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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Объектно-ориентированное программирование»
Тема: Сериализация состояния программы

Студент гр. 8303 _____ Хохлов Г.О.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

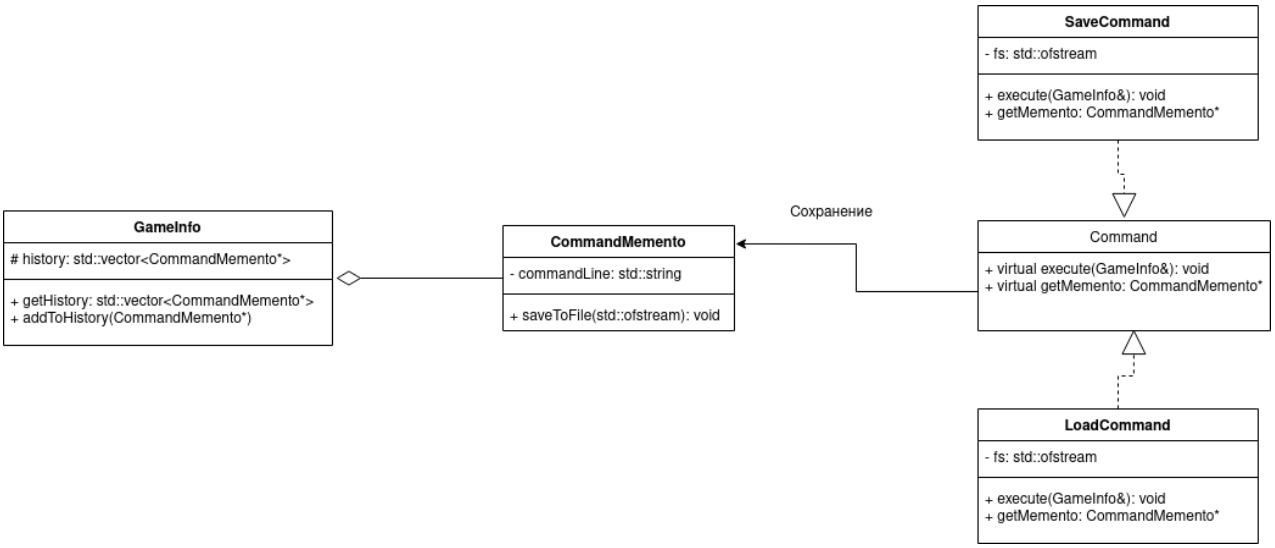
Реализация сохранения и загрузки состояния программы. Основные требования:

- Возможность записать состояние программы в файл
- Возможность считать состояние программы из файла

Ход выполнения работы.

1. Были реализованы команды «save» и «load» для сохранения и загрузки состояния.
2. Класс SaveCommand открывает файл в конструкторе и закрывает в деструкторе. Таким образом, он соответствует идиоме RAII
3. Был написан класс CommandMemento, хранящий состояние команды. При выполнении команды ее состояние сохраняется в истории.
4. Был реализован контроль за корректностью файла с помощью класса LoadCommandInterpreter. Он игнорирует части файла, которые не могут быть выполнены.

UML-диаграмма.



Вывод.

В ходе выполнения лабораторной работы была изучена сериализация состояния программы путем написание функции сохранения с использованием паттерна «Состояние».

.

Приложение А. Исходный код программы

1. CommandMemento.h

```
#include <string>
#include <fstream>
#include <utility>

class CommandMemento{

private:

    std::string commandLine;

public:

    CommandMemento(std::string commandLine):
commandLine(std::move(commandLine)){}
    void saveToFile(std::ofstream &fs) const{

        fs << commandLine;

    }

};
```

2. LoadCommand.h

```
#include "../../LoadCommandInterpreter.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;
        while (std::getline(fs, cmd)){

            CommandPtr command = interpreter.handle(cmd);
            command->execute(gameInfo);
```

```
            gameInfo.addToHistory(command->getMemento());
            gameInfo.nextUser();

        }

    }

    ~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>();

    }

};
```

3. SaveCommand.h

```
#include "../Command.h"

class SaveCommand: public Command {

private:

    std::string filename;
    std::ofstream fs;

public:

    explicit SaveCommand(std::string &filename): fs(filename){
```

```
        game::log << "File opened" << game::logend;
        game::log << "File is open: " << fs.is_open() << game::logend;

    }
    void execute(GameInfo &gameInfo) override{

        game::log << "Saving..." << game::logend;
        auto history =  gameInfo.getHistory();
        for (auto m: history){

            m->saveToFile(fs);

        }

    }

    ~SaveCommand() override{
        game::log << "File closed" << game::logend;
        fs.close();
        game::log << "File is open: " << fs.is_open() << game::logend;
    }

};

class SaveCommandHandler: public CommandHandler{

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

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

    }

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

        if (canHandle(cmd)){
            return CommandPtr(new SaveCommand(cmd[1]));
        }

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

        return  std::make_unique<Command>();

    }

};
```

4. LoadCommandInterpreter

```
#include "Commands/Command.h"

#include "Commands/Attack/AttackCommand.h"
#include "Commands/Create/CreateCommand.h"
#include "Commands/Move/MoveCommand.h"
#include "Commands/Show/ShowCommand.h"
```

```
#include "Commands/Exit/ExitCommand.h"

class LoadCommandInterpreter  {

private:

    AttackCommandHandler *attackHandler;
    CreateCommandHandler *createHandler;
    MoveCommandHandler *moveHandler;
    ShowCommandHandler *showHandler;
    ExitCommandHandler *exitHandler;

public:

    LoadCommandInterpreter(){

        attackHandler = new AttackCommandHandler();
        createHandler = new CreateCommandHandler();
        moveHandler = new MoveCommandHandler();
        showHandler = new ShowCommandHandler();
        exitHandler = new  ExitCommandHandler();


        attackHandler->setNext(createHandler);
        createHandler->setNext(moveHandler);
        moveHandler->setNext(showHandler);
        showHandler->setNext(exitHandler);
    }

    CommandPtr handle(std::string commandString){

        std::vector <std::string> commandSplitted;
        std::stringstream ss(commandString);
        std::string commandWord;
        while (ss >> commandWord)
            commandSplitted.push_back(commandWord);

        return attackHandler->handle(commandSplitted);

    }

    ~LoadCommandInterpreter(){

        delete attackHandler;
        delete createHandler;
        delete moveHandler;
        delete showHandler;
        delete exitHandler;

    }


};
```