ЛАБОРАТОРНАЯ РАБОТА №4

Курс «Объектно-ориентированное программирование»

Тема: Агрегация и композиция. Дружественные функции и классы. Исключения.

Цель: Научиться реализовывать на C++ межклассовое отношение агрегации / композиции, писать код генерации и обработки исключений, перегружать операторы с помощью дружественных функций.

Ход работы

Вариант 5

Задание 1.

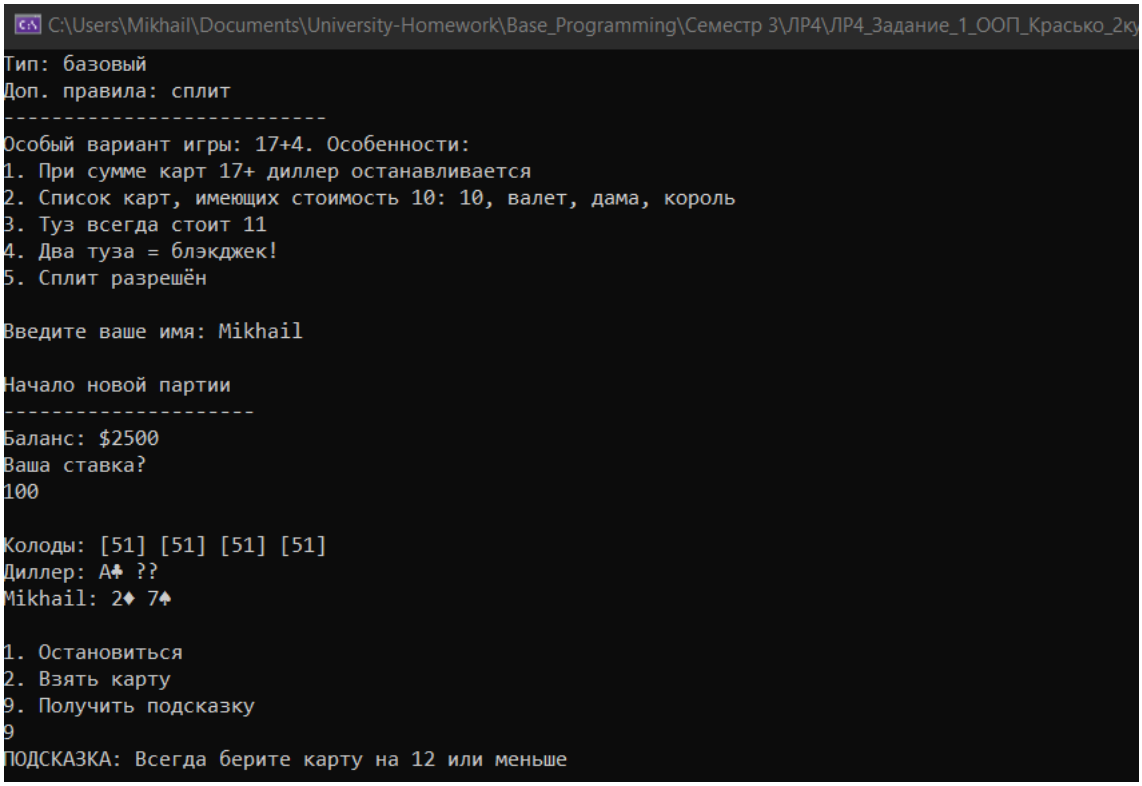
1. Написать простой консольный вариант карточной игры BlackJack для игры один-на-один с дилером, в соответствии с вариантом. В приложении А приведены вариации и особенности правил игры для каждого варианта. В ходе работы необходимо сделать как минимум следующее:
   - создать и связать отношением агрегации/композиции и/или наследования классы КАРТА, КОЛОДА, ДИЛЕР, ИГРОК, ИГРА. В целом, Вы можете предлагать здесь свои варианты объектно-ориентированного проектирования;
   - в начале игры генерировать случайным образом 4 колоды с 36 или 52 картами, в зависимости от варианта;
   - имитировать действия дилера, в соответствии с вариантом игры;
   - запрограммировать обработку всех потенциально возможных вариантов исхода: блек-джек, перебор, ровно, выигрыш по очкам, проигрыш по очкам;
   - бросать и отлавливать исключение при «переборе» («перебор» рассматривать как исключительную ситуацию);
   - перегрузить операцию потокового вывода объекта класса карты на экран с помощью дружественной функции. Выводить карту в виде 2♠, Q♦ и т.д. (символы карточных мастей имеют ASCII-коды 3, 4, 5, 6 и UNICODE-коды "\u2665", "\u2666", "\u2663", "\u2660", соответственно).

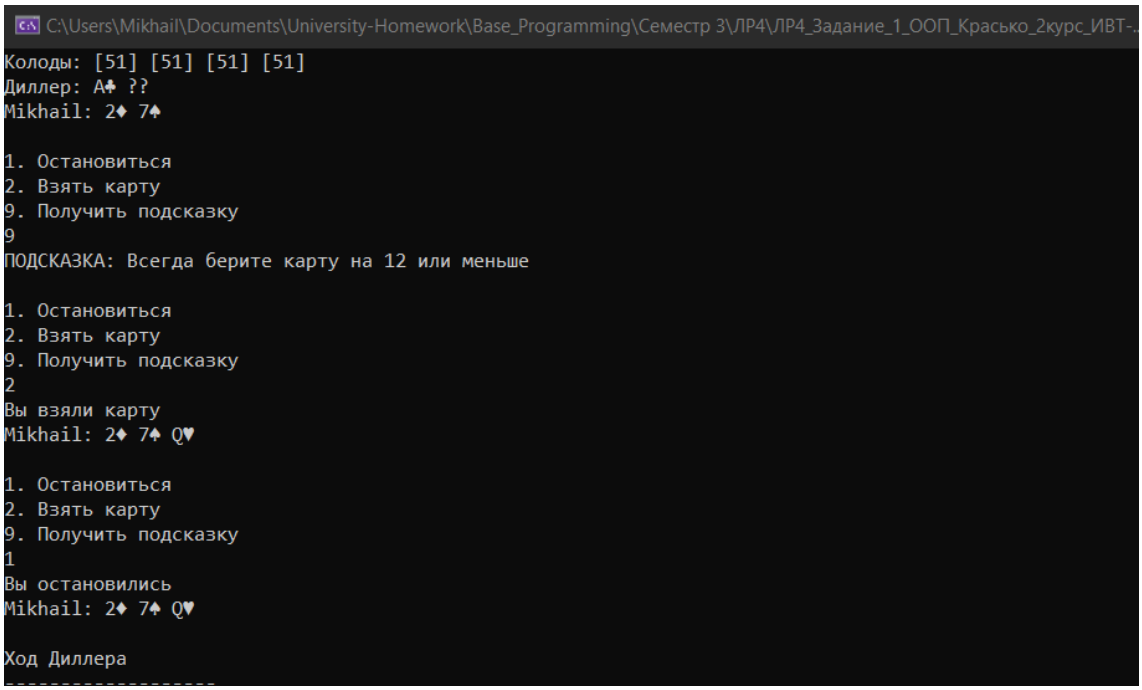
Приложение А. Варианты индивидуальных заданий.

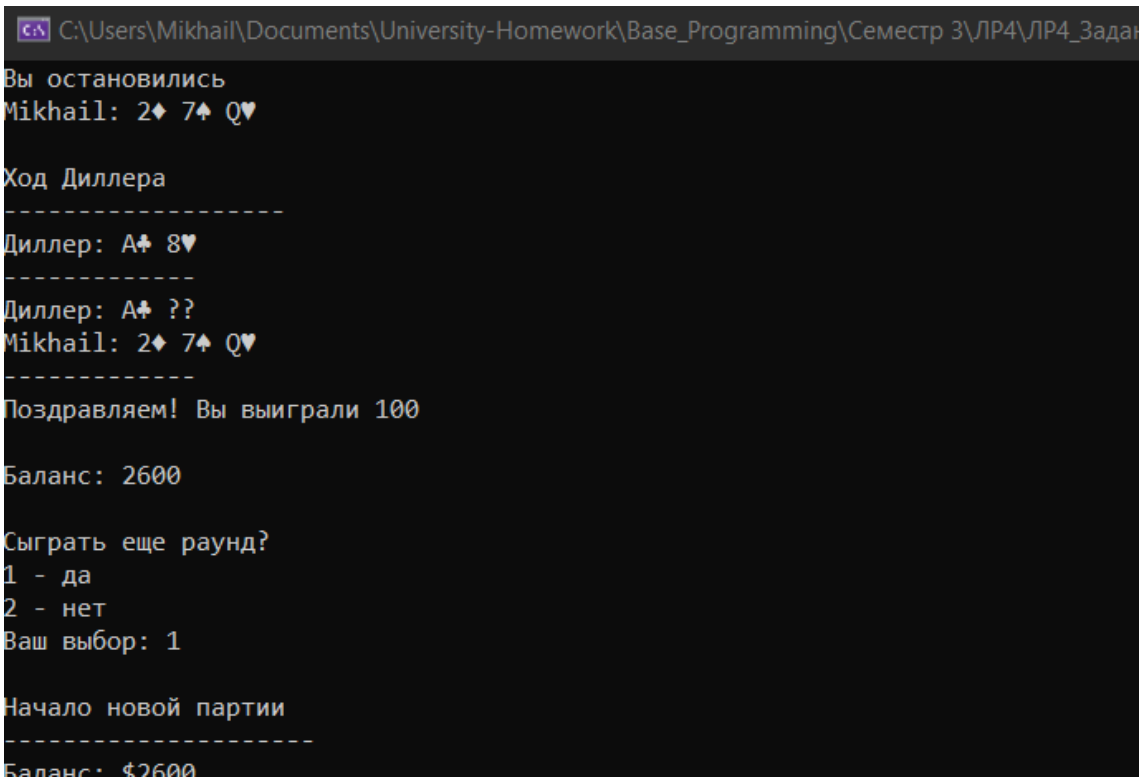
Задание 1.

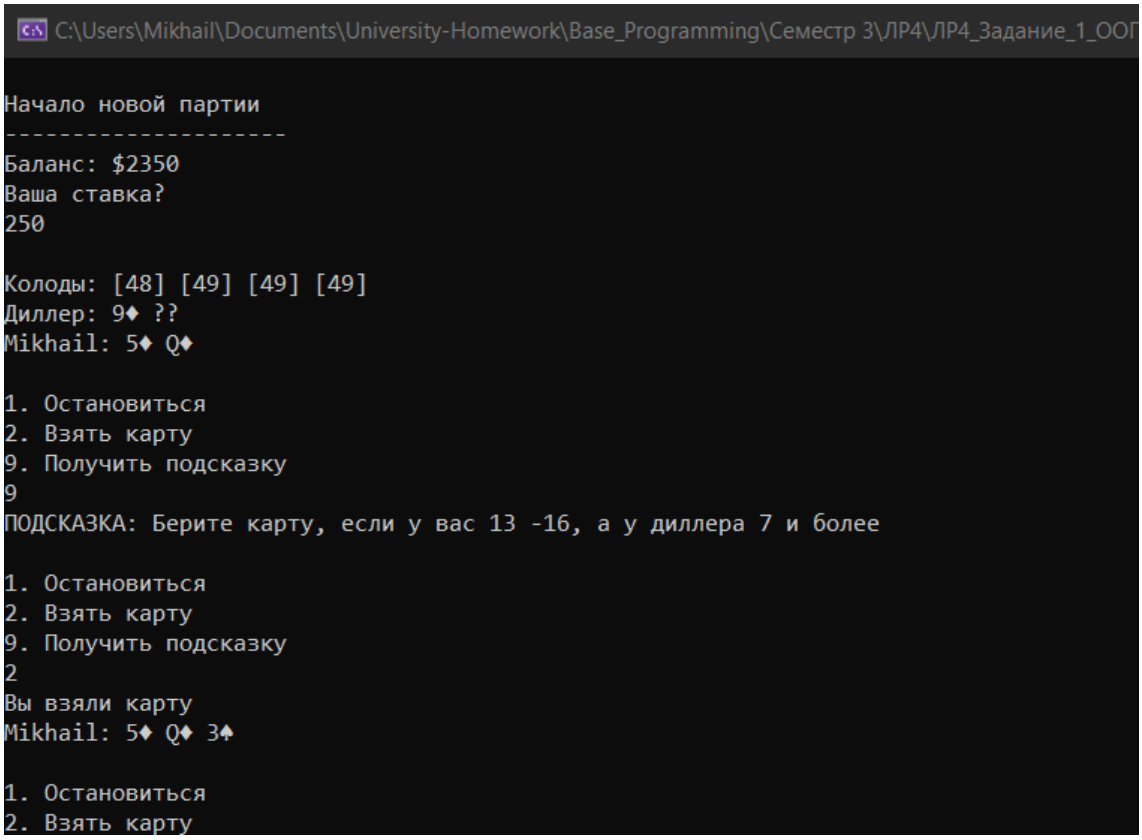
№ вар.	*Особый вариант игры*	*Тип игры*	*Доп. правила*
1	ПАРА ТУЗОВ	базовый	Сплит
2	777	европейский	Дабл
3	ОДНОМАСТНЫЙ БЛЕК-ДЖЕК	базовый	Трипл
4	МАКСИМУМ КАРТ	европейский	Саррендер
5	17 + 4	базовый	Сплит
6	ПАРА ТУЗОВ	европейский	Дабл
7	777	базовый	Трипл
8	ОДНОМАСТНЫЙ БЛЕК-ДЖЕК	европейский	Саррендер

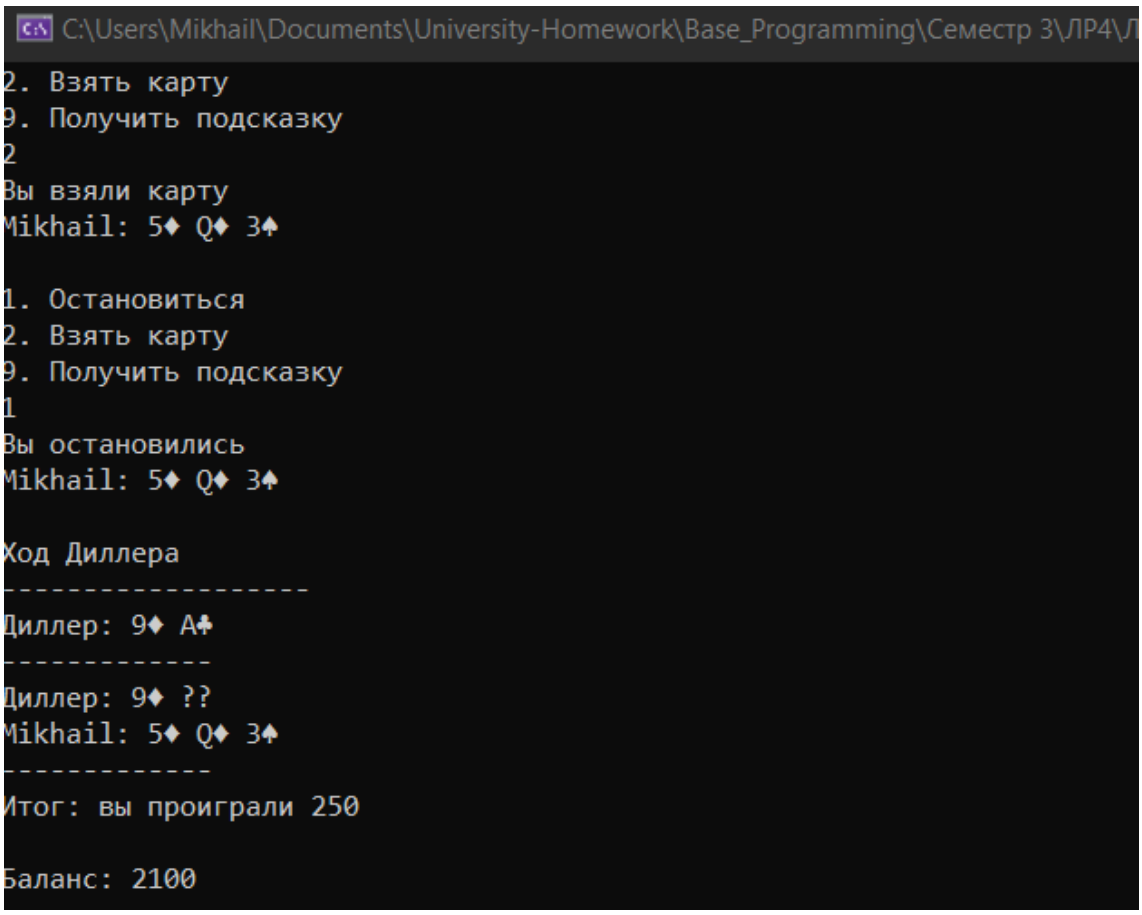
Результат работы программы:

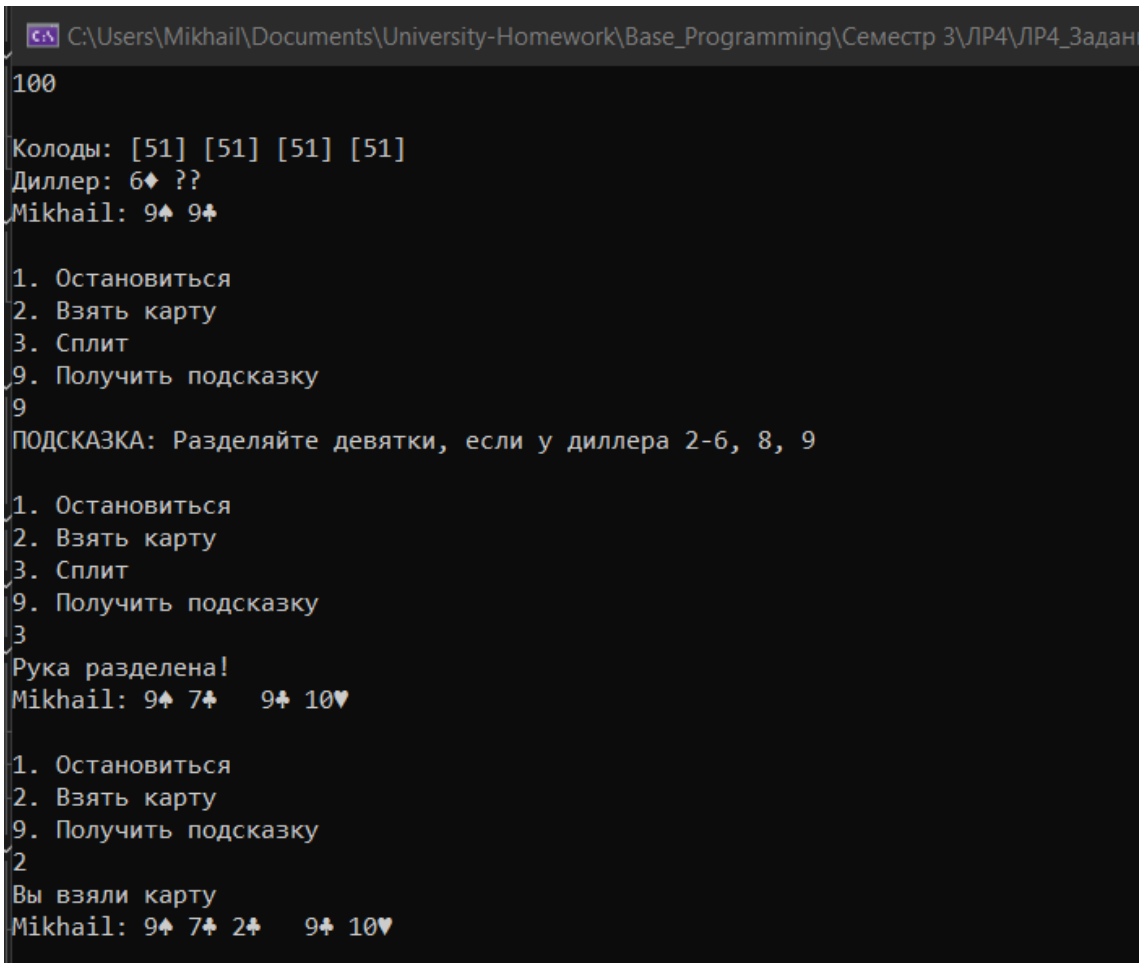


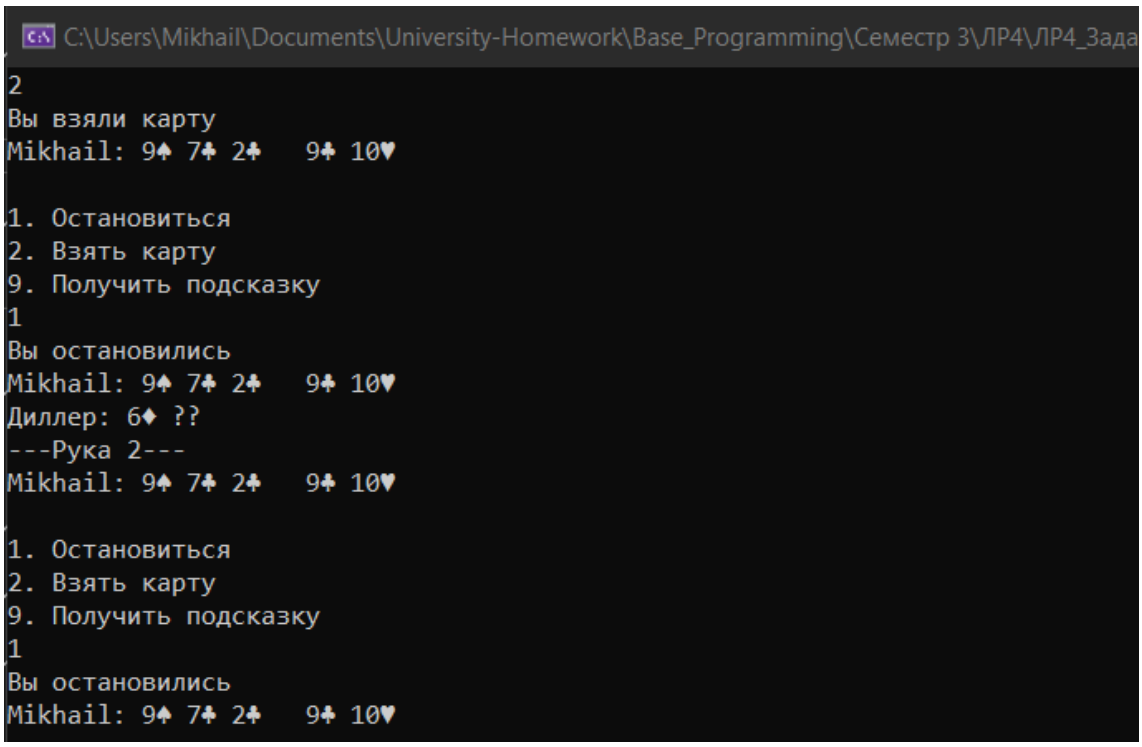


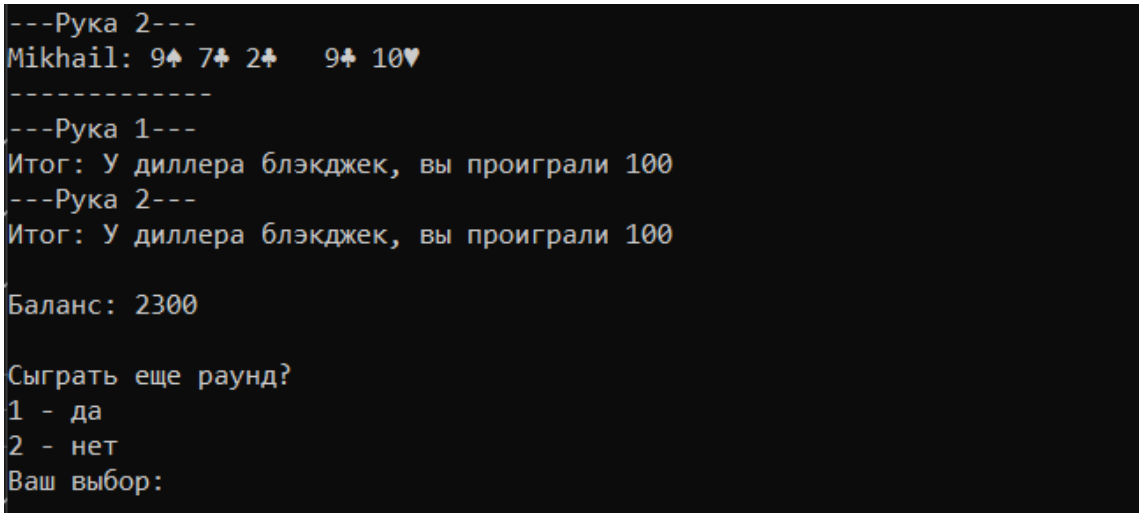

C:\Users\Mikhail\Documents\University-Homework\Base_Programming\Семестр 3\ЛР4\ЛР4_Зада
Вы остановились
Mikhail: 2♦ 7♠ Q♥
Ход Диллера
Диллер: A♣ 8♥
Диллер: A♣ ??
Mikhail: 2♦ 7♠ Q♥
Поздравляем! Вы выиграли 100
Баланс: 2600
Сыграть еще раунд?
1 - да
2 - нет
Ваш выбор: 1
Начало новой партии
Баланс: $2600


C:\Users\Mikhail\Documents\University-Homework\Base_Programming\Семестр 3\ЛР4\ЛР4_Задание_1_ООП
Начало новой партии
Баланс: $2350
Ваша ставка?
250
Колоды: [48] [49] [49] [49]
Диллер: 9♦ ??
Mikhail: 5♦ Q♦
1. Остановиться
2. Взять карту
9. Получить подсказку
9
ПОДСКАЗКА: Берите карту, если у вас 13 -16, а у диллера 7 и более
1. Остановиться
2. Взять карту
9. Получить подсказку
2
Вы взяли карту
Mikhail: 5♦ Q♦ 3♠
1. Остановиться
2. Взять карту


C:\Users\Mikhail\Documents\University-Homework\Base_Programming\Семестр 3\ЛР4\Л
2. Взять карту
9. Получить подсказку
2
Вы взяли карту
Mikhail: 5♦ Q♦ 3♠
1. Остановиться
2. Взять карту
9. Получить подсказку
1
Вы остановились
Mikhail: 5♦ Q♦ 3♠
Ход Диллера
Диллер: 9♦ A♣
Диллер: 9♦ ??
Mikhail: 5♦ Q♦ 3♠
Итог: вы проиграли 250
Баланс: 2100


C:\Users\Mikhail\Documents\University-Homework\Base_Programming\Семестр 3\ЛР4\ЛР4_Задан
100
Колоды: [51] [51] [51] [51]
Диллер: 6♦ ??
Mikhail: 9♠ 9♣
1. Остановиться
2. Взять карту
3. Сплит
9. Получить подсказку
9
ПОДСКАЗКА: Разделяйте девятки, если у диллера 2-6, 8, 9
1. Остановиться
2. Взять карту
3. Сплит
9. Получить подсказку
3
Рука разделена!
Mikhail: 9♠ 7♣ 9♣ 10♥
1. Остановиться
2. Взять карту
9. Получить подсказку
2
Вы взяли карту
Mikhail: 9♠ 7♣ 2♣ 9♣ 10♥





Код:

card.h

```
#ifndef CARD_H
#define CARD_H

#include <string>
#include <iostream>

// Suit - масть
enum class Suit { HEARTS, DIAMONDS, CLUBS, SPADES };

// Ace - туз
enum class Rank {
    ACE = 11, TWO = 2, THREE = 3, FOUR = 4, FIVE = 5, SIX =
6, SEVEN = 7,
```

```
        EIGHT = 8, NINE = 9, TEN = 10, JACK = 12, QUEEN = 13,
KING = 14
    };

    class Card {
    private:
        Suit suit;
        Rank rank;
        bool faceUp; // Нужно для определения, какие карты
диллера показывать, а какие нет

    public:
        Card(Suit s, Rank r);
        int getValue() const;
        Rank getRank() const;
        bool isAce() const;
        bool isTenValue() const;
        void flip(); // Показать карту диллера
        bool isFaceUp() const; // Показано ли содержание карты.
Нужно для подсказок т.к. они делаются только на основе той
информации,
        // которая есть у пользователя
        std::string suitToString() const;
        std::string rankToString() const;
        void display() const;
    };

    #endif
```

card.cpp

```
    #include "card.h"

    Card::Card(Suit s, Rank r): suit(s), rank(r), faceUp(true) {
    }

    int Card::getValue() const {

        if (isTenValue())
            return 10;
        return static_cast<int>(rank);
    }

    Rank Card::getRank() const {
        return rank;
    }

    bool Card::isAce() const {
        return rank == Rank::ACE;
    }
```

```
bool Card::isTenValue() const {
    return rank == Rank::TEN || rank == Rank::JACK || rank ==
Rank::QUEEN || rank == Rank::KING;
}

void Card::flip() {
    faceUp = !faceUp;
}

bool Card::isFaceUp() const {
    return faceUp;
}

std::string Card::suitToString() const {

    switch (suit) {
    case Suit::HEARTS: return std::string(1, char(3));    //
Сердце
    case Suit::DIAMONDS: return std::string(1, char(4));  //
Ромб
    case Suit::CLUBS: return std::string(1, char(5));     //
Клевер
    case Suit::SPADES: return std::string(1, char(6));    //
пика
    default: return "?";
    }
}

std::string Card::rankToString() const {
    switch (rank) {
        case Rank::ACE: return "A";
        case Rank::JACK: return "J";
        case Rank::QUEEN: return "Q";
        case Rank::KING: return "K";
    default: return std::to_string(getValue());
    }
}
void Card::display() const {

    // Встроенная функция показа только тех карт, которые
можно видеть игроку
    if (faceUp) {
        std::cout << rankToString() << suitToString();
    }
    else {
        std::cout << "??";
    }
}
```

dealer.h

```
#ifndef DEALER_H
#define DEALER_H

#include "participant.h"


// Player и Dealer наследуют у Participante
class Dealer : public Participant {
public:
    Dealer();
    void displayHand(bool showAll = false) const override;
    bool shouldHit() const; // Нужно для подсказок
    int getVisibleCardValue() const; // Тоже нужно для
подсказок, так как они будут на основе только видимых карт диллера
};

#endif
```

dealer.cpp

```
#include "dealer.h"

Dealer::Dealer() : Participant() {}

void Dealer::displayHand(bool showAll) const {

    if (!hands.empty()) {
        if (showAll) {
            hands[0].display();
        }
        else {
            if (hands[0].getCardAmount() > 0) {
                hands[0].getCard(0).display();
                std::cout << " ??";
            }
        }
    }
}

bool Dealer::shouldHit() const {
    int total = hands[0].getTotal();
    bool result = total < 17;
    return result;
}

int Dealer::getVisibleCardValue() const {
    if (hands.empty()) {
        return 0;
    }
```

```
        // Проверяем, есть ли карты в первой руке
        if (hands[0].getCardAmount() == 0) {
            return 0;
        }

        // Получаем видимую карту диллера
        Card firstCard = hands[0].getCard(0);

        return firstCard.getValue();
    }
```

deck.h

```
    #ifndef DECK_H
    #define DECK_H

    #include "card.h"
    #include <vector>
    #include <random>
    #include <ctime>

    class Deck {
    private:
        std::vector<std::vector<Card>> decks;  // для отдельных
колод
        std::vector<int> currentIndices;       // Текущая позиция
в каждой колоде
        int nextDeckIndex;                     // Следующая карта
для каждой колоды

    public:
        Deck();
        void initialize();
        Card giveCard();
        void shuffleDeck(int deckIndex); // shuffle - перемешать
        void displayDecksRemainingCards() const; // отобразить
оставшиеся карты в каждой колоде
        int getRemainingAmountInDeck(int deckIndex) const; //
Оставшееся колво карт в конкретной колоде
    };

    #endif
```

deck.cpp

```
    #include "deck.h"

    Deck::Deck() {
        initialize();
```

```
    }

    void Deck::initialize() {
        decks.clear();
        currentIndices.clear();

        // Создаем 4 отдельные колоды
        for (int deckNum = 0; deckNum < 4; deckNum++) {
            std::vector<Card> deck;

            for (int suitIndex = 0; suitIndex < 4; suitIndex++) {
                Suit suit = static_cast<Suit>(suitIndex);

                for (int rankValue = 1; rankValue <= 13;
rankValue++) {
                    Rank rank;
                    //if ((rankValue == 11) || (rankValue == 12)
|| (rankValue == 13)) {
                    //    rank = static_cast<Rank>(10);
                    //}
                    //else {
                        rank = static_cast<Rank>(rankValue);
                    //}

                    deck.emplace_back(suit, rank);
                }
            }

            // Сохраняем созданную колоду
            decks.push_back(deck);
            currentIndices.push_back(0);
        }

        for (int i = 0; i < 4; i++) {
            shuffleDeck(i);
        }

        nextDeckIndex = 0;
    }


    void Deck::shuffleDeck(int deckIndex) {
        // Создаем генератор случайных чисел
        std::random_device rd;
        std::mt19937 g(rd());

        // Перемешиваем карты в указанной колоде
        std::shuffle(decks[deckIndex].begin(),
decks[deckIndex].end(), g);

        currentIndices[deckIndex] = 0;
```

```
}

// Дать одну карту и перейти к следующей колоде
Card Deck::giveCard() {
    // Если текущая колода закончилась, перетасовать ее
    if (currentIndices[nextDeckIndex] >= 52) {
        shuffleDeck(nextDeckIndex);
    }

    Card card =
decks[nextDeckIndex][currentIndices[nextDeckIndex]];
    currentIndices[nextDeckIndex]++;

    nextDeckIndex = (nextDeckIndex + 1) % 4;

    return card;
}

// Показать оставшиеся карты в каждой колоде
void Deck::displayDecksRemainingCards() const {
    std::cout << "Колоды: ";
    for (int i = 0; i < 4; i++) {
        int remaining = 52 - currentIndices[i];
        std::cout << "[" << remaining << "] ";
    }

    std::cout << std::endl;
}


// Получить оставшиеся карты в конкретной колоде
int Deck::getRemainingAmountInDeck(int deckIndex) const {
    if (deckIndex >= 0 && deckIndex < 4) {
        return 52 - currentIndices[deckIndex];
    }
    return 0;
}
```

game.h

```
#ifndef GAME_H
#define GAME_H

#include "deck.h"
#include "player.h"
#include "dealer.h"
#include <memory>

class Game {
private:
    Deck deck;
```

```
        std::unique_ptr<Player> player; // unique_ptr для
автоматического освобождения памяти
        std::unique_ptr<Dealer> dealer;
        bool gameOver;

        void betResults();
        std::string getHint();
        void playPlayerHand();
        void playDealerHand();

    public:
        Game();
        void initialize();
        void playRound();
        void run(); // Нужно, чтобы все этапы игры были в классе,
а не часть в main
        void printCards();
    };

    #endif
```

game.cpp

```
    #include "game.h"
    #include <iostream>
    #include <limits>

    Game::Game() : gameOver(false) {
        dealer = std::make_unique<Dealer>();
    }

    // initialize отдельно от конструктора, чтобы не было ввода
данных в конструкторе
    void Game::initialize() {
        std::string playerName;

        std::cout << "Тип: базовый\nДоп. правила: сплит\n--------
------------------\nОсобый вариант игры: 17+4. Особенности:\n";
        std::cout << "1. При сумме карт 17+ диллер
останавливается\n";
        std::cout << "2. Список карт, имеющих стоимость 10: 10,
валет, дама, король\n";
        std::cout << "3. Туз всегда стоит 11\n";
        std::cout << "4. Два туза = блэкджек!\n";
        std::cout << "5. Сплит разрешён\n";

        std::cout << "\nВведите ваше имя: ";
        std::getline(std::cin, playerName);

        player = std::make_unique<Player>(playerName);
```

```
    }

    std::string Game::getHint() {

        int playerTotal = player->getCurrentHand().getTotal();
        int dealerCardValue = dealer->getVisibleCardValue();
        bool canSplitHand = player->canSplit();

        // Если есть возможность сплита
        if (canSplitHand) {

            int cardValue = player-
>getCurrentHand().getCard(0).getValue();

            switch (cardValue) {
            case 10: return "Никогда не разделяйте карты
стоимостью 10. Остановитесь";
            case 9:
                if (dealerCardValue == 7 || dealerCardValue == 10
|| dealerCardValue == 11) {
                    return "Не разделяйте девятки, если у дилера
7, 10 или туз. Остановитесь";
                }
                else {
                    return "Разделяйте девятки, если у диллера 2-
6, 8, 9";
                }
            case 8: return "Всегда разделяйте восьмерки";
            case 7:
                if (dealerCardValue >= 2 && dealerCardValue <= 7)
{
                    return "Разделяйте семерки, если у дилера 2-
7";
                }
                else {

                    return "Не разделяйте семерки, если у диллера
8 и больше в сумме. Берите карту";
                }
            case 6:
                if (dealerCardValue >= 2 && dealerCardValue <= 6)
{
                    return "Разделяйте шестерки, если у диллера
2-6";
                }
                else {
                    return "Не разделяйте шестерки, если у
диллера 7 и выше. Берите карту";
                }
            case 5: return "Никогда не разделяйте пятерки. Берите
карту если у вас меньше 17";
```

```
            case 4:
                if (dealerCardValue == 5 || dealerCardValue == 6)
{
                    return "Разделяйте четверки, если у диллера 5
или 6";
                }
                else {
                    return "Не разделяйте четверки. Берите
карту";
                }
            case 3:
            case 2:
                if (dealerCardValue >= 2 && dealerCardValue <= 7)
{
                    return "Разделяйте 2 или 3, если у диллера 2-
7";
                }
                else {
                    return "Не разделяйте 2 или  3, если у
диллера 8 или более. Берите карту";
                }
            }
        }

        if (playerTotal >= 17) {
            return "Если у вас больше 17, всегда
останавливайтесь";
        }

        else if (playerTotal >= 13 && playerTotal <= 16) {
            if (dealerCardValue >= 2 && dealerCardValue <= 6) {
                return "Останавливайтесь, если у вас 13-16, а у
диллера 2-6";
            }
            else {
                return "Берите карту, если у вас 13 -16, а у
диллера 7 и более";
            }
        }
        else {
            return "Всегда берите карту на 12 или меньше";
        }
    }

    void Game::playRound() {
        player->resetRound();
        dealer->clearHands();

        if (player->getMoney() <= 0) {
            std::cout << "\nУ вас закончились деньги. Игра
окончена\n";
```

```
            gameOver = true;
            return;
        }

        std::cout << "\nНачало новой партии\n--------------------
-\n";

        std::cout << "Баланс: $" << player->getMoney() << "\n";

        int betAmount;

        // выполняем пока пользователь не укажет допустимую сумму
        do {
            std::cout << "Ваша ставка?\n";
            std::cin >> betAmount;
            if (betAmount <= 0 || betAmount > player->getMoney())
{
                std::cout << "Невозможная сумма ставки\n";
            }
        } while (betAmount <= 0 || betAmount > player-
>getMoney());

        player->placeBet(betAmount);

        // Раздача первой партии карт
        player->addCard(deck.giveCard());
        dealer->addCard(deck.giveCard());
        player->addCard(deck.giveCard());
        dealer->addCard(deck.giveCard());

        // Показать количество карт в колодах
        std::cout << "\n";
        deck.displayDecksRemainingCards();

        // Проверить на блэкджек диллера
        if (dealer->getCurrentHand().hasBlackjack()) {
            betResults();
            return;
        }

        // Проверить на блэкДжек игрока
        if (player->getCurrentHand().hasBlackjack()) {
            betResults();
            return;
        }

        // Ход игрока для каждой руки
        player->resetToFirstHand();

        while (!player->allHandsDone()) {
            if (!player->getNextHandToPlay()) {
```

```
                break;
            }

            playPlayerHand();
        }


        // Ход диллера если у игрока есть руки, в которых не
более 21 очка
        bool hasPlayableHands = false;
        for (const auto& hand : player->getHands()) {
            if (!hand.isBust()) {
                hasPlayableHands = true;
                break;
            }
        }

        if (hasPlayableHands) {
            playDealerHand();
        }
        else {
            std::cout << "\nВы взяли слишком много карт! Диллер
выигрывает\n";
        }

        betResults();
    }


    void Game::printCards() {
        std::cout << "Диллер: ";
        dealer->displayHand(false);
        // Выводим сообщение про руку только если больше одной
руки

        std::cout << "\n";
        if (player->getTotalHands() > 1) {
            std::cout << "---Рука " << (player-
>getCurrentHandIndex() + 1) << "---\n";
        }
        player->displayHands();
    }

    void Game::playPlayerHand() {
        bool handDone = false;

        printCards();

        while (!handDone && !player->getCurrentHand().isBust()) {
```

```
                std::cout << "\n1. Остановиться\n2. Взять карту\n";
                if (player->canSplit()) {
                    std::cout << "3. Сплит\n";
                }
                std::cout << "9. Получить подсказку\n";

                int choice;
                std::cin >> choice;

                switch (choice) {
                case 1: // Остановиться
                    std::cout << "Вы остановились\n";
                    player->standCurrentHand();
                    handDone = true;
                    player->displayHands();
                    break;

                case 2: // Взять карту
                    player->addCard(deck.giveCard());
                    std::cout << "Вы взяли карту\n";
                    player->displayHands();
                    break;

                case 3: // Сплит
                    if (player->canSplit()) {
                        if (player->split()) {
                            std::cout << "Рука разделена!\n";

                            // Раздать карты в обе руки
                            player->addCard(deck.giveCard());

                            if (player->getHands().size() > 1) {
                                Card newCard = deck.giveCard();
                                player-
>getHands()[1].addCard(newCard);
                            }

                            //printCards();
                            player->displayHands();
                        }
                        else {
                            std::cout << "Невозможно сделать
сплит\n";
                        }
                    }
                    else {
                        std::cout << "Невозможно сделать сплит\n";
                    }
                    break;

                case 9: // Подсказка
```

```
                std::cout << "ПОДСКАЗКА: " << getHint() << "\n";
                break;

            default:
                std::cout << "Неверный выбор\n";
                break;
            }
        }

        if (player->getCurrentHand().isBust()) {
            player->markCurrentHandDone();
        }
    }

    void Game::playDealerHand() {
        std::cout << "\nХод Диллера\n--------------------\n";
        std::cout << "Диллер: ";
        dealer->displayHand(true); // показать все карты диллера

        while (dealer->shouldHit() && !dealer->isBust()) {
            std::cout << "\nДиллер берет карту...\n";
            dealer->addCard(deck.giveCard());

            std::cout << "Диллер: ";
            dealer->displayHand(true);
        }
        std::cout << "\n";

        if (dealer->isBust()) {
            std::cout << "Диллер перебрал\n";
            // Такая ситуация однако не произойдёт, т.к. мы
обрабатываем перебор как исключение
        }
    }

    void Game::betResults() {
        int dealerTotal = dealer->getCurrentHand().getTotal();
        bool dealerBust = dealer->isBust();
        bool dealerBlackjack = dealer-
>getCurrentHand().hasBlackjack();

        std::cout << "-------------\n";
        printCards();
        std::cout << "-------------\n";

        for (int i = 0; i < player->getTotalHands(); i++) {
            Hand& hand = player->getHands()[i];
            int playerTotal = hand.getTotal();
            bool playerBlackjack = hand.hasBlackjack();
            if (player->getTotalHands() > 1)
                std::cout << "---Рука " << i + 1 << "---\n";
```

```
            if (hand.isBust()) {
                std::cout << "Итог: вы проиграли " << player-
>getCurrentBet() << "\n";
                player->lose(i);
            }
            else if (playerBlackjack && !dealerBlackjack) {
                std::cout << "Поздравляем! Блэкджек! Вы выиграли
" << (player->getCurrentBet() * 1.5) << "\n";

                player->win(player->getCurrentBet() * 1.5, i);
            }
            else if (dealerBlackjack && !playerBlackjack) {
                std::cout << "Итог: У диллера блэкджек, вы
проиграли " << player->getCurrentBet() << "\n";
                player->lose(i);
            }
            else if (playerBlackjack && dealerBlackjack) {
                std::cout << "Итог: У обоих блэкджек, ничья\n";
                player->draw(i);
            }
            else if (dealerBust) {
                std::cout << "Поздравляем! Диллер перебрал, вы
выиграли " << player->getCurrentBet() << "\n";
                player->win(player->getCurrentBet(), i);
            }
            else if (playerTotal > dealerTotal) {
                std::cout << "Поздравляем! Вы выиграли " <<
player->getCurrentBet() << "\n";
                player->win(player->getCurrentBet(), i);
            }
            else if (playerTotal < dealerTotal) {
                std::cout << "Итог: вы проиграли " << player-
>getCurrentBet() << "\n";
                player->lose(i);
            }
            else {
                std::cout << "Итог: Ничья, ставка возвращена\n";
                player->draw(i);
            }
        }

        std::cout << "\nБаланс: " << player->getMoney() << "\n";
    }


    void Game::run() {
        initialize();

        while (!gameOver) {
```

```
            playRound();

            if (!gameOver) {
                std::cout << "\nСыграть еще раунд?\n1 - да\n2 -
нет\nВаш выбор: ";
                std::string choice;
                std::cin >> choice;

                if (choice == "1") {
                    continue;
                }
                else if (choice == "2") {
                    gameOver = true;
                    std::cout << "\nСпасибо за игру! Итоговый
баланс: "
                        << player->getMoney() << "\n";
                }
                else {
                    std::cout << "Неправильный вариант ответа\n";
                }
            }
        }
    }
```

hand.h

```
#ifndef HAND_H
#define HAND_H

#include "card.h"
#include <vector>
#include <stdexcept>

class Hand {
private:
    std::vector<Card> cards;

public:
    Hand() = default;
    void addCard(const Card& card); // Добавить карту в руку
    void clear(); // Убрать карты из руки
    int getTotal() const; // Общая стоимость руки

    Card getCard(int index) const; // Вернуть карту из руки
под определённым индексом
    int getCardAmount() const;
    const std::vector<Card>& getCards() const;  // Все карты
руки
    bool isBust() const; // Был ли привышен лимит в 21 очко
    bool hasBlackjack() const;
```

```
        bool canSplit() const;
        void display() const;
    };

    #endif
```

hand.cpp

```
    #include "hand.h"

    // Добавить карту в руку и проверить на перебор
    void Hand::addCard(const Card& card) {
        cards.push_back(card);
        // Исключение если перебор
        if (getTotal() > 21) {
            throw std::runtime_error("ПЕРЕБОР! Сумма карт: " +
std::to_string(getTotal()));
        }
    }

    void Hand::clear() {
        cards.clear();
    }


    int Hand::getTotal() const {
        int total = 0;
        int aceCount = 0;

        for (const auto& card : cards) {
            // Учитываем только стоимость открытых карт (часть
карт диллера может быть закрыта)
            if (card.isFaceUp()) {
                total += card.getValue();
                if (card.isAce()) {
                    aceCount++;
                }
            }
        }

        if (cards.size() == 2 && aceCount == 2) {
            return 21;
        }

        return total;
    }

    int Hand::getCardAmount() const {
        return cards.size();
    }
```

```
Card Hand::getCard(int index) const {
    if (index >= 0 && index < static_cast<int>(cards.size()))
{
        return cards[index];
    }

    throw std::out_of_range("Неверный индекс карты");
}

const std::vector<Card>& Hand::getCards() const {
    return cards;
}

bool Hand::isBust() const {
    return getTotal() > 21;
}

bool Hand::hasBlackjack() const {
    if (getTotal() == 21)
        return true;
    return false;
}

bool Hand::canSplit() const {
    if (cards.size() == 2) {
        return cards[0].getRank() == cards[1].getRank();
    }
    return false; // на случай, если карт не 2
}

void Hand::display() const {
    for (const auto& card : cards) {
        card.display();
        std::cout << " ";
    }
}
```

participant.h

```
#ifndef PARTICIPANT_H
#define PARTICIPANT_H

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

// из этого класса наследуются диллер и игрок
class Participant {
protected:
    std::vector<Hand> hands;
```

```
        int currentHandIndex;

    public:
        Participant();
        virtual void addCard(const Card& card);
        Hand& getCurrentHand();
        std::vector<Hand>& getHands();
        const std::vector<Hand>& getHands() const;
        void clearHands();
        bool isBust() const;
        virtual void displayHand(bool showAll = true) const;
        virtual ~Participant() = default;
    };

    #endif
```

participant.cpp

```
    #include "participant.h"

    Participant::Participant() : currentHandIndex(0) {
        hands.emplace_back(); // добавляет новый объект Hand в
конец вектора
    }

    void Participant::addCard(const Card& card) {
        hands[currentHandIndex].addCard(card);
    }

    Hand& Participant::getCurrentHand() {
        return hands[currentHandIndex];
    }

    // для безопасности и совместимости 2 метода, константный и
не константный
    std::vector<Hand>& Participant::getHands() {
        return hands;
    }

    const std::vector<Hand>& Participant::getHands() const {
        return hands;
    }

    void Participant::clearHands() {

        hands.clear();
        hands.emplace_back();
        currentHandIndex = 0;
    }
```

```
bool Participant::isBust() const {
    return hands[currentHandIndex].isBust();
}

void Participant::displayHand(bool showAll) const {
    hands[currentHandIndex].display();
}
```

player.h

```
#ifndef PLAYER_H
#define PLAYER_H

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

class Player : public Participant {
private:
    std::string name;
    int money;
    std::vector<int> bets; // может быть несколько ставок на
несколько рук
    bool hasSplit; // был ли сплит
    std::vector<bool> handDone;
    std::vector<bool> handStand;

public:
    Player(std::string n);
    bool placeBet(int amount, int handIndex = 0);
    bool split();
    bool canSplit() const;
    bool hasCurrentHandStood() const; // Была ли остановлена
текущая рука
    void standCurrentHand(); // Остановить текущую руку,
больше карт не брать
    void markCurrentHandDone(); // Завершить партию для
    void resetToFirstHand(); // выбрать первую руку
    bool allHandsDone() const; // Завершить партию для всех
рук
    void resetRound();
    void win(int amount, int handIndex = 0);
    void lose(int handIndex = 0);
    void draw(int handIndex = 0);
    bool getNextHandToPlay();

    // Методы получения данных
    std::string getName() const;
    int getMoney() const;
    int getCurrentBet() const;
```

```
        bool getHasSplit() const;
        int getCurrentHandIndex() const;
        int getTotalHands() const;
        void displayHands() const;
        int getBetForHand(int handIndex) const;
    };

    #endif
```

player.cpp

```
    #include "player.h"

    Player::Player(std::string n) : name(std::move(n)),
money(2500), hasSplit(false) {

        // name(std::move(n)) передаёт владение строкой без
копирования
        bets.push_back(0);
        handDone.push_back(false);
        handStand.push_back(false);
    }

    bool Player::placeBet(int amount, int handIndex) {
        // Если номер текущей руки больше текущего размера
массива ставок, то добавляем новое место под ставку
        if (handIndex >= static_cast<int>(bets.size())) {
            bets.resize(handIndex + 1, 0);
        }

        if (money >= amount) {
            money -= amount;
            bets[handIndex] = amount;

            return true;
        }
        return false;
    }


    bool Player::split() {
        // Нужно чтобы руку можно было поделить, она не была уже
поделена и количество карт было равно 2
        if (hands[currentHandIndex].canSplit() && !hasSplit &&
            hands[currentHandIndex].getCardAmount() == 2) {

            Hand newHand;
            Card secondCard = hands[currentHandIndex].getCard(1);
            newHand.addCard(secondCard);
```

```
            Hand currentHand;

currentHand.addCard(hands[currentHandIndex].getCard(0));
            hands[currentHandIndex] = currentHand;

            hands.push_back(newHand);

            bets.push_back(bets[currentHandIndex]);
            money -= bets[currentHandIndex];

            handDone.push_back(false);
            handStand.push_back(false);

            hasSplit = true;
            return true;
        }
        return false;
    }

    bool Player::canSplit() const {
        return hands[currentHandIndex].canSplit() && !hasSplit &&
            hands[currentHandIndex].getCardAmount() == 2;
    }

    bool Player::hasCurrentHandStood() const {
        // проверка, что не будет выхода за границу вектора
handStand
        if (currentHandIndex <
static_cast<int>(handStand.size())) {
            return handStand[currentHandIndex];

        }
        return false;
    }

    void Player::standCurrentHand() {
        if (currentHandIndex <
static_cast<int>(handStand.size())) {
            handStand[currentHandIndex] = true;
            handDone[currentHandIndex] = true;
        }
    }

    void Player::markCurrentHandDone() {
        if (currentHandIndex < static_cast<int>(handDone.size()))
{
            handDone[currentHandIndex] = true;
        }
    }

    bool Player::getNextHandToPlay() {
```

```
        for (int i = 0; i < static_cast<int>(hands.size()); i++)
{
            if (!handDone[i] && !hands[i].isBust()) {
                currentHandIndex = i;
                return true;
            }

        }
        return false;
    }

    void Player::resetToFirstHand() {
        currentHandIndex = 0;
    }

    bool Player::allHandsDone() const {
        for (size_t i = 0; i < hands.size(); i++) {
            if (!handDone[i] && !hands[i].isBust()) {
                return false;
            }
        }
        return true;
    }

    void Player::resetRound() {
        clearHands();
        bets.clear();
        bets.push_back(0);
        handDone.clear();

        handDone.push_back(false);
        handStand.clear();
        handStand.push_back(false);
        currentHandIndex = 0;
        hasSplit = false;
    }

    void Player::win(int amount, int handIndex) {

        if (handIndex < static_cast<int>(bets.size())) {
            money += amount + bets[handIndex]; // возврат ставки
+ выигрыш
            bets[handIndex] = 0;
        }
    }

    void Player::lose(int handIndex) {
        if (handIndex < static_cast<int>(bets.size())) {
            bets[handIndex] = 0;
        }
    }
```

```
void Player::draw(int handIndex) {
    if (handIndex < static_cast<int>(bets.size())) {
        money += bets[handIndex];
        bets[handIndex] = 0;
    }
}

std::string Player::getName() const {
    return name;
}

int Player::getMoney() const {
    return money;
}

int Player::getCurrentBet() const {
    if (currentHandIndex < static_cast<int>(bets.size())) {
        return bets[currentHandIndex];
    }
    else {
        return 0;
    }
}

bool Player::getHasSplit() const {
    return hasSplit;
}


int Player::getCurrentHandIndex() const {
    return currentHandIndex;
}

int Player::getTotalHands() const {
    return hands.size();
}

void Player::displayHands() const {
    std::cout << name << ": ";
    for (size_t i = 0; i < hands.size(); i++) {
        if (i > 0) std::cout << "  ";
        hands[i].display();
    }
    std::cout << std::endl;
}

int Player::getBetForHand(int handIndex) const {
    if (handIndex >= 0 && handIndex <
static_cast<int>(bets.size())) {
        return bets[handIndex];
```

```
        }
        return 0;
    }
```

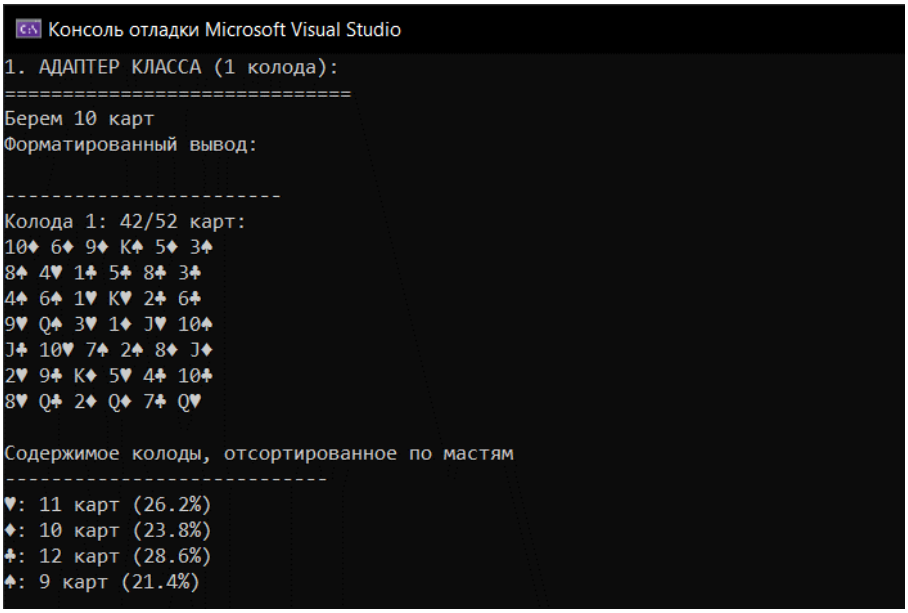
main.cpp

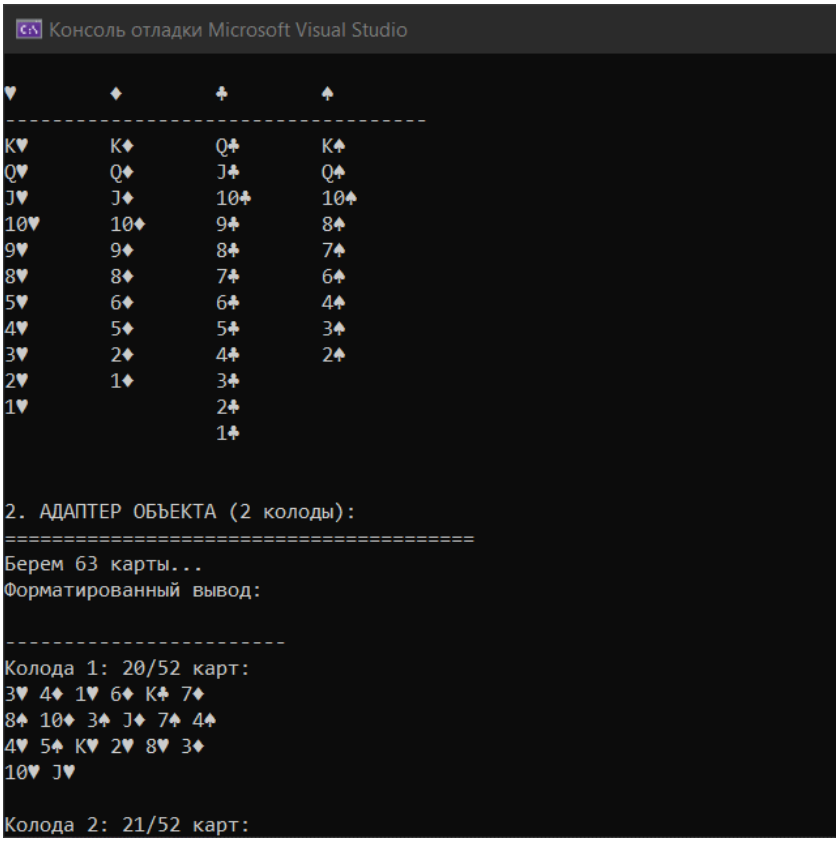
```
#include "game.h"

int main() {
    setlocale(LC_ALL, "Russian");

    // Рандом
    std::srand(static_cast<unsigned
int>(std::time(nullptr)));

    // Всё делается внутри класса
    Game blackjackGame;
    blackjackGame.run();

    return 0;
}
```

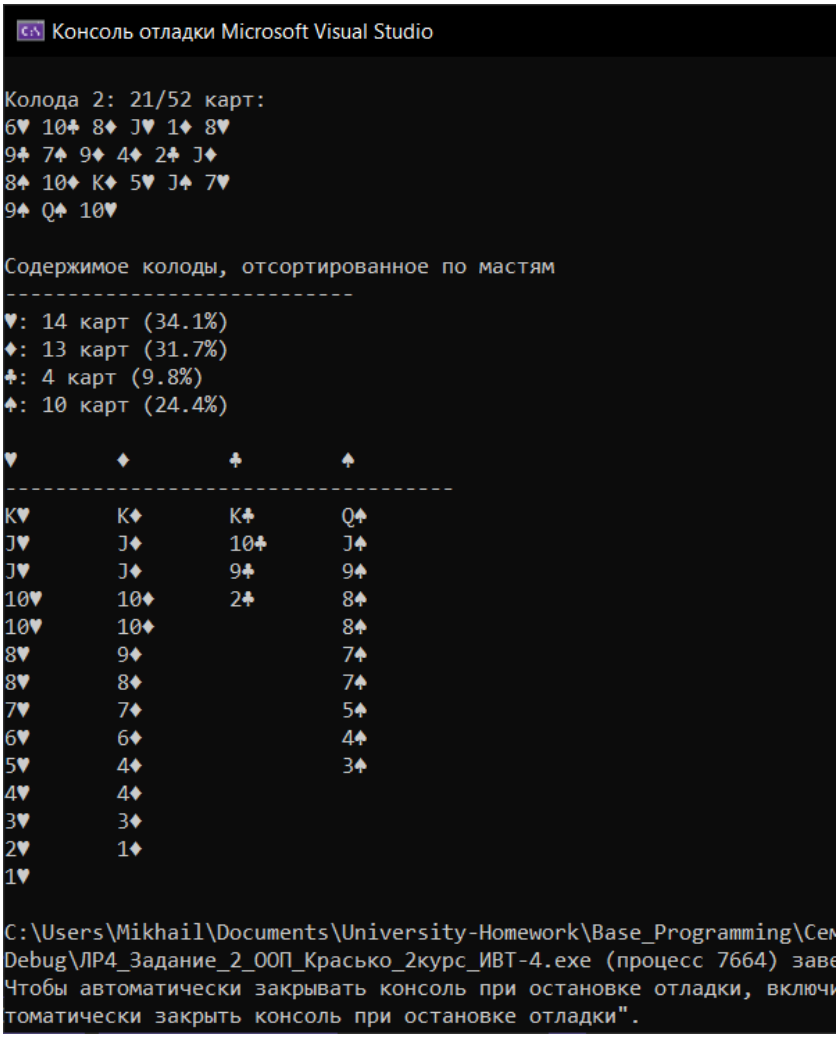
Задание 2.

2. Реализуйте паттерн GOF «Адаптер» в обоих видах (адаптер класса и адаптер объекта) на следующем примере. Напишите интерфейс `IFormattable` с методом `std::string format()` для форматирования объекта, реализующего данный интерфейс, а также функцию `void prettyPrint(const IFormattable& object)`, которая выводит на экран объект-параметр в отформатированном виде. Сделайте так, чтобы в эту функцию можно было передавать ссылку на Вашу колоду карт, разработанную в задании 1. Алгоритм форматирования придумайте произвольный сами.

Вывод в консоль:




Консоль отладки Microsoft Visual Studio
♥ ♦ ♣ ♠
K♥ K♦ Q♣ K♠
Q♥ Q♦ J♣ Q♠
J♥ J♦ 10♣ 10♠
10♥ 10♦ 9♣ 8♠
9♥ 9♦ 8♣ 7♠
8♥ 8♦ 7♣ 6♠
5♥ 6♦ 6♣ 4♠
4♥ 5♦ 5♣ 3♠
3♥ 2♦ 4♣ 2♠
2♥ 1♦ 3♣
1♥ 2♣
1♣
2. АДАПТЕР ОБЪЕКТА (2 колоды):
Берем 63 карты...
Форматированный вывод:
Колода 1: 20/52 карт:
3♥ 4♦ 1♥ 6♦ K♣ 7♦
8♠ 10♦ 3♠ J♦ 7♠ 4♠
4♥ 5♠ K♥ 2♥ 8♥ 3♦
10♥ J♥
Колода 2: 21/52 карт:


Консоль отладки Microsoft Visual Studio
Колода 2: 21/52 карт:
6♥ 10♣ 8♦ J♥ 1♦ 8♥
9♣ 7♠ 9♦ 4♦ 2♣ J♦
8♠ 10♦ K♦ 5♥ J♠ 7♥
9♠ Q♠ 10♥
Содержимое колоды, отсортированное по мастям
♥: 14 карт (34.1%)
♦: 13 карт (31.7%)
♣: 4 карт (9.8%)
♠: 10 карт (24.4%)
♥ ♦ ♣ ♠
K♥ K♦ K♣ Q♠
J♥ J♦ 10♣ J♠
J♥ J♦ 9♣ 9♠
10♥ 10♦ 2♣ 8♠
10♥ 10♦ 8♠
8♥ 9♦ 7♠
8♥ 8♦ 7♠
7♥ 7♦ 5♠
6♥ 6♦ 4♠
5♥ 4♦ 3♠
4♥ 4♦
3♥ 3♦
2♥ 1♦
1♥
C:\Users\Mikhail\Documents\University-Homework\Base_Programming\Се
Debug\ЛР4_Задание_2_ООП_Красько_2курс_ИВТ-4.exe (процесс 7664) заве
Чтобы автоматически закрывать консоль при остановке отладки, включи
томатически закрыть консоль при остановке отладки".

Код:

Card.h

```
#ifndef CARD_H
#define CARD_H

#include <string>
#include <iostream>

// Suit - масть
enum class Suit { HEARTS, DIAMONDS, CLUBS, SPADES };

// Ace - туз
enum class Rank {
    TWO = 2, THREE = 3, FOUR = 4, FIVE = 5, SIX = 6, SEVEN = 7,
    EIGHT = 8, NINE = 9, TEN = 10, JACK = 11, QUEEN = 12, KING =
13, ACE = 14
};

class Card {
private:
    Suit suit;
    Rank rank;
    bool faceUp; // Нужно для определения, какие карты диллера
показывать, а какие нет

public:
    Card(Suit s, Rank r);

    int getValue() const;
    Rank getRank() const;
    bool isAce() const;
    bool isTenValue() const;
    void flip();// Показать карту диллера
    bool isFaceUp() const; // Показано ли содержание карты. Нужно
для подсказок т.к. они делаются только на основе той информации,
    // которая есть у пользователя
    std::string suitToString() const;
    std::string rankToString() const;
    void display() const;

    Suit getSuit() const { return suit; } // Необходимо для
сортировки по масте
};

#endif // CARD_H
```

Card.cpp

```
#include "Card.h"

Card::Card(Suit s, Rank r) : suit(s), rank(r), faceUp(true) {
}

int Card::getValue() const {
    if (isTenValue())
```

```
            return 10;
        return static_cast<int>(rank);
    }

    Rank Card::getRank() const {
        return rank;
    }

    bool Card::isAce() const {
        return rank == Rank::ACE;
    }

    bool Card::isTenValue() const {
        return rank == Rank::TEN || rank == Rank::JACK ||
            rank == Rank::QUEEN || rank == Rank::KING;
    }

    void Card::flip() {
        faceUp = !faceUp;
    }

    bool Card::isFaceUp() const {
        return faceUp;
    }

    std::string Card::suitToString() const {
        switch (suit) {
            case Suit::HEARTS: return std::string(1, char(3));    //
Сердце
            case Suit::DIAMONDS: return std::string(1, char(4));  //
Ромб
            case Suit::CLUBS: return std::string(1, char(5));     //
Клевер
            case Suit::SPADES: return std::string(1, char(6));    //
пика
        default: return "?";
        }
    }


    std::string Card::rankToString() const {
        switch (rank) {
            case Rank::ACE: return "A";
            case Rank::JACK: return "J";
            case Rank::QUEEN: return "Q";
            case Rank::KING: return "K";
        default: return std::to_string(getValue());
        }
    }

    void Card::display() const {
        // Встроенная функция показа только тех карт, которые можно
видеть игроку
        if (faceUp) {
            std::cout << rankToString() << suitToString();
        }
```

```
        else {
            std::cout << "??";
        }
    }
```

Deck.h

```
#ifndef DECK_H
#define DECK_H

#include "Card.h"
#include <vector>
#include <random>
#include <ctime>

class Deck {
private:
    std::vector<std::vector<Card>> decks;
    std::vector<int> currentIndices; // Позиция в каждой колоде
    int nextDeckIndex;  // Следующая карта для каждой колоды
    int deckCount; // Количество колод. Я добавил возможность
выбирать число колод,
    // чтобы вывод всех карт не был громоздким

public:
    Deck(int numDecks = 4);
    void initialize();
    Card giveCard();
    void shuffleDeck(int deckIndex); // shuffle - перемешать
    void displayDecksRemainingCards() const; // отобразить
оставшиеся карты в каждой колоде
    int getRemainingAmountInDeck(int deckIndex) const; //
Оставшееся колво карт в конкретной колод

    // Геттеры для адаптеров
    int getDeckCount() const { return deckCount; }
    int getTotalCards() const { return deckCount * 52; }
    int getRemainingCards() const;

    // Новый метод для адаптеров
    std::vector<std::vector<Card>> getAllCards() const; // Получить
все карты всех колод
};

#endif
```

Dech.cpp

```
#include "Deck.h"
#include <algorithm>
#include <iostream>

Deck::Deck(int numDecks) : deckCount(numDecks), nextDeckIndex(0) {
    initialize();
}

void Deck::initialize() {
```

```
        decks.clear();
        currentIndices.clear();

        // Создаем указанное количество колод
        for (int deckNum = 0; deckNum < deckCount; deckNum++) {
            std::vector<Card> deck;

            for (int suitIndex = 0; suitIndex < 4; suitIndex++) {
                Suit suit = static_cast<Suit>(suitIndex);

                for (int rankValue = 1; rankValue <= 13; rankValue++) {
                    Rank rank = static_cast<Rank>(rankValue);
                    deck.emplace_back(suit, rank);
                }
            }

            // Сохраняем созданную колоду
            decks.push_back(deck);
            currentIndices.push_back(0);
        }

        for (int i = 0; i < deckCount; i++) {
            shuffleDeck(i);
        }

        nextDeckIndex = 0;
    }

    void Deck::shuffleDeck(int deckIndex) {
        // Создаем генератор случайных чисел
        std::random_device rd;
        std::mt19937 g(rd());

        // Перемешиваем карты в указанной колоде
        std::shuffle(decks[deckIndex].begin(), decks[deckIndex].end(),
g);

        currentIndices[deckIndex] = 0;
    }

    // Дать одну карту и перейти к следующей колоде
    Card Deck::giveCard() {
        // Если текущая колода закончилась, перетасовать ее
        if (currentIndices[nextDeckIndex] >= 52) {
            shuffleDeck(nextDeckIndex);
        }

        Card card =
decks[nextDeckIndex][currentIndices[nextDeckIndex]];
        currentIndices[nextDeckIndex]++;

        nextDeckIndex = (nextDeckIndex + 1) % deckCount;

        return card;
    }
```

```
// Показать оставшиеся карты в каждой колоде
void Deck::displayDecksRemainingCards() const {
    std::cout << "Колоды: ";
    for (int i = 0; i < deckCount; i++) {
        int remaining = 52 - currentIndices[i];
        std::cout << "[" << remaining << "] ";
    }

    std::cout << std::endl;
}

// Получить оставшиеся карты в конкретной колоде
int Deck::getRemainingAmountInDeck(int deckIndex) const {
    if (deckIndex >= 0 && deckIndex < deckCount) {
        return 52 - currentIndices[deckIndex];
    }
    return 0;
}

// Получить все карты всех колод
std::vector<std::vector<Card>> Deck::getAllCards() const {
    return decks;
}

// Получить общее количество оставшихся карт
int Deck::getRemainingCards() const {
    int total = 0;
    for (int i = 0; i < deckCount; i++) {
        total += getRemainingAmountInDeck(i);
    }
    return total;
}
```

DeckClassAdapter.h

```
#ifndef DECKCLASSADAPTER_H
#define DECKCLASSADAPTER_H

#include "Deck.h"
#include "IFormattable.h"

// Наследует Deck и IFormatable
class DeckClassAdapter : public Deck, public IFormattable {
public:
    DeckClassAdapter(int numDecks = 1) : Deck(numDecks) {}
    std::string format() const override;

private:
    std::string formatAllCardsInOrder() const; // Выводит все карты
в консоль в том порядке, в котором они были в колоде
    std::string formatCardsBySuit() const; // Выводит карты,
отсортированные через getCardsSortedbySuit в консоль

    std::vector<Card> getAllCardsFlat() const; // Превращает
двухмерный вектор в одномерный для удобства работы
```

```
        std::vector<std::vector<Card>> getCardsSortedBySuit() const; //
Сортирует карты по масти
    };

    #endif
```

DeckClassAdapter.cpp

```
    #include "DeckClassAdapter.h"
    #include <sstream>
    #include <iomanip>
    #include <algorithm>

    // Соединить все карты в один вектор из разных колод
    std::vector<Card> DeckClassAdapter::getAllCardsFlat() const {
        std::vector<Card> allCards;
        auto decks = Deck::getAllCards();

        for (int deckIdx = 0; deckIdx < getDeckCount(); deckIdx++) {
            const auto& deck = decks[deckIdx];
            int currentIndex = getRemainingAmountInDeck(deckIdx); //
Нужно оставшееся количество карт т.к. в векторах колод не удаляются
карты при раздачи,
            // просто смещается начальный индекс

            for (int i = 52 - currentIndex; i < 52; i++) {
                if (i >= 0 && i < static_cast<int>(deck.size())) {
                    allCards.push_back(deck[i]);
                }
            }
        }
        return allCards;
    }

    // Отсортировать все карты по мастям
    std::vector<std::vector<Card>>
DeckClassAdapter::getCardsSortedBySuit() const {
        std::vector<std::vector<Card>> suits(4);

        auto allCards = getAllCardsFlat();

        for (const auto& card : allCards) {
            std::string suitSymbol;

            int suitIndex = -1;
            switch (card.getSuit()) {
            case Suit::HEARTS:   suitIndex = 0; break;
            case Suit::DIAMONDS: suitIndex = 1; break;
            case Suit::CLUBS:    suitIndex = 2; break;
            case Suit::SPADES:   suitIndex = 3; break;
            }

            if (suitIndex != -1) {
                suits[suitIndex].push_back(card);
            }
        }
```

```
        // Сортируем карты по ценности
        for (int suitIndex = 0; suitIndex < 4; suitIndex++) {
            std::vector<Card>& currentSuit = suits[suitIndex];
            int n = currentSuit.size();

            // Пузырьковая сортировка
            for (int i = 0; i < n - 1; i++) {
                for (int j = 0; j < n - i - 1; j++) {
                    if (currentSuit[j].getRank() < currentSuit[j +
1].getRank()) {
                        Card temp = currentSuit[j];
                        currentSuit[j] = currentSuit[j + 1];
                        currentSuit[j + 1] = temp;
                    }
                }
            }
        }

        return suits;
    }

    std::string DeckClassAdapter::format() const {
        std::ostringstream resultString;

        resultString << "\n-------------------------\n";

        resultString << formatAllCardsInOrder();
        resultString << "Содержимое колоды, отсортированное по
мастям\n----------------------------\n";

        resultString << formatCardsBySuit();

        return resultString.str();
    }

    std::string DeckClassAdapter::formatAllCardsInOrder() const {
        std::ostringstream resultString;

        auto decks = Deck::getAllCards();

        for (int deckIdx = 0; deckIdx < getDeckCount(); deckIdx++) {
            int remaining = getRemainingAmountInDeck(deckIdx);

            resultString << "Колода " << (deckIdx + 1) << ": " <<
remaining << "/52 карт:\n";

            if (remaining == 0) continue; // Пропускаем пустые колоды

            const auto& deck = decks[deckIdx];
            int startIndex = 52 - remaining; // Начинаем с первой
нераздannой карты

            for (int i = startIndex; i < 52; i++) {
                const auto& card = deck[i];
```

```
                resultString << card.rankToString() <<
card.suitToString();

                // Перенос строки для читаемости
                if ((i - startIndex + 1) % 6 == 0) {
                    resultString << "\n";
                }
                else {
                    resultString << " ";
                }
            }
            resultString << "\n";
        }

        return resultString.str();
    }

    std::string DeckClassAdapter::formatCardsBySuit() const {
        std::ostringstream resultString;

        auto suits = getCardsSortedBySuit();

        // Названия мастей
        std::vector<std::string> suitNames = {
        std::string(1, char(3)),
        std::string(1, char(4)),
        std::string(1, char(5)),
        std::string(1, char(6))
        };

        int totalRemaining = getRemainingCards();
        for (int i = 0; i < 4; i++) {
            // Вывод информации о том, какие масти остались в каком
процентном соотношении
            int suitCount = suits[i].size();
            double percentage;
            if (totalRemaining > 0) {
                percentage = suitCount * 100.0 / totalRemaining;
            }
            else {
                percentage = 0;
            }
            resultString << suitNames[i] << ": " << suitCount << " карт
("
                << std::fixed << std::setprecision(1) << percentage <<
"%)\n";
        }
        resultString << "\n";

        // Максимальное количество карт в одной масти
        size_t maxCards = 0; // Необходимо для того, чтобы определить
сколько строк в таблице
        for (const auto& suitCards : suits) {
            maxCards = std::max(maxCards, suitCards.size());
        }
```

```
        // Заголовки
        for (int i = 0; i < 4; i++) {
            resultString << suitNames[i] << "        ";
        }
        resultString << "\n";

        for (int i = 0; i < 4; i++) {
            resultString << "---------";
        }
        resultString << "\n";

        // 4 столбика
        for (size_t row = 0; row < maxCards; row++) {
            for (int suitIdx = 0; suitIdx < 4; suitIdx++) {
                if (row < suits[suitIdx].size()) {
                    const auto& card = suits[suitIdx][row];
                    std::string cardStr = card.rankToString() +
card.suitToString();
                    resultString << std::setw(9) << std::left <<
cardStr;
                }
                else {
                    resultString << std::setw(9) << " ";
                }
            }
            resultString << "\n";
        }

        return resultString.str();
    }
```

DeckObjectAdapter.h

```
    #ifndef DECKOBJECTADAPTER_H
    #define DECKOBJECTADAPTER_H

    #include "Deck.h"
    #include "IFormattable.h"

    class DeckObjectAdapter : public IFormattable {
    private:
        const Deck& deck;

    public:
        explicit DeckObjectAdapter(const Deck& d) : deck(d) {} //
Защита от неявного преобразования
        std::string format() const override;

    private:
        std::string formatAllCardsInOrder() const;
        std::string formatCardsBySuit() const;

        std::vector<Card> getAllCardsFlat() const;
        std::vector<std::vector<Card>> getCardsSortedBySuit() const;
    };
```

```
#endif
```

DeckObjectAdapter.cpp

```
#include "DeckObjectAdapter.h"
#include <sstream>
#include <iomanip>
#include <algorithm>

// Соединить все карты в один вектор из разных колод
std::vector<Card> DeckObjectAdapter::getAllCardsFlat() const {
    std::vector<Card> allCards;
    auto decks = deck.getAllCards();

    for (int deckIdx = 0; deckIdx < deck.getDeckCount(); deckIdx++)
{
        const auto& deckCards = decks[deckIdx];
        int currentIndex = deck.getRemainingAmountInDeck(deckIdx);
// Нужно оставшееся количество карт т.к. в векторах колод не удаляются
карты при раздачи,
        // просто смещается начальный индекс

        for (int i = 52 - currentIndex; i < 52; i++) {
            if (i >= 0 && i < static_cast<int>(deckCards.size())) {
                allCards.push_back(deckCards[i]);
            }
        }
    }
    return allCards;
}

// Отсортировать все карты по мастям
std::vector<std::vector<Card>>
DeckObjectAdapter::getCardsSortedBySuit() const {
    std::vector<std::vector<Card>> suits(4);
    auto allCards = getAllCardsFlat();


    for (const auto& card : allCards) {
        int suitIndex = -1;
        switch (card.getSuit()) {
        case Suit::HEARTS:   suitIndex = 0; break;
        case Suit::DIAMONDS: suitIndex = 1; break;
        case Suit::CLUBS:    suitIndex = 2; break;
        case Suit::SPADES:   suitIndex = 3; break;
        }

        if (suitIndex != -1) {
            suits[suitIndex].push_back(card);
        }
    }

    // Сортируем карты по ценности
    for (int suitIndex = 0; suitIndex < 4; suitIndex++) {
        std::vector<Card>& currentSuit = suits[suitIndex];
        int n = currentSuit.size();
```

```
            // Пузырьковая сортировка
            for (int i = 0; i < n - 1; i++) {
                for (int j = 0; j < n - i - 1; j++) {
                    if (currentSuit[j].getRank() < currentSuit[j +
1].getRank()) {
                        Card temp = currentSuit[j];
                        currentSuit[j] = currentSuit[j + 1];
                        currentSuit[j + 1] = temp;
                    }
                }
            }
        }

        return suits;
    }

    std::string DeckObjectAdapter::format() const {
        std::ostringstream resultString;

        resultString << "\n-----------------------\n";

        resultString << formatAllCardsInOrder();
        resultString << "Содержимое колоды, отсортированное по
мастям\n--------------------------\n";

        resultString << formatCardsBySuit();

        return resultString.str();
    }

    std::string DeckObjectAdapter::formatAllCardsInOrder() const {
        std::ostringstream resultString;

        auto decks = deck.getAllCards();

        for (int deckIdx = 0; deckIdx < deck.getDeckCount(); deckIdx++)
{
            int remaining = deck.getRemainingAmountInDeck(deckIdx);
            resultString << "Колода " << (deckIdx + 1) << ": "
                << remaining << "/52 карт:\n";

            if (remaining == 0) continue; // Пропускаем пустые колоды

            const auto& deckCards = decks[deckIdx];
            int startIndex = 52 - remaining; // Начинаем с первой
неразданной карты

            for (int i = startIndex; i < 52; i++) {
                const auto& card = deckCards[i];
                resultString << card.rankToString() <<
card.suitToString();

                // Перенос строки для читаемости
                if ((i - startIndex + 1) % 6 == 0) {
                    resultString << "\n";
```

```
                }
                else {
                    resultString << " ";
                }
            }
            resultString << "\n\n";
        }

        return resultString.str();
    }

    std::string DeckObjectAdapter::formatCardsBySuit() const {
        std::ostringstream resultString;

        auto suits = getCardsSortedBySuit();

        // Названия мастей
        std::vector<std::string> suitNames = {
            std::string(1, char(3)),
            std::string(1, char(4)),
            std::string(1, char(5)),
            std::string(1, char(6))
        };

        int totalRemaining = deck.getRemainingCards();
        for (int i = 0; i < 4; i++) {
            // Вывод информации о том, какие масти остались в каком
процентном соотношении
            int suitCount = suits[i].size();
            double percentage;
            if (totalRemaining > 0) {
                percentage = suitCount * 100.0 / totalRemaining;
            }
            else {
                percentage = 0;
            }
            resultString << suitNames[i] << ": " << suitCount << " карт
("
                << std::fixed << std::setprecision(1) << percentage <<
"%)\n";
        }
        resultString << "\n";

        // Максимальное количество карт в одной масти
        size_t maxCards = 0; // Необходимо для того, чтобы определить
сколько строк в таблице
        for (const auto& suitCards : suits) {
            maxCards = std::max(maxCards, suitCards.size());
        }

        // Заголовки
        for (int i = 0; i < 4; i++) {
            resultString << suitNames[i] << "        ";
        }
        resultString << "\n";
```

```
            for (int i = 0; i < 4; i++) {
                resultString << "---------";
            }
            resultString << "\n";

            // 4 столбика
            for (size_t row = 0; row < maxCards; row++) {
                for (int suitIdx = 0; suitIdx < 4; suitIdx++) {
                    if (row < suits[suitIdx].size()) {
                        const auto& card = suits[suitIdx][row];
                        std::string cardStr = card.rankToString() +
card.suitToString();
                        resultString << std::setw(9) << std::left <<
cardStr;
                    }
                    else {
                        resultString << std::setw(9) << " ";
                    }
                }
                resultString << "\n";
            }

            return resultString.str();
        }
```

IFormattable.h

```
        #ifndef IFORMATTABLE_H
        #define IFORMATTABLE_H

        #include <string>

        class IFormattable {
        public:
            virtual ~IFormattable() = default; // сначала деструкторы
производных классов, а после этот
            virtual std::string format() const = 0; // метод нельзя создать
напрямую
        };

        void prettyPrint(const IFormattable& object);

        #endif
```

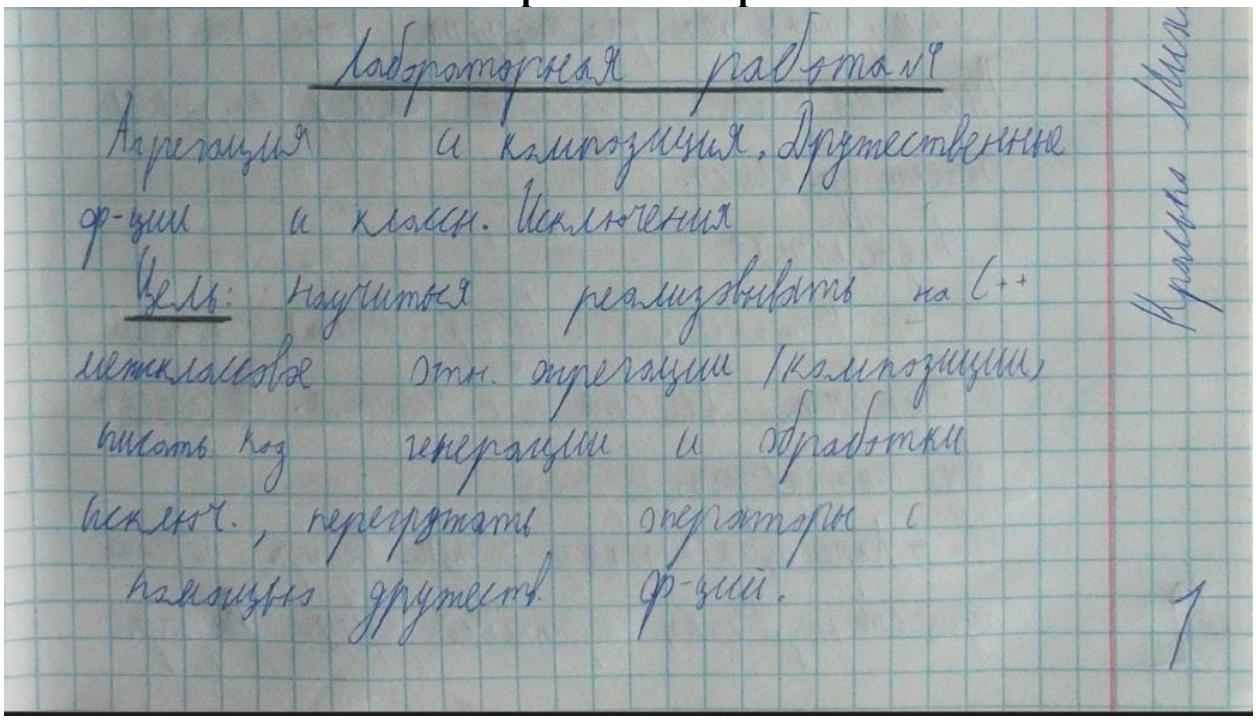
IFormattable.cpp

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

        void prettyPrint(const IFormattable& object) {
            std::cout << "Форматированный вывод:\n";
            std::cout << object.format();
        }
```

main.cpp

```cpp
#include <iostream>
#include "IFormattable.h"
#include "DeckClassAdapter.h"
#include "DeckObjectAdapter.h"

int main() {
    setlocale(LC_ALL, "Russian");

    std::cout << "1. АДАПТЕР КЛАССА (1 колода):\n";
    std::cout << std::string(40, '=') << "\n";

    DeckClassAdapter adapter(1);
    std::cout << "Берём 10 карт\n";
    for (int i = 0; i < 10; i++) {
        adapter.giveCard();
    }
    prettyPrint(adapter);

    std::cout << "\n\n2. АДАПТЕР ОБЪЕКТА (2 колоды):\n";
    std::cout << std::string(40, '=') << "\n";

    Deck deck2(2);
    std::cout << "Берём 63 карты...\n";
    for (int i = 0; i < 63; i++) {
        deck2.giveCard();
    }
    DeckObjectAdapter objectAdapter(deck2);
    prettyPrint(objectAdapter);

    return 0;
}
```

Контрольные вопросы



Контрольные вопросы

1. Приведите 3 примера пар классов, наслед. в отн. „has-a"

Car и Engine

University и Department (отделение, факультет)

Computer и ~~Pentium~~ Processor

2. Укажите сходства и различия композиции и агрегации

Сходства: оба реализуют отн. „has-a". Внешний Класс содержит ссылку/указатель/ объект др. класса.

Различия:

- Композиция: внутр. объект создаётся и уничт. вместе с внешним. Не существ. без интерфейса.

- Агрегация (слабая связь): внутр. объект существ. независимо, передаётся извне. Внутр.

2

класс только исп. его.

3. Опишите суть паттерна делегирования и приведите пример с композицией.

Класс передаёт выполнение некоторой задачи др классу, вместо реализации логики самост.

Пример:

Класс Printer содержит объект PrintFormatter и делегирует ему форматирование текста перед печатью.

4. Какими способами можно реализовать композицию в C++?

- Прямое вкл. объекта как члена класса.
- Хранение умного указателя (std::unique_ptr) с созданием объекта в конструкторе.
- хранение вектора / массива объектов (в случае "has-many").

Кравенко Михаил ИВТ-4

3

5. Что такое исключение? Как генерируются и обр. исключ. в C++?

Исключение - механизм обр. ошибок, прерывающий нормальный поток выполнения.

Генерация: throw exception_object;

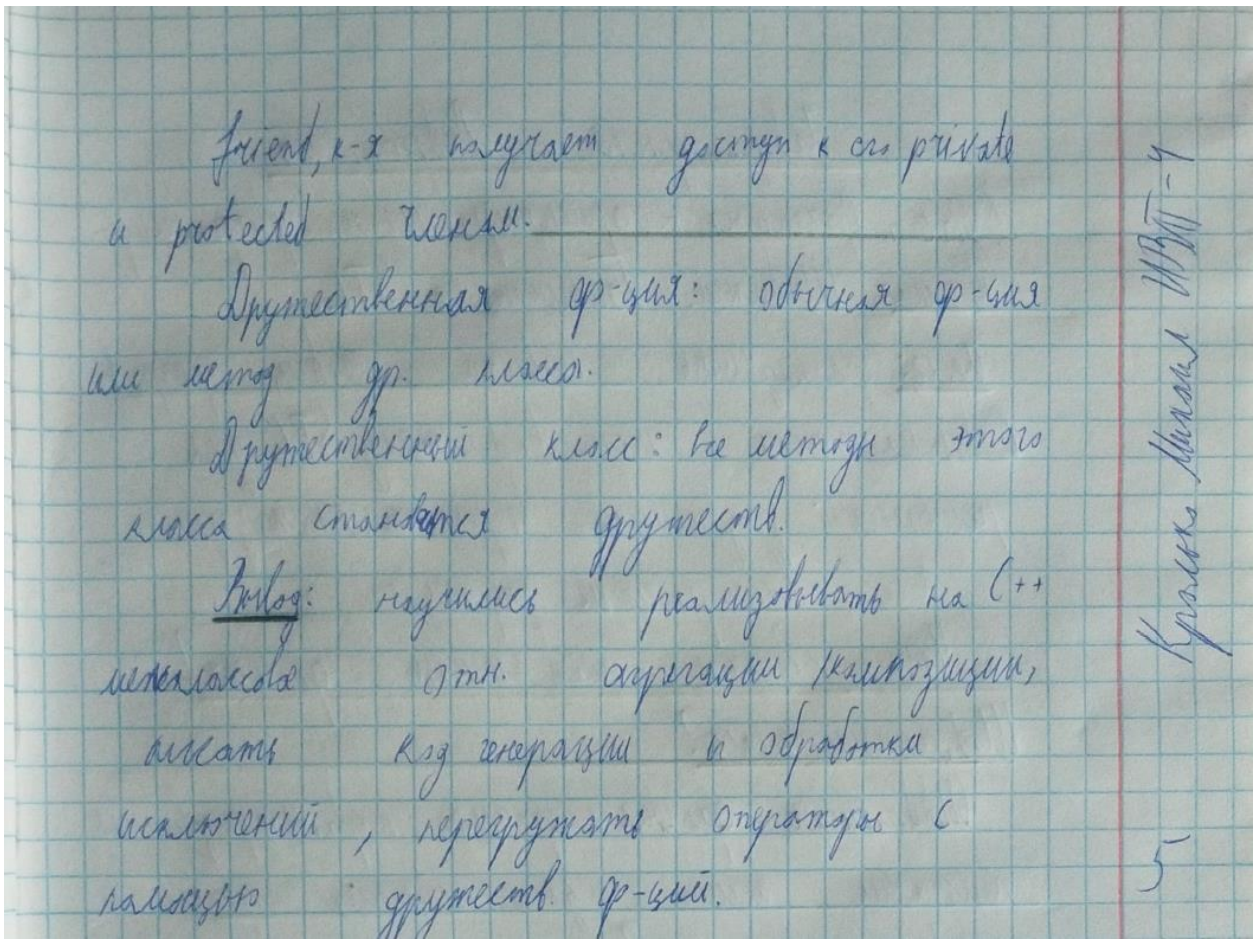
try { Код, в-й может выбросить исключ. };
catch (const ExceptionType & e) { обработка }
catch(...) { обработка любых исключ. }

6. Какие есть стандартные типы исключений в C++?

std:: exception (базовый)
std:: runtime_error, std:: logic_error
std:: invalid_argument, std:: out_of_range
std:: bad_alloc (нехватка памяти)

7. Что такое дружественный класс и дружественная ф-ция?

Дружеств. класс/ф-ция - внешняя сущн., объявленная в классе с ключевым словом



Вывод: научились реализовывать на C++ межклассовое отношение агрегации / композиции, писать код генерации и обработки исключений, перегружать операторы с помощью дружественных функций.