**Universidad CENFOTEC**



**Estructura de Datos 1.**

**Segundo Proyecto Programado**

**Profesor:** Christhian Sibaja.

**Estudiantes:**

Denis Mauricio Granados Ly

Rafael Alberto Briceño Aguilar.

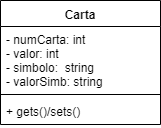
Denilson Ruiz Vargas

**Fecha de entrega:** 29/04/19

Cuatrimestre No.1 2019

# TAD Carta

**Representación abstracta**



**Definición de la invariante**

* numCarta no aceptará valores de menores a 1 y mayores a 52.
* El valor va del 2 al 14.
* La simbología solo va a tener 4 valores
* Tendrá valores del 2 al nueve o J, Q y K.

**Definición de operaciones**

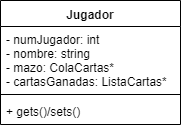
* Carta: (int, int, string, string) -> void // Constructor por defecto (Constructora)
* mostrarCarta: (void) -> void // Muestra valor simbólico y el símbolo de la carta. (Salida en panatalla)
* siguienteSimbolo: (int) -> string //Devuelve un símbolo a para ser asignado a una carta, de acuerdo con el número entero que reciba. (Analizadora)

**Manejo del error**

En este objeto no se implementa el manejo del error

# TAD Jugador

**Representación abstracta**



**Definición de la invariante**

* Cada jugador ingresado al juego tendrá una cola de cartas representadas como mazo, con el que jugará en el juego.
* Las cartas ganadas en el juego serán almacenas en una lista, dentro de la misma instancia del objeto jugador.

**Definición de operaciones**

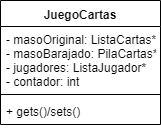
* obtenerPuntaje: (void) -> int //Devuelve la longitud de la lista de cartas ganadas por el jugador. (Analizadora)
* mostrarJugador: (void) -> void //Muestra en pantalla el numero del jugador, el nombre y su puntaje. (Salida en pantalla)

**Manejo del error**

Al atributo numJugador solo se le asignarán números enteros positivos mayores a 0.

# TAD JuegoCartas

**Representación abstracta**



**Definición de la invariante**

Se gestionan las estructuras de acuerdo con la abstracción y análisis del juego.

**Definición de operaciones**

* reiniciarContador: (void) -> void //Asigna 0 a la variable. (Modificadora)
* generarMaso: (void) -> bool //Genera cada una de las cartas que componen el maso. Retorna true si las cartas fueron generadas con éxito. (Modificadora)
* sumarContador: (void) -> void //Aumenta el valor de la variable en uno. (Modificadora)
* restarContador: (void) -> void //Disminuye el valor de la variable en uno. (Modificadora)
* buscarCarta: (int) -> Carta //Recibe como parámetro un numero entero con el que busca y retorna una carta de acuerdo con este. (Analizadora)
* barajarMaso: (void) -> void //Toma de manera aleatoria cartas de la lista de cartas generada, para almacenarlas en una pila. (Modificadora)
* ingresarJugadores: (void) -> void //Se ingresan el número y el nombre de los jugadores que van a participar en el juego y los añade a una lista. (Modificadora)
* repartirCartas: (void) -> void //Se reparten las cartas del maso barajado anteriormente generado a cada uno de los jugadores ingresados al juego. (Analizadora)
* darCarta: (int) -> void //Se da una carta a un jugador de acuerdo con el parámetro recibido. (Modificadora)
* iniciarJuego: (void) -> void //Da inicio al juego de cartas. (Analizadora)
* imprimirMenuJuego: (void) -> void //Imprime el menú del juego. (Salida en pantalla)
* tirarCartas: (void) -> void //Analiza la cantidad de jugadores en el juego para determinar los resultados del turno. (Analizadora)
* evaluarTurnoDosJugadores: (void) -> void //Determina el ganador del turno entre dos jugadores y acumula las cartas de los perdedores en el maso de cartas ganadas de jugador ganador. (Modificadora)
* evaluarTurnoTresJugadores: (void) -> void //Determina el ganador del turno entre tres jugadores y acumula las cartas de los perdedores en el maso de cartas ganadas de jugador ganador. (Modificadora)
* evaluarTurnoCuatroJugadores: (void) -> void //Determina el ganador del turno entre cuatro jugadores y acumula las cartas de los perdedores en el maso de cartas ganadas de jugador ganador. (Modificadora)
* generarNumeroRandom: (void) -> int //Devuelve un numero cualquiera entre 1 y 52. (Analizadora)

**Manejo del error**

* Se controla que el número de cartas creadas no sea mayor a 52.
* Se maneja como máximo el ingreso de 4 jugadores y 2 como mínimo.

# TAD ListaCartas

**Representación abstracta**

<Carta 1, Carta 2, …, Carta n>, donde Carta != NULL.

**Definición de la invariante**

* Los elementos de la lista son del tipo Carta.
* Los elementos de las listas se ordenarán de acuerdo con el numero de la carta.

**Definición de operaciones**

* ListaCartas: (void) -> ListaCartas // Constructor por defecto, devuelve una lista vacía. (Constructora)
* insertarCarta: (Carta) -> void //Ingresa los objetos tipo Carta a la lista y los ordena de acuerdo con el numero de la carta. (Modificadora)
* mostrarListaCartas: (void) -> void //Muestra las cartas que fueron ingresadas en la lista. (Salida en pantalla)
* buscarCarta: (int) -> Carta //Busca una carta de acuerdo con el número de valor. (Analizadora)
* buscarCartaPorNum: (int) -> Carta //Busca una carta de acuerdo con su numero en el maso. (Analizadora)

**Manejo del error**

* Todas las operaciones de la lista deben mantener el invariante del TAD.
* Cuando ocurra una operación indebida en la lista o que genere un error se abortará su ejecución.

# TAD ListaJugador

**Representación abstracta**

<Jugador 1, Jugador 2, …, Jugador n>, donde Jugador != NULL.

**Definición de la invariante**

* Los elementos de la lista son del tipo Jugador.
* Los elementos de las listas se ordenarán de acuerdo con el número del jugador.

**Definición de operaciones**

* ListaJugador: (void) -> ListaJugador // Constructor por defecto, devuelve una lista vacía. (Constructora)
* annadirJugador: (Jugador) -> void //Ingresa los objetos tipo Jugador a la lista y los ordena de acuerdo con el número del jugador. (Modificadora)
* mostrarListaJugadores: (void) -> void //Muestra los jugadores que fueron ingresados en la lista. (Salida en pantalla)
* sacarJugador: (int) -> void // Elimina a un jugador de la lista. (Modificadora)
* buscarJugador: (int) -> Jugador //Busca un jugador de acuerdo con el número entero recibido como parámetro y retorna al jugador encontrado. (Analizadora)
* modificarJugador: (Jugador) -> void //Modifica los datos de un jugador en la lista. (Modificadora)

**Manejo del error**

* Todas las operaciones de la lista deben mantener el invariante del TAD.
* Cuando ocurra una operación indebida en la lista o que genere un error se abortará su ejecución.

# TAD ColaCartas

**Representación abstracta**

Carta1, Carta 2, …, Carta n-1, Carta n

**Definición de la invariante**

Cada carta que se obtenga será de acuerdo con el orden en la cola.

**Definición de operaciones**

* ColaCartas: (void) -> ColaCartas // Constructor por defecto, devuelve una cola vacía. (Constructora)
* esVacia: (void) -> bool // Retorna si la cola es vacía. (Analizadora)
* insertarCarta: (Carta) -> void // Se ingresan cartas a la cola. (Modificadora)
* tomarCarta: (void) -> Nodo\* // Devuelve una carta ubicado al inicio de la cola. (Modificadora)

**Manejo del error**

Se reporta al usuario cuando se trate de obtener una carta y la cola se encuentra vacía.

# TAD PilaCartas

**Representación abstracta**

Cartan, Cartan-1, …, Carta1 donde Carta != NULL.

**Definición de la invariante**

Todos los elementos de la pila son de tipo Carta

**Definición de operaciones**

* PilaCartas: (void) -> PilaCartas // Constructor por defecto, devuelve una pila vacía. (Constructora)
* esVaciaPila: (void) -> bool // Determina si la pila es vacía. (Analizadora)
* pushCarta: (Carta) -> void // Añade cartas a la pila. (Modificadora)
* popCarta: (void) -> (Carta) // Elimina la primera carta de la pila y la retorna. (Modificadora)
* buscarEnPila: (int) -> bool // Determina si el número de una carta se encuentra en la pila, devolviendo true si la encuentra. (Analizadora)

**Manejo del error**

Se reporta al usuario cuando la pila se encuentre vacía.

# Diagrama de Clases