МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**ЛАБОРАТОРНАЯ РАБОТА № 3**

по дисциплине

**«Объектно-ориентированное программирование»**

на тему:

**«Динамический выбор типа объекта»**

Выполнил:

Студент группы

КТбо2-6

Калитин А.В.

Проверил:

доцент кафедры

МОП ЭВМ

Тарасов С. А.

Таганрог 2020

# **1 Цель работы**

Ознакомление с механизмом выбора типа создаваемых объектов во время выполнения программы.

# **2 Задание**

# Реализовать симуляции карточной игры для определенного количества игроков с определенной колодой. Реализовать характеры игроков.

# **3 Ход работы**

**3.1 Спецификации классов**

Диаграмма используемых классов приведена на Рисунке 1.

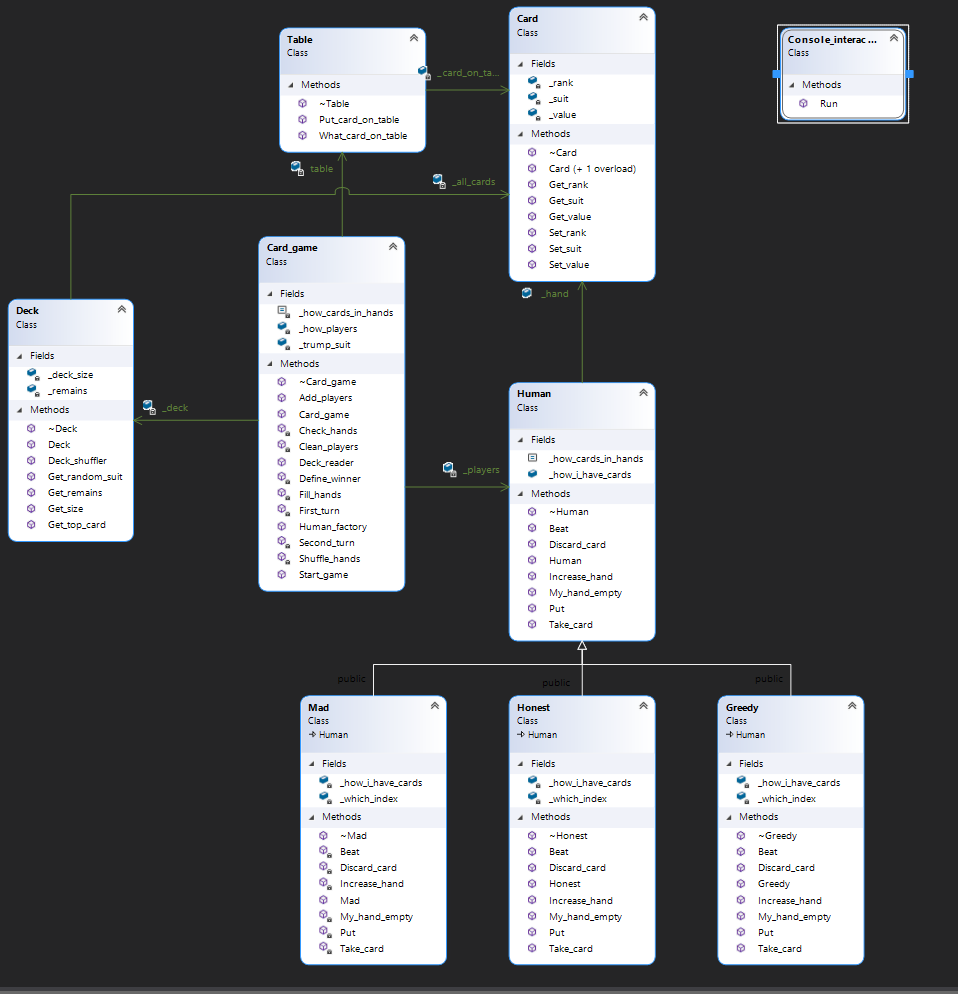


Рисунок 1 - Диаграмма классов

**3.2 Пояснение к коду**

# Колода может быть собственной, всего лишь нужно соблюсти требования для корректного считывания. Для изначального выбора имеется 3 колоды на 32, 52, 104 карт. Количество игроков контролируется количеством карт по умолчанию(4).

# **Характеры игроков**, которые имеются на данный момент:

# Безумный (бьёт, с 50% шансом)

# 2)Жадный(никогда не бьёт и не отдаёт козырную карту)

# 3)Честный(Всегда бьёт, если может)

# **Правила игры таковы:**

# -Игрок, у которого право хода, кладет на стол карту, а следующий за ним игрок **с непустой рукой** пытается найти бьющую карту у себя в руке.

# -Если игрок не имеет возможности ответить на выпад, то он забирает карту себе. Рука не ограничена в размерах.

# -Игрок берет из колоды карту, если на данный момент он имеет не более 3.

4. Листинг

Main.cpp

#define \_\_CRTDBG\_MAP\_ALLOC

#include <crtdbg.h>

#define DEBUG\_NEW new(\_NORMAL\_BLOCK, \_\_FILE\_\_, \_\_LINE\_\_)

#define new DEBUG\_NEW

#include "Console\_interactor.h"

int main()

{

\_CrtMemState \_ms;

\_CrtMemCheckpoint(&\_ms);

Console\_interactor console;

console.Run();

\_CrtMemDumpAllObjectsSince(&\_ms);

\_CrtSetReportMode(\_CRT\_WARN, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_WARN, \_CRTDBG\_FILE\_STDOUT);

\_CrtDumpMemoryLeaks();

}

Console\_interactor.h

#pragma once

#include <iostream>

#include "Card\_game.h"

#include <ctime>

#include <typeinfo>

class Console\_interactor

{

public:

void Run();

};

Console\_interactor.cpp

#include "Console\_interactor.h"

#define new new( \_NORMAL\_BLOCK, \_\_FILE\_\_, \_\_LINE\_\_)

void Console\_interactor::Run()

{

int command = 1;

Card\_game\* my\_game = new Card\_game;

std::cout << "Enter the number of cards in the deck(32 or 52 or 104):\n";

std::string path;

std::cin >> path;

path += ".txt";

try

{

int how;

my\_game->Deck\_reader(path);

std::cout << "How players?\n";

std::cin >> how;

int\* characters = new int[how];

std::cout << "1-mad\n2-honest\n3-greedy\nEnter characters through a space\n";

for (int i = 0; i < how; i++)

{

std::cin >> characters[i];

}

my\_game->Add\_players(how, characters);

delete[] characters;

}

catch (const std::exception& err)

{

std::cerr << "Error: " << err.what();

exit(1);

}

int how;

srand((unsigned int)time(NULL) / 2);

std::cout << "1 - Add players\n2 - Start game \n3 - Change deck\n4 - \n5 - \n0 - Exit.\n";

std::cout << "Enter a command: ";

while (command)

{

std::cout << "Enter a command: ";

std::cin >> command;

switch (command)

{

case 0:

{

delete my\_game;

break;

}

case 1:

{

try

{

std::cout << "How?\n";

std::cin >> how;

int\* characters = new int[how];

std::cout << "1-mad\n2-honest\n3-greedy\nEnter characters through a space\n";

for (int i = 0; i < how; i++)

{

std::cin >> characters[i];

}

my\_game->Add\_players(how, characters);

delete[] characters;

}

catch (std::exception& ex)

{

std::cout << ex.what();

exit(1);

}

}

break;

case 2:

{

// 32

// 2

// 1 3

// 2

try

{

srand((unsigned int)time(NULL) / 2);

Human\* winner = my\_game->Start\_game();

std::cout << "\nWinner: \n";

std::cout << typeid(\*winner).name() << "\n";

std::cout << "\n";

}

catch (std::exception& ex)

{

std::cout << ex.what();

}

break;

}

case 3:

{

std::cout << "Enter the number of cards in the deck(32 or 52 or 104):\n";

std::string path;

std::cin >> path;

path += ".txt";

try

{

my\_game->Deck\_reader(path);

}

catch (const std::exception& err)

{

std::cerr << "Error: " << err.what();

exit(1);

}

}

break;

case 4:

{

//52

//1

//4

//1 2 3 2

//4

break;

}

case 5:

break;

default:

std::cout << "Invalid command. Try again.\n";

break;

}

}

}

Card\_game.cpp

#include "Card\_game.h"

#include <sstream>

Card\_game::~Card\_game()

{

Clean\_players();

if(\_deck!=nullptr)

delete \_deck;

if(table!=nullptr)

delete table;

}

void Card\_game::Deck\_reader(const std::string& path)

{

\_deck = new Deck(path);

}

Human\* Card\_game::Human\_factory(int which)

{

Human\* player;

switch (which)

{

case 1:

player = new Mad;

break;

case 2:

player = new Honest;

break;

case 3:

player = new Greedy;

break;

default:

throw std::exception("player isnt create");

}

return player;

}

void Card\_game::Add\_players(int how, int\* characters)

{

if (how <= \_deck->Get\_size() / \_how\_cards\_in\_hands)

{

Clean\_players();

\_how\_players = how;

\_players = new Human \* [how];

for (int i = 0; i < how; i++)

{

\_players[i] = Human\_factory(characters[i]);

}

}

else

{

throw std::exception("Many players");

}

}

bool Card\_game::Check\_hands(Human\*\* \_players, int how) const

{

int answer = 0, res = 0;

for (int i = 0; i < how; i++)

{

for (int j = 0; j < \_players[i]->\_how\_i\_have\_cards; j++)

{

if (\_players[i]->\_hand[j] == nullptr)

{

res++;

}

}

if (res == \_players[i]->\_how\_i\_have\_cards)

{

answer++;

}

res = 0;

}

if (answer >= how - 1)

{

return false;

}

else

{

return true;

}

}

int Card\_game::First\_turn(Human\*\* \_players, int which, Table\* table, int how) const

{

Card\* temp = \_players[which]->Put(\_trump\_suit);

if (temp == nullptr)

{

return -1;

}

for (int i = 1; i < \_how\_players; i++)

{

if (!\_players[(which + i) % how]->My\_hand\_empty())

{

table->Put\_card\_on\_table(temp);

std::cout << "Player " << which << " put: " << \*table->What\_card\_on\_table() << "\n";

return i;

}

}

}

bool Card\_game::Second\_turn(Human\*\* \_players, int which, Table\* table, const std::string& trump\_suit)

{

return \_players[which]->Beat(trump\_suit, table->What\_card\_on\_table());

}

Human\* Card\_game::Define\_winner(Card\* card, Human\*\* \_players, int how\_players) const

{

int res = 0;

Human\* winner = nullptr;

if (card == nullptr)

{

for (int i = 0; i < how\_players; i++)

{

if (\_players[i]->My\_hand\_empty())

{

res++;

}

}

if (res >= how\_players-1)

{

for (int i = 0; i < how\_players; i++)

{

if (\_players[i]->My\_hand\_empty())

{

winner = \_players[i];

}

}

}

res = 0;

return winner;

}

}

Human\* Card\_game::Start\_game()

{

if (\_players != nullptr)

{

\_trump\_suit = \_deck->Get\_random\_suit();

for (int i = 0; i < 15; i++)

{

\_deck->Deck\_shuffler();

}

Fill\_hands(\_players);

table = new Table;

int result, tmp;

int index = 0;

Human\* winner;

winner = nullptr;

while (Check\_hands(\_players, \_how\_players))// пока у 3 из 4 есть карты => играем

{

Fill\_hands(\_players); //каждый раз заполняем 4 первых карты в руке

tmp = First\_turn(\_players, index % \_how\_players, table, \_how\_players);//кладет первую попавшиюся карту справа

if (tmp == -1)

{

index++;

continue;

}

index += tmp;

result = Second\_turn(\_players, (index) % \_how\_players, table, \_trump\_suit);

if (!result)

{

std::cout << "Player " << index % \_how\_players << " grab card\n";

\_players[(index) % \_how\_players]->Take\_card(table->What\_card\_on\_table());

\_players[(index - tmp) % \_how\_players]->Discard\_card();

index++;

}

else

{

\_players[(index - tmp) % \_how\_players]->Discard\_card();

}

winner = Define\_winner(\_deck->Get\_top\_card(), \_players, \_how\_players);

if (index % 5 == 0)

{

Shuffle\_hands(\_players, \_how\_players);

}

}

//1 ходит на 2

//если забирает то пропускает

//игра продолжается пока

//у 3 человек не кончатся карты

return winner;

}

}

void Card\_game::Shuffle\_hands(Human\*\* \_players, int how\_players) const

{

for (int i = 0; i < how\_players; i++)

{

int n = \_players[i]->\_how\_i\_have\_cards;

if (n < 2)

{

break;

}

while (n > 1)

{

int ind = rand() % n;

n--;

Card\* temp = \_players[i]->\_hand[n];

\_players[i]->\_hand[n] = \_players[i]->\_hand[ind];

\_players[i]->\_hand[ind] = temp;

}

}

}

void Card\_game::Fill\_hands(Human\*\* \_players) const

{

int count = 0;

for (int i = 0; i < \_how\_players; i++)

{

for (int j = 0; j < \_players[i]->\_how\_i\_have\_cards; j++)

{

if (\_players[i]->\_hand[j] != nullptr)

{

count++;

}

}

if (count < \_how\_cards\_in\_hands)

{

for (int j = 0; j < \_how\_cards\_in\_hands; j++)

{

if (\_players[i]->\_hand[j] == nullptr)

{

\_players[i]->\_hand[j] = \_deck->Get\_top\_card();

}

}

}

count = 0;

}

}

void Card\_game::Clean\_players()

{

if (\_players != nullptr)

{

for (int i = 0; i < \_how\_players; i++)

{

if (\_players[i] != nullptr)

{

delete[] \_players[i];

}

}

delete[] \_players;

}

}

Card.cpp

#include "Card.h"

Card::Card(const std::string& suit, const std::string& rank, int value) :\_suit(suit), \_rank(rank), \_value(value)

{}

int Card::Get\_value() const

{

return \_value;

}

void Card::Set\_value(int value)

{

\_value = value;

}

std::string Card::Get\_suit() const

{

return \_suit;

}

std::string Card::Get\_rank() const

{

return \_rank;

}

void Card::Set\_suit(const std::string& suit)

{

\_suit = suit;

}

void Card::Set\_rank(const std::string& rank)

{

\_rank = rank;

}

std::ostream& operator<<(std::ostream& out, const Card& card)

{

out << card.Get\_rank() << " " << card.Get\_suit();

return out;

}

Deck.cpp

#include "Deck.h"

#define new new( \_NORMAL\_BLOCK, \_\_FILE\_\_, \_\_LINE\_\_)

Deck::Deck(const std::string& path)

{

std::ifstream file;

file.open(path);

std::string suit, rank;

int value;

if (file.is\_open())

{

file >> \_deck\_size;

\_remains = \_deck\_size - 1;

\_all\_cards = new Card[\_deck\_size];

for (int i = 0; i < \_deck\_size; i++)

{

file >> rank >> suit >> value;

\_all\_cards[i] = Card(suit, rank, value);

}

file.close();

}

else

{

file.close();

throw std::exception("File not found.");

}

}

Deck::~Deck()

{

if (\_all\_cards != nullptr)

{

delete[] \_all\_cards;

}

\_all\_cards = nullptr;

}

void Deck::Deck\_shuffler() const

{

int n = \_deck\_size;

while (n > 1)

{

int ind = rand() % n;

n--;

Card temp = \_all\_cards[n];

\_all\_cards[n] = \_all\_cards[ind];

\_all\_cards[ind] = temp;

}

}

Card\* Deck::Get\_top\_card()

{

if(\_remains>=0)

return &\_all\_cards[\_remains--];

else

return nullptr;

}

const std::string Deck::Get\_random\_suit()

{

return \_all\_cards[rand() % \_deck\_size].Get\_suit();

}

std::ostream& operator<<(std::ostream& out, const Deck& deck)

{

out << "Deck have " << deck.\_deck\_size << " cards";

return out;

}

Greedy.cpp

#include "Greedy.h"

Greedy::Greedy()

{

\_hand = new Card \* [\_how\_i\_have\_cards];

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

\_hand[i] = nullptr;

}

}

Greedy::~Greedy()

{

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

\_hand[i] = nullptr;

}

}

bool Greedy::My\_hand\_empty()

{

bool flag = true;

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

if (\_hand[i] != nullptr)

{

flag = false;

}

}

return flag;

}

bool Greedy::Beat(const std::string& trump\_suit, Card\* card)

{

bool can = false;

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

if (\_hand[i] != nullptr)

{

if (\_hand[i]->Get\_suit() == card->Get\_suit() && \_hand[i]->Get\_value() > card->Get\_value())

{

if (card->Get\_suit() == trump\_suit)

{

can = false;

}

else

{

can = true;

std::cout << "Next player beat it with: " << \*\_hand[i] << "\n";

\_hand[i] = nullptr;

break;

}

}

}

}

return can;

}

void Greedy::Discard\_card()

{

\_hand[\_which\_index] = nullptr;

}

void Greedy::Increase\_hand(int how)

{

Card\*\* temp = new Card \* [\_how\_cards\_in\_hands + how];

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

temp[i] = \_hand[i];

\_hand[i] = nullptr;

}

for (int i = 0; i < how; i++)

{

temp[\_how\_i\_have\_cards + i] = nullptr;

}

\_hand = temp;

\_how\_i\_have\_cards += how;

}

void Greedy::Take\_card(Card\* card)

{

bool flag = false;

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

if (\_hand[i] == nullptr)

{

\_hand[i] = card;

flag = true;

break;

}

}

if (!flag)

{

Increase\_hand(1);

\_hand[\_how\_i\_have\_cards - 1] = card;

}

}

Card\* Greedy::Put(const std::string& trump\_suit)

{

for (int i = \_how\_i\_have\_cards - 1; i > -1; i--)

{

if (\_hand[i] != nullptr && \_hand[i]->Get\_suit() != trump\_suit)

{

\_which\_index = i;

return \_hand[i];

}

}

return nullptr;

}

Mad.cpp

#include "Mad.h"

#include <ctime>

Mad::Mad()

{

\_hand = new Card \* [\_how\_i\_have\_cards];

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

\_hand[i] = nullptr;

}

}

Mad::~Mad()

{

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

\_hand[i] = nullptr;

}

}

bool Mad::Beat(const std::string& trump\_suit, Card\* card)

{

bool can = false;

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

if (\_hand[i] != nullptr)

{

if (\_hand[i]->Get\_suit() == card->Get\_suit() && \_hand[i]->Get\_value() >= card->Get\_value())

{

if (rand() % 2)

{

can = true;

std::cout << "Next player beat it with: " << \*\_hand[i]<<"\n";

\_hand[i] = nullptr;

break;

}

}

}

}

return can;

}

Card\* Mad::Put(const std::string& trump\_suit)

{

for (int i = \_how\_i\_have\_cards-1; i > -1; i--)

{

if (\_hand[i] != nullptr)

{

\_which\_index = i;

return \_hand[i];

}

}

return nullptr;

}

void Mad::Discard\_card()

{

\_hand[\_which\_index] = nullptr;

}

bool Mad::My\_hand\_empty()

{

bool flag = true;

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

if (\_hand[i] != nullptr)

{

flag = false;

}

}

return flag;

}

void Mad::Increase\_hand(int how)

{

Card\*\* temp = new Card \* [\_how\_cards\_in\_hands + how];

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

temp[i] = \_hand[i];

\_hand[i] = nullptr;

}

for (int i = 0; i < how; i++)

{

temp[\_how\_i\_have\_cards + i] = nullptr;

}

\_hand = temp;

\_how\_i\_have\_cards += how;

}

void Mad::Take\_card(Card\* card)

{

bool flag = false;

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

if (\_hand[i] == nullptr)

{

\_hand[i] = card;

flag = true;

break;

}

}

if (!flag)

{

Increase\_hand(1);

\_hand[\_how\_i\_have\_cards-1] = card;

}

}

Honest.cpp

#include "Honest.h"

#include <ctime>

Honest::Honest()

{

\_hand = new Card \* [\_how\_i\_have\_cards];

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

\_hand[i] = nullptr;

}

}

Honest::~Honest()

{

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

\_hand[i] = nullptr;

}

}

bool Honest::My\_hand\_empty()

{

bool flag = true;

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

if (\_hand[i] != nullptr)

{

flag = false;

}

}

return flag;

}

void Honest::Discard\_card()

{

\_hand[\_which\_index] = nullptr;

}

bool Honest::Beat(const std::string& trump\_suit, Card\* card)

{

bool can = false;

for (int i = \_how\_i\_have\_cards - 1; i > -1; i--)

{

if (\_hand[i] != nullptr)

{

if (\_hand[i]->Get\_suit() == card->Get\_suit() && \_hand[i]->Get\_value() >= card->Get\_value())

{

if (\_hand[i]->Get\_suit() == card->Get\_suit() && \_hand[i]->Get\_value() >= card->Get\_value())

{

can = true;

std::cout << "Next player beat it with: " << \*\_hand[i] << "\n";

\_hand[i] = nullptr;

break;

}

}

}

}

return can;

}

Card\* Honest::Put(const std::string& trump\_suit)

{

for (int i = \_how\_i\_have\_cards - 1; i > -1; i--)

{

if (\_hand[i] != nullptr)

{

\_which\_index = i;

return \_hand[i];

}

}

return nullptr;

}

void Honest::Increase\_hand(int how)

{

Card\*\* temp = new Card \* [\_how\_cards\_in\_hands + how];

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

temp[i] = \_hand[i];

\_hand[i] = nullptr;

}

for (int i = 0; i < how; i++)

{

temp[\_how\_i\_have\_cards + i] = nullptr;

}

\_hand = temp;

\_how\_i\_have\_cards += how;

}

void Honest::Take\_card(Card\* card)

{

bool flag = false;

for (int i = 0; i < \_how\_i\_have\_cards; i++)

{

if (\_hand[i] == nullptr)

{

\_hand[i] = card;

flag = true;

break;

}

}

if (!flag)

{

Increase\_hand(1);

\_hand[\_how\_i\_have\_cards - 1] = card;

}

}

Table.cpp

#include "Table.h"

Table::~Table()

{

if (\_card\_on\_table != nullptr)

{

\_card\_on\_table = nullptr;

}

}

Card\* Table::What\_card\_on\_table()

{

return \_card\_on\_table;

}

void Table::Put\_card\_on\_table(Card\* card)

{

\_card\_on\_table = card;

}