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

Alumno:	GUERRERO, Steven
Número de padrón:	110067
Email:	sguerrero@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

ManoDe5: Se toma que una carta que pertenezca a ManoDe5 (que ha sido seleccionada) 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, serán clases distintas ya que se podrán modificar aisladamente.

Mazo: Un mazo unicamente tendrá las 52 cartas de la baraja inglesa.

Carta: El valor del AS es 1 y el valor de la J, Q, K; 11, 12 y 13 respectivamente (a diferencia del juego original donde AS vale 11 y las figuras 10).

3. Diagramas de clase

3.1. Relación entre Jugador, Mano, ManoDe5 y Mazo

En la figura 1, se muestra la relación entre Jugador, Mano, ManoDe5 y Mazo. Como podemos ver, mazo tiene una dependencia con Mano, ya que le llega por parámetro al momento de repartir las cartas. Además el Jugador tiene dos manos, la de ocho y la de cinco. Notemos, que esta ultima hereda adhiriéndose totalmente a cumplir el contrato de su clase madre.

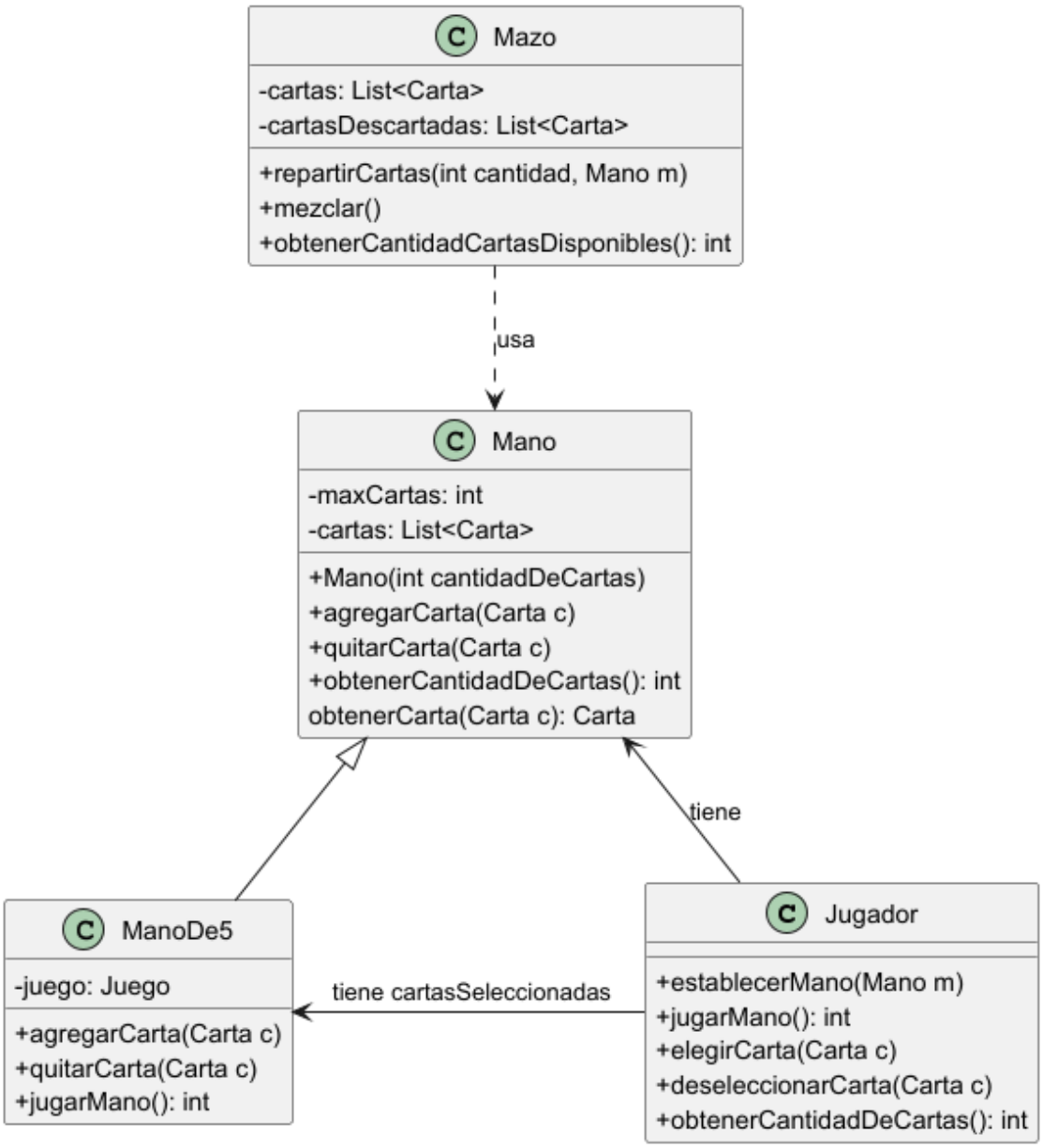

Figura 1: Diagrama de la relación entre Jugador, Mano, ManoDe5 y Mazo.

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, interfaces.

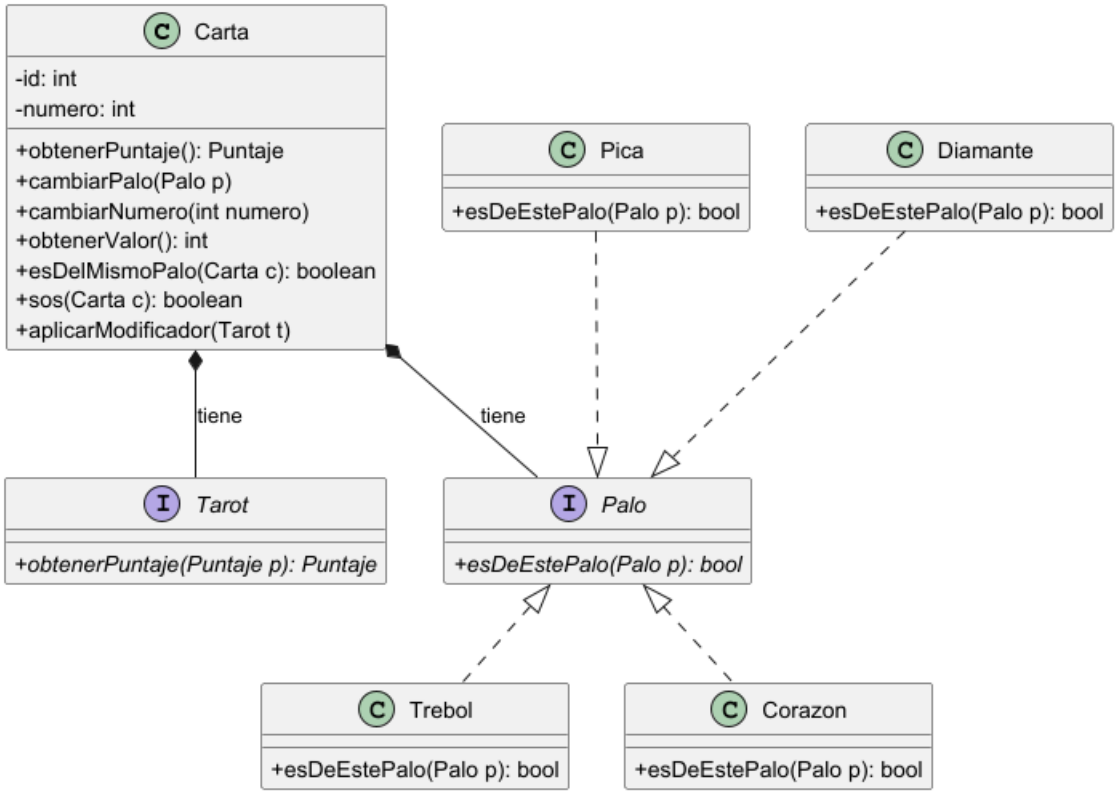

Figura 2: Carta.

3.3. Tarot

Al igual que con los otros dos diagramas vistos, aquí se ve Tator más a detalle, con sus respectivas clases que lo implementan.

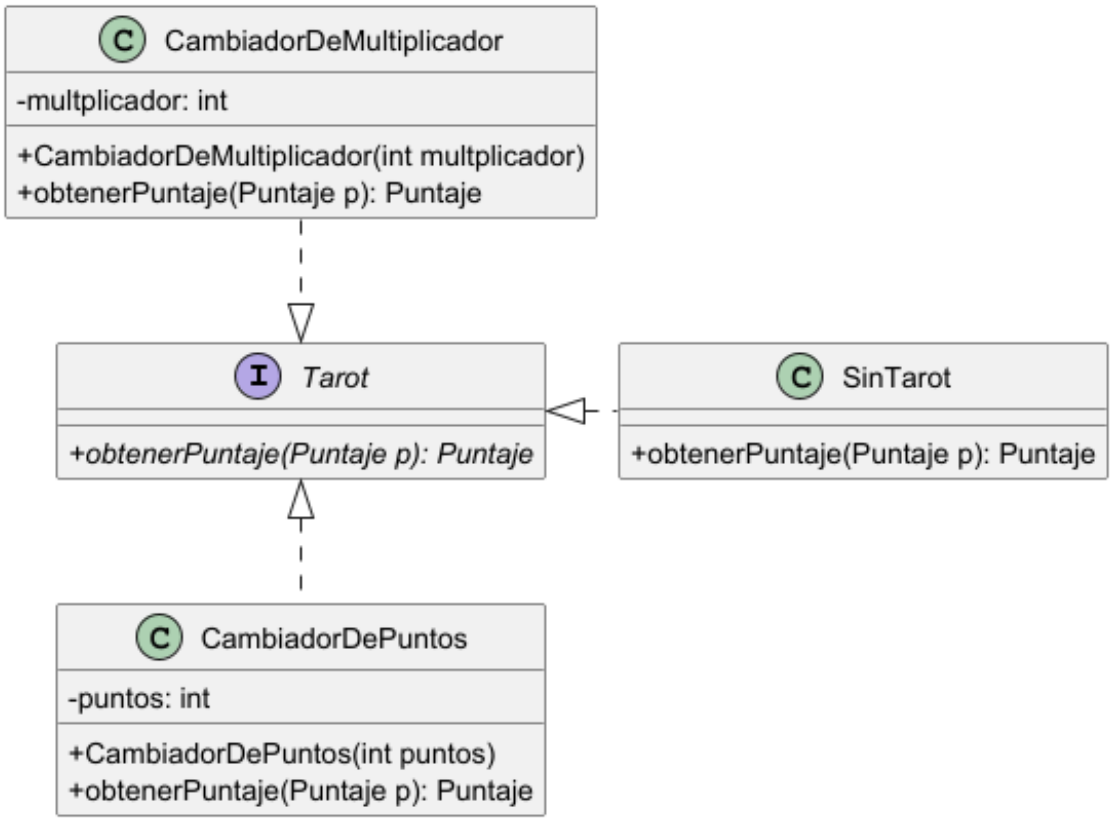

Figura 3: Tarot.

3.4. Juego

En este diagrama (figura 4) 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 modificar la mano de 5

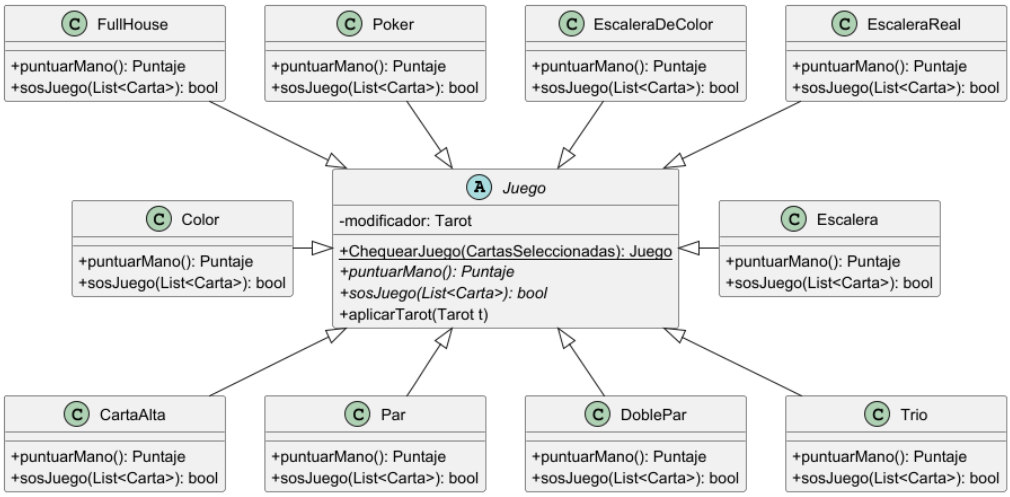

Figura 4: Juego.

3.5. Puntaje

En este diagrama (figura 5) se explicita las clases que hacen uso de Puntaje.

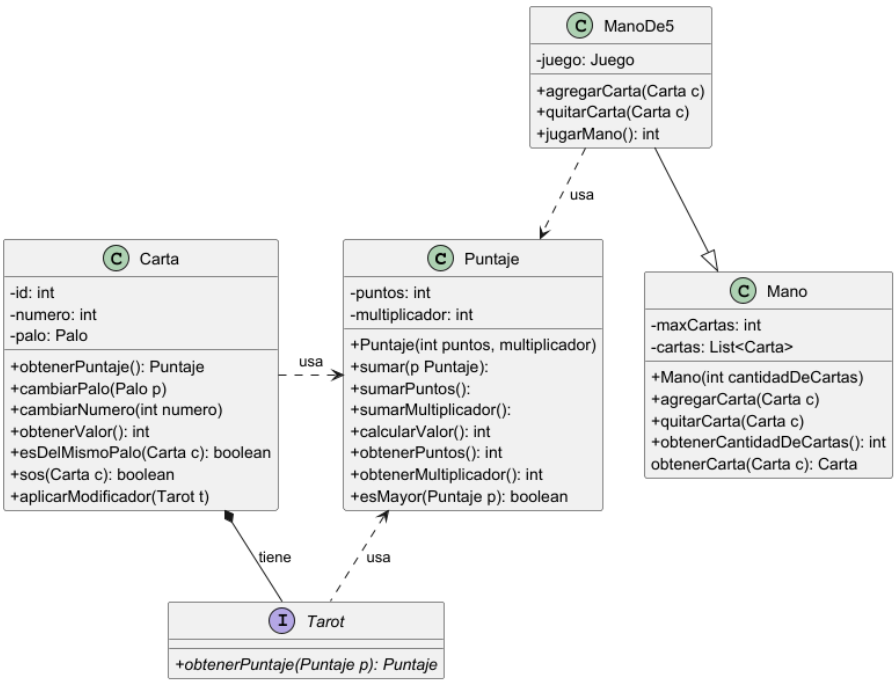

Figura 5: Puntaje.

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() aqui se usa el partón Strategy. Allí se recibe una lista de cartas y a partir de ellas, comprobando cada juego, se devuelve una instancia de juego que maximice el puntaje, inyectando la dependencia. Además como al menos ha de haber una carta para que este método sea invocado, siempre CartaAlta será una opción válida.

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

Se ve como queda la variable juegoSeleccionado se inicializa con una instancia de CartaAlta, 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:

```
for (Juego juegoActual : juegos) {
    if (juegoActual.sosJuego(cartas)) {
        ...
        if (puntajeJuegoActual.esMayor(puntajeJuegoSeleccionado)) {
            juegoSeleccionado = juegoActual;
        }
    }
}
```

Además compara si JuegoActual, juego de la lista de juegos, es juego de las cartas provistas.

4.2. Tarot

Para resolver el problema de los tarots, usamos el patrón Strategy, donde delegamos el comportamiento en clases concretas: CambiadorDeMultiplicador, CambiadorDePuntos, SinTarot las cuales implementan la interfaz 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 Carta {
    Tarot modificador;
    ...
    public Carta(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 Carta {
    Tarot modificador;
    ...
    public void aplicarModificador(Tarot tarot) {
```

```
            this.modificador = tarot;
        }
    }
```

4.3. Herencia de ManoDe5

ManoDe5 hereda directamente de la clase concreta Mano, usualmente esto tendría altas chances de violar el Principio de Sustitución de Liskov. No obstante, no ocurre en este caso porque la clase hija se compromete a cumplir la totalidad del contrato de la clase madre. Una diferencia en el comportamiento de ambas es que a diferencia de Mano, ManoDe5 no permite tener cartas repetidas, de hecho al intentar agregarle una carta, esta misma si estaba ya en la ManoDe5, esta se quita de la misma.

```
@Override
public void agregarCarta(Carta carta) {
    ..
    if (this.cartas.contains(carta)) {
       this.quitarCarta(carta);
    } else {
        this.cartas.add(carta);
    }
    ...
}
```

Ademas de redefinir el método agregarCarta(), redefine el método quitarCarta(), agregando en ambos una comprobacion de juego[1]

```
{
    ...
    this.juego = Juego.chequearJuego(this.cartas);
}
```

5. Excepciones

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.

[1] Explicado en Reconocer el juego Óptimo (pag4.1).

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.

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

6. Diagramas de secuencia

6.1. Diagrama de secuencia del método jugarMano()

Este diagrama muestra como se ejecuta el mensaje jugarMano(), enviado por el jugador de forma genérica.

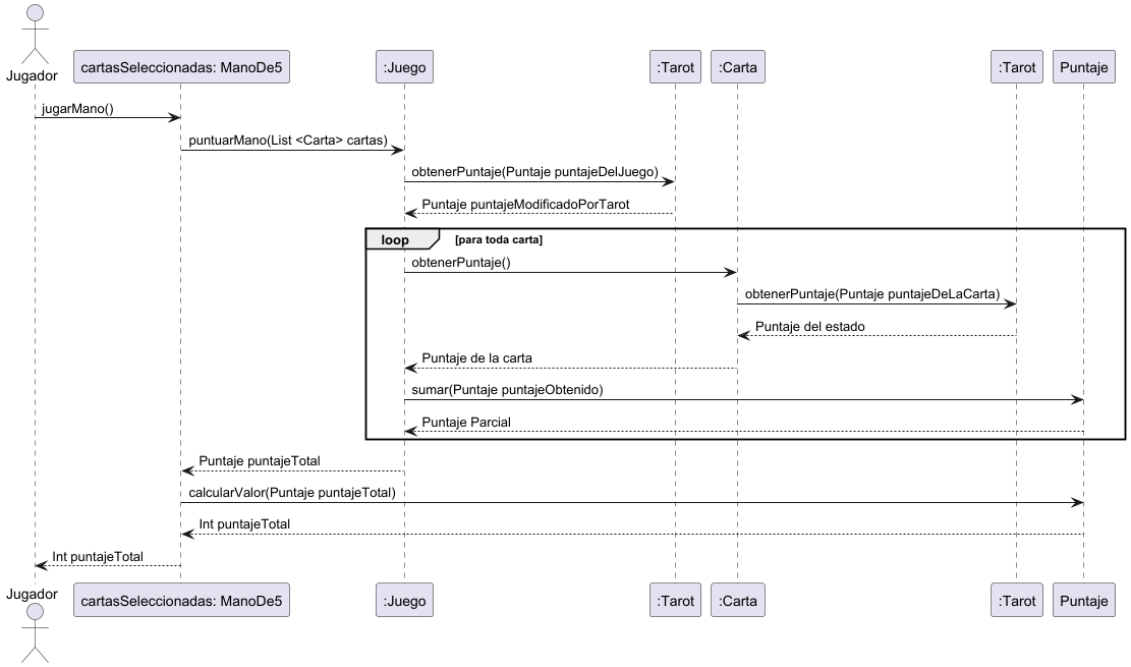

Figura 6: Diagrama de secuencia del método puntuarMano().

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. Nótese que para tanto la clase Juego como la clase Tarot, los objetos que dependen de estas, interactúan únicamente con el paquete de las mismas.

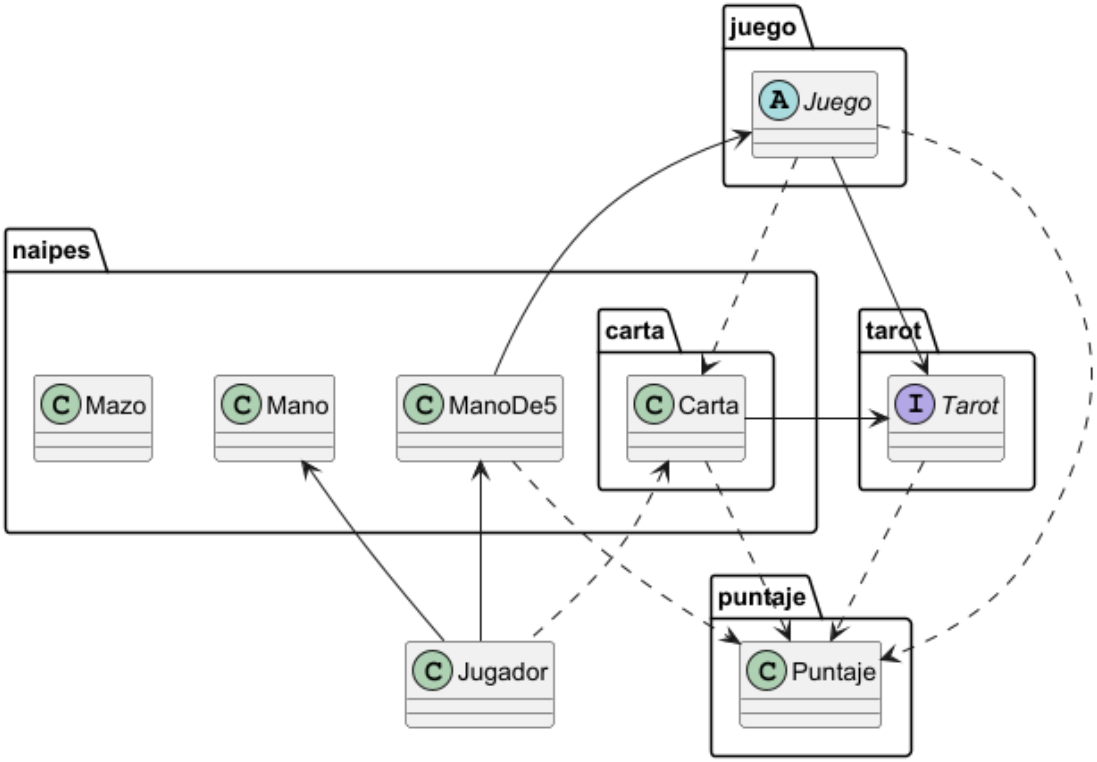

Figura 7: Diagrama de paquetes del modelo.