|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** \_***ИУК «Информатика и управление»*\_\_**\_\_\_\_\_\_\_\_\_\_\_\_

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА**

**«Наследование и иерархия классов»**

**ДИСЦИПЛИНА: «Высокоуровневое программирование»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-22Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) (Ф.И.О.) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Козина А.В. )  (Подпись) (Ф.И.О.) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |
| Калуга , 2021 | | |

**Цель:** приобретение практических навыков проектирования иерархии классов.

**Задачи:**

1. Изучить понятие иерархии классов.
2. Познакомиться со способами наследования классов.
3. Изучить понятие абстрактный класс.
4. Изучить понятие интерфейса.
5. Изучить работу с виртуальными функциями.
6. Научиться выполнять инициализацию объекта разными способами.

**Вариант 8**

**Задание:**

**Общая задача**

Вам будет предложено написать программу – «Автоматизированная система диалога (чат бот)». Которая будет включать следующий функционал:

* Ведение базы пользователей
  + Создание / удаление / редактирование записей
  + Сортировка / фильтрация
* Ведение базы диалогов, тем, интересов и напоминаний
* Возможность авторизации
* Создание файлов-отчётов и сохранения состояния

**Базовая часть**

*Задача 1*

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

*Задача 2*

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

*Задача 3*

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m\_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m\_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char\* «m\_title», которое будет содержать ссылку на C-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size\_t «m\_count», которое будет хранить количество пунктов меню.

*Задача 4*

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m\_item\_name» типа char\*, который будет содержать указатель на C-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m\_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

*Задача 5*

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(\*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на C-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

*Задача 6*

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m\_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

*Задача 7*

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

**Индивидуальная часть**

*Задача 1*

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

*Задача 2*

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

**UML-диаграмма классов:**



**Рисунок 1.1.** UML-диаграмма классов



**Рисунок 1.2.** UML-диаграмма классов

**Листинг:**

**Menu.h**

#ifndef MENU\_H

#define MENU\_H

#include "AbstractMenuItem.h"

namespace KMK

{

class Menu

{

public:

Menu(char\* title, MenuItem\*\* items, size\_t count);

int GetSelect();

bool GetRunning();

char\* GetTitle();

size\_t GetCount();

MenuItem\*\* GetItems();

void Print();

int RunCommand();

private:

int m\_select = -1;

bool m\_running = false;

char\* m\_title = nullptr;

size\_t m\_count{};

MenuItem\*\* m\_items = nullptr;

};

}

#endif // !MENU\_H

**Menu.cpp**

#include "Menu.h"

#include <iostream>

using namespace KMK;

Menu::Menu(char\* title, MenuItem\*\* items, size\_t count)

{

m\_title = new char[LENGTH\_OF\_FIELD];

strcpy\_s(m\_title, LENGTH\_OF\_FIELD, title);

m\_items = items;

m\_count = count;

}

int Menu::GetSelect()

{

return m\_select;

}

bool Menu::GetRunning()

{

return m\_running;

}

char\* Menu::GetTitle()

{

return m\_title;

}

size\_t Menu::GetCount()

{

return m\_count;

}

MenuItem\*\* Menu::GetItems()

{

return m\_items;

}

void Menu::Print()

{

for (short i = 0; i < m\_count; i++)

{

std::cout << i << ". ";

m\_items[i]->PrintItemName();

std::cout << '\n';

}

}

int Menu::RunCommand()

{

Print();

std::cout << "Enter command: ";

short command;

std::cin >> command;

return m\_items[command]->Run();

}

**AbstractMenuItem.h**

#ifndef ABSTRACT\_MENU\_ITEM\_H

#define ABSTRACT\_MENU\_ITEM\_H

namespace KMK

{

extern const int LENGTH\_OF\_FIELD;

class MenuItem

{

public:

MenuItem(char\* itemName);

virtual char\* GetItemName();

virtual void PrintItemName();

virtual int Run() = 0;

private:

char\* m\_itemName = nullptr;

};

}

#endif // !ABSTRACT\_MENU\_ITEM\_H

**AbstractMenuItem.cpp**

#include "AbstractMenuItem.h"

#include <iostream>

using namespace KMK;

const int KMK::LENGTH\_OF\_FIELD = 255;

MenuItem::MenuItem(char\* itemName)

{

m\_itemName = new char[LENGTH\_OF\_FIELD];

strcpy\_s(m\_itemName, LENGTH\_OF\_FIELD, itemName);

}

char\* MenuItem::GetItemName()

{

return m\_itemName;

}

void MenuItem::PrintItemName()

{

std::cout << m\_itemName;

}

**LogInItem.h**

#ifndef LOG\_IN\_ITEM\_H

#define LOG\_IN\_ITEM\_H

#include "AbstractMenuItem.h"

namespace KMK

{

class LogInItem : public MenuItem

{

public:

LogInItem(char\* itemName, char\* tempDataBase, char\* dataBase);

int Run();

private:

char\* m\_tempDataBase = nullptr;

char\* m\_dataBase = nullptr;

};

}

#endif // !LOG\_IN\_ITEM\_H

**LogInItem.cpp**

#include "LogInItem.h"

#include <iostream>

using namespace KMK;

LogInItem::LogInItem(char\* itemName, char\* tempDataBase, char\* dataBase) : MenuItem(itemName)

{

m\_tempDataBase = tempDataBase;

m\_dataBase = dataBase;

}

int LogInItem::Run()

{

std::cout << "Enter login and password...\n";

std::cout << "What data do you want to change?..\n";

return 1;

}

**ItemWithMenu.h**

#ifndef ITEM\_WITH\_MENU\_H

#define ITEM\_WITH\_MENU\_H

#include "AbstractMenuItem.h"

namespace KMK

{

class ItemWithMenu : public MenuItem

{

public:

ItemWithMenu(char\* itemName, MenuItem\*\* items, size\_t count);

void SetItems(MenuItem\*\* items);

void SetCount(size\_t count);

int Run();

private:

MenuItem\*\* m\_items = nullptr;

size\_t m\_count{};

};

}

#endif // !ITEM\_WITH\_MENU\_H

**ItemWithMenu.cpp**

#include "ItemWIthMenu.h"

#include <iostream>

using namespace KMK;

ItemWithMenu::ItemWithMenu(char\* itemName, MenuItem\*\* items, size\_t count) : MenuItem(itemName)

{

m\_items = items;

m\_count = count;

}

void ItemWithMenu::SetItems(MenuItem\*\* items)

{

m\_items = items;

}

void ItemWithMenu::SetCount(size\_t count)

{

m\_count = count;

}

int ItemWithMenu::Run()

{

for (short i = 0; i < m\_count; i++)

{

std::cout << i << ". ";

m\_items[i]->GetItemName();

std::cout << '\n';

}

std::cout << "Enter command: ";

short command;

std::cin >> command;

return m\_items[command]->Run();

}

**UserListItem.h**

#ifndef USER\_LIST\_ITEM\_H

#define USER\_LIST\_ITEM\_H

#include "ItemWIthMenu.h"

namespace KMK

{

class UserListItem : public ItemWithMenu

{

public:

UserListItem(char\* itemName, char\* tempDataBase, char\* dataBase);

int Run();

private:

char\* m\_tempDataBase = nullptr;

char\* m\_dataBase = nullptr;

};

}

#endif // !USER\_LIST\_ITEM\_H

**UserListItem.cpp**

#include "UserListItem.h"

#include <iostream>

using namespace KMK;

UserListItem::UserListItem(char\* itemName, char\* tempDataBase, char\* dataBase):

ItemWithMenu(itemName, nullptr, 0)

{

m\_tempDataBase = tempDataBase;

m\_dataBase = dataBase;

}

int UserListItem::Run()

{

std::cout << "Create new user\n";

std::cout << "Format list\n";

std::cout << "[List of users]\n";

std::cout << "Enter command: ...\n";

return 1;

}

**Main.cpp**

#include "Menu.h"

#include "UserListItem.h"

#include "LogInItem.h"

using namespace KMK;

int main()

{

UserListItem users = UserListItem((char\*)"Users", nullptr, nullptr);

LogInItem logIn = LogInItem((char\*)"Log in", nullptr, nullptr);

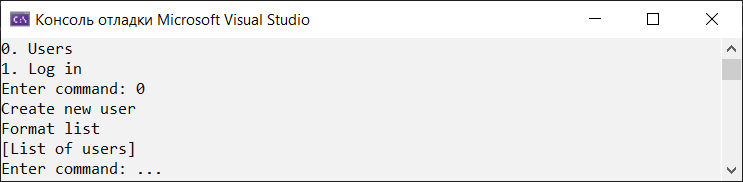
Menu menu = Menu((char\*)"Chat bot", new MenuItem\*[2]{ &users, &logIn }, 2);

menu.RunCommand();

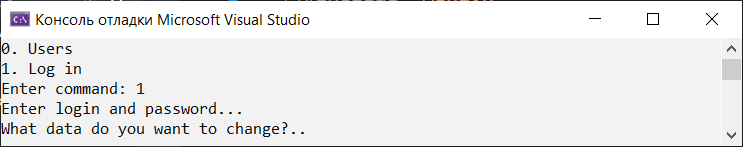
return 0;

}

**Демонстрация:**



**Рисунок 2.** Консоль при вводе 0



**Рисунок 3.** Консоль при вводе 1

**Вывод:** в ходе работы были получены навыки создания родительских и дочерних классов, абстрактных классов, виртуальных и чисто виртуальных методов, переопределения методов, создания конструкторов в дочерних классах, UML-диаграмм, пространств имен.