Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

(ФГАОУ ВО «ЮФУ»)

Институт компьютерных технологий и информационной безопасности

Кафедра вычислительной техники

ОТЧЁТ

о выполнении лабораторной работы № 2

по дисциплине «Объектно-ориентированное программирование»

Выполнил

студент группы КТбо2-1 Д. В. Манькус

Принял

преподаватель

ИКТИБ И. А. Кажарова

Таганрог 2023

Оглавление

[Техническое задание 3](#_Toc148642897)

[Выполнение работы 4](#_Toc148642898)

[Исходный код программы 5](#_Toc148642899)

[Примеры работы 16](#_Toc148642900)

[Вывод 19](#_Toc148642901)

# Техническое задание

**Задание**

Спроектировать и реализовать иерархию классов для игры в шахматы.

В качестве варианта использования рассмотреть задачу анализа ситуации на поле. В задаче участвует несколько черных фигур (любого вида) и одна белая – король. Провести анализ положения белого короля в условии хода белых фигур - шах, пат, мат или белому королю ничего не угрожает.

В иерархии классов реализовать:

* проверку попадания фигуры на доску;
* проверку установки фигуры на свободную клетку;
* отслеживание количества экземпляров каждого вида фигур.

Разработанная иерархия должна допускать использование:

int main()

{

TKing bKing(black,’h’,5);

THorse bHorse1(black,’c’,2);

TKing bKing2(black,’a’,1); // эта строчка работать не должна, потому что двух

// одинаковых королей быть не может

THorse bHorse2(black,’с’,2); // эта строчка работать не должна, потому что клетка

// с2 уже занята

THorse bHorse3(black,’z’,20); // эта строчка работать не должна из-за

// неправильных координат

TFigure \*figures[32];

figures[0]=&bKing;

figures[1]=&bHorse;

/\*for (int i = 0; i < 32; i++)

figures[i]->mapStep();\*/

TKing wKing(white,’a’,4);

cout << endl << wKing.Situation();

return 0;

}

В тестовой программе информацию о черных фигурах считывать из файла, координаты белого короля получать с клавиатуры.

Для удобства проверки правильности работы программы реализовать метод вывода доски на экран.

# Выполнение работы

Изображение выглядит как текст, снимок экрана, Параллельный, число

Автоматически созданное описание

1 - UML-диаграмма классов программы

# Исходный код программы

**Chess.h:**

#pragma once

//#define DEBUG

// Лабораторная работа №2

enum FIGURECOLOR {

white, black

};

enum FIGURETYPE {

empty = 0, pawn, horse, bishop, king, rook, queen

};

struct figurePosition {

char xPos;

char yPos;

};

class TFigure {

protected:

figurePosition position;

public:

TFigure(FIGURECOLOR color, char letter, char number);

void show();

virtual void mapStep() {}

virtual void Situation() {}

virtual ~TFigure();

};

class TBoard {

private:

static TBoard\* instancePtr;

figurePosition figuresMap[128];

char mapPtr = 0;

bool mapAct = true;

TBoard() {}

public:

FIGURETYPE chessboard[8][8] = {empty, empty, empty, empty, empty, empty, empty, empty,

empty, empty, empty, empty, empty, empty, empty, empty,

empty, empty, empty, empty, empty, empty, empty, empty,

empty, empty, empty, empty, empty, empty, empty, empty,

empty, empty, empty, empty, empty, empty, empty, empty,

empty, empty, empty, empty, empty, empty, empty, empty,

empty, empty, empty, empty, empty, empty, empty, empty,

empty, empty, empty, empty, empty, empty, empty, empty};

char bking = 0;

char wking = 0;

char bqueen = 0;

char bbishop = 0;

char bhorse = 0;

char brook = 0;

char bpawn = 0;

TBoard(const TBoard& obj) = delete;

static TBoard\* getInstance();

void show();

bool add(int x, int y);

bool checkCell(int x, int y);

bool checkHit(int x, int y);

void reloadMap();

bool set(FIGURETYPE type, FIGURECOLOR color, char letter, char number);

};

class TKing : public TFigure {

public:

TKing(FIGURECOLOR color, char letter, char number);

void mapStep();

void Situation();

~TKing();

};

class TQueen : public TFigure {

public:

TQueen(FIGURECOLOR color, char letter, char number);

void mapStep();

void Situation();

~TQueen();

};

class TPawn : public TFigure {

public:

TPawn(FIGURECOLOR color, char letter, char number);

void mapStep();

void Situation();

~TPawn();

};

class THorse : public TFigure {

public:

THorse(FIGURECOLOR color, char letter, char number);

void mapStep();

void Situation();

~THorse();

};

class TBishop : public TFigure {

public:

TBishop(FIGURECOLOR color, char letter, char number);

void mapStep();

void Situation();

~TBishop();

};

class TRook : public TFigure {

public:

TRook(FIGURECOLOR color, char letter, char number);

void mapStep();

void Situation();

~TRook();

};

**Chess.cpp:**

#include <iostream>

#include "Chess.h"

TBoard\* TBoard::instancePtr = nullptr;

TBoard\* TBoard::getInstance() {

if (instancePtr == nullptr)

{

instancePtr = new TBoard();

return instancePtr;

}

else

{

return instancePtr;

}

}

bool TBoard::set(FIGURETYPE type, FIGURECOLOR color, char letter, char number) {

int xpos = letter - 97;

int ypos = number - 1;

if (xpos > 7 || xpos < 0) return false;

if (ypos > 7 || ypos < 0) return false;

if (chessboard[ypos][xpos] != empty) return false;

if (type == king) {

if (color == white) {

if (wking > 0) return false;

else wking++;

}

if (color == black) {

if (bking > 0) return false;

else bking++;

}

}

if (type == queen) {

if (color == black && bqueen > 0) return false;

else bqueen++;

}

if (type == bishop) {

if (color == black && bbishop > 1) return false;

else bbishop++;

}

if (type == horse) {

if (color == black && bhorse > 1) return false;

else bhorse++;

}

if (type == rook) {

if (color == black && brook > 1) return false;

else brook++;

}

if (type == pawn) {

if (color == black && bpawn > 7) return false;

else bpawn++;

}

chessboard[ypos][xpos] = type;

return true;

};

void TBoard::show() {

char names[8] = { ".eNBKRQ" };

std::cout << "abcdefgh" << std::endl;

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 8; j++) {

std::cout << names[chessboard[i][j]];

}

std::cout << i+1;

std::cout << std::endl;

}

}

bool TBoard::add(int x, int y) {

figurePosition pos = { x, y };

if (mapAct) {

for (int i = 0; i < 128; i++) {

figuresMap[i].xPos = 0;

figuresMap[i].yPos = 0;

}

mapPtr = 0;

mapAct = false;

}

if (pos.xPos < 0 || pos.xPos > 7) return false;

if (pos.yPos < 0 || pos.yPos > 7) return false;

figuresMap[mapPtr] = pos;

mapPtr++;

if (mapPtr > 127) mapAct = true;

return true;

}

bool TBoard::checkCell(int x, int y) {

if (x < 0 || x > 7) return false;

if (y < 0 || y > 7) return false;

if (chessboard[y][x] == empty) return true;

return false;

}

bool TBoard::checkHit(int x, int y) {

if (x < 0 || x > 7) return true;

if (y < 0 || y > 7) return true;

for (int i = 0; i < mapPtr; i++) {

if (figuresMap[i].xPos == x && figuresMap[i].yPos == y) return true;

}

return false;

}

void TBoard::reloadMap() {

mapAct = true;

}

TFigure::TFigure(FIGURECOLOR color, char letter, char number) {

#ifdef DEBUG

std::cout << "constructorTFigure" << std::endl;

#endif

}

TFigure::~TFigure() {

#ifdef DEBUG

std::cout << "destructorTFigure" << std::endl;

#endif

}

TKing::TKing(FIGURECOLOR color, char letter, char number):TFigure(color, letter, number) {

#ifdef DEBUG

std::cout << "constructorTKing" << std::endl;

#endif

TBoard\* p = TBoard::getInstance();

if (!p->set(king, color, letter, number)) throw std::logic\_error("incorrect placement");

position.xPos = letter - 97;

position.yPos = number - 1;

}

void TKing::mapStep() {

TBoard\* p = TBoard::getInstance();

p->add(position.xPos + 1, position.yPos);

p->add(position.xPos + 1, position.yPos + 1);

p->add(position.xPos, position.yPos + 1);

p->add(position.xPos - 1, position.yPos + 1);

p->add(position.xPos - 1, position.yPos);

p->add(position.xPos - 1, position.yPos - 1);

p->add(position.xPos, position.yPos - 1);

p->add(position.xPos + 1, position.yPos - 1);

}

void TKing::Situation() {

TBoard\* p = TBoard::getInstance();

p->reloadMap();

bool isMove = !p->checkHit(position.xPos + 1, position.yPos) ||

!p->checkHit(position.xPos + 1, position.yPos + 1) ||

!p->checkHit(position.xPos, position.yPos + 1) ||

!p->checkHit(position.xPos - 1, position.yPos + 1) ||

!p->checkHit(position.xPos - 1, position.yPos) ||

!p->checkHit(position.xPos - 1, position.yPos - 1) ||

!p->checkHit(position.xPos, position.yPos - 1)||

!p->checkHit(position.xPos + 1, position.yPos - 1);

bool isHit = p->checkHit(position.xPos, position.yPos);

if (!isMove && isHit) { std::cout << "Checkmate!" << std::endl; return; }

if (isMove && isHit) { std::cout << "Check!" << std::endl; return; }

if (!isMove && !isHit) { std::cout << "Stalemate!" << std::endl; return; }

std::cout << "The king is fine!" << std::endl;

}

void TFigure::show() {

TBoard\* p = TBoard::getInstance();

p->show();

}

TKing::~TKing() {

#ifdef DEBUG

std::cout << "destructorTKing" << std::endl;

#endif

}

TPawn::TPawn(FIGURECOLOR color, char letter, char number) :TFigure(color, letter, number) {

#ifdef DEBUG

std::cout << "constructorTPawn" << std::endl;

#endif

TBoard\* p = TBoard::getInstance();

if (!p->set(pawn, color, letter, number)) throw std::logic\_error("incorrect placement");

position.xPos = letter - 97;

position.yPos = number - 1;

}

void TPawn::mapStep() {

TBoard\* p = TBoard::getInstance();

p->add(position.xPos + 1, position.yPos - 1);

p->add(position.xPos - 1, position.yPos - 1);

}

void TPawn::Situation() {}

TQueen::TQueen(FIGURECOLOR color, char letter, char number) :TFigure(color, letter, number) {

#ifdef DEBUG

std::cout << "constructorTQueen" << std::endl;

#endif

TBoard\* p = TBoard::getInstance();

if (!p->set(queen, color, letter, number)) throw std::logic\_error("incorrect placement");

position.xPos = letter - 97;

position.yPos = number - 1;

}

void TQueen::mapStep() {

TBoard\* p = TBoard::getInstance();

int i = 0;

do {

i++;

if (position.xPos + i > 7) break;

if (position.yPos + i > 7) break;

p->add(position.xPos + i, position.yPos + i);

} while (p->checkCell(position.xPos + i, position.yPos + i));

i = 0;

do {

i++;

if (position.xPos - i < 0) break;

if (position.yPos - i < 0) break;

p->add(position.xPos - i, position.yPos - i);

} while (p->checkCell(position.xPos - i, position.yPos - i));

i = 0;

do {

i++;

if (position.xPos - i < 0) break;

if (position.yPos + i > 7) break;

p->add(position.xPos - i, position.yPos + i);

} while (p->checkCell(position.xPos - i, position.yPos + i));

i = 0;

do {

i++;

if (position.xPos + i > 7) break;

if (position.yPos - i < 0) break;

p->add(position.xPos + i, position.yPos - i);

} while (p->checkCell(position.xPos + i, position.yPos - i));

i = 0;

do {

i++;

if (position.xPos + i > 7) break;

p->add(position.xPos + i, position.yPos);

} while (p->checkCell(position.xPos + i, position.yPos));

i = 0;

do {

i++;

if (position.xPos - i < 0) break;

p->add(position.xPos - i, position.yPos);

} while (p->checkCell(position.xPos - i, position.yPos));

i = 0;

do {

i++;

if (position.yPos + i > 7) break;

p->add(position.xPos, position.yPos + i);

} while (p->checkCell(position.xPos, position.yPos + i));

i = 0;

do {

i++;

if (position.yPos - i < 0) break;

p->add(position.xPos, position.yPos - i);

} while (p->checkCell(position.xPos, position.yPos - i));

}

void TQueen::Situation() {}

TQueen::~TQueen() {

#ifdef DEBUG

std::cout << "destructorTQueen" << std::endl;

#endif

}

TPawn::~TPawn() {

#ifdef DEBUG

std::cout << "destructorTPawn" << std::endl;

#endif

}

THorse::THorse(FIGURECOLOR color, char letter, char number) :TFigure(color, letter, number) {

#ifdef DEBUG

std::cout << "constructorTHorse" << std::endl;

#endif

TBoard\* p = TBoard::getInstance();

if (!p->set(horse, color, letter, number)) throw std::logic\_error("incorrect placement");

position.xPos = letter - 97;

position.yPos = number - 1;

}

void THorse::mapStep() {

TBoard\* p = TBoard::getInstance();

p->add(position.xPos + 1, position.yPos + 2);

p->add(position.xPos - 1, position.yPos + 2);

p->add(position.xPos + 2, position.yPos + 1);

p->add(position.xPos - 2, position.yPos + 1);

p->add(position.xPos + 1, position.yPos - 2);

p->add(position.xPos - 1, position.yPos - 2);

p->add(position.xPos + 2, position.yPos - 1);

p->add(position.xPos - 2, position.yPos - 1);

}

void THorse::Situation() {}

THorse::~THorse() {

#ifdef DEBUG

std::cout << "destructorTHorse" << std::endl;

#endif

}

TBishop::TBishop(FIGURECOLOR color, char letter, char number) :TFigure(color, letter, number) {

#ifdef DEBUG

std::cout << "constructorTBishop" << std::endl;

#endif

TBoard\* p = TBoard::getInstance();

if (!p->set(bishop, color, letter, number)) throw std::logic\_error("incorrect placement");

position.xPos = letter - 97;

position.yPos = number - 1;

}

void TBishop::mapStep() {

TBoard\* p = TBoard::getInstance();

int i = 0;

do {

i++;

if (position.xPos + i > 7) break;

if (position.yPos + i > 7) break;

p->add(position.xPos + i, position.yPos + i);

} while (p->checkCell(position.xPos + i, position.yPos + i));

i = 0;

do {

i++;

if (position.xPos - i < 0) break;

if (position.yPos - i < 0) break;

p->add(position.xPos - i, position.yPos - i);

} while (p->checkCell(position.xPos - i, position.yPos - i));

i = 0;

do {

i++;

if (position.xPos - i < 0) break;

if (position.yPos + i > 7) break;

p->add(position.xPos - i, position.yPos + i);

} while (p->checkCell(position.xPos - i, position.yPos + i));

i = 0;

do {

i++;

if (position.xPos + i > 7) break;

if (position.yPos - i < 0) break;

p->add(position.xPos + i, position.yPos - i);

} while (p->checkCell(position.xPos + i, position.yPos - i));

}

void TBishop::Situation() {}

TBishop::~TBishop() {

#ifdef DEBUG

std::cout << "destructorTBishop" << std::endl;

#endif

}

TRook::TRook(FIGURECOLOR color, char letter, char number) :TFigure(color, letter, number) {

#ifdef DEBUG

std::cout << "constructorTRook" << std::endl;

#endif

TBoard\* p = TBoard::getInstance();

if (!p->set(rook, color, letter, number)) throw std::logic\_error("incorrect placement");

position.xPos = letter - 97;

position.yPos = number - 1;

}

void TRook::mapStep() {

TBoard\* p = TBoard::getInstance();

int i = 0;

do {

i++;

if (position.xPos + i > 7) break;

p->add(position.xPos + i, position.yPos);

} while (p->checkCell(position.xPos + i, position.yPos));

i = 0;

do {

i++;

if (position.xPos - i < 0) break;

p->add(position.xPos - i, position.yPos);

} while (p->checkCell(position.xPos - i, position.yPos));

i = 0;

do {

i++;

if (position.yPos + i > 7) break;

p->add(position.xPos, position.yPos + i);

} while (p->checkCell(position.xPos, position.yPos + i));

i = 0;

do {

i++;

if (position.yPos - i < 0) break;

p->add(position.xPos, position.yPos - i);

} while (p->checkCell(position.xPos, position.yPos - i));

}

void TRook::Situation() {}

TRook::~TRook() {

#ifdef DEBUG

std::cout << "destructorTRook" << std::endl;

#endif

}

**ConsoleApplication2.cpp:**

#include <iostream>

#include <stdexcept>

#include "Chess.h"

int main()

{

FILE\* inputFile;

fopen\_s(&inputFile, "input.txt", "r");

if (inputFile == NULL) {

printf("Не удалось открыть файл input.txt\n");

return 1;

}

char note[5];

int i = 0;

TFigure\* figures[32];

while (fgets(note, sizeof(note), inputFile) != NULL) {

note[3] = '\0';

try {

switch (note[0]) {

case 'e':

figures[i] = new TPawn(black, note[1], note[2] - 48);

#ifdef DEBUG

std::cout << "pawn" << std::endl;

#endif

break;

case 'R':

figures[i] = new TRook(black, note[1], note[2] - 48);

#ifdef DEBUG

std::cout << "rook" << std::endl;

#endif

break;

case 'N':

figures[i] = new THorse(black, note[1], note[2] - 48);

#ifdef DEBUG

std::cout << "knight" << std::endl;

#endif

break;

case 'B':

figures[i] = new TBishop(black, note[1], note[2] - 48);

#ifdef DEBUG

std::cout << "bishop" << std::endl;

#endif

break;

case 'Q':

figures[i] = new TQueen(black, note[1], note[2] - 48);

#ifdef DEBUG

std::cout << "queen" << std::endl;

#endif

break;

case 'K':

figures[i] = new TKing(black, note[1], note[2] - 48);

#ifdef DEBUG

std::cout << "king" << std::endl;

#endif

break;

default:

std::cout << "Figure doesn't exist" << std::endl;

i--;

}

}

catch (const std::exception& e) {

std::cout << "line " << note << " error: ";

std::cout << e.what() << std::endl;

i--;

}

i++;

}

fclose(inputFile);

//setlocale(LC\_ALL, "Russian");

//std::cout.imbue(std::locale(""));

//std::cout.sync\_with\_stdio(false);

//TKing bKing(black, 'h', 5);

//THorse bHorse1(black, 'c', 5);

//TPawn bPawn1(black, 'c', 3);

//TRook bRook1(black, 'h', 2);

//TKing bKing2(black, 'a', 1); // эта строчка работать не должна, потому что двух

// одинаковых королей быть не может

//THorse bHorse2(black, ’с’, 2); // эта строчка работать не должна, потому что клетка

// с2 уже занята

//THorse bHorse3(black, ’z’, 20); // эта строчка работать не должна из-за

// неправильных координат

/\*

figures[0] = &bKing;

figures[1] = &bHorse1;

figures[2] = &bPawn1;

figures[3] = &bRook1;

\*/

for (int j = 0; j < i; j++) figures[j]->mapStep();

std::cout << "e - pawn\nR - rook\nN - knight\nB - bishop\nQ - queen\nK - king" << std::endl;

figures[0]->show();

std::cout << "Enter the position of the white king (leter and a number): ";

fgets(note, sizeof(note), stdin);

note[2] = '\0';

try {

TKing wKing(white, note[0], note[1] - 48);

wKing.show();

wKing.Situation();

}

catch (const std::exception& e) {

std::cout << "line " << note << " error: ";

std::cout << e.what() << std::endl;

}

return 0;

}

# Примеры работы

В файле «input.txt» указываеются позиции черных фигур на доске в формате код фигуры, позиция по горизонтали, позиция по вертикали, без пробелов. Коды фигур:

* K – Король
* Q – Ферзь
* B – Слон
* N – Конь
* R – Ладья
* e – Пешка

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

1 - Король в безопасности

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

2 - Патовая ситуация

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

3 - Шаховая ситуация

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

4 - Шах и мат

# Вывод

Объктно ориентированное программирование структурирует программу, облегчает создание разного поведения у похожих сущностей при этом объединяя их общие свойства и позволяя не дублировать код. Использование полиморфизма позволяет хранить разные объекты в одной и той же структуре данных и применять к ним общие методы.