**ПОЯСНЕНИЕ КОДА**

**Практическая работа №8. Симулятор Пиццерии**

**Описание программы:**

Программа имитирует процесс заказа в пиццерии и включает в себя управление меню, возможность кастомизации пиццы, обработку разных типов заказов, а также управление списком клиентов.

РАЗБОР КОДА

#include "Pizzeria.h"

#include <iostream>

#include <limits>

void clearInputBuffer() {

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

}

**Очищает буфера ввода, чтобы избежать ошибок при вводе данных пользователем**

void printMainMenu() {

std::cout << "\n";

std::cout << "=====================================\n";

std::cout << " PIZZERIA SIMULATOR \n";

std::cout << "=====================================\n";

std::cout << " 1.View Menu\n";

std::cout << " 2.Place Order\n";

std::cout << " 3.Add Customer\n";

std::cout << " 4.View Orders\n";

std::cout << " 5.Exit\n";

std::cout << "=====================================\n";

std::cout << "Enter your choice: ";

}

**Отрисовка главного меню**

int main() {

Pizzeria pizzeria;

int choice;

while (true) {

printMainMenu();

std::cin >> choice;

**Бесконечный цикл – программа будет работать пока пользователь сам не выберет выход**

if (std::cin.fail()) {

std::cin.clear();

clearInputBuffer();

std::cout << "Invalid input. Please enter a number.\n";

continue;

}

**Если пользователь ввёл что то не то - очищает ошибку, буфера. Сообщение об ошибке**

switch (choice) {

case 1:

pizzeria.displayMenu();

break;

case 2:

pizzeria.createOrderInteractively();

break;

case 3:

pizzeria.createCustomerInteractively();

break;

case 4:

pizzeria.displayOrders();

break;

case 5:

std::cout << "\nThank you for using Pizzeria Simulator! Goodbye!\n";

return 0;

default:

std::cout << "Invalid choice. Please try again.\n";

}

}

return 0;

}

**Проверяет что именно выбрал пользователь, если выбрал номер которого не существует – сообщит об ошибке**

#include "Customer.h" **подключение описания класса**

Customer::Customer(const std::string& n, const std::string& contact)

: name(n), contactInfo(contact) {}

**Создает новый клиент**

std::string Customer::getName() const {

return name;

}

**Возвращает имя клиента**

std::string Customer::getContactInfo() const {

return contactInfo;

}

**Возвращает контактную информацию**

void Customer::addOrder(const Order& order) {

orders.push\_back(order);

}

**Принимает заказ**

void Customer::displayOrders() const {

std::cout << "\nCustomer: " << name << ", Contact: " << contactInfo << "\nOrders:\n";

for (const auto& order : orders) {

order.display();

}

}

**Показать все заказы клиентов**

#include "Drink.h" – **подключение описания класса**

Drink::Drink(const std::string& n, double price, const std::string& vol)

: MenuItem(n, price), volume(vol) {

std::cout << "Drink '" << name << "' created." << std::endl;

}

**Создание объект и передает название цена объем**

double Drink::calculatePrice() const {

return basePrice;

}

**Рассчитать цену напитка**

void Drink::display() const {

std::cout << "Drink: " << name << ", Volume: " << volume

<< ", Price: $" << calculatePrice() << std::endl;

}

**Показывает информацию о напитке**

#include "Menu.h" – подключение

Menu::Menu() {

availablePizzas.emplace\_back("Margherita", 10.0, Size::MEDIUM, BaseType::TRADITIONAL);

availablePizzas.emplace\_back("Pepperoni", 12.0, Size::LARGE, BaseType::THIN);

availableDrinks.emplace\_back("Cola", 2.0, "0.5L");

availableDrinks.emplace\_back("Juice", 2.5, "1L");

availableSideDishes.emplace\_back("Fries", 3.0, "Small");

availableSideDishes.emplace\_back("Salad", 4.0, "Large");

availableToppings.emplace\_back("Cheese", 1.0);

availableToppings.emplace\_back("Mushrooms", 1.5);

}

**При создании объекта, в нем сразу добавляются стандартные позиции**

void Menu::addPizza(const Pizza& pizza) {

availablePizzas.push\_back(pizza);

}

Добавление новой пиццы

void Menu::addDrink(const Drink& drink) {

availableDrinks.push\_back(drink);

}

**Добавление нового напитка**

void Menu::addSideDish(const SideDish& sideDish) {

availableSideDishes.push\_back(sideDish);

}

**Добавление нового гарнира**

void Menu::addTopping(const Topping& topping) {

availableToppings.push\_back(topping);

}

**Добавление нового топпинга**

void Menu::displayMenu() const {

std::cout << "\n--- Menu ---\n";

std::cout << "Pizzas:\n";

for (const auto& pizza : availablePizzas) {

pizza.display();

}

std::cout << "\nDrinks:\n";

for (const auto& drink : availableDrinks) {

drink.display();

}

std::cout << "\nSide Dishes:\n";

for (const auto& sideDish : availableSideDishes) {

sideDish.display();

}

std::cout << "\nToppings:\n";

for (const auto& topping : availableToppings) {

std::cout << topping.name << ": $" << topping.price << std::endl;

}

}

**Печатает всё меню**

const std::vector<Pizza>& Menu::getAvailablePizzas() const {

return availablePizzas;

}

**Показывает список пиццы**

const std::vector<Drink>& Menu::getAvailableDrinks() const {

return availableDrinks;

}

**Список напитка**

const std::vector<SideDish>& Menu::getAvailableSideDishes() const {

return availableSideDishes;

}

**Список гарнира**

const std::vector<Topping>& Menu::getAvailableToppings() const {

return availableToppings;

}

**Список топпинга**

#include "MenuItem.h"

MenuItem::MenuItem(const std::string& n, double price) : name(n), basePrice(price) {

std::cout << "MenuItem '" << name << "' created." << std::endl;

}

**Создает объект и задает название и цену**

std::string MenuItem::getName() const {

return name;

}

double MenuItem::getBasePrice() const {

return basePrice;

}

**Получить название и цену**

#include "Order.h"

Order::Order(int id, OrderType type) : orderId(id), orderType(type) {}

**Инициализирует объект заказа с уникальным ID**

void Order::addPizza(const Pizza& pizza) {

orderedPizzas.push\_back(pizza);

}

void Order::addDrink(const Drink& drink) {

orderedDrinks.push\_back(drink);

}

void Order::addSideDish(const SideDish& sideDish) {

orderedSideDishes.push\_back(sideDish);

}

Добавление пиццы напитка гарнира

double Order::calculateTotal() const {

double total = 0.0;

for (const auto& pizza : orderedPizzas) {

total += pizza.calculatePrice();

}

for (const auto& drink : orderedDrinks) {

total += drink.calculatePrice();

}

for (const auto& sideDish : orderedSideDishes) {

total += sideDish.calculatePrice();

}

if (orderType == OrderType::DELIVERY) {

total += 5.0; // Доставка

}

return total;

}

**Этот метод рассчитывает общую стоимость заказа, складывая цену каждой пиццы, напитка и гарнира.**

void Order::display() const {

std::cout << "\nOrder #" << orderId << " (" << (orderType == OrderType::DINE\_IN ? "Dine-in" :

orderType == OrderType::TAKEOUT ? "Takeout" : "Delivery") << ")\n";

std::cout << "Pizzas:\n";

for (const auto& pizza : orderedPizzas) {

pizza.display();

}

std::cout << "Drinks:\n";

for (const auto& drink : orderedDrinks) {

drink.display();

}

std::cout << "Side Dishes:\n";

for (const auto& sideDish : orderedSideDishes) {

sideDish.display();

}

std::cout << "Total: $" << calculateTotal() << std::endl;

}

**Этот метод выводит информацию о заказе на экран:**

#include "Pizza.h"

Pizza::Pizza(const std::string& n, double price, Size s, BaseType bt)

: MenuItem(n, price), size(s), baseType(bt) {

std::cout << "Pizza '" << name << "' created." << std::endl;

}

**Конструктор инициализирует пиццу с ее названием, ценой, размером**

void Pizza::addTopping(const Topping& t) {

toppings.push\_back(t);

}

void Pizza::setSize(Size s) {

size = s;

}

void Pizza::setBaseType(BaseType bt) {

baseType = bt;

}

**Добавление топпинга, размер, типы основы**

double Pizza::calculatePrice() const {

double finalPrice = basePrice;

for (const auto& t : toppings) {

finalPrice += t.price;

}

switch (size) {

case Size::SMALL: finalPrice \*= 0.8; break;

case Size::LARGE: finalPrice \*= 1.2; break;

default: break;

}

return finalPrice;

}

void Pizza::display() const {

std::cout << "Pizza: " << name << ", Base Price: $" << basePrice;

std::cout << ", Size: ";

switch (size) {

case Size::SMALL: std::cout << "Small"; break;

case Size::MEDIUM: std::cout << "Medium"; break;

case Size::LARGE: std::cout << "Large"; break;

}

std::cout << ", Base: ";

switch (baseType) {

case BaseType::THIN: std::cout << "Thin"; break;

case BaseType::TRADITIONAL: std::cout << "Traditional"; break;

case BaseType::THICK: std::cout << "Thick"; break;

}

std::cout << ", Toppings: ";

if (toppings.empty()) {

std::cout << "None";

} else {

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

std::cout << toppings[i].name << (i == toppings.size() - 1 ? "" : ", ");

}

}

std::cout << ", Total Price: $" << calculatePrice() << std::endl;

}

**рассчитывает итоговую цену пиццы:**

#include "Pizzeria.h"

#include <iostream>

#include <limits>

Pizzeria::Pizzeria() : nextOrderId(1) {}

void Pizzeria::addCustomer(const Customer& customer) {

customers.push\_back(customer);

}

**Добавление клиента**

void Pizzeria::placeOrder(const Customer& customer, const Order& order) {

orders.push\_back(order);

for (auto& c : customers) {

if (c.getName() == customer.getName()) {

c.addOrder(order);

}

}

nextOrderId++;

}

**Оформление заказа**

void Pizzeria::displayMenu() const {

menu.displayMenu();

}

void Pizzeria::displayOrders() const {

std::cout << "\nAll Orders:\n";

for (const auto& order : orders) {

order.display();

}

}

**Отображение меню и всех заказов**

const Menu& Pizzeria::getMenu() const {

return menu;

}

**Получение меню**

void Pizzeria::createCustomerInteractively() {

std::string name, contact;

std::cout << "\nEnter customer name: ";

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

std::getline(std::cin, name);

std::cout << "Enter contact info (e.g., email or phone): ";

std::getline(std::cin, contact);

Customer customer(name, contact);

addCustomer(customer);

std::cout << "Customer '" << name << "' added.\n";

}

**позволяет интерактивно создать нового клиента, запрашивая его имя и контактные данные, после чего добавляет клиента в пиццерию.**

void Pizzeria::createOrderInteractively() {

if (customers.empty()) {

std::cout << "No customers available. Please add a customer first.\n";

return;

}

std::cout << "\nAvailable customers:\n";

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

std::cout << i + 1 << ". " << customers[i].getName() << " (" << customers[i].getContactInfo() << ")\n";

}

std::cout << "Select customer by number: ";

int customerIndex;

std::cin >> customerIndex;

if (customerIndex < 1 || customerIndex > static\_cast<int>(customers.size())) {

std::cout << "Invalid customer selection.\n";

return;

}

Customer& selectedCustomer = customers[customerIndex - 1];

std::cout << "\nSelect order type:\n1. Dine-in\n2. Takeout\n3. Delivery\n";

int orderTypeChoice;

std::cin >> orderTypeChoice;

OrderType orderType;

switch (orderTypeChoice) {

case 1: orderType = OrderType::DINE\_IN; break;

case 2: orderType = OrderType::TAKEOUT; break;

case 3: orderType = OrderType::DELIVERY; break;

default: std::cout << "Invalid order type.\n"; return;

}

Order order(nextOrderId, orderType);

while (true) {

std::cout << "\nAdd a pizza? (y/n): ";

char choice;

std::cin >> choice;

if (choice != 'y' && choice != 'Y') break;

std::cout << "\nAvailable pizzas:\n";

const auto& pizzas = menu.getAvailablePizzas();

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

std::cout << i + 1 << ". ";

pizzas[i].display();

}

std::cout << "Select pizza by number: ";

int pizzaIndex;

std::cin >> pizzaIndex;

if (pizzaIndex < 1 || pizzaIndex > static\_cast<int>(pizzas.size())) {

std::cout << "Invalid pizza selection.\n";

continue;

}

Pizza pizza = pizzas[pizzaIndex - 1];

std::cout << "\nSelect size:\n1. Small\n2. Medium\n3. Large\n";

int sizeChoice;

std::cin >> sizeChoice;

switch (sizeChoice) {

case 1: pizza.setSize(Size::SMALL); break;

case 2: pizza.setSize(Size::MEDIUM); break;

case 3: pizza.setSize(Size::LARGE); break;

default: std::cout << "Invalid size, using Medium.\n"; break;

}

std::cout << "\nSelect base type:\n1. Thin\n2. Traditional\n3. Thick\n";

int baseChoice;

std::cin >> baseChoice;

switch (baseChoice) {

case 1: pizza.setBaseType(BaseType::THIN); break;

case 2: pizza.setBaseType(BaseType::TRADITIONAL); break;

case 3: pizza.setBaseType(BaseType::THICK); break;

default: std::cout << "Invalid base type, using Traditional.\n"; break;

}

while (true) {

std::cout << "\nAdd a topping? (y/n): ";

std::cin >> choice;

if (choice != 'y' && choice != 'Y') break;

std::cout << "\nAvailable toppings:\n";

const auto& toppings = menu.getAvailableToppings();

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

std::cout << i + 1 << ". " << toppings[i].name << " ($" << toppings[i].price << ")\n";

}

std::cout << "Select topping by number: ";

int toppingIndex;

std::cin >> toppingIndex;

if (toppingIndex < 1 || toppingIndex > static\_cast<int>(toppings.size())) {

std::cout << "Invalid topping selection.\n";

continue;

}

pizza.addTopping(toppings[toppingIndex - 1]);

}

order.addPizza(pizza);

}

while (true) {

std::cout << "\nAdd a drink? (y/n): ";

char choice;

std::cin >> choice;

if (choice != 'y' && choice != 'Y') break;

std::cout << "\nAvailable drinks:\n";

const auto& drinks = menu.getAvailableDrinks();

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

std::cout << i + 1 << ". ";

drinks[i].display();

}

std::cout << "Select drink by number: ";

int drinkIndex;

std::cin >> drinkIndex;

if (drinkIndex < 1 || drinkIndex > static\_cast<int>(drinks.size())) {

std::cout << "Invalid drink selection.\n";

continue;

}

order.addDrink(drinks[drinkIndex - 1]);

}

while (true) {

std::cout << "\nAdd a side dish? (y/n): ";

char choice;

std::cin >> choice;

if (choice != 'y' && choice != 'Y') break;

std::cout << "\nAvailable side dishes:\n";

const auto& sideDishes = menu.getAvailableSideDishes();

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

std::cout << i + 1 << ". ";

sideDishes[i].display();

}

std::cout << "Select side dish by number: ";

int sideDishIndex;

std::cin >> sideDishIndex;

if (sideDishIndex < 1 || sideDishIndex > static\_cast<int>(sideDishes.size())) {

std::cout << "Invalid side dish selection.\n";

continue;

}

order.addSideDish(sideDishes[sideDishIndex - 1]);

}

placeOrder(selectedCustomer, order);

std::cout << "\nOrder placed successfully!\n";

order.display();

}

**позволяет создать заказ для выбранного клиента. Процесс создания заказа проходит несколько этапов:**

* **Сначала проверяется, есть ли клиенты. Если нет — выводится сообщение об ошибке.**
* **Затем пользователю показываются все доступные клиенты, и он может выбрать одного.**
* **Далее выбирается тип заказа (питание в ресторане, на вынос или доставка).**
* **После этого можно добавить пиццу в заказ, выбрав размер, тип основы и начинки.**
* **В процессе добавления пиццы пользователю предлагается выбрать напитки и гарниры, которые можно добавить в заказ.**

#include "SideDish.h"

SideDish::SideDish(const std::string& n, double price, const std::string& portion)

: MenuItem(n, price), portionSize(portion) {

std::cout << "SideDish '" << name << "' created." << std::endl;

}

**Принимает 3 параметры название гарниры цены размер порции**

double SideDish::calculatePrice() const {

double price = basePrice;

if (portionSize == "Large") {

price \*= 1.5;

}

return price;

}

**Вычисляет цену гарнира**

void SideDish::display() const {

std::cout << "SideDish: " << name << ", Portion: " << portionSize

<< ", Price: $" << calculatePrice() << std::endl;

}

**выводит на экран информацию о гарнире, включая его название, размер порции и цену (вычисленную с использованием метода calculatePrice()).**