



UNIVERSIDAD DEL VALLE

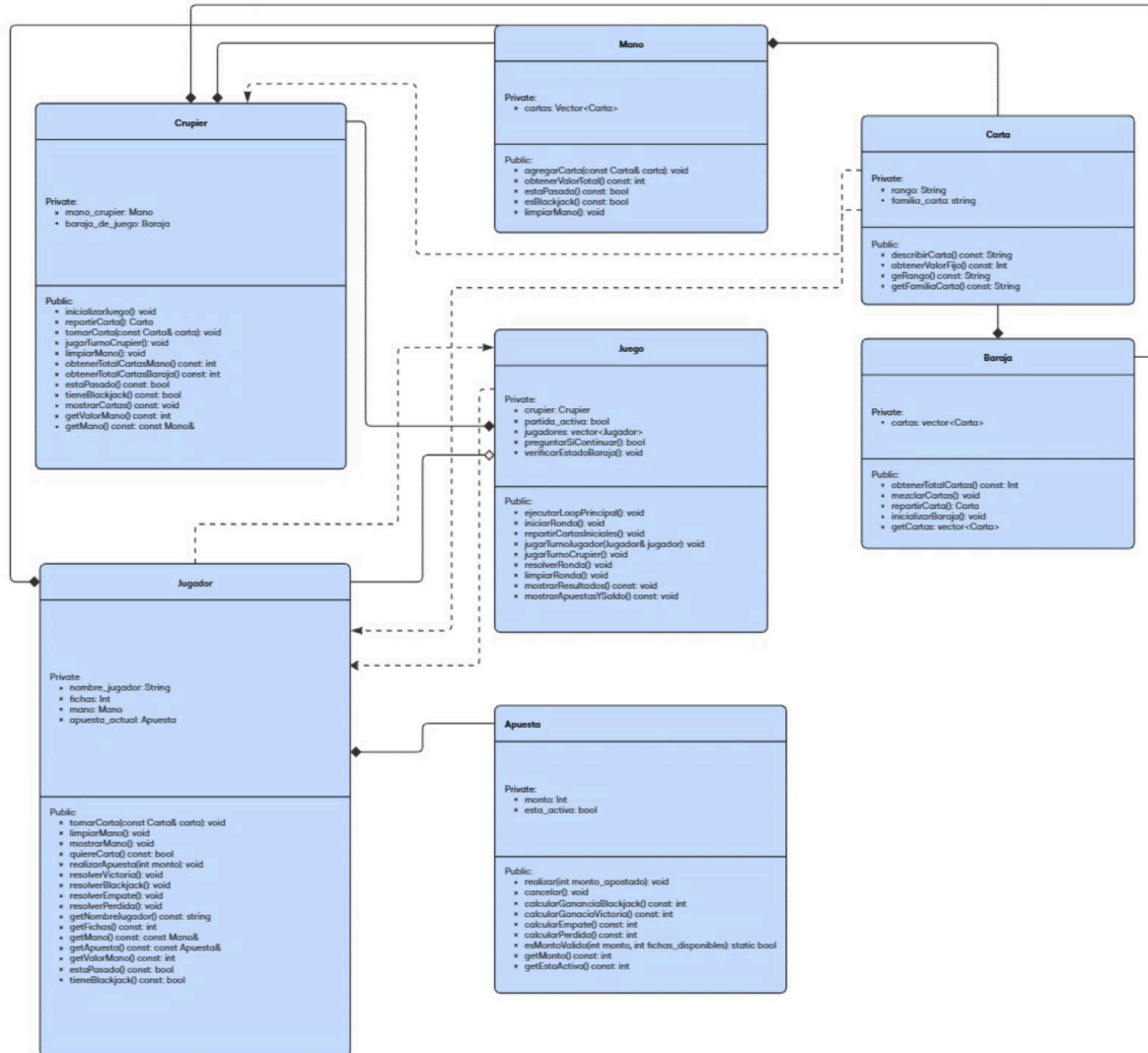
FACULTAD DE INGENIERÍA

PROYECTO BLACKJACK ENTREGA FINAL

Marilyn Grijalba - 2515702
Salomon Jimenez - 2517743

Identificación del requerimiento	Descripción	Entradas	Resultados (salidas)
R1	Una clase que determinar el flujo/control del juego	Número de jugadores	Mostrar los resultados al resolver la ronda.
R2	Crear una clase base para las cartas	String: 2 - 10 en las cartas, cartas especiales como As, J, Q, K.	Leer la descripción de la carta y mostrarla de manera entendible.
R3	Crear una clase que contenga las cartas para crear una baraja	La clase Cartas.	Valor total del jugador, método que restara la cartas repartidas, barajar las cartas, crear un mazo para cada participante.
R4	Crear una clase que tenga las cartas del jugador, a esto se le llama mano del jugador.	Agrega cartas a la Mano del jugador (grupo de cartas)	Lleva la puntuación total de la mano del jugador, agrega cartas al jugador.
R5	Crear una clase para el crupier.	Toma cartas de la baraja para repartir	Reparte las cartas, pone cartas en su mano y obtiene su valor total.
R6	Crear una clase para el jugador/jugadores.	Monto a apostar, recibe las cartas que le da el crupier.	Permitir tomar cartas, apostar, obtener el total de cartas, el nombre del jugador.
R7	Crear una clase para las apuestas.	Monto de apuesta y multiplicador de la apuesta	Monto, resultado de apuesta y cálculos

Diagrama de flujo, mejor calidad:





UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

Información CRC

Clase: Carta

Responsabilidades:

- Almacenar el rango (ej: As, 5, 7, J, K).
- Almacenar la familia/palo (ej: Diamantes).
- Devolver su valor numérico fijo (ej: 10 = J, Q, K).
- Descripción legible para el usuario.

Colaboradoras:

- No tiene colaborador, es una clase base, sólo almacenará datos.

```
C/C++
#ifndef CARTA_H
#define CARTA_H

#include <string>

class Carta {
    private:
        std::string rango; //string ya que necesitamos poner el "As", "2",
        "J"/"K"
        std::string familia_carta; //Las familias son: diamantes, corazones,
        picas y trovoles

    public:
        //Constructor
        Carta(const std::string& r, const std::string& f); //r: rango, f:
        familiaCarta

        //Getters
        std::string getRango() const { return rango; }
        std::string getFamiliaCarta() const { return familia_carta; }

        //Metodos
        std::string describirCarta() const; //"Traducir" el numero de las
        cartas y su familia a texto para mejor legibilidad
        int obtenerValorFijo() const; //Obtener el valor de las cartas como
        J, Q y K, el valor de esas cartas es 10

};

#endif //CARTA_H
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

```
C/C++
#include "Carta.h"
#include <string>
#include <stdexcept> //Libreria para manejar errores y excepciones

Carta::Carta(const std::string& r, const std::string& f)
    : rango(r), familia_carta(f)
{}

int Carta::obtenerValorFijo() const {
    //Cartas J, Q, K = 10
    if (rango == "J" || rango == "Q" || rango == "K") {
        return 10;
    }

    //Carta As (1 u 11)
    else if (rango == "As") {
        return 11;
    }

    //Cartas 2 al 10
    else {
        //try-catch para manejar el error si std::stoi falla
        try {
            //Convertir el string en entero usando std::stoi (string to integer)
            return std::stoi(rango);
        }
        catch (const std::exception& e){
            //Si se genera una falla, obtendremos un 0
            return 0;
        }
    }
}

std::string Carta::describirCarta() const {
    std::string simbolo;

    if (familia_carta == "Corazones") {
        simbolo = "♥";
    }
    else if (familia_carta == "Diamantes") {
        simbolo = "♦";
    }
    else if (familia_carta == "Picas") {
        simbolo = "♠";
    }
    else if (familia_carta == "Treboles") {
        simbolo = "♣";
    }
    else {
        //Fallback: si la familia no se reconoce, usar el texto
        return rango + " de " + familia_carta;
    }

    return rango + simbolo;
}
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

Información CRC

Clase: Baraja

Responsabilidades:

- Realizar un vector con todas las cartas (52 cartas en total).
- Almacenar la lista de cartas restantes.
- Barajar las cartas para tener un orden aleatorio.
- Repartir las cartas para el jugador y el crupier.
- Informar el número de cartas restantes.

Colaboradores:

- Carta.h

```
C/C++

#ifndef BARAJA_H
#define BARAJA_H

#include <vector>
#include <string>
#include "Carta.h"

//La clase Baraja es el contenedor de la la clase Carta.
class Baraja {
    private:
        std::vector<Carta> cartas; //Por este medio tendremos en un vector las 52
        cartas.

    public:
        //Constructor
        Baraja();

        //Getters
        const std::vector<Carta>& getCartas() const { return cartas; }

        //Metodos
        int obtenerTotalCartas() const; //Ira quitando las cartas de la baraja para
        que no se repitan.
        void mezclarCartas(); //Originalmente tenia un int pero se cambio a un void ya
        que solamente vamos a cambiar de orden los elementos.
        Carta repartirCarta(); //Llamamos a clase ya que de ahi se le daran los
        atributos a la carta
        void inicializarBaraja();
};

#endif //BARAJA_H
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

```
C/C++

#include "Carta.h"
#include <string>
#include <stdexcept> //Libreria para manejar errores y excepciones

Carta::Carta(const std::string& r, const std::string& f)
    : rango(r), familia_carta(f)
{}

int Carta::obtenerValorFijo() const {
    //Cartas J, Q, K = 10
    if (rango == "J" || rango == "Q" || rango == "K") {
        return 10;
    }

    //Carta As (1 u 11)
    else if (rango == "As") {
        return 11;
    }

    //Cartas 2 al 10
    else {
        //try-catch para manejar el error si std::stoi falla
        try {
            //Convertir el string en entero usando std::stoi (string to integer)
            return std::stoi(rango);
        }
        catch (const std::exception& e){
            //Si se genera una falla, obtendremos un 0
            return 0;
        }
    }
}

std::string Carta::describirCarta() const {
    std::string simbolo;

    if (familia_carta == "Corazones") {
        simbolo = "♥";
    }
    else if (familia_carta == "Diamantes") {
        simbolo = "♦";
    }
    else if (familia_carta == "Picas") {
        simbolo = "♠";
    }
    else if (familia_carta == "Treboles") {
        simbolo = "♣";
    }
    else {
        //Fallback: si la familia no se reconoce, usar el texto
        return rango + " de " + familia_carta;
    }

    return rango + simbolo;
}
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

Información CRC

Clase: Mano

Responsabilidades:

- Almacenar las cartas recibidas.
- Agregar cartas a la mano.
- Ajustar el valor total de la mano si As es 1 u 11.
- Determinar si el valor total de la mano se ha pasado de 21 (pierde)
- Determinar si es Black Jack, la mano debe contener las cartas As (11) y un 10 (J, Q o K) para tener un total de 21.
- Limpiar la mano, dejar sin cartas la mano cada nueva ronda.

Colaboradores:

- Carta.h

```
C/C++

#ifndef MANO_H
#define MANO_H

#include <vector>
#include "Carta.h"

class Mano{
    private:
        std::vector<Carta> cartas;

    public:
        //Constructor
        Mano();

        //Getters
        const std::vector<Carta>& getCartas() const { return cartas; }

        //Metodo
        void agregarCarta(const Carta& carta);
        int obtenerValorTotal() const; //Metodo para el valor total que
        tiene la mano/mazo del jugador.
        bool estaPasado() const; //Verificacion si el jugador se ha pasado
        de 21.
        bool esBlackjack() const; //Metodo para verificar si el jugador
        tiene un As y 10 o J,Q,K.
        void limpiarMano(); //Metodo para borrar las cartas e iniciar una
        ronda.
};

#endif //MANO_H
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

C/C++

```
#include "Mano.h"
```

```
Mano::Mano() {}
```

```
int Mano::obtenerValorTotal() const {
```

```
    int valor_total = 0;
```

```
    int contador_ases = 0;
```

```
    for (const Carta& carta: cartas) {
```

```
        int valor = carta.obtenerValorFijo();
```

```
        if (valor == 11) {
```

```
            contador_ases ++;
```

```
        }
```

```
        valor_total += valor;
```

```
    }
```

```
    while (valor_total > 21 && contador_ases > 0){
```

```
        valor_total -= 10;
```

```
        contador_ases --;
```

```
    }
```

```
    return valor_total;
```

```
}
```

```
bool Mano::estaPasado() const {
```

```
    return obtenerValorTotal() > 21;
```

```
}
```

```
bool Mano::esBlackjack() const {
```

```
    return (obtenerValorTotal() == 21) && (cartas.size() == 2);
```

```
}
```

```
void Mano::agregarCarta(const Carta& carta) {
```

```
    cartas.push_back(carta);
```

```
}
```

```
void Mano::limpiarMano() {
```

```
    cartas.clear();
```

```
}
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

Información CRC

Clase: Crupier

Responsabilidades:

- Pedir **Carta** a la **Baraja** y entregarla al **Jugador**
- Tomar cartas.
- Contar cartas, total de su mano.
- Mostrar sus cartas al jugador.

Colaboradores:

- Mano.h

```
C/C++

#ifndef CRUPIER_H
#define CRUPIER_H

#include "Mano.h"
#include "Baraja.h"

class Crupier{
private:
    Mano mano_crupier;
    Baraja baraja_de_juego;

public:
    //Constructor
    Crupier();

    //Metodos de gestion Baraja
    void inicializarJuego();
    Carta repartirCarta();

    //Metodos de juego
    void tomarCarta(const Carta& carta);
    void jugarTurnoCrupier();
    void limpiarMano();

    int obtenerTotalCartasMano() const;
    int obtenerTotalCartasBaraja() const;
    bool estaPasado() const;
    bool tieneBlackjack() const;

    //Metodos de Visualización
    void mostrarCartas() const;

    //Getters
    int getValorMano() const;
    const Mano& getMano() const { return mano_crupier; }
};

#endif //CRUPIER_H
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

C/C++

```
#include "Crupier.h"
#include <iostream>

Crupier::Crupier() {}

void Crupier::inicializarJuego() {
    baraja_de_juego.inicializarBaraja();
    baraja_de_juego.mezclarCartas();
}

Carta Crupier::repartirCarta() {
    return baraja_de_juego.repartirCarta();
}

void Crupier::tomarCarta(const Carta& carta) {
    mano_crupier.agregarCarta(carta);
}

int Crupier::getValorMano() const {
    return mano_crupier.obtenerValorTotal();
}

void Crupier::jugarTurnoCrupier() {
    while (getValorMano() < 17) {
        Carta nueva_carta = repartirCarta();
        tomarCarta(nueva_carta);
    }
}

int Crupier::obtenerTotalCartasMano() const {
    return mano_crupier.getCartas().size();
}

void Crupier::mostrarCartas() const {
    const std::vector<Carta>& mano = mano_crupier.getCartas();

    std::cout << "=== Cartas del Crupier (" << getValorMano() << ") ===" << std::endl;

    for (const Carta& carta : mano) {
        std::cout << " === " << carta.describirCarta() << std::endl;
    }

    std::cout << "===== " << std::endl;
}

void Crupier::limpiarMano() {
    mano_crupier.limpiarMano();
}

int Crupier::obtenerTotalCartasBaraja() const {
    return baraja_de_juego.obtenerTotalCartas();
}

bool Crupier::estaPasado() const {
    return mano_crupier.estaPasado();
}

bool Crupier::tieneBlackjack() const {
    return mano_crupier.esBlackjack();
}
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

Información CRC

Clase: Jugador

Responsabilidades:

- Representar la identidad y el estado de fichas (saldo)
- Gestionar la Mano de carta (tomar, limpiar, mostrar).
- Gestionar la Apuesta (realizarApuesta).
- Calcular el estado de su mano (valor total, si esta pasado o tiene blackjack).
- Determinar la acción del jugador (quiereCarta).
- Resolver el resultado de la ronda (resolverVictoria/Perdida/Empate/Blackjack).

Colaboradores:

- Mano.h
- Apuesta.h

C/C++

```
#ifndef JUGADOR_H
#define JUGADOR_H

#include <string>
#include <vector>
#include "Mano.h"
#include "Apuesta.h"

class Jugador {
private:
    std::string nombre_jugador;
    int fichas;
    Mano mano;
    Apuesta apuesta_actual;

public:
    //constructor
    Jugador(const std::string& nj, int saldo_inicial);

    //getters
    std::string getNombreJugador() const { return nombre_jugador; }
    int getFichas() const { return fichas; }
    const Apuesta& getApuesta() const { return apuesta_actual; }
    const Mano& getMano() const { return mano; }

    int getValorMano() const { return mano.obtenerValorTotal(); }
    bool estaPasado() const { return mano.estaPasado(); }
    bool tieneBlackjack() const { return mano.esBlackjack(); }

    //metodos
    void tomarCarta(const Carta& carta);
    void limpiarMano();
    void mostrarMano() const;
    bool quiereCarta() const;

    void realizarApuesta(int monto);
    void resolverVictoria();
    void resolverBlackjack();
    void resolverEmpate();
    void resolverPerdida();
};

#endif //JUGADOR_H
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

```
C/C++

#include "Jugador.h"
#include <iostream>
#include <limits>

Jugador::Jugador(const std::string& nj, int saldo_inicial)
    : nombre_jugador(nj), fichas(saldo_inicial) {}

void Jugador::realizarApuesta(int monto) {
    if (!Apuesta::esMontoValido(monto, fichas)) {
        throw std::runtime_error("Apuesta invalida. Fichas insuficientes o
monto invalido.");
    }

    fichas -= monto;
    apuesta_actual.realizar(monto);
}

void Jugador::resolverVictoria() {
    int ganancia = apuesta_actual.calcularGananciaVictoria();
    fichas += ganancia;
    apuesta_actual.cancelar();

    std::cout << nombre_jugador << " gana " << ganancia << " fichas!" <<
std::endl;
}

void Jugador::resolverBlackjack() {
    int ganancia = apuesta_actual.calcularGananciaBlackjack();
    fichas += ganancia;
    apuesta_actual.cancelar();

    std::cout << "BLACKJACK!!! " << nombre_jugador << " gana " << ganancia
<< " fichas!" << std::endl;
}

void Jugador::resolverEmpate() {
    int devolucion = apuesta_actual.calcularEmpate();
    fichas += devolucion;
    apuesta_actual.cancelar();

    std::cout << "Empate. " << nombre_jugador << " recupera " << devolucion
<< " fichas." << std::endl;
}
```



UNIVERSIDAD DEL VALLE FACULTAD DE INGENIERÍA

```
void Jugador::resolverPerdida() {
    apuesta_actual.cancelar();

    std::cout << nombre_jugador << " pierde la apuesta." << std::endl;
}

void Jugador::tomarCarta(const Carta& carta) {
    mano.agregarCarta(carta);
}

void Jugador::limpiarMano() {
    mano.limpiarMano();
}

void Jugador::mostrarMano() const {
    std::cout << "\n=== Mano de " << nombre_jugador << " ===" << std::endl;
    for (const Carta& carta : mano.getCartas()) {
        std::cout << " - " << carta.describirCarta() << std::endl;
    }
    std::cout << "Total: " << mano.obtenerValorTotal() << std::endl;
}

bool Jugador::quiereCarta() const {
    char opcion;
    std::cout << nombre_jugador << ", ¿quieres otra carta? (s/n): ";
    if (!(std::cin >> opcion)) {
        // Limpiar buffer de entrada en caso de entrada inválida
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        return false;
    }

    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    return (opcion == 's' || opcion == 'S');
}
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

Información CRC

Clase: Apuesta

Responsabilidades:

- Almacenar y gestionar el monto apostado (monto).
- Registrar si la apuesta esta activa o no (estado).
- Validar si un monto es elegible para apostar (esMontoValido).
- Permitir realizar y cancelar apuesta.
- Calcular los ajustes de fichas basados en el resultado de la apuesta (Ganancia, empate, perdida, blackjack).

Colaboradores:

- Jugador.h

```
C/C++

#ifndef APUESTA_H
#define APUESTA_H

#include <stdexcept>
class Apuesta{
    private:
        int monto;
        bool esta_activa;

    public:
        //Constructor
        Apuesta();

        void realizar(int monto_apostado);
        void cancelar();

        int calcularGananciaBlackjack() const;
        int calcularGananciaVictoria() const;
        int calcularEmpate() const;
        int calcularPerdida() const;

        static bool esMontoValido(int monto, int fichas_disponibles);

        //Getters
        int getMonto() const { return monto; }
        bool getEstaActiva() const { return esta_activa; }
};

#endif //APUESTA_H
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

```
C/C++
#include "Apuesta.h"

Apuesta::Apuesta()
    : monto(0), esta_activa(false) {}

void Apuesta::realizar(int monto_apostado) {
    if (monto_apostado <= 0) {
        throw std::invalid_argument("El monto para apostar debe ser mayor a 0.");
    }

    monto = monto_apostado;
    esta_activa = true;
}

void Apuesta::cancelar() {
    monto = 0;
    esta_activa = false;
}

int Apuesta::calcularGananciaVictoria() const {
    if (!esta_activa) {
        throw std::runtime_error("No hay apuesta activa.");
    }

    return monto * 2;
}

int Apuesta::calcularGananciaBlackjack() const {
    if (!esta_activa){
        throw std::runtime_error("No hay apuesta activa.");
    }

    return static_cast<int>(monto * 2.5f); //(Apuesta + 1.5x ganacia)
}

int Apuesta::calcularEmpate() const {
    if (!esta_activa) {
        throw std::runtime_error("No hay apuesta activa.");
    }
    return monto;
}

int Apuesta::calcularPerdida() const {
    return 0;
}

bool Apuesta::esMontoValido(int monto, int fichas_disponibles) {
    return (monto > 0) && (monto <= fichas_disponibles);
}
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

Información CRC

Clase: Juego

Responsabilidades:

- Coordinar el flujo principal del juego (ejecutarLoopPrincipal).
- Mantener la lista de Jugador es activo.
- Gestionar el Crupier que maneja las cartas.
- Inicializar y controlar la secuencia de la Ronda (iniciarRonda, repartirCartasIniciales).
- Controlar los turnos de juego para el Jugador y el Crupier.
- Resolver la ronda: determinar resultados, pagar apuestas y limpiar el estado.
- Monitorear el estado de la bajara (verificarEstadoBaraja).
- Controlar la continuación del juego (preguntarSiContinuar).

Colaboradores:

- Jugador
- Crupier

C/C++

```
#ifndef JUEGO_H
#define JUEGO_H

#include <vector>
#include <string>
#include "Jugador.h"
#include "Crupier.h"

class Juego {
private:
    Crupier crupier;
    bool partida_activa;
    std::vector<Jugador> jugadores;
    bool preguntarSiContinuar();
    void verificarEstadoBaraja(); //Para mezclar o inicializar si hay pocas cartas.

public:
    Juego(const std::string& nombre_jugador);

    void ejecutarLoopPrincipal(); //Loop contiene rondas

    void iniciarRonda(); //Crea los jugadores, mezclar y repartir
    void repartirCartasIniciales();
    void jugarTurnoJugador(Jugador& jugador);
    void jugarTurnoCrupier();
    void resolverRonda(); //Determinar el ganador, pagar y cobrar a los jugadores
    void limpiarRonda();

    void mostrarResultados() const;
    void mostrarApuestasYSaldo() const;

};

#endif //JUEGO_H
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

```
C/C++

#include "Juego.h"
#include <iostream>
#include <string>
#include <limits>

using namespace std;

Juego::Juego(const std::string& nombre_jugador)
    : partida_activa(false) {
    jugadores.emplace_back(nombre_jugador, 1000);
    crupier.inicializarJuego();

    std::cout << "=== Bienvenido al Blackjack, " << nombre_jugador << " ===" <<
std::endl;
    std::cout << "Tienes 1000 fichas para empezar." << std::endl;
}

//Verificar la baraja, si quedan menos de 10 cartas, reinicia el mazo
void Juego::verificarEstadoBaraja() {
    if (crupier.obtenerTotalCartasBaraja() < 10) {
        cout << "\n[MANTENIMIENTO] Quedan pocas cartas. Reiniciando y mezclando la
baraja..." << endl;
        crupier.inicializarJuego();
    }
}

void Juego::iniciarRonda() {
    if (jugadores.empty()) return;

    Jugador& jugador = jugadores[0];

    //Limpiar mesa
    crupier.limpiarMano();
    jugador.limpiarMano();

    //Pedir apuesta
    int apuesta_monto = 0;
    cout << "\n=== NUEVA RONDA ===" << endl;
    cout << jugador.getNombreJugador() << ", tienes " << jugador.getFichas() << "
fichas. ¿Cuanto deseas apostar? ";

    while(!(cin >> apuesta_monto) || apuesta_monto <= 0 || apuesta_monto >
jugador.getFichas()) {
        cout << "Apuesta invalida. Introduce un monto valido (max " <<
jugador.getFichas() << "): ";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
}
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

```
try {
    jugador.realizarApuesta(apuesta_monto);
} catch (const runtime_error& e) {
    cerr << "Error: " << e.what() << endl;
    return;
}

repartirCartasIniciales();
}

void Juego::repartirCartasIniciales() {
    Jugador& jugador = jugadores[0];

    //Repatir 2 cartas a cada uno (Jugador -> Crupier -> Jugador -> Crupier -> ...)

    //La primera carta la recibe el jugador
    jugador.tomarCarta(crupier.repartirCarta());

    //La primera carta la recibe el crupier y la mostrara.
    crupier.tomarCarta(crupier.repartirCarta());

    //Segunda carta que recibe el jugador
    jugador.tomarCarta(crupier.repartirCarta());

    //Segunda carta que recibe el crupier pero esta no se mostrara
    //Se mostrara con el metodo mostrarMesa()
    crupier.tomarCarta(crupier.repartirCarta());

    cout << "\n=== CARTAS REPARTIDA ===" << endl;
    jugador.mostrarMano();

    cout << "Mano del crupier: " << crupier.getMano().getCartas()[0].describirCarta()
    << " y una carta OCULTA." << endl;
}

void Juego::jugarTurnoJugador(Jugador& jugador) {
    cout << "\n=== TURNO DE " << jugador.getNombreJugador() << " ===" << endl;

    while (jugador.quiereCarta() && !jugador.estaPasado()) {
        Carta nueva_carta = crupier.repartirCarta();
        jugador.tomarCarta(nueva_carta);

        cout << jugador.getNombreJugador() << " pide carta: " <<
        nueva_carta.describirCarta() << endl;

        jugador.mostrarMano();

        if (jugador.estaPasado()) {
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

```
        cout << jugador.getNombreJugador() << " Se ha pasado de 21 (" <<
jugador.getValorMano() << ") PIERDE la ronda." << endl;
        break;
    }
}

void Juego::ejecutarLoopPrincipal() {
    partida_activa = true;

    while (partida_activa && !jugadores.empty()) {
        Jugador& jugador = jugadores[0];

        if (jugador.getFichas() <= 0) {
            cout << "\n Te quedaste sin fichas. Fin del juego." << endl;
            partida_activa = false;
            break;
        }

        verificarEstadoBaraja();

        iniciarRonda();

        if (jugador.getApuesta().getMonto() == 0) {
            continue;
        }

        if (jugador.tieneBlackjack()) {
            cout << "\nBLACKJACK INSTANTANEO" << endl;
            if (crupier.tieneBlackjack()) {
                cout << "El crupier tambien tiene blackjack. EMPATE" << endl;
                crupier.mostrarCartas();
                jugador.resolverEmpate();
            } else {
                cout << "El crupier NO tiene blackjack !GANASTE con blackjack" <<
endl;

                crupier.mostrarCartas();
                jugador.resolverBlackjack();
            }
        }

        mostrarApuestasYSaldo();

        if (!preguntarSiContinuar()) {
            partida_activa = false;
            break;
        }

        continue;
    }
}
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

```
    jugarTurnoJugador(jugador);

    if (jugador.estaPasado()) {
        mostrarResultados();
        jugador.resolverPerdida();
        mostrarApuestasYSaldo();

        if (!preguntarSiContinuar()) {
            partida_activa = false;
            break;
        }

        continue;
    }

    jugarTurnoCrupier();

    mostrarResultados();

    resolverRonda();

    mostrarApuestasYSaldo();

    if (!preguntarSiContinuar()) {
        partida_activa = false;
        break;
    }
}

//Mensaje de despedida.
cout << "\n=== FIN DEL JUEGO ===" << endl;
if (!jugadores.empty()) {
    Jugador& jugador = jugadores [0];
    cout << jugador.getNombreJugador() << ", terminaste con " <<
jugador.getFichas() << " fichas." << endl;

    if (jugador.getFichas() > 1000) {
        cout << "FELICIDADES GANASTE " << (jugador.getFichas() - 1000) << "
fichas." << endl;
    } else if (jugador.getFichas() < 1000) {
        cout << "Perdiste " << (1000 - jugador.getFichas()) << " fichas." << endl;
    } else {
        cout << "Terminaste con el mismo saldo inicial." << endl;
    }
}

cout << "Gracias por jugar." << endl;
}

bool Juego::preguntarSiContinuar() {
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

```
char respuesta;
cout << "\nQuieres jugar otra ronda? (s/n): ";

while (true) {
    if (cin >> respuesta) {
        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        if (respuesta == 's' || respuesta == 'S') {
            return true;
        } else if (respuesta == 'n' || respuesta == 'N') {
            return false;
        }
    } else {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }

    cout << "Respuesta invalida. Ingrese 's' para continuar o 'n' para salir: ";
}

}

void Juego::jugarTurnoCrupier() {
    cout << "\n=== TURNO DEL CRUPIER ===" << endl;
    crupier.mostrarCartas();
    crupier.jugarTurnoCrupier();
    crupier.mostrarCartas();
}

void Juego::resolverRonda() {
    Jugador& jugador = jugadores[0];

    //Si el jugador se paso, pierde
    if (jugador.estaPasado()) {
        jugador.resolverPerdida();
        return;
    }

    //Si el crupier pierde, el jugador gana
    if (crupier.estaPasado()) {
        if (jugador.tieneBlackjack()) {
            jugador.resolverBlackjack();
        } else {
            jugador.resolverVictoria();
        }
        return;
    }

    int valor_jugador = jugador.getValorMano();
    int valor_crupier = crupier.getValorMano();
```



UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

```
if (valor_jugador > valor_crupier) {
    if (jugador.tieneBlackjack() && !crupier.tieneBlackjack()) {
        jugador.resolverBlackjack();
    } else {
        jugador.resolverVictoria();
    }
} else if (valor_jugador < valor_crupier) {
    jugador.resolverPerdida();
} else {
    jugador.resolverEmpate();
}
}

void Juego::limpiarRonda() {
    crupier.limpiarMano();
    for (Jugador& jugador : jugadores) {
        jugador.limpiarMano();
    }
}

void Juego::mostrarResultados() const {
    cout << "\n=== RESULTADO FINAL ===" << endl;
    const Jugador& jugador = jugadores[0];
    jugador.mostrarMano();
    crupier.mostrarCartas();
}

void Juego::mostrarApuestasYSaldo() const {
    const Jugador& jugador = jugadores[0];
    cout << "\nSaldo actual: " << jugador.getFichas() << " fichas" << endl;
}
```