МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ

ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»

Факультет Информационных систем и технологий

Кафедра Безопасности информационных систем

КУРСОВОЙ ПРОЕКТ

|  |
| --- |
| Технологии программирования |

по дисциплине

(наименование учебной дисциплины согласно учебному плану)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

|  |
| --- |
| «Доставка пиццы»: ассортимент, клиенты, заказы и |

|  |
| --- |
| скидки... |

Тема проекта:

|  |
| --- |
| ИСТ-312 |

|  |
| --- |
|  |

|  |
| --- |
| Кандиков М.В. |

**/**

**/**

Автор: студент гр.

(шифр группы) (подпись) (Ф.И.О.)

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
| ст. преподаватель кафедры БИС |

Проверил:

руководитель проекта

|  |
| --- |
|  |

|  |
| --- |
| Ликарь А. И. |

(должность)

**/**

(подпись) (Ф.И.О.)

Санкт-Петербург

2024 г.

**Оглавление**

**Описание проекта3**

**Основная логика программы7**

**Заключение14**

**Список источников15**

**Приложение16**

**Описание проекта**

Программа курсового проекта представляет собой консольное приложение, которое симулирует работу пиццерии, предоставляя пользователю возможность просматривать меню, скидки, филиалы, топ клиентов, а также делать заказы и применять промокоды. Ниже приведено детальное описание основных компонентов и логики программы.

**Библиотеки:**

* iostream – для вывода на экран и работы с вводом;
* vector – для использования контейнера vector;
* string – для использования строки string
* fstream – для работы с файлами (чтение/запись)
* locale – для установки локали
* unordered\_set – для использования контейнера unordered\_set
* limits – для работы с пределами числовых типов, очистки ввода

**Структуры данных:**

Pizza, Drink, Snack – структуры для хранения информации о пицце, напитках и снэках. Они содержат в себе название и цену.

Branch – структура для хранения информации о филиале. Содержит в себе только адрес филиала.

Client – структура для хранения информации о клиенте. Содержит внутри себя имя клиента, любимую позицию меню, общую сумму заказов и филиал, который клиент выбирает чаще всего.

**Классы:**

Order – класс для работы с заказами. Представляет собой заказ, который пользователь может создать и оформить. Основная цель класса – хранить информацию о заказе, управлять добавлением товаров в заказ, применять промокоды и выводить чек.

Поля класса Order:

* `vector<Pizza> pizzas`: Список пицц, добавленных в заказ.
* `vector<Drink > drinks`: Список напитков, добавленных в заказ.
* `vector<Snack > snacks`: Список снэков, добавленных в заказ.
* `double total`: Общая сумма заказа.
* `string branch`: Адрес филиала, где был сделан заказ.
* `string promocode`: Промокод, применённый к заказу

Методы класса Order:

* Конструктор `Order()` - инициализирует общее значение заказа (`total`) равным нулю.
* Методы `addPizza(const Pizza& pizza)`, `addDrink(const Drink& drink)` и `addSnack(const Snack& snack)` - добавляют пиццу, напиток и снэки соответственно в заказ и увеличивает общую сумму заказа на их цену.
* Метод `applyPromoCode`(const string& code) – применяет промокод к заказу, устанавливая поле `promocode` и уменьшая общую сумму на 20%.
* Метод `printReceipt() const` - выводит чек заказа в файл “Receipt.txt”, включая филиал, все товары, их цены, промокод и общую сумму.

Pizzeria – класс, управляющий всей пиццерией: хранит данные о меню, филиалах, скидках и топ клиентах, а также предоставляет интерфейс для создания и обработки заказов.

Поля класса `Pizzeria`:

* `vector<Pizza> pizzas`: Список доступных пицц.
* `vector<Drink > drinks`: Список доступных напитков.
* `vector<Snack > snacks`: Список доступных снэков.
* `vector<Branch> branches`: Список филиалов пиццерии.
* `vector<Client> topClients`: Список топ клиентов.
* `vector<string> discounts`: Список доступных скидок.
* `unordered\_set<string> promoCodes`: Набор доступных промокодов.

Методы класса `Pizzeria`:

* Конструктор `Pizzeria()` - инициализирует данные о пиццах, напитках снэках, филиалах, топе клиентов, и скидках. Загружает промокоды из файла “Promocodes.txt”.
* Метод `loadPromoCodes()` - загружает промокоды из файла “Promocodes.txt” и сохраняет из в набор `promoCodes`.
* Метод `showMenu()` - выводит на экран меню пиццерии, включая пиццы, напитки и снэки с их ценами.
* Метод `showDiscounts()` - выводит на экран текущие скидки пиццерии.
* Метод `showBranches()` - выводит на экран филиалы пиццерии.
* Метод `showTopClients()` - выводит на экран лучших клиентов пиццерии.
* Метод `createOrder` - создаёт заказ, в котором учитывается выбор: филиала пиццерии, позиций из меню, ввод промокода и печать чека.
* Метод `loadPromoCodes` - загружает доступные промокоды из файла Promocodes.txt.
* Метод `chooseBranch` - нужен для выбора филиала в заказе.
* Методы `addPizzaToOrder`, `addDrinkToOrder` и `addSnackToOrder` - добывляют пиццу, напитки и снэки в заказ.
* Метод `applyPromocode` - применяет промокод к заказу, давая скидку в 20%.

**Функции для чтения и записи данных:**

* loadPromoCodes() – открывает файл “Promocodes.txt”, читает промокоды и добавляет их в множество `promoCodes` после чего закрывает файл.

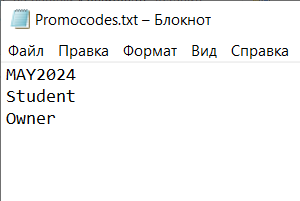


Рисунок 1. Файл “Promocodes.txt“, содержащий промокоды на скидку

* printReceipt() – открывает файл “Receipt.txt”, печатает чек заказа, включая информацию о филиале, заказанных позициях и их ценах, промокоде и итоговой стоимости, а далее закрывает файл.

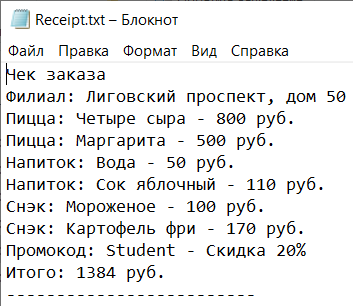


Рисунок 2. Файл “Receipt.txt”, содержащий чек заказа пользователя

**Основная логика программы**

1. **Запуск программы:**

* Пользователь запускает приложение.
* Отображается главное меню с несколькими опциями: просмотр меню, просмотр скидок, создание заказа, просмотр филиалов, просмотр топ клиентов и выход из программы.

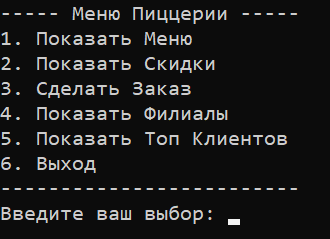


Рисунок 3. Меню пиццерии

* Пользователь выбирает интересующий его пункт меню, вводя соответствующую цифру.

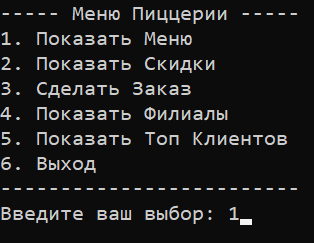


Рисунок 4. Выбор пункта меню

1. **Просмотр меню (showMenu()):**

* Если пользователь выбирает опцию просмотра меню, на экран выводится список доступных пицц, напитков и снэков с их ценами.

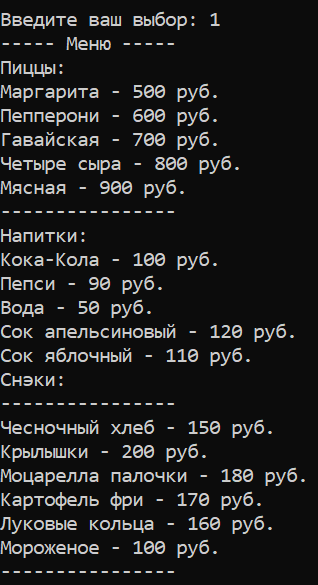


Рисунок 5. Отображение ассортимента пиццерии

1. **Просмотр скидок (showDiscounts()):**

* Если пользователь выбирает опцию просмотра скидок, на экран выводится список текущих скидок пиццерии.

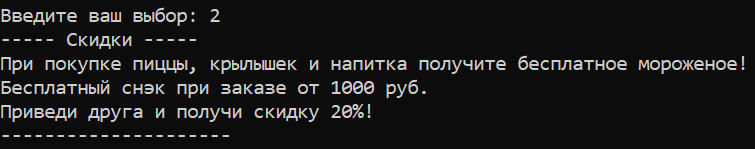


Рисунок 6. Просмотр скидок

1. **Создание заказа (createOrder()):**

* Если пользователь выбирает опцию создания заказа, программа начинает процесс оформления заказа.

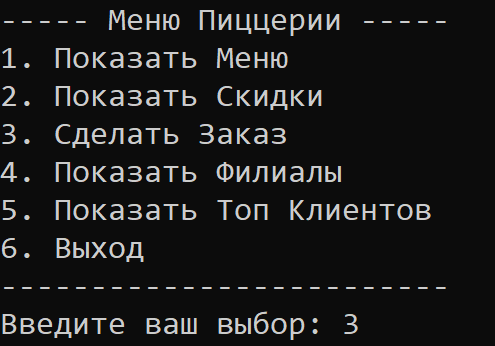


Рисунок 7. Выбор пункта создания заказа

* Пользователю предлагается выбрать филиал, куда он хочет сделать заказ. После выбора филиала создается объект класса Order, который будет хранить информацию о заказе.

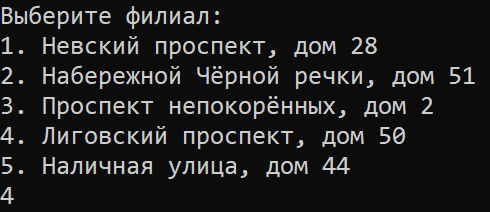


Рисунок 8. Выбор филиала пиццерии

* Пользователю предлагается выбрать категорию товара для добавления в заказ: пиццы, напитки или снэки.

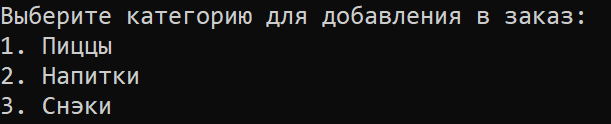


Рисунок 9. Отображение категорий, доступных для добавления в заказ

* После выбора категории пользователь может добавить в заказ необходимое количество товаров, указав их название.

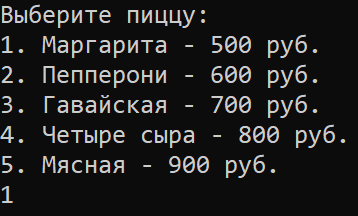


Рисунок 10. Выбор категории пицц

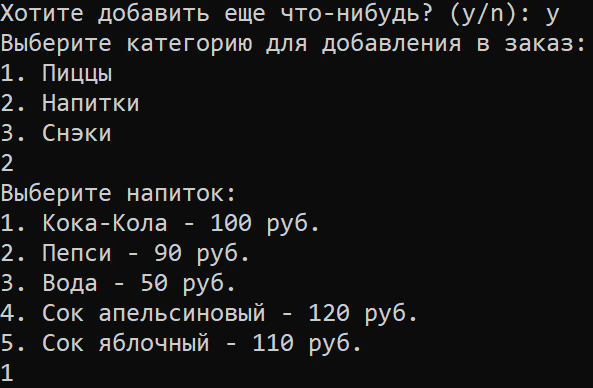


Рисунок 11. Выбор категории напитков

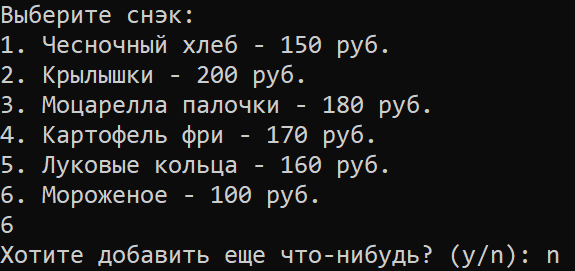


Рисунок 12. Выбор категории снэков

* Пользователю также предоставляется возможность применить промокод, если у него есть.

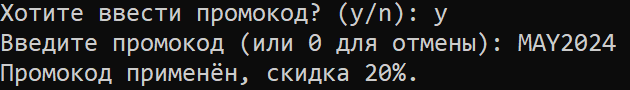


Рисунок 13. Ввод промокода на скидку

* После завершения заказа программа печатает чек, содержащий информацию о заказанных товарах, примененном промокоде и общей сумме заказа.

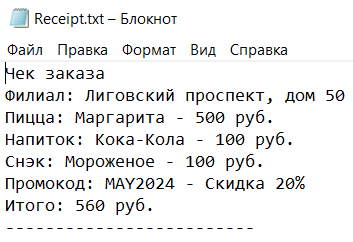


Рисунок 14. Чек заказа, итоговой стоимостью в 560 рублей

1. **Просмотр филиалов (showBranches()):**

* Если пользователь выбирает опцию просмотра филиалов, на экран выводится список всех филиалов пиццерии.

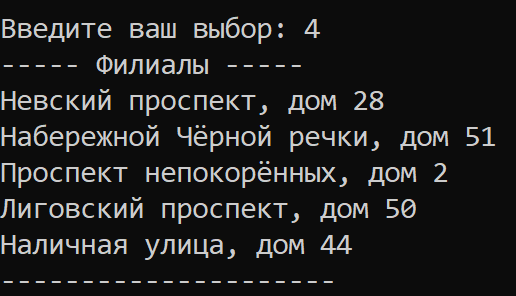


Рисунок 15. Доступные адреса филиалов пиццерии

1. **Просмотр топ клиентов (showTopClients()):**

* Если пользователь выбирает опцию просмотра топ клиентов, на экран выводится информация о самых активных клиентах пиццерии: их имена, любимые блюда, общая сумма потраченных денег и часто посещаемые филиалы.

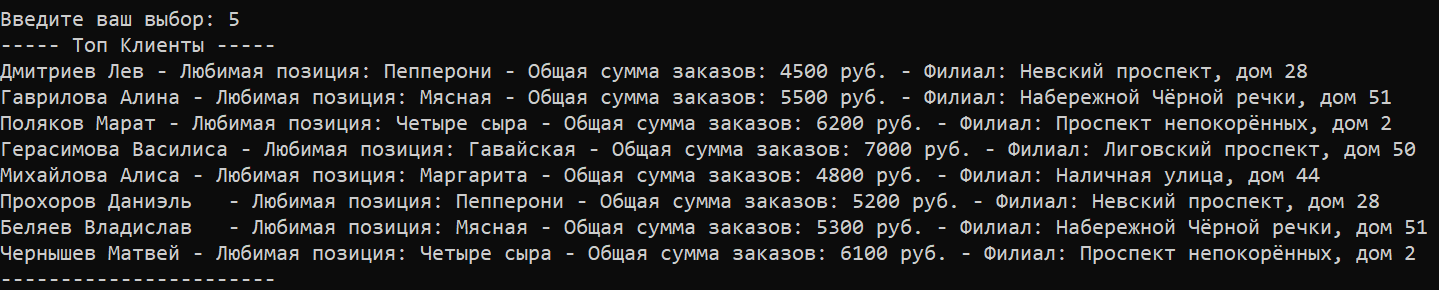


Рисунок 16. Топ самых активных клиентов пиццерии

1. **Выход из программы:**

* Если пользователь выбирает опцию выхода, программа завершает свою работу.

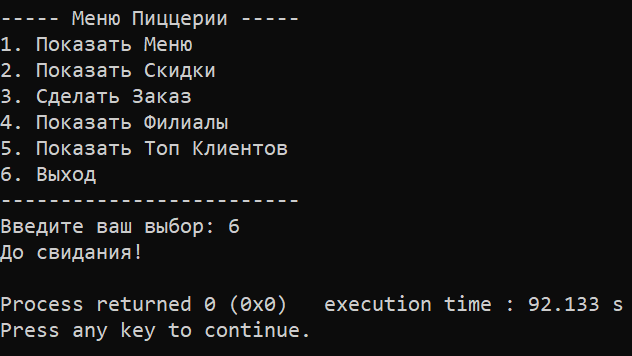


Рисунок 17. Выбор опции выхода из программы

**Заключение**

В ходе разработки данной курсовой работы было создано консольное приложение "Доставка пиццы", моделирующее работу реальной пиццерии. Программа позволяет пользователям просматривать меню, делать заказы, применять скидки и промокоды, а также анализировать активность клиентов.

Разработка этой программы позволила мне глубже понять принципы объектно-ориентированного программирования и их применение на практике. Я научился проектировать классы и структуры данных, адаптировать их для решения конкретных задач. Работа с вводом и выводом данных, включая валидацию пользовательского ввода, помогла мне лучше понять механизмы взаимодействия с пользователем и обеспечить удобный интерфейс приложения.

Обработка ошибок и исключений стала важной частью моего опыта, помогая обеспечить корректное выполнение программы в различных сценариях использования. Анализ требований пользователей и проектирование удобного интерфейса приложения улучшили мои навыки в создании программного продукта, который соответствует потребностям пользователей.

Все это в совокупности улучшило мои навыки программирования и разработки программного обеспечения, приближая меня к статусу опытного разработчика.

**Список источников**

1. Керниган Б. Язык программирования Си / Керниган Б., Ритчи Д. – СПб, 2012. – 352 с.

2. Шилдт, Герберт. С++: методики программирования Шилдта.: Пер. с англ. -

М. : ООО "И.Д. Вильямс", 2017 - 480 с.

3. Страуструп, Бьёрн. Язык программирования С++ (стандарт С++11). Краткий курс. Пер. с англ. – М.: ИД Бином, 2019. – 176 с.

4. Павловская Т. А. C/C++. Программирование на языке высокого уровня. –

СПб.: Питер, 2018. – 461 с.

5. Шилдт, Герберт. С++: базовый курс; 3-е изд.: Пер. с англ. - М. : ООО "И.Д.

Вильямс", 2019 - 624 с.

.

**Приложение**

#include <iostream> // Для вывода на экран и работы с вводом

#include <vector> // Для использования контейнера vector

#include <string> // Для использования строки string

#include <fstream> // Для работы с файлами (чтение/запись)

#include <locale> // Для установки локали

#include <unordered\_set> // Для использования контейнера unordered\_set

#include <limits> // Для работы с пределами числовых типов, очистки ввода

using namespace std;

// Структура для хранения информации о пицце

struct Pizza {

string name; // Название пиццы

double price; // Цена пиццы

};

// Структура для хранения информации о напитках

struct Drink {

string name; // Название напитка

double price; // Цена напитка

};

// Структура для хранения информации о снэках

struct Snack {

string name; // Название снэка

double price; // Цена снэка

};

// Структура для хранения информации о филиале

struct Branch {

string address; // Адрес филиала

};

// Структура для хранения информации о клиенте

struct Client {

string name; // Имя клиента

string favoriteItem; // Любимая позиция клиента

double totalSpent; // Общая сумма заказов клиента

string branch; // Филиал, который клиент выбирает чаще всего

};

// Класс для работы с заказами

class Order {

public:

vector<Pizza> pizzas; // Вектор для хранения пицц в заказе

vector<Drink> drinks; // Вектор для хранения напитков в заказе

vector<Snack> snacks; // Вектор для хранения снэков в заказе

double total; // Общая сумма заказа

string branch; // Филиал, в который делается заказ

string promoCode; // Промокод, если применен

// Конструктор по умолчанию, инициализирующий общую сумму заказа нулем

Order() : total(0) {}

// Метод для добавления пиццы в заказ

void addPizza(const Pizza& pizza) {

pizzas.push\_back(pizza);

total += pizza.price;

}

// Метод для добавления напитка в заказ

void addDrink(const Drink& drink) {

drinks.push\_back(drink);

total += drink.price;

}

// Метод для добавления снэка в заказ

void addSnack(const Snack& snack) {

snacks.push\_back(snack);

total += snack.price;

}

// Метод для применения промокода к заказу

void applyPromoCode(const string& code) {

promoCode = code;

total \*= 0.8; // Применение скидки 20%

}

// Метод для печати чека заказа в файл

void printReceipt() const {

ofstream receipt("Receipt.txt"); // Открытие файла для записи

receipt << "Чек заказа\n";

receipt << "Филиал: " << branch << "\n";

for (const auto& pizza : pizzas) {

receipt << "Пицца: " << pizza.name << " - " << pizza.price << " руб.\n";

}

for (const auto& drink : drinks) {

receipt << "Напиток: " << drink.name << " - " << drink.price << " руб.\n";

}

for (const auto& snack : snacks) {

receipt << "Снэк: " << snack.name << " - " << snack.price << " руб.\n";

}

if (!promoCode.empty()) {

receipt << "Промокод: " << promoCode << " - Скидка 20%\n";

}

receipt << "Итого: " << total << " руб.\n";

receipt << "-------------------------\n";

receipt.close(); // Закрытие файла

}

};

// Класс для работы с пиццерией

class Pizzeria {

public:

vector<Pizza> pizzas; // Вектор для хранения ассортимента пицц

vector<Drink> drinks; // Вектор для хранения ассортимента напитков

vector<Snack> snacks; // Вектор для хранения ассортимента снэков

vector<Branch> branches; // Вектор для хранения адресов филиалов

vector<Client> topClients; // Вектор для хранения информации о топ клиентах

vector<string> discounts; // Вектор для хранения информации о скидках

unordered\_set<string> promoCodes; // Множество для хранения промокодов

// Конструктор класса, инициализирующий данные

Pizzeria() {

// Инициализации ассортиментов пицц, напитков и снеков

pizzas = { {"Маргарита", 500.0}, {"Пепперони", 600.0}, {"Гавайская", 700.0}, {"Четыре сыра", 800.0}, {"Мясная", 900.0} };

drinks = { {"Кока-Кола", 100.0}, {"Пепси", 90.0}, {"Вода", 50.0}, {"Сок апельсиновый", 120.0}, {"Сок яблочный", 110.0} };

snacks = { {"Чесночный хлеб", 150.0}, {"Крылышки", 200.0}, {"Моцарелла палочки", 180.0}, {"Картофель фри", 170.0}, {"Луковые кольца", 160.0}, {"Мороженое", 100.0} };

// Инициализация адресов филиалов

branches = { {"Невский проспект, дом 28"}, {"Набережной Чёрной речки, дом 51"}, {"Проспект непокорённых, дом 2"}, {"Лиговский проспект, дом 50"}, {"Наличная улица, дом 44"} };

// Инициализация топа клиентов

topClients = {

{"Дмитриев Лев", "Пепперони", 4500.0, "Невский проспект, дом 28"},

{"Гаврилова Алина", "Мясная", 5500.0, "Набережной Чёрной речки, дом 51"},

{"Поляков Марат", "Четыре сыра", 6200.0, "Проспект непокорённых, дом 2"},

{"Герасимова Василиса", "Гавайская", 7000.0, "Лиговский проспект, дом 50"},

{"Михайлова Алиса", "Маргарита", 4800.0, "Наличная улица, дом 44"},

{"Прохоров Даниэль ", "Пепперони", 5200.0, "Невский проспект, дом 28"},

{"Беляев Владислав ", "Мясная", 5300.0, "Набережной Чёрной речки, дом 51"},

{"Чернышев Матвей", "Четыре сыра", 6100.0, "Проспект непокорённых, дом 2"},

};

// Инициализация скидок

discounts = { "При покупке пиццы, крылышек и напитка получите бесплатное мороженое!", "Бесплатный снэк при заказе от 1000 руб.", "Приведи друга и получи скидку 20%!" };

// Загрузка промокодов из файла

loadPromoCodes();

}

// Метод для отображения меню

void showMenu() const {

cout << "----- Меню -----\n";

cout << "Пиццы:\n";

for (const auto& pizza : pizzas) {

cout << pizza.name << " - " << pizza.price << " руб.\n";

}

cout << "----------------\n";

cout << "Напитки:\n";

for (const auto& drink : drinks) {

cout << drink.name << " - " << drink.price << " руб.\n";

}

cout << "Снэки:\n";

cout << "----------------\n";

for (const auto& snack : snacks) {

cout << snack.name << " - " << snack.price << " руб.\n";

}

cout << "----------------\n";

}

// Метод для отображения скидок

void showDiscounts() const {

cout << "----- Скидки -----\n";

for (const auto& discount : discounts) {

cout << discount << endl;

}

cout << "---------------------\n";

}

// Метод для отображения филиалов

void showBranches() const {

cout << "----- Филиалы -----\n";

for (const auto& branch : branches) {

cout << branch.address << endl;

}

cout << "---------------------\n";

}

// Метод для отображения топ клиентов

void showTopClients() const {

cout << "----- Топ Клиенты -----\n";

for (const auto& client : topClients) {

cout << client.name << " - Любимая позиция: " << client.favoriteItem

<< " - Общая сумма заказов: " << client.totalSpent << " руб."

<< " - Филиал: " << client.branch << "\n";

}

cout << "-----------------------\n";

}

// Метод для создания заказа

void createOrder() {

Order order; // Создание нового заказа

chooseBranch(order); // Выбор филиала для заказа

char choice;

do {

cout << "Выберите категорию для добавления в заказ:\n";

cout << "1. Пиццы\n2. Напитки\n3. Снэки\n";

int category;

cin >> category;

// Проверка корректности ввода

while(cin.fail()) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Неверный выбор! Пожалуйста, выберите снова:\n";

cin >> category;

}

switch (category) {

case 1:

addPizzaToOrder(order); // Добавление пиццы в заказ

break;

case 2:

addDrinkToOrder(order); // Добавление напитка в заказ

break;

case 3:

addSnackToOrder(order); // Добавление снэка в заказ

break;

default:

cout << "Неверный выбор!\n";

}

cout << "Хотите добавить еще что-нибудь? (y/n): ";

cin >> choice;

} while (choice == 'y'); // Повторение цикла, пока пользователь не выберет 'n'

applyPromoCode(order); // Применение промокода

order.printReceipt(); // Печать чека заказа

cout << "Заказ оформлен. Проверьте 'Receipt.txt' для деталей.\n";

}

private:

// Метод для загрузки промокодов из файла

void loadPromoCodes() {

ifstream promoFile("Promocodes.txt"); // Открытие файла для чтения

if (promoFile.is\_open()) {

string code;

while (getline(promoFile, code)) {

promoCodes.insert(code); // Добавление промокода в множество

}

promoFile.close(); // Закрытие файла

} else {

cout << "Не удалось открыть файл 'Promocodes.txt'.\n";

}

}

// Метод для выбора филиала для заказа

void chooseBranch(Order& order) const {

cout << "Выберите филиал:\n";

for (size\_t i = 0; i < branches.size(); ++i) {

cout << i + 1 << ". " << branches[i].address << "\n";

}

int choice;

cin >> choice;

// Проверка корректности ввода

while(cin.fail() || choice <= 0 || choice > branches.size()) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Неверный выбор! Пожалуйста, выберите снова:\n";

cin >> choice;

}

order.branch = branches[choice - 1].address; // Установка выбранного филиала в заказе

}

// Метод для добавления пиццы в заказ

void addPizzaToOrder(Order& order) const {

cout << "Выберите пиццу:\n";

for (size\_t i = 0; i < pizzas.size(); ++i) {

cout << i + 1 << ". " << pizzas[i].name << " - " << pizzas[i].price << " руб.\n";

}

int choice;

cin >> choice;

// Проверка корректности ввода

while(cin.fail() || choice <= 0 || choice > pizzas.size()) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Неверный выбор! Пожалуйста, выберите снова:\n";

cin >> choice;

}

order.addPizza(pizzas[choice - 1]); // Добавление выбранной пиццы в заказ

}

// Метод для добавления напитка в заказ

void addDrinkToOrder(Order& order) const {

cout << "Выберите напиток:\n";

for (size\_t i = 0; i < drinks.size(); ++i) {

cout << i + 1 << ". " << drinks[i].name << " - " << drinks[i].price << " руб.\n";

}

int choice;

cin >> choice;

// Проверка корректности ввода

while(cin.fail() || choice <= 0 || choice > drinks.size()) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Неверный выбор! Пожалуйста, выберите снова:\n";

cin >> choice;

}

order.addDrink(drinks[choice - 1]); // Добавление выбранного напитка в заказ

}

// Метод для добавления снэка в заказ

void addSnackToOrder(Order& order) const {

cout << "Выберите снэк:\n";

for (size\_t i = 0; i < snacks.size(); ++i) {

cout << i + 1 << ". " << snacks[i].name << " - " << snacks[i].price << " руб.\n";

}

int choice;

cin >> choice;

// Проверка корректности ввода

while(cin.fail() || choice <= 0 || choice > snacks.size()) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Неверный выбор! Пожалуйста, выберите снова:\n";

cin >> choice;

}

order.addSnack(snacks[choice - 1]); // Добавление выбранного снэка в заказ

}

// Метод для применения промокода к заказу

void applyPromoCode(Order& order) {

cout << "Хотите ввести промокод? (y/n): ";

char choice;

cin >> choice;

if (choice == 'y') {

string promoCode;

while (true) {

cout << "Введите промокод (или 0 для отмены): ";

cin >> promoCode;

if (promoCode == "0") {

cout << "Промокод не применён.\n";

break;

}

if (promoCodes.find(promoCode) != promoCodes.end()) {

order.applyPromoCode(promoCode); // Применение промокода, если он найден

cout << "Промокод применён, скидка 20%.\n";

break;

} else {

cout << "Неверный промокод! Попробуйте снова.\n";

}

}

}

}

};

// Основная функция

int main() {

setlocale(LC\_ALL, "Russian"); // Установка локали для поддержки русского языка

Pizzeria pizzeria; // Создание объекта класса Pizzeria

int choice;

// Основной цикл меню

do {

cout << "----- Меню Пиццерии -----\n";

cout << "1. Показать Меню\n";

cout << "2. Показать Скидки\n";

cout << "3. Сделать Заказ\n";

cout << "4. Показать Филиалы\n";

cout << "5. Показать Топ Клиентов\n";

cout << "6. Выход\n";

cout << "-------------------------\n";

cout << "Введите ваш выбор: ";

cin >> choice;

// Проверка корректности ввода

while(cin.fail()) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Неверный выбор! Попробуйте снова: ";

cin >> choice;

}

// Обработка выбора пользователя

switch (choice) {

case 1:

pizzeria.showMenu(); // Отображение меню

break;

case 2:

pizzeria.showDiscounts(); // Отображение скидок

break;

case 3:

pizzeria.createOrder(); // Создание заказа

break;

case 4:

pizzeria.showBranches(); // Отображение филиалов

break;

case 5:

pizzeria.showTopClients(); // Отображение топ клиентов

break;

case 6:

cout << "До свидания!\n"; // Выход из программы

break;

default:

cout << "Неверный выбор! Попробуйте снова.\n";

}

} while (choice != 6); // Повторение цикла, пока пользователь не выберет выход

return 0;

}