МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования
«Новосибирский государственный университет экономики и управления «НИНХ»
(ФГБОУ ВО «НГУЭУ», НГУЭУ)

Кафедра информационных технологий

КУРСОВАЯ РАБОТА

Дисциплина: Языки программирования
Ф.И.О студента: Небылицин Максим Николаевич
Направление: 02.03.02 Фундаментальная информатика и информационные технологии
Специализация: Программная инженерия
Номер группы: ФИ202
Номер зачетной книжки: 220130
Номер варианта: 10
Проверил: Ковригин Алексей Викторович, канд. педагогических наук, преподаватель

Новосибирск 2024

Оглавление

1. Задача проекта

Создать программу по управлению объектами данных на тему «Управление проектами» с использованием ООП и реализовать изученные паттерны.

Необходимо создать минимум 3 объекта сущности разных видов и указать взаимодействие между объектами. Реализовать интерфейсы ввода, вывода и редактирования сущностей. Реализовать ручной ввод сущностей, загрузку из файла и сохранения в файл, удаление выбранных сущностей.

Данные между запусками программы хранить в структурированном текстовом файле.

2. Структура проекта

2.1 Описание структуры проекта

Тема программы – «Управление проектами». В процессе создания использовалось объектно-ориентированное программирование и паттерн Model-View-Controller (MVC).

Программа состоит из нескольких файлов, в соответствие с паттерном MVC. Это сделано для более удобного изменения различных компонентов.

Главный файл с исходным кодом, из которого запускается программа, называется project_nebylitsin.cpp. Вот так выглядит код этого файла:

```
#include <iostream>
#include "controller.h"
int main() {
    Model model;
    View view;
    Controller controller(model, view);
    controller.run();
    return 0;
}
```

С помощью директивы #include, включается заголовочный файл controller.h. К нему уже подключается model.h и view.h. Также есть файлы project.h, task.h и текстовый файл projects.txt, в котором хранятся данные между запусками программы.

В процессе работы программы пользователь также может создавать текстовые файлы с названием проекта и расширением .txt, в которых будут содержаться задачи проекта. После корректного выхода из программы эти файлы и их содержимое будут сохранены в той же папке, откуда была запущена программа.

2.2 Класс Task

Класс Task отвечает за создание и изменение задачи и состояния задачи. Он имеет две приватных переменных: строку name и логическое значение status. Конструктор класса принимает строку name и устанавливает значение status в false. Класс также содержит методы для установки статуса задачи (set_status), изменения статуса задачи (change_status), получения статуса задачи (get_status), получения статуса задачи в виде строки (get_status_string) и получения имени задачи (get_name). Директива #ifndef используется для защиты от множественного включения заголовочного файла task.h.

```
#ifndef TASK_H
#define TASK_H
#include <string>
using namespace std;
class Task {
private:
    string name;
    bool status;
public:
    Task(string name) {
        this->name = name;
        status = false;
    }
    void set_status(bool status) {
        this->status = status;
    }
    void change_status(bool status) {
        this->status = !status;
    }
```

```
    bool get_status() {
        return status;
    }
    string get_status_string() {
        if (status) {
            return "done";
        } else {
            return "not done";
        }
    }
    string get_name() {
        return name;
    }
};
#endif
```

2.3 Класс Project

Класс Project имеет три приватных переменных: строку name, логическую переменную data-loaded и вектор объектов класса Task под названием tasks. Конструктор класса принимает строку "name" и устанавливает значение "data_loaded" в false. Класс также содержит методы для добавления задачи в проект ("add_task"), получения задачи по индексу ("get_task"), получения списка задач ("get_tasks"), получения имени проекта ("get_name"), получения количества задач в проекте ("get_num_tasks") и удаления задачи из проекта ("remove_task"). Директива "#ifndef" используется для защиты от множественного включения заголовочного файла "project.h".

```
#ifndef PROJECT_H
#define PROJECT_H
#include <string>
#include <vector>
#include <fstream>
#include "task.h"
class Project {
private:
    string name;
    bool data_loaded;
    vector<Task> tasks;
```

```
public:
    Project(string name) {
        this->name = name;
        this->data_loaded = false;
    }
    bool get_data_loaded() {
        return data_loaded;
    }
    void change_data_loaded() {
        data_loaded = true;
    }
    void add_task(Task task) {
        tasks.push_back(task);
    }
    Task& get_task(int index) {
        return tasks.at(index);
    }
    vector<Task>& get_tasks() {
        return tasks;
    }
    string get_name() {
        return name;
    }
    int get_num_tasks() {
        return tasks.size();
    }
    void remove_task(Task task) {
        for (auto it = tasks.begin(); it != tasks.end(); ++it) {
            if (it->get_name() == task.get_name()) {
                tasks.erase(it);
                break;
            }
        }
    }
};
#endif
```

2.4 Класс Model

Класс Model в паттерне MVC (Model-View-Controller) представляет собой модель данных, которая отвечает за обработку, хранение, изменение данных. Этот класс содержит важнейшие методы, которые использует класс Controller для управления программой. Он содержит методы для создания, удаления, получения и изменения проектов и задач, а также для загрузки и сохранения данных в файлы. Например, здесь находится метод для загрузки (load_project_data) данных из текстового файла projects.txt и для сохранения (save_project_data) данных в этот файл. Управление проектами и задачами в процессе работы программы осуществляется с помощью вектора объектов класса Project, потовый представляет список проектов, и вектора объектов класса Task, который представляет список задач.

В целом класс Model является основной частью модели в паттерне MVC и отвечает за управление данными в приложении.

```
#ifndef MODEL_H
#define MODEL_H
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include "project.h"
using namespace std;
class Model {
private:
    vector<Project> projects;
    vector<Task> tasks;
public:
    void create_project(Project project) {
        projects.push_back(project);
        ofstream file(project.get_name() + ".txt");
        if (!file) {
                cout << "Error, project has not created!\n\n";
        }
        else {
```

```
                cout << "Project created\n\n";
            }
            file.close();
        }
        void delete_project(Project project, int index) {
            auto iter = projects.cbegin();
            projects.erase(iter + index);
            string filename = project.get_name() + ".txt";
            if (remove(filename.c_str()) != 0) {
                cout << "Error deleting project" << endl;
            } else {
                cout << "Project deleted" << endl;
            }
        }
        Project& get_project(int index) {
            return projects.at(index);
        }
        vector<Project>& get_projects() {
            return projects;
        }
        void create_task(Project& project, Task& task) {
            project.add_task(task);
            ofstream file(project.get_name() + ".txt", std::ios::app);
            for (int i = project.get_tasks().size() - 1; i <
project.get_tasks().size(); i++) {
                Task task = project.get_task(i);
                cout << i + 1 << ". " << task.get_name() << " (" <<
task.get_status_string() << ")" << endl;
            }
            file.close();
        }
        void change_status(Project& project, int index) {
            project.get_task(index);
        }
        void load_data() {
            ifstream file("projects.txt");
            string line;
            if (file.is_open()) {
                while (getline(file, line)) {
```

```
                projects.push_back(line);
            }
        } else {
            cout << "ERROR! THE DATA HAS NOT BEEN LOADED" << endl;
        }
        file.close();
    }
    void save_data() {
        ofstream file("projects.txt");
        if (file.is_open()) {
            for (int i = 0; i < projects.size(); i++) {
                file << projects.at(i).get_name() << endl;
            }
        }
        else {
            cout << "ERROR! THE DATA HAS BEEN DELETED" << endl;
        }
        file.close();
    }
    void load_project_data(Project& project) {
        ifstream file(project.get_name() + ".txt");
        if (file.is_open()) {
            string line;
            while (getline(file, line)) {
                int task_num;
                string task_name, status_str;
                if (sscanf(line.c_str(), "%d. ", &task_num) != 1) {
                    continue;
                }
                size_t name_start = line.find_first_not_of(" ",
line.find(". ") + 2);
                size_t name_end = line.find(" (", name_start);
                task_name = line.substr(name_start, name_end -
name_start);
                bool status = (line.substr(name_end + 2, 4) ==
"done");
                project.add_task(task_name);
                project.get_tasks().back().set_status(status);
            }
```

```
        } else {
            cout << "ERROR! THE PROJECT DATA HAS NOT BEEN LOADED" <<
endl;
        }
        file.close();
    }
    void save_project_data(Project& project) {
        ofstream file(project.get_name() + ".txt");
        if (file) {
            for (int i = 0; i < project.get_num_tasks(); i++) {
                Task& task = project.get_task(i);
                string status_str = task.get_status() ? "done" : "not
done";
                file << i + 1 << ". " << task.get_name() << " (" <<
status_str << ")\n";
            }
        } else {
            cout << "ERROR! THE PROJECT DATA HAS BEEN DELETED" <<
endl;
        }
    }
};
#endif
```

2.5 Класс View

Класс View в модели MVC отвечает за представление данных пользователю. Он содержит методы для вывода информации на экран, получения ввода от пользователя и обработки ошибок. View отвечает за вывод информации о проектах и задачах на экран, получение ввода от пользователя, связанного с проектами и задачами, и вывод сообщений об ошибках и статусе сохранения данных. View не содержит логики приложения и не изменяет состояние данных. Его задача - только отображать данные и получать ввод от пользователя. Это позволяет разделить логику приложения и представление данных, что упрощает поддержку и расширение приложения.

```
#ifndef VIEW_H
#define VIEW_H
#include <iostream>
```

```
#include <fstream>
#include <string>
#include <vector>
#include "project.h"
#include "task.h"
using namespace std;
class View {
public:
    void press_enter() {
        cout << "Press Enter to continue..." << endl;
        getchar();
    }
    void invalid_input() {
        cout << "Invalid input" << endl;
    }
    void print_projects(vector<Project>& projects) {
        cout << "Projects:" << endl;
        for (int i = 0; i < projects.size(); i++) {
            cout << i + 1 << ". " << projects.at(i).get_name() <<
endl;
        }
    }
    int input_project_index() {
        string input;
        cout << "Enter project index: ";
        getline(cin, input);
        return stoi(input);
    }
    void print_project_details(Project& project) {
        cout << "Project: " << project.get_name() << endl;
        cout << "Tasks:" << endl;
        for (int i = 0; i < project.get_tasks().size(); i++) {
            Task task = project.get_task(i);
            cout << i + 1 << ". " << task.get_name() << " (" <<
task.get_status_string() << ")" << endl;
        }
    }
    int input_task_index() {
        string input;
```

```
        cout << "Enter task index: ";
        getline(cin, input);
        return stoi(input);
    }
    Project input_project_name() {
        string name;
        cout << "Enter project name: ";
        getline(cin, name);
        return Project(name);
    }
    Task input_task() {
        string name;
        cout << "Enter task name: ";
        getline(cin, name);
        return Task(name);
    }
    void error() {
        cout << "ERROR! Probably, some of the data was not saved to
the file" << endl;
    }
    void save_info() {
        cout << "(the data will be saved in a file after quitting)\n"
<< endl;
    }
    void save_info_back() {
        cout << "(the data will be saved in a file after go back)\n"
<< endl;
    }
    void print_menu() {
        cout << "Enter 'p' to print projects, "
            << "or 't' to view project tasks, "
            << "or 'n' to create a new project, "
            << "or 'd' to delete a project, "
            << "or 'q' to quit: ";
    }
    void print_project_menu(Project& project) {
        cout << "Enter 'n' to create a new task, "
            << "or 'c' to change a task status, "
            << "or 'r' to remove a task, "
```

```
            << "or 'p' to print tasks, "
            << "or 'b' to go back: ";
    }
};
#endif
```

2.6 Класс Controller

Класс Controller в модели MVC отвечает за обработку пользовательского ввода и управление моделью и представлением. Он получает данные от View, обрабатывает их и передает соответствующие команды Model и View. Он содержит метод run(), который загружает данные из файла, выводит меню на экран и ожидает ввода от пользователя. В зависимости от выбора пользователя, Controller вызывает соответствующие методы Model и View для создания, удаления, изменения или вывода информации о проектах и задачах. Также при запуске или выходе из цикла автоматически загружаются данные из текстового файла projects.txt. Controller также отвечает за обработку ошибок и вывод сообщений об ошибках и статусе сохранения данных. Если пользователь вводит некорректные данные, Controller вызывает методы View для вывода сообщений об ошибке и просит пользователя повторить ввод.

```
#include <iostream>
#include <fstream>
#include "model.h"
#include "view.h"
using namespace std;
class Controller {
private:
    Model& model;
    View& view;
public:
    Controller(Model& model, View& view) : model(model), view(view) {}
    void run() {
        model.load_data();
        while (true) {
            try {
```

```
                view.save_info();
                view.print_menu();
                string choice;
                getline(cin, choice);
                if (choice == "p") {
                    view.print_projects(model.get_projects());
                    view.press_enter();
                } else if (choice == "n") {
                    Project project = view.input_project_name();
                    model.create_project(project);
                    view.press_enter();
                } else if (choice == "d") {
                    int index = view.input_project_index() - 1;
                    Project project = model.get_project(index);
                    model.delete_project(project, index);
                    view.press_enter();
                } else if (choice == "q") {
                    model.save_data();
                    break;
                } else if (choice == "t") {
                    int index = view.input_project_index() - 1;
                    Project& project = model.get_project(index);
                    if (!project.get_data_loaded()) {
                        project.change_data_loaded();
                        model.load_project_data(project);
                    }
                    view.print_project_details(project);
                    view.press_enter();
                    while (true) {
                        view.save_info_back();
                        view.print_project_menu(project);
                        getline(cin, choice);
                        if (choice == "n") {
                            Project& project =
model.get_project(index);
                            Task task = view.input_task();
                            model.create_task(project, task);
                            view.press_enter();
                        } else if (choice == "c") {
```

```
                                index = view.input_task_index() - 1;
                                Task& task = project.get_task(index);
                                task.change_status(task.get_status());
                                view.press_enter();
                            } else if (choice == "r") {
                                index = view.input_task_index() - 1;
                                Task& task = project.get_task(index);
                                project.remove_task(task);
                                view.press_enter();
                            } else if (choice == "p") {
                                view.print_project_details(project);
                                view.press_enter();
                            } else if (choice == "b") {
                                model.save_project_data(project);
                                view.press_enter();
                                break;
                            }
                        }
                    }
                } catch (...) {
                    view.invalid_input();
                    view.error();
                    view.press_enter();
                }
            }
        }
};
```