Софийски университет „Св. Климент Охридски“
Факултет по математика и информатика

Курс по Обектно - ориентирано програмиране
за специалност Информатика
Летен семестър на учебната 2019/2020 година

Traveller’s app

Автор: Мирослав Николаев Парапанов
ФН:45560
ГРУПА: 1

1. Увод

1.1. Описание и идея на проекта

Проектът е проектиран да съхранява потребители чрез ипозлване на изключително опростена база от данниа. Въпреки че проектът е в пъти по-симплифициран от съвременните бази от данни, той е функционален и добре структуриран.

1.2. Цел и задачи на разработката

Една от основните цели на задача бе да се построи максимално добре работеща и оптимизационна архитектура. Също така капсуловането на информацията и съхранението под формата на последователен файл също бе предизвикателство .Използвани са множество от идеи, които да поставят едно по-абстрактно виждане върху самото приложение и ,разбира се, те са съвместими с ООП (Обектно-ориентирано програмиране) парадигмите. Използвани са patterns(патърни / модели) за REPL (read-eval-print loop), чрез които се симлифицира за по-нататъшно развитие и опримизиране на проекта. Разделното компилиране, управлението на паметта, глобалните и статичните променливи, и реализирането на помощни класове с цел подобряване на функционалността са задачи, които са имплементирани във приложението.

С цел също да се намали сложността на повечето задачи са използвани помощни функции.

1.3. Структура на Документацията

Първо ще се разгедат архитектурата и йерархията на класовете и как по-точно са свързание техните функционалности. Ще покажем как всеки клас си има собствена фунция и се грижи, в зависимост от това как са имплементирани методи му, за входните и изходните данни.

2. Преглед на предметната област

2,1 . Основни дефиниции, концепции и алгоритми, които ще бъдат използвани

2.1.1. Клас

2.1.2. Полиморфизъм

2.1.3. Конструктори, Деструктори

2.1.4. Единично и множествено наследяване

2.1.6. Абстрактни класове

2.1.7. Виртуални и чисто виртуални функции

2.1.8. Файлове

2.1.9. Command Pattern (патърн / модел)

2.1.10. Стилизиране на кода

2.1.11. Smart pointers (умни указатели)

2.1.12. Гит хранилище (GitHub)

2.2. Дефиниране на проблеми и сложност на поставената задача:

2.2.1. Работа с последователни файл и в конкретност пресмятане на байтовете за записването на записите.

2.2.2. Изготвяне на интерфейс класове, които в бъдеще ще послужат за по-лесно развитие на проекта.

2.2.3. Комуникацията между отделните класове и архитектурата.

2.2.4. Използване на smart pointers (умни указатели) и правенето на на такива

2.3. Подходи и методи за решаване на проблемите:

2.3.1. Използване на абстрактни класове за по-абстрактен вид върху приложението

2.3.2. Използване на смарт пойнтъри за по - лесен достъп до класовете команда

2.3.2. Използване на интърфейс-класове за по-нататъшно развитие на проекта понеже сега базата от данни е само под формата на текстови файл, а тя може да бъде и SQL или JSON, или EXCELL и т.н. Аналогично това се отнася и за командите.

2.4. Потребителски изисквания:

2.4.1. Валидност на входните данни

2.4.2. За персонална информация всички данни да са на различни редове като следва този форамт ->Град->Държава->дата на пристигане (написани с точка между цифрите пр. 23.01.2019) ->дата на заминаване (—//—)->Рейтинг->коментар->снимка/-и(ако са много снимки да са на единт ред с запетайки без спейсове).

2.4.3

3. Проектиране

3.1. Обща архитектура :

3.1.1. Използван е Command Pattern, който опростява модела за писане. Имаме два абстрактни класа. Единият е ICommand, който се грижи за името на командата и една чисто виртуална функция чрез,която ще се извършва

полиморфизмът., а другия е System, който ще се грижи изцяло за базата от данни.

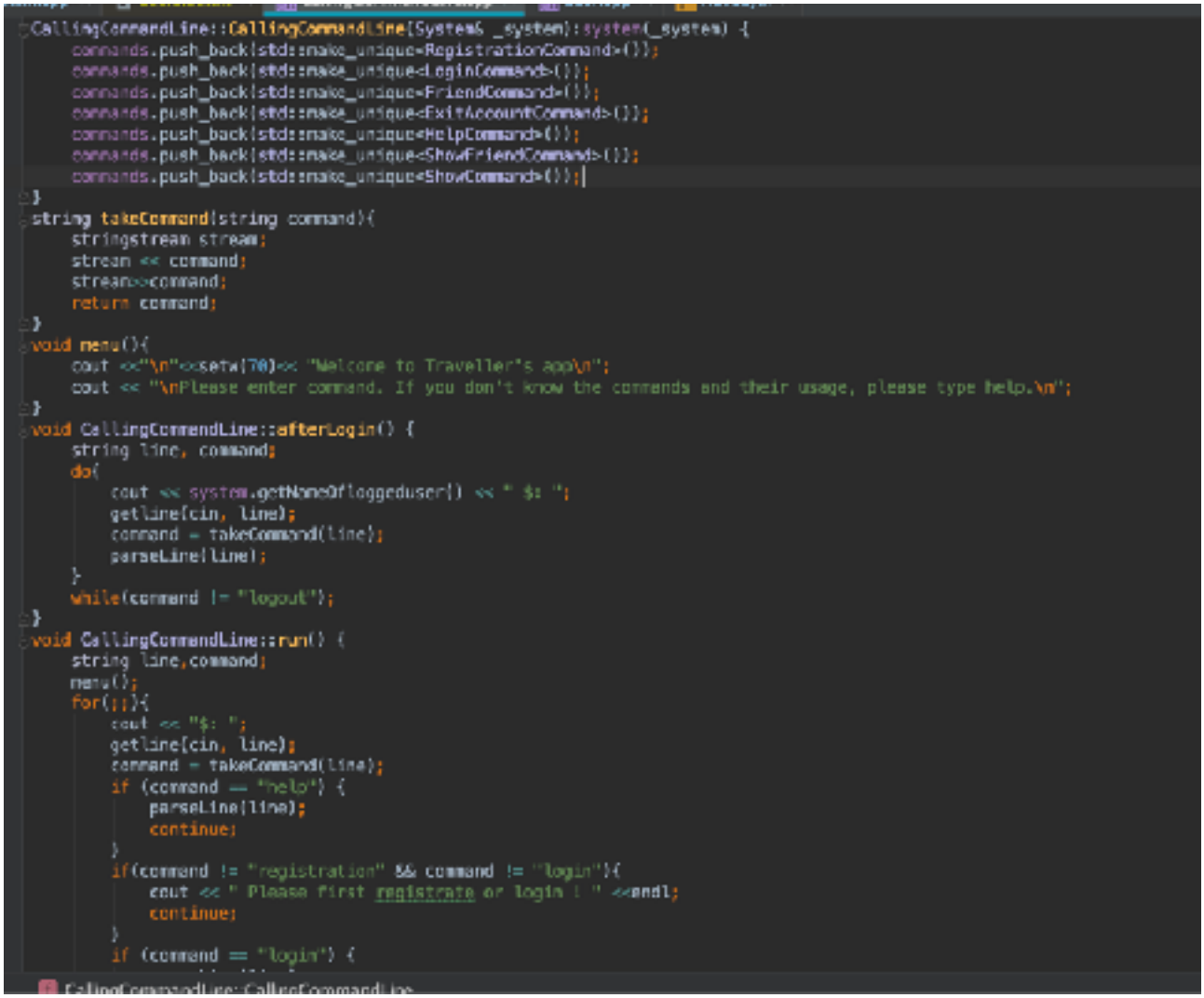
```
class ICommand {
protected:
    std::string commandName;
    std::vector<std::string> arguments;

public:
    ICommand(){};
    ICommand(const std::string& commandNameTemp): commandName(commandNameTemp)
    virtual void execute(TXTPlaneManager &) {};
    void setArguments(std::vector<std::string>& arguments) {
        this->arguments = arguments;
    }
    std::string& convertToString() {
        return commandName;
    }
    virtual ~ICommand() {};
};
```

Използван е също така смарт пойнтър и в конкретност

std::unique_ptr<T> понеже е доста по - лесно поради факта, че не нужно собственоръчно да се грижиш за паметта а, тя автоматично се трие след това.Също така при този вид пойнтър няма как да се направи нова референция и за опростение на дивелопъра operator= и копи-конструкторът са изтрити.

В Invoker класа единственото нещо, което се случва е още при създаването на обект от този вид той да запълва вектора от смарт пойнтъри с командите като ги прави смарт пойнтър и при написването на дадената команда тя да се оцени и да се приложи дадения метод , на когото принадлежи командата!

```
std::vector<std::unique_ptr<ICommand>> commands;
```



```
CallingCommandLine::CallingCommandLine(System& _system):system(_system) {
    commands.push_back(std::make_unique<RegistrationCommand>());
    commands.push_back(std::make_unique<LoginCommand>());
    commands.push_back(std::make_unique<FriendCommand>());
    commands.push_back(std::make_unique<ExitAccountCommand>());
    commands.push_back(std::make_unique<HelpCommand>());
    commands.push_back(std::make_unique<ShowFriendCommand>());
    commands.push_back(std::make_unique<ShowCommand>());
}
string takeCommand(string command){
    stringstream stream;
    stream << command;
    stream>>command;
    return command;
}
void menu(){
    cout <<"\n"<<setw(70)<< "Welcome to Traveller's app\n";
    cout << "\nPlease enter command. If you don't know the commands and their usage, please type help.\n";
}
void CallingCommandLine::afterLogin() {
    string line, command;
    do{
        cout << system.getNameOfloggeduser() << " $: ";
        getline(cin, line);
        command = takeCommand(line);
        parseLine(line);
    }
    while(command != "logout");
}
void CallingCommandLine::run() {
    string line,command;
    menu();
    for(;;){
        cout << "$: ";
        getline(cin, line);
        command = takeCommand(line);
        if (command == "help") {
            parseLine(line);
            continue;
        }
        if(command != "registration" && command != "login"){
            cout << " Please first registrate or login ! " <<endl;
            continue;
        }
        if (command == "login") {
```

Класът System се грижи изцяло за базата от данни под формата на текстови файл.

```
class System {
private:
    vector<User> users;
    bool wrongConnection;
    string nameOfLoggedUser;
    bool isDuplicate(const string&,const string&,const string&)const;
    bool isExisting(const string&)const;
    bool isLogged()const;
public:
    System();
    ~System();
    void help()const;
    void logout(const string&);
    User getUserByName(const string&)const;
    void registration(string&, string&, string&);
    void logIn(const string&,const string&);
    void friends(const string&);
    bool getWrongConnection()const;
    string getNameOfloggeduser()const;
    void showFriendDest(const string&,const string&)const;
    void show(const string&)const;
};
```

4. Реализация и тестване :

4.1. Реализация на класове:

4.1.1. По горе на картинките са показани част от по важните класове , които са задължителен слой от архитекртурата

5. Заключение:

5.1.Крайният продукт е работещо приложение, което отговаря на ООП парадигмите и желанията на клиента.

5.2. Оптимизацията на проекта може да се осъществи като се добавят нови бази от данни или се разработят нови команди, които подобряват базата от данни.Могат също така да се добавят нови функционалности и поради факта, че кодът е написан за бъдещо разширение на този продукт, няма опасност от спиране на извървашнето на старите такива.

6. Използвана Литература:

6.1.https://www.youtube.com/

6.2.http://www.cplusplus.com/reference/

6.3. https://www.wikipedia.org/

6.4.https://www.geeksforgeeks.org/

6.5. https://stackoverflow.com/