МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №7

по дисциплине «Объектно-ориентированное программирование»

Тема: Написание исключений

Студент гр. 8303 _____ Удод М.Н.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Разработать и реализовать набор исключений. Основные требования к исключениям:

- Исключения покрывают как минимум все тривиальные случаи возникновения ошибки
- Все реализованные исключения обрабатываются в программе
- Исключения должны хранить подробную информацию об ошибке, а не только строку с сообщением об ошибке

Ход выполнения работы.

1. Были реализованы классы исключений, наследующийся от std::exception. Они покрывают все ошибки, возникающие в классах GameField, GameInfo и LoadCommand.
2. Были написаны обработчики для каждого исключения при выполнении команд.

Вывод.

В ходе выполнения лабораторной работы были изучено написание и обработка исключений путем модификации программы из предыдущих лабораторных.

.

Приложение А. Исходный код программы

1. DoublePlacingOnFieldException.h

```
class DoublePlacingOnFieldException: std::exception {

};
```

2. DoubleBasePlacingException.h

```
class DoubleBasePlacingException: std::exception {

public:

    int playerIndex;
    explicit DoubleBasePlacingException(int playerIndex):
playerIndex(playerIndex){}

};
```

3. GameFieldOutOfRangeException.h

```
class GameFieldOutOfRangeException: std::exception {

public:

    int x;
    int y;

    GameFieldOutOfRangeException(int x, int y): x(x), y(y){}

};
```

4. ImpossibleMoveException.h

```
class ImpossibleMoveException: std::exception {

};
```

5. InvalidFileLoaddingException.h

```
class InvalidFileLoaddingException: std::exception {

};
```

6. GameFacade.h

```
#include <sstream>
#include "GameInfo.h"
#include "UI/MainCommandInterpreter.h"
#include "Exceptions/DoublePlacingOnFieldException.h"
#include "Exceptions/GameFieldOutOfRangeException.h"
#include "Exceptions/ImpossibleMoveException.h"

template<typename Rule, int playersCount>
class GameFacade: public GameInfo {

private:

    MainCommandInterpreter interpreter;
    GameFacade(int fieldWidth, int fieldHeight): GameInfo(playersCount,
fieldWidth, fieldWidth, new Rule){}
```

```
    Rule rule;

public:

    static GameFacade& instance(){
        Rule rule;
        static GameFacade singleInstance(rule.fieldWidth, rule.fieldHeight);
        return singleInstance;
    }

    void nextTurn(){

        std::string commandString;
        std::getline(std::cin, commandString);

        CommandPtr command = interpreter.handle(commandString);
        try {
            command->execute(*this);
        } catch(DoubleBasePlacingException &exception) {
            std::cout << "Player " << exception.playerIndex << " trying to
place base second time." << std::endl;
        } catch (DoublePlacingOnFieldException &exception){
            std::cout << "This cell is busy by other object." << std::endl;
        } catch (GameFieldOutOfRangeException &exception){
            std::cout << "Out of range. Cell " << exception.x << " " <<
exception.y << " is not exist." << std::endl;
        } catch (ImpossibleMoveException &exception){
            std::cout << "Can't move to this cell. They busy by other object."
<< std::endl;
        } catch (InvalidFileLoaddingException &exception){
            std::cout << "Wrong file." << std::endl;
        } catch (...){
            std::cout << "Undefined error." << std::endl;
        }
        history.push_back(command->getMemento());

        nextUser();

    }

    friend std::ostream &operator<<(std::ostream &stream, const GameFacade
&game){

        stream << "Now player: " << game.nowPlayerIndex << std::endl;
        stream << game.gameField << std::endl;
        return stream;

    }

    bool isOver(){

        return rule.isOver(*this);

    }

};
```

7. GameInfo.h

```
#include "UI/Commands/CommandMemento.h"
#include "GameField/GameField.h"
#include "Rules/GameRule.h"
#include "Exceptions/DoubleBasePlacingException.h"

class GameInfo {

protected:

    GameField gameField;
    std::vector<Base*> playersBases;
    int nowPlayerIndex;
    std::vector<CommandMemento*> history;
    GameRule *rule;

public:

    GameInfo(int playersCount, int fieldWidth, int fieldHeight, GameRule
*rule):
    gameField(fieldHeight, fieldWidth),
    playersBases(playersCount, nullptr),
    nowPlayerIndex(0),
    rule(rule)
    {}

    Base *getNowPlayerBase(){ return playersBases[nowPlayerIndex]; }
    bool setNowPlayerBase(Base *base){

        if (playersBases[nowPlayerIndex]){
            throw DoubleBasePlacingException(nowPlayerIndex);
        } else{

            playersBases[nowPlayerIndex] = base;
            return true;

        }

    }
    int getNowPlayerIndex(){ return nowPlayerIndex; }

    void newGame(){

        int playersCount = playersBases.size();

        gameField.reset();
        playersBases.clear();
        history.clear();
        playersBases.resize(playersCount, nullptr);

    }
    void nextUser(){
        nowPlayerIndex = rule->nextUser(*this);
    }
```

```
    void addToHistory(CommandMemento *memento){

        history.push_back(memento);

    }

    std::vector<CommandMemento*> getHistory(){ return history; };
    GameField &getField(){ return gameField; }
    const std::vector<Base*> &getBases(){
        return playersBases;
    }

};
```

8. LoadCommand.h

```
#include "../../LoadCommandInterpreter.h"
#include "../../../Utils/utils.h"
#include "../../../Exceptions/InvalidFileLoaddingException.h"
#include "../../../Exceptions/DoublePlacingOnFieldException.h"
#include "../../../Exceptions/GameFieldOutOfRangeException.h"
#include "../../../Exceptions/ImpossibleMoveException.h"

class LoadCommand: public Command {

private:

    std::ifstream fs;
    LoadCommandInterpreter interpreter;

public:

    explicit LoadCommand(std::string &filename): fs(filename){}
    void execute(GameInfo &gameInfo) override{

        gameInfo.newGame();

        std::string cmd;

        std::hash<std::string> hashFunc;
        unsigned long int calculatedHash = 0;
        unsigned long int fileHash = 0;

        std::string fileHashStr;
        std::getline(fs, fileHashStr);

        fileHash = utils::StrToInt(fileHashStr);

        while (std::getline(fs, cmd)){

            CommandPtr command = interpreter.handle(cmd);
            try {
                command->execute(gameInfo);
            } catch(DoubleBasePlacingException &exception) {
                game::log << "[FileLoader]" << "Player " <<
```

```
exception.playerIndex << " trying to place base second time." << game::logend;
            } catch (DoublePlacingOnFieldException &exception){
                game::log << "[FileLoader]" << "This cell is busy by other
object." << game::logend;
            } catch (GameFieldOutOfRangeException &exception){
                game::log << "[FileLoader]" << "Out of range. Cell " <<
exception.x << " " << exception.y << " is not exist." << game::logend;
            } catch (ImpossibleMoveException &exception){
                game::log << "[FileLoader]" << "Can't move to this cell. They
busy by other object." << game::logend;
            } catch (...){
                game::log << "[FileLoader]" << "Undefined error." <<
game::logend;
            }
            auto memento = command->getMemento();
            gameInfo.addToHistory(memento);
            calculatedHash += memento->getHash(hashFunc);
            gameInfo.nextUser();

        }

        game::log << "String hash from file: " << fileHashStr << game::logend;
        game::log << "Int hash from file: " << fileHash << game::logend;
        game::log << "Calculated hash: " << calculatedHash << game::logend;
        game::log << "Read commands count: " << gameInfo.getHistory().size() <<
game::logend;

        if (fileHash != calculatedHash){
            game::log << "Wrong file format. The correctness of the loaded
field is not guaranteed." << game::logend;
            throw InvalidFileLoaddingException();
        }

    }

    ~LoadCommand() override{

        fs.close();

    }

};

class LoadCommandHandler: public CommandHandler{

public:

    bool canHandle(std::vector<std::string> &cmd) override{

        return cmd.size() == 2 && cmd[0] == "load";

    }

    CommandPtr handle(std::vector<std::string> &cmd) override{

        if (canHandle(cmd)){
```

```
            return CommandPtr(new LoadCommand(cmd[1]));
        }

        if (next) return next->handle(cmd);

        return std::make_unique<Command>();

    }

};
```