín

Notar que para realizar los comodines, hemos usado el patrón Composite, donde Comodín es una clase abstracta que implementa Modificador y las que hereden de la misma serán clases concretas. Ademas, tenemos multicomodín el cual también implementa Modificador y tiene una lista de los mismos, donde al momento de calcular el puntaje, no hace mas que recorrerlos y aplicarlos uno por uno.

```
public class MultiComodin implements Modificador {
    private final ArrayList<Modificador> comodines = new ArrayList<>();

    ...

    public void componerComodin(Modificador comodin) {
        comodines.add(comodin);
    }

    @Override
    public void aplicarModificador(Puntaje puntaje, Juego juego) {
        for (Modificador comodin : comodines) {
            comodin.aplicarModificador(puntaje, juego);
        }
    }
```

4.4. Implementación patrón observer

Para poder realizar la vista .ªctualizable.ên los momentos donde se cambia un estado el cual se está mostrando, usamos el patrón Observer. La notificación fue de tipo "Pull"donde solo se notificaba a los subscriptores que el estado había cambiado, siendo ellos los encargados de mediante getters al Observado, actualizarse.

Aquí se ve cuando la ronda -cuando inicia- notifica a todos sus observadores que inició. En Particular le interesa a las cartas en mano que se muestran en vista a través de CartasEnMano, que es uno de los observadores que tiene ronda, ya que la ronda al inciar reparte las cartas del mazo a la mano.

```
public void iniciarRonda() {
        this.mano.agregarCartas(this.mazo.repartirCartas(8));
        this.notificarObservadores();
    }
```

Es la vista que entre incializaciones varias se encarga de agregar al CartasEnMano como observador de la ronda.

```
    ...
    ronda.agregarObservador(new CartasEnMano(...));
    ...
```

4.5. Implementación patrón MVC

Para la vista, creamos varios tipos de clases y controladores, haciendo uso del patrón MVC, separando así, la aplicación en 3 partes. Toda la lógica corresponde al modelo, el cual efectivamente tiene un bajo acoplamiento con vistas y controladores. Notar como el modelo siquiera conoce la vista y los controladores, siendo estos notificados de cambios de estados ya que han sido añadidos como observadores.

4.6. Implementación Aleatorio

Para la implementación de Ejecución, interfaz que implementa la clase Aleatorio, a esta se le envía el mensaje ejecuta(), enviando dentro una acción, para que esta la ejecute si es que la probabilidad resulta.

```
public interface Ejecucion {
    void ejecuta(Runnable accion);
}
```

En esta implementación se inclina a usar una función lambda más propia del paradigma funcional, únicamente por su simplicidad de uso y la poca lógica que tiene, ademas de lo poco que otras clases interactúa con esta. Pero es importante detallar que pudo -quizás debió- ser utilizado el patrón Command.

5. Excepciones

ProbabilidadInvalidaException Es un error que se lanza cuando se le asigna una probabilidad menor a cero a un objeto de tipo Aleatorio.

CantidadComodinInvalidaException Es un error que se lanza cuando el valor asignado al crear un objeto de tipo Comodin es menor a cero.

ComodinNoEnComodineraException Es un error que se lanza cuando se quiere quitar un Comodin que no esta en la Comodinera.

NumeroInvalidoException Es un error que se lanza cuando se quieren crear una carta con un valor que no este entre uno y trece.

CartaNoEnManoException Es un error que se lanza cuando se quiere seleccionar o deseleccionar una carta de la mano la cual no esta en la mano.

CartasInsuficientesException Es un error que se lanza cuando se le pide al mazo repartir una mayor cantidad de cartas a las que tiene.

ManoLlenaException Es un error que se lanza cuando se quiere agregar una carta nueva en una mano que ya tiene la cantidad limite.

MaximoCartasSeleccionadasException Es un error que se lanza cuando se quiere agregar una sexta carta a la mano de cinco.

SinCartasSeleccionadasException Es un error que se lanza cuando se quiere jugar una mano de 5 pero no se ha seleccionado ninguna carta.

MultiplicadorInvalidosException Es un error que se lanza cuando se quiere instanciar al Puntaje con un multiplicador menor a cero.

PuntosInvalidosException Es un error que se lanza cuando se quiere instanciar al Puntaje con unos puntos bases menores a cero.

CantidadDeDescartesInvalidaExcepcion Es un error que se lanza cuando la ronda se instancia con una cantidad de descartes menores o iguales a cero.

CantidadDeManosInvalidaExcepcion Es un error que se lanza cuando la ronda se instancia con una cantidad de manos menores o iguales a cero

PasoLaRondaException Es un error que se lanza cuando el jugador supera el puntaje de la ronda.

PerdioLaRondaException Es un error que se lanza cuando el jugador no supera el puntaje de la ronda y ya no tiene manos para jugar.

MultiplicadorInvalidoTarotException Es un error que se lanza cuando se quiere modificar el multiplicador por uno menor o igual a cero.

PuntosNegativosTarotException Es un error que se lanza cuando se quiere modificar los puntos por unos nuevos los cuales son menores a cero.

TarotNoEnTaroteraExcepcion Es un error que se lanza cuando se quiere quitar un Tarot que no esta en la Tarotera.

MaximoProductosComprablesExcepcion Es un error que se lanza cuando se quiere comprar mas productos de los permitidos.

ProductoNoEnTiendaExcepcion Es un error que se lanza cuando se quiere comprar un producto el cual no se encuentra en la tienda.

6. Diagramas de secuencia

6.1. Diagrama de secuencia de la inicialización de Cartas

Aquí se ve la inicialización de las cartas en una lista que luego los demás diagramas de secuencia emplearán para mejorar la visibilidad.

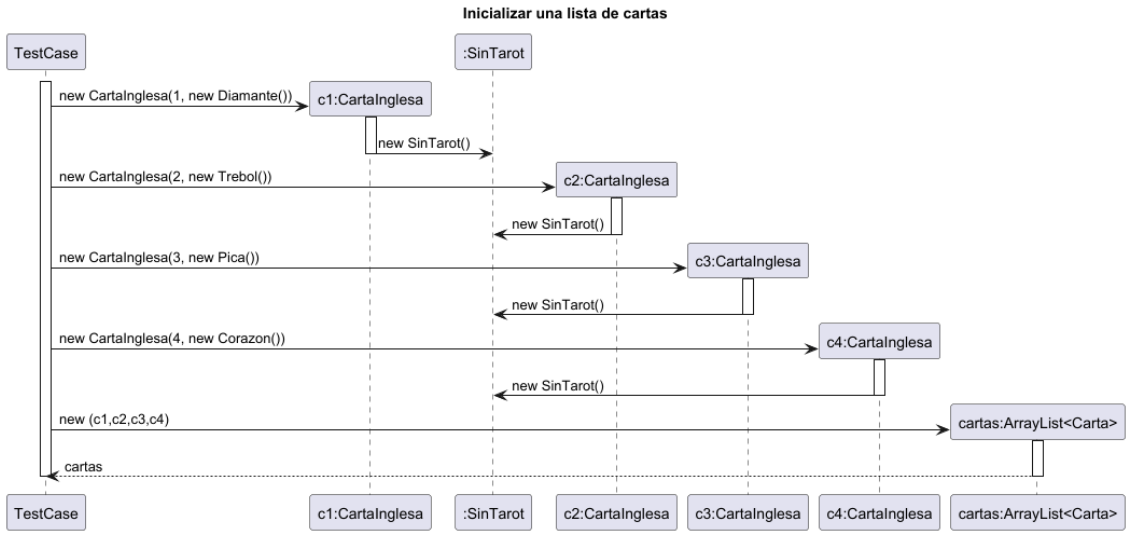

Figura 10: Inicializar una lista de cartas

6.2. Diagrama de secuencia del caso de uso 1

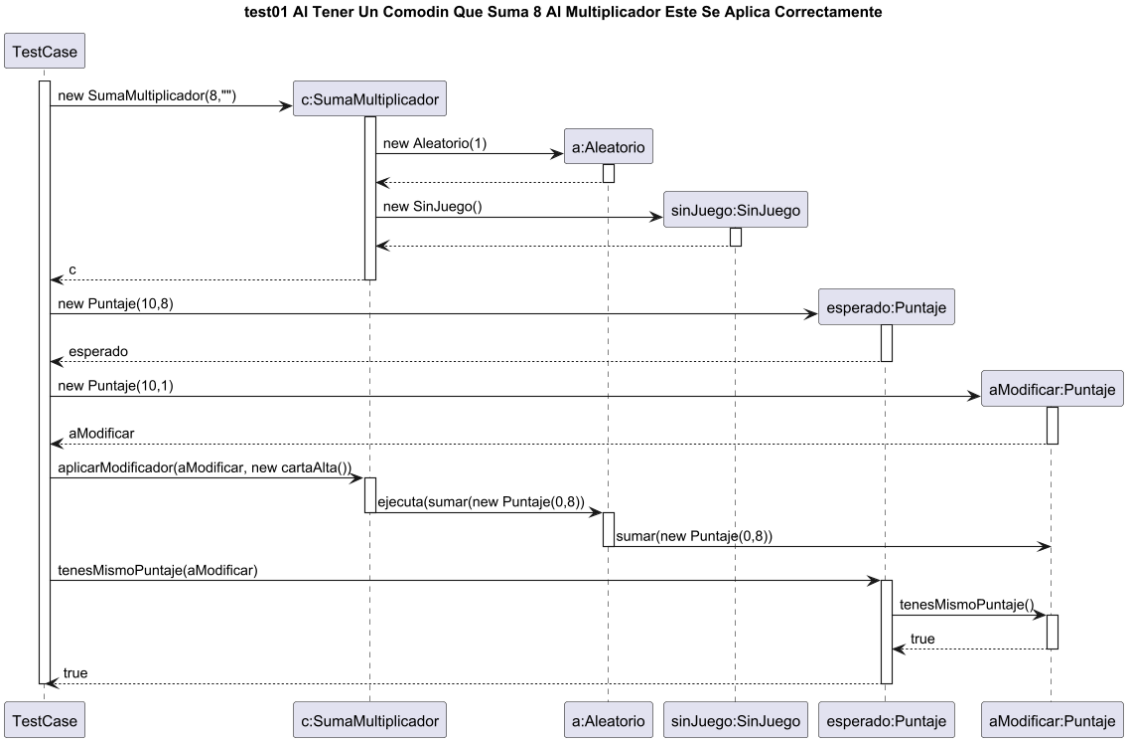

Figura 11: Al Tener Un Comodin Que Suma 8 Al Multiplicador Este Se Aplica Correctamente

6.3. Diagrama de secuencia del caso de uso 2

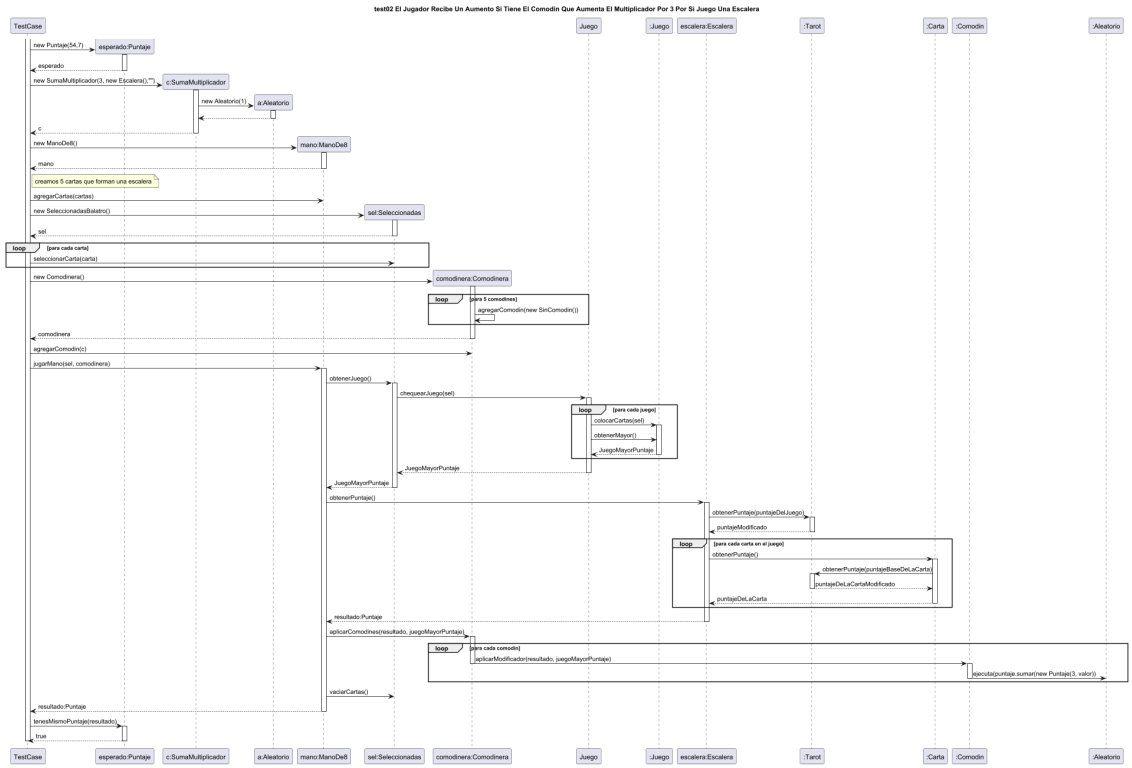

Figura 12: El Jugador Recibe Un Aumento Si Tiene El Comodin Que Aumenta El Multiplicador Por 3 Por Si Juego Una Escalera

6.4. Diagrama de secuencia del caso de uso 3

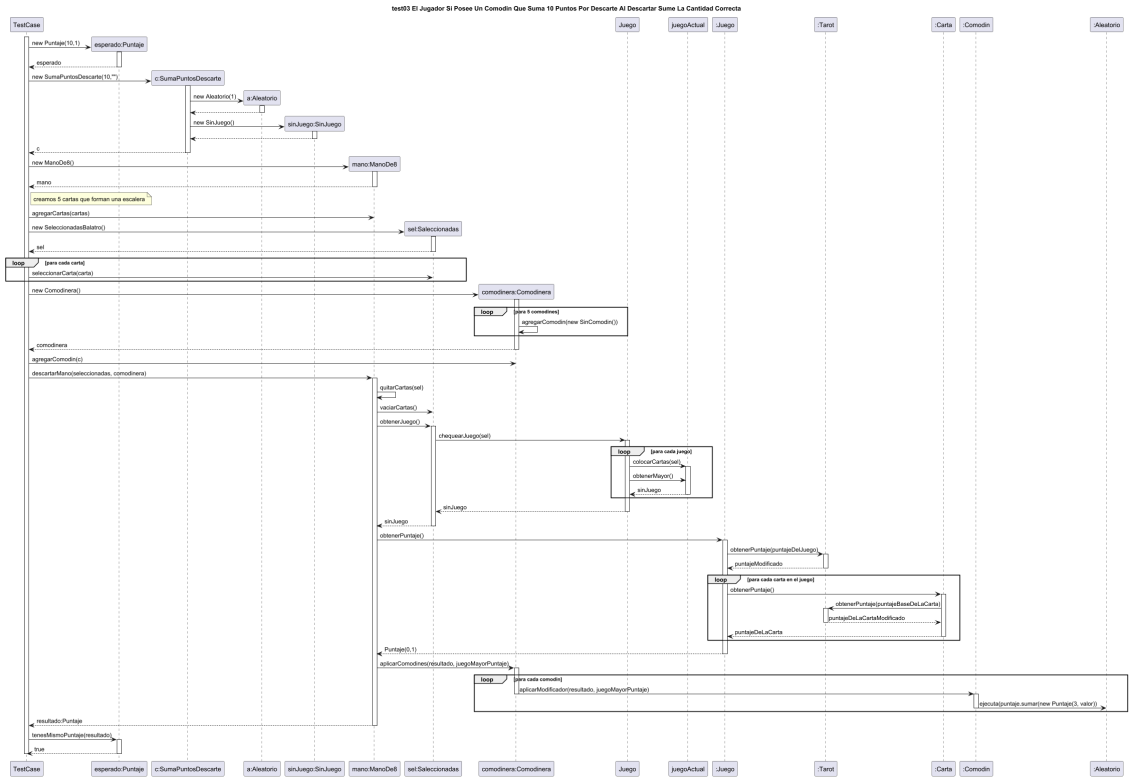

Figura 13: El Jugador Si Posee Un Comodin Que Suma 10 Puntos Por Descarte Al Descartar Sume La Cantidad Correcta

6.5. Diagrama de secuencia del caso de uso 4

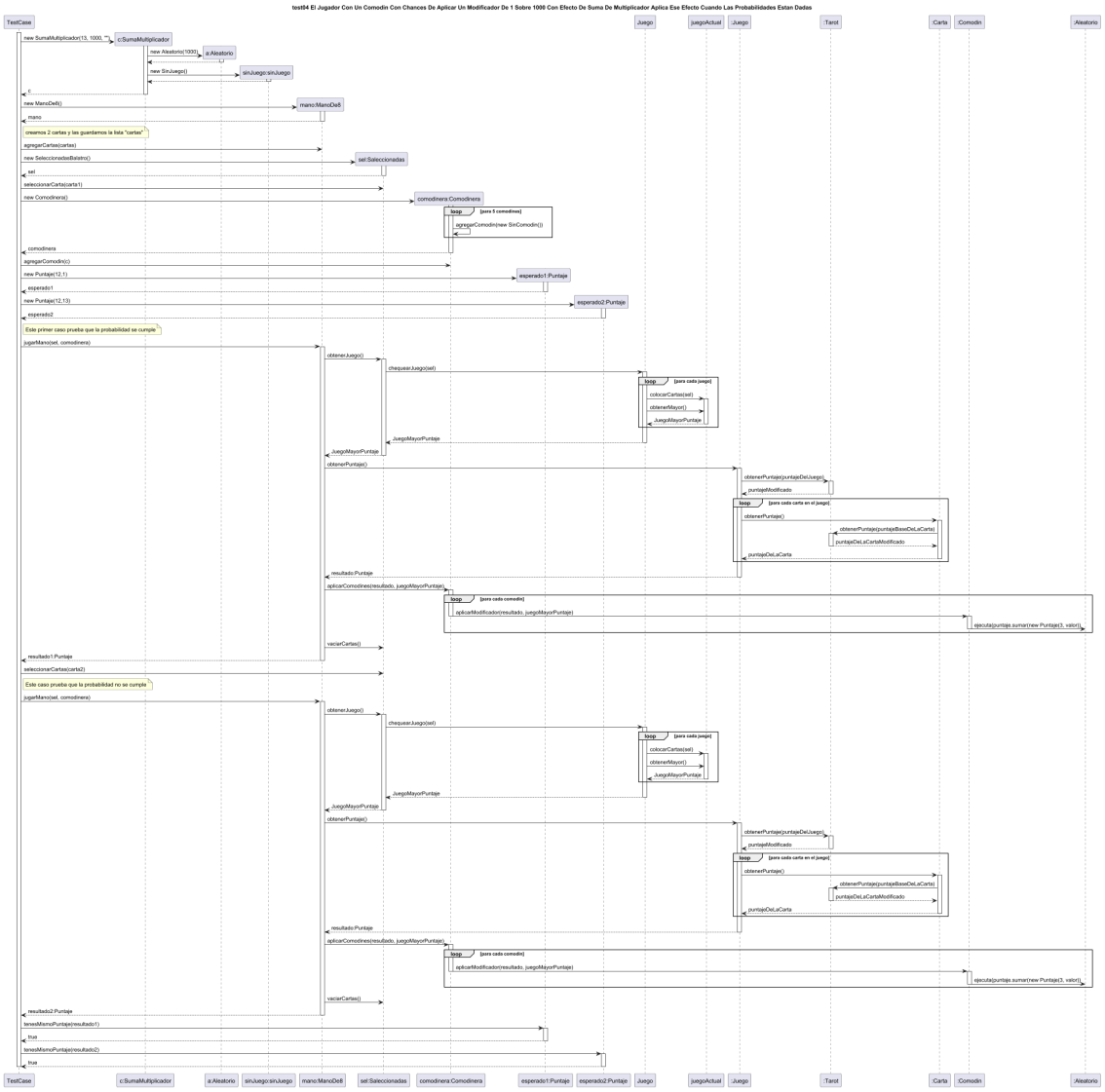

Figura 14: El Jugador Con Un Comodin Con Chances De Aplicar Un Modificador De 1 Sobre 1000 Con Efecto De Suma De Multiplicador Aplica Ese Efecto Cuando Las Probabilidades Estan Dadas

6.6. Diagrama de secuencia del caso de uso 5

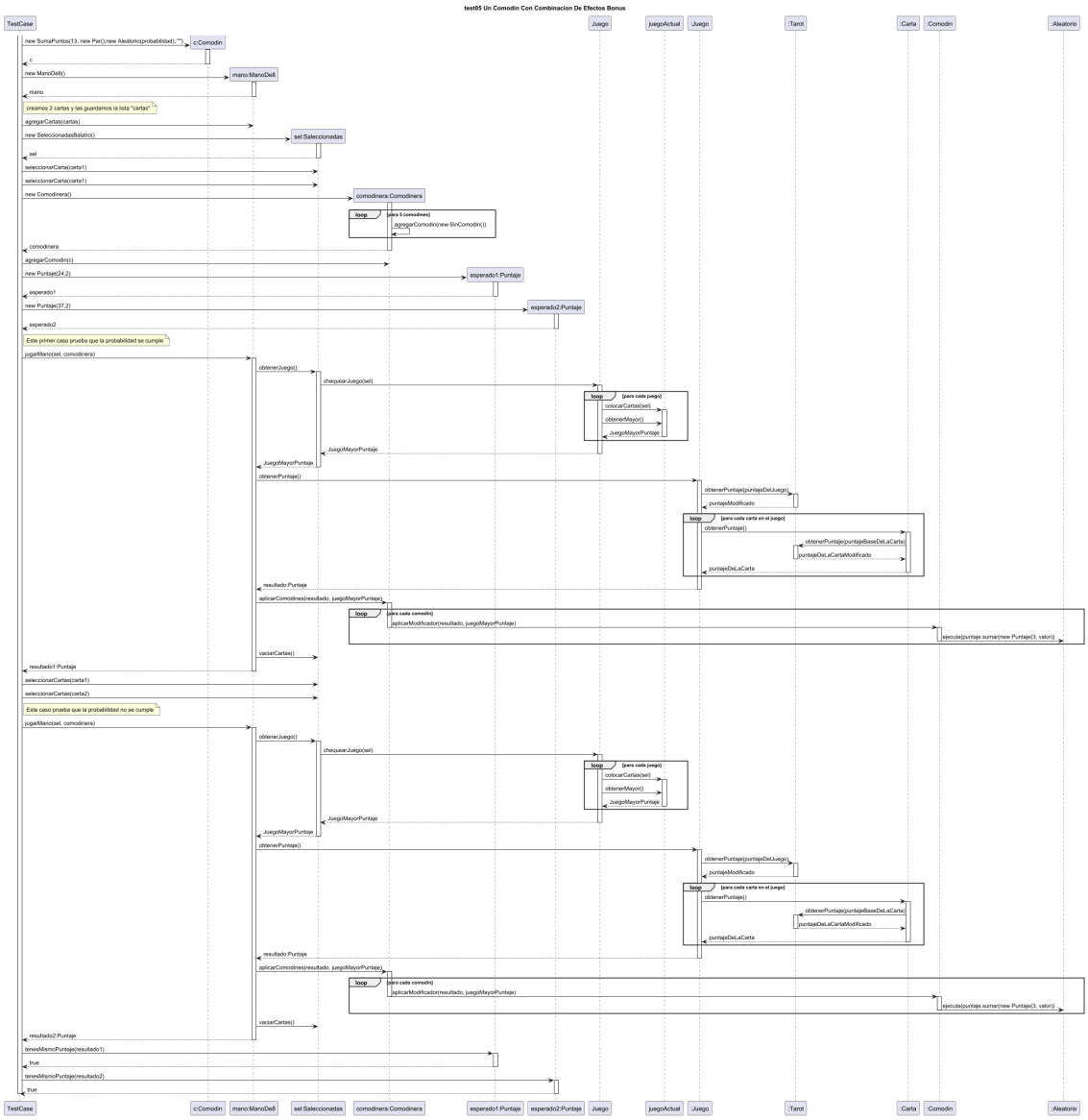

Figura 15: Un Comodin Con Combinacion De Efectos Bonus

7. Diagramas de Paquetes

7.1. Diagrama de paquetes del modelo

En este diagrama de paquete muestra los distintos paquetes del modelo y sus relaciones e interacciones.

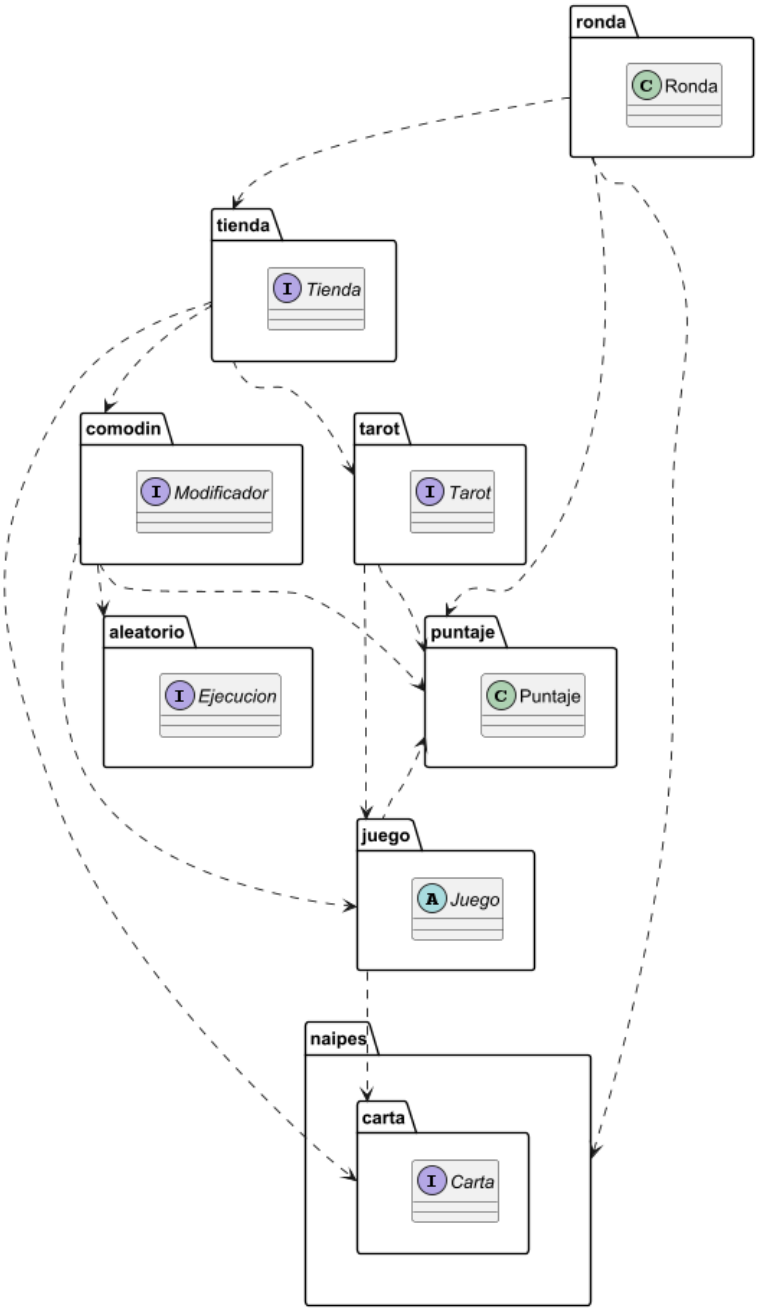

Figura 16: Diagrama de paquetes del modelo.