МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій

Кафедра комп'ютерної інженерії та інформаційних систем

Лабораторна робота № 4

з дисципліни

«Об'єктно-орієнтоване програмування»

на тему :

«Наслідування, множинне наслідування»

Виконала:

студентка 2 курсу, групи КІ2-23-4 Атанелова А.Т.

(Підпис)

Перевірив: Кисіль В.В.

(Підпис)

Хмельницький – 2024

**Завдання:**

Третій рівень

До проєкту з Лабораторної роботи №3 додається функціонал:

1. Визначити ієрархію класів відповідно до варіанта (не менше 8 полів на клас).

2. Продемонструвати керування доступу до членів базового класу (трьох методів та трьох даних) у трьох

варіантах: public,private,protected, шляхом генерації об’єктів конструкторами).

3. Визначити додатковий похідний клас (подати самостійно по відношенню до варіанту), який демонструє

закритий спосіб наслідування із створенням об’єктів конструкторами; об’єкти , які оперують методами з

різних класів ієрархії.

4. Побудувати 3 додаткових класів в наявну ієрархію, яка дозволить реалізувати множинне наслідування з

подальшою побудовою відповідних об’єктів.

5. Продемонструвати передачу параметрів конструктору базового класу (не менше 5-ти аргументів);побудувати

відповідні об’єкти, а також побудувати конструктор похідного класу, який використовує всі параметри, які

вказані в його оголошенні ( передача параметрів конструктору базового класу відбувається всередині

похідного класу).

6. Продемонструвати доступ до членів та методів базового класу з похідних класів засобами оператора

глобального доступу та оператора using.

7. Продемонструвати для усіх класів порядок виклику конструкторів та деструкторів як для простого наслідування , так і для множинного.

8. Визначити в класі деструктор з метою відслідковування входження у блоки програми та виходу з них, для

якого визначений об'єкт цього класу.

9. Продемонструвати сценарій функціонування системи на прикладі створення усіх необхідних об’єктів, їх

функціонування та взаємодії.

10. В тексті програми вказати в коментарях реалізацію кожного пункту даного завдання.

11. В звіті графічно подати ієрархію класів за допомогою UML-діаграми (діаграма класів UML DESIGNER).

12. Оформлення програмного коду повинно дотримуватися правил code style

(https://google.github.io/styleguide/cppguide.html).

13. Програма використає такі файли: заголовні h-файл із визначенням класів, cpp-файли із реалізаціями кожного

класів, сpp-файл демонстраційної програми.

14. Завантажити проект на GitHub.

**C++ код програми:**

*Administrator.h:*

#ifndef ADMINISTRATOR\_H

#define ADMINISTRATOR\_H

#include <iostream>

#include <vector>

#include <string>

class Administrator {

private:

std::string name;

std::string email;

std::string password;

int accessLevel;

float salary;

float timeOnline;

int numberOfProcessedOrders;

int rating;

public:

// Конструктор за замовчуванням

Administrator() : accessLevel(0), salary(0.0f), timeOnline(0.0f), numberOfProcessedOrders(0), rating(0) {}

// Конструктор з параметрами

Administrator(const std::string& name, const std::string& email, const std::string& password,

int accessLevel, float salary, float timeOnline, int numberOfProcessedOrders, int rating)

: name(name), email(email), password(password), accessLevel(accessLevel),

salary(salary), timeOnline(timeOnline), numberOfProcessedOrders(numberOfProcessedOrders), rating(rating) {}

~Administrator() {}

// Геттери

std::string getName() const { return name; }

std::string getEmail() const { return email; }

std::string getPassword() const { return password; }

int getAccessLevel() const { return accessLevel; }

float getSalary() const { return salary; }

float getTimeOnline() const { return timeOnline; }

int getNumberOfProcessedOrders() const { return numberOfProcessedOrders; }

int getRating() const { return rating; }

// Сеттери

void setName(const std::string& name) { this->name = name; }

void setEmail(const std::string& email) { this->email = email; }

void setPassword(const std::string& password) { this->password = password; }

void setAccessLevel(int accessLevel) { this->accessLevel = accessLevel; }

void setSalary(float salary) { this->salary = salary; }

void setTimeOnline(float timeOnline) { this->timeOnline = timeOnline; }

void setNumberOfProcessedOrders(int numberOfProcessedOrders) { this->numberOfProcessedOrders = numberOfProcessedOrders; }

void setRating(int rating) { this->rating = rating; }

};

#endif

*Cart.h:*

#ifndef CART\_H

#define CART\_H

#include <iostream>

#include <vector>

#include "Product.h"

#include "Client.h"

class Cart {

private:

int cartID;

Client client;

float totalAmount;

int itemCount;

float discount;

std::string status;

std::string creationDate;

std::vector<Product> products; // Вектор для зберігання продуктів у кошику

public:

// Конструктор базового класу

Cart(int cartID, const Client& client, float totalAmount = 0.0, float discount = 0.0, const std::string& status = "Active", const std::string& creationDate = "")

: cartID(cartID), client(client), totalAmount(totalAmount), discount(discount), status(status), creationDate(creationDate) {}

// Деструктор для очищення кошика

~Cart() {

products.clear(); // Очищення продуктів

std::cout << "Cart cleared." << std::endl;

}

// Геттери

int getCartID() const { return cartID; }

Client getClient() const { return client; }

float getTotalAmount() const { return totalAmount; }

int getItemCount() const { return itemCount; }

float getDiscount() const { return discount; }

std::string getStatus() const { return status; }

std::string getCreationDate() const { return creationDate; }

std::vector<Product> getProducts() const { return products; }

// Сеттери

void setTotalAmount(float amount) { totalAmount = amount; }

void setItemCount(int count) { itemCount = count; }

void setDiscount(float discount) { this->discount = discount; }

void setStatus(const std::string& status) { this->status = status; }

void setCreationDate(const std::string& date) { creationDate = date; }

// Метод для отримання загальної вартості

float getTotalCost() const {

float total = 0.0;

for (const auto& product : products) {

total += product.getPrice() \* product.getQuantity(); // Передбачається, що є getPrice() та getQuantity() для товарів

}

return total;

}

// Метод для отримання кількості товарів у кошику

int getItemCountInCart() const {

return products.size(); // Повертає кількість продуктів у векторі

}

// Метод для додавання продукту до кошика

void addProduct(const Product& product) {

products.push\_back(product); // Додає продукт до вектора

}

// Метод для відображення продуктів у кошику

void displayProducts() const {

std::cout << "Products in Cart:" << std::endl;

for (const auto& product : products) {

std::cout << "- " << product.getName() << " (Quantity: " << product.getQuantity() << ", Price: " << product.getPrice() << ")" << std::endl;

}

}

};

#endif

*CartOrder.h:*

#ifndef CART\_ORDER\_H

#define CART\_ORDER\_H

#include "Cart.h"

#include "Order.h"

class CartOrder : public Cart, public Order {

public:

// Конструктор

CartOrder(const Cart& cart, const Order& order)

: Cart(cart), Order(order) {}

~CartOrder() {}

// Метод для відображення інформації про замовлення з кошика

void displayCartOrderInfo() const {

std::cout << "Cart Order Information:" << std::endl;

std::cout << "Order ID: " << getOrderID() << std::endl;

std::cout << "Items in Cart: " << getItemCount() << std::endl;

std::cout << "Total Price: " << getTotalPrice() << std::endl;

}

};

#endif

*Client.h:*

#ifndef CLIENT\_H

#define CLIENT\_H

#include <iostream>

#include <string>

#include <fstream>

#include <vector>

#include <algorithm>

#include "Product.h"

#include "Order.h"

#include "PaymentMethod.h"

class Client {

private:

std::string name;

int clientID;

std::string email;

std::string phone;

std::string address;

PaymentMethod paymentMethod;

std::string membershipType;

float totalSpent;

protected:

//2)

std::string lastName;

public:

//2)

int number;

// Конструктор

Client() : clientID(0), totalSpent(0.0f) {}

// Конструктор з параметрами

Client(const std::string& name, int clientID, const std::string& email, const std::string& phone,

const std::string& address, const PaymentMethod& paymentMethod, const std::string& membershipType, float totalSpent)

: name(name), clientID(clientID), email(email), phone(phone), address(address),

paymentMethod(paymentMethod), membershipType(membershipType), totalSpent(totalSpent) {}

Client(const std::string& lastName, const std::string& email, int number) : lastName(lastName), email(email), number(number) {}

// Деструктор

~Client() {

std::cout << "Client destructor called for Client ID: " << clientID << std::endl;

}

// Геттери

std::string getName() const { return name; }

int getClientID() const { return clientID; }

std::string getEmail() const { return email; }

std::string getPhone() const { return phone; }

std::string getAddress() const { return address; }

PaymentMethod getPaymentMethod() const { return paymentMethod; }

std::string getMembershipType() const { return membershipType; }

float getTotalSpent() const { return totalSpent; }

std::string getLastName() const { return lastName; }

int getNumber() const { return number; }

// Сеттери

void setName(const std::string& name) { this->name = name; }

void setClientID(int clientID) { this->clientID = clientID; }

void setEmail(const std::string& email) { this->email = email; }

void setPhone(const std::string& phone) { this->phone = phone; }

void setAddress(const std::string& address) { this->address = address; }

void setPaymentMethod(const PaymentMethod& paymentMethod) { this->paymentMethod = paymentMethod; }

void setMembershipType(const std::string& membershipType) { this->membershipType = membershipType; }

void setTotalSpent(float totalSpent) { this->totalSpent = totalSpent; }

// Публічний метод

void displayInfo() const {

std::cout << "Name: " << name << ", Client ID: " << clientID << ", Email: " << email << std::endl;

}

};

#endif

*ClientOrder.h:*

#ifndef CLIENT\_ORDER\_H

#define CLIENT\_ORDER\_H

#include "Client.h"

#include "Order.h"

#include "PaymentMethod.h"

// 4) Множинне наслідування

class ClientOrder : public Client, public Order, public PaymentMethod {

public:

// 8) Конструктор за замовчуванням

ClientOrder() : Client(), Order(), PaymentMethod() {}

// 8) Конструктор з параметрами

ClientOrder(const std::string& clientName, int clientID, const std::string& email, const std::string& phone,

const std::string& address, const PaymentMethod& paymentMethod, const std::string& membershipType,

float totalSpent, int orderID, const Product& product, int quantity,

const PaymentMethod& orderPaymentMethod, bool isPaid, const std::string& deliveryAddress,

const std::string& orderStatus)

: Client(clientName, clientID, email, phone, address, paymentMethod, membershipType, totalSpent),

Order(orderID, product, quantity, orderPaymentMethod, isPaid, deliveryAddress, orderStatus) {}

// 7) Деструктор множинного наслідування

~ClientOrder() {

// Можна вказати код для звільнення ресурсів, якщо це необхідно

std::cout << "ClientOrder destructor called." << std::endl;

}

// Методи для отримання інформації про клієнта та замовлення

void displayClientOrderInfo() const {

std::cout << "Client Name: " << getName() << "\n"

<< "Client ID: " << getClientID() << "\n"

<< "Order ID: " << getOrderID() << "\n"

<< "Product: " << getProduct().getName() << "\n"

<< "Quantity: " << getQuantity() << "\n"

<< "Total Price: " << getTotalPrice() << "\n"

<< "Payment Method: " << Order::getPaymentMethod().getMethodName() << "\n"

<< "Delivery Address: " << getDeliveryAddress() << "\n"

<< "Order Status: " << getOrderStatus() << std::endl;

}

};

#endif

*DiscountedCart.h:*

#ifndef DISCOUNTED\_CART\_H

#define DISCOUNTED\_CART\_H

#include "Cart.h"

#include <string>

#include <ctime>

// 3) Додатковий похідний клас, який демонструє закритий спосіб наслідування із створенням об’єктів конструкторами

class DiscountedCart : private Cart {

private:

float discountPercentage;

std::string discountCode;

std::time\_t expiryDate;

bool isValidDiscount;

public:

// Конструктор з параметрами

DiscountedCart(int cartID, const Client& client, float discountPercentage, const std::string& discountCode, std::time\_t expiryDate, float totalAmount = 0.0, float discount = 0.0, const std::string& status = "Active", const std::string& creationDate = "")

: Cart(cartID, client, totalAmount, discount, status, creationDate), // Виклик конструктора базового класу

discountPercentage(discountPercentage), discountCode(discountCode), expiryDate(expiryDate), isValidDiscount(true) {}

~DiscountedCart() {}

void applyDiscount() {

if (validateDiscount()) {

float totalCost = getTotalCost();

float discountedCost = totalCost - (totalCost \* (discountPercentage / 100));

std::cout << "Total after discount: " << discountedCost << std::endl;

}

else {

std::cout << "Discount expired or invalid.\n";

}

}

bool validateDiscount() {

std::time\_t now = std::time(nullptr);

isValidDiscount = (now <= expiryDate);

return isValidDiscount;

}

};

#endif

*Order.h:*

#ifndef ORDER\_H

#define ORDER\_H

#include <iostream>

#include <string>

#include <fstream>

#include <vector>

#include <algorithm>

#include "Product.h"

#include "PaymentMethod.h"

class Order {

private:

int orderID; // ID замовлення

Product product; //продукт

int quantity; //кількість

double totalPrice; //загальна вартість

PaymentMethod paymentMethod; //метод оплати

bool isPaid; //статус оплати

std::string deliveryAddress; //адреса доставки

std::string orderStatus; //статус замовлення

static int orderCount; // статичне поле для підрахунку кількості замовлень

public:

// конструктор без параметрів

Order()

: orderID(0), product(), quantity(0), totalPrice(0.0),

paymentMethod(), isPaid(false), deliveryAddress(""), orderStatus("") {}

// конструктор з параметрами

Order(int id, const Product& p, int q, PaymentMethod pm, bool paid, std::string address, std::string status)

: orderID(id), product(p), quantity(q),

totalPrice(p.getPrice()\* q), paymentMethod(pm),

isPaid(paid), deliveryAddress(address), orderStatus(status) {}

// конструктор копіювання

Order(const Order& other)

: orderID(other.orderID), product(other.product),

quantity(other.quantity), totalPrice(other.totalPrice),

paymentMethod(other.paymentMethod), isPaid(other.isPaid),

deliveryAddress(other.deliveryAddress), orderStatus(other.orderStatus) {}

~Order() {

std::cout << "Order destroyed: ID " << orderID << std::endl;

}

//гет, сет

void setOrderID(int id) { orderID = id; }

int getOrderID() const { return orderID; }

void setProduct(Product p) { product = p; }

Product getProduct() const { return product; }

void setQuantity(int q) { quantity = q; }

int getQuantity() const { return quantity; }

void setTotalPrice(double tp) { totalPrice = tp; }

double getTotalPrice() const { return totalPrice; }

void setPaymentMethod(PaymentMethod pm) { paymentMethod = pm; }

PaymentMethod getPaymentMethod() const { return paymentMethod; }

void setIsPaid(bool paid) { isPaid = paid; }

bool getIsPaid() const { return isPaid; }

void setDeliveryAddress(std::string address) { deliveryAddress = address; }

std::string getDeliveryAddress() const { return deliveryAddress; }

void setOrderStatus(std::string status) { orderStatus = status; }

std::string getOrderStatus() const { return orderStatus; }

};

#endif

*OrderHistory.h:*

#ifndef ORDER\_HISTORY\_H

#define ORDER\_HISTORY\_H

#include <iostream>

#include <vector>

#include "Client.h"

#include "Order.h"

#include "Product.h"

class OrderHistory : public Client {

private:

std::vector<Order> previousOrders; // Зберігає попередні замовлення

public:

// Конструктор

OrderHistory(const std::string& name, int clientID, const std::string& email,

const std::string& phone, const std::string& address,

const PaymentMethod& paymentMethod, const std::string& membershipType, float totalSpent)

: Client(name, clientID, email, phone, address, paymentMethod, membershipType, totalSpent) {}

// 2)

OrderHistory(const std::string& lastName, const std::string& email, int number) : Client(lastName, email, number) {}

// Додає замовлення до історії

void addOrder(const Order& order) {

previousOrders.push\_back(order);

}

// Відображає історію замовлень

void displayOrderHistory() const {

std::cout << "Order History for Client: " << getName() << std::endl;

if (previousOrders.empty()) {

std::cout << "No previous orders." << std::endl;

return;

}

for (const auto& order : previousOrders) {

std::cout << "Order ID: " << order.getOrderID()

<< ", Product: " << order.getProduct().getName()

<< ", Quantity: " << order.getQuantity()

<< ", Total Price: $" << (order.getQuantity() \* order.getProduct().getPrice())

<< std::endl;

}

}

// 2) Додатковий метод для демонстрації доступу в похідному класі з базового

void showClientDetails() {

// Доступ до protected

std::cout << "Last Name: " << getLastName() << std::endl;

// Доступ до private

std::cout << "Email: " << getEmail() << std::endl;

// Доступ до public

std::cout << "Number: " << getNumber() << std::endl;

}

// 6) using

using Client::displayInfo;

};

#endif

*Payment.h:*

#ifndef PAYMENT\_H

#define PAYMENT\_H

#include <iostream>

#include <vector>

#include "Order.h"

#include "PaymentMethod.h"

class Payment

{

private:

int paymentID;

float amount;

PaymentMethod paymentMethod;

Order order;

std::string paymentDate;

std::string status;

std::string transactionID;

bool isRefundable;

public:

Payment(int paymentID, float amount, const PaymentMethod& paymentMethod, const Order& order, const std::string& paymentDate,

const std::string& status, const std::string& transactionID, bool isRefundable) : paymentID(paymentID), amount(amount),

paymentMethod(paymentMethod), order(order), paymentDate(paymentDate), status(status), transactionID(transactionID),

isRefundable(isRefundable) {}

~Payment() {}

};

#endif

*PaymentMethod.h:*

#ifndef PAYMENT\_METHOD\_H

#define PAYMENT\_METHOD\_H

#include <iostream>

class PaymentMethod {

private:

int methodID;

std::string methodName;

float transactionFee;

bool isSecure;

std::string currency;

int maxLimit;

std::string provider;

float discountRate;

public:

// Конструктор

PaymentMethod()

: methodID(0), transactionFee(0.0f), isSecure(false), maxLimit(0), discountRate(0.0f) {}

// Конструктор з параметрами

PaymentMethod(int methodID, const std::string& methodName, float transactionFee,

bool isSecure, const std::string& currency, int maxLimit,

const std::string& provider, float discountRate)

: methodID(methodID), methodName(methodName), transactionFee(transactionFee),

isSecure(isSecure), currency(currency), maxLimit(maxLimit),

provider(provider), discountRate(discountRate) {}

// Геттери

int getMethodID() const { return methodID; }

std::string getMethodName() const { return methodName; }

float getTransactionFee() const { return transactionFee; }

bool getIsSecure() const { return isSecure; }

std::string getCurrency() const { return currency; }

int getMaxLimit() const { return maxLimit; }

std::string getProvider() const { return provider; }

float getDiscountRate() const { return discountRate; }

// Сеттери

void setMethodID(int methodID) { this->methodID = methodID; }

void setMethodName(const std::string& methodName) { this->methodName = methodName; }

void setTransactionFee(float transactionFee) { this->transactionFee = transactionFee; }

void setIsSecure(bool isSecure) { this->isSecure = isSecure; }

void setCurrency(const std::string& currency) { this->currency = currency; }

void setMaxLimit(int maxLimit) { this->maxLimit = maxLimit; }

void setProvider(const std::string& provider) { this->provider = provider; }

void setDiscountRate(float discountRate) { this->discountRate = discountRate; }

};

#endif

*Product.h:*

#ifndef PRODUCT\_H

#define PRODUCT\_H

#include <iostream>

#include <string>

#include <fstream>

#include <vector>

#include <algorithm>

class Product {

private:

std::string name; //назва

int productID; //ID

double price; //ціна

int quantity; //кількість

std::string category; //категорія

std::string brand; //бренд

int rating; //рейтинг

bool availability; //доступність

static int productCount; // статичне поле для підрахунку кількості продуктів

public:

// конструктор без параметрів

Product()

: name(""), productID(0), price(0.0), quantity(0),

category(""), brand(""), rating(0), availability(true) {}

// конструктор з параметрами

Product(std::string n, int id, double p, int q, std::string c,

std::string b, int r, bool a)

: name(n), productID(id), price(p), quantity(q),

category(c), brand(b), rating(r), availability(a) {}

// конструктор копіювання

Product(const Product& other)

: name(other.name), productID(other.productID),

price(other.price), quantity(other.quantity),

category(other.category), brand(other.brand),

rating(other.rating), availability(other.availability) {}

// Деструктор

~Product() {

std::cout << "Product destroyed: " << name << std::endl;

}

//гет,сет

void setName(std::string n) { name = n; }

std::string getName() const { return name; }

void setProductID(int id) { productID = id; }

int getProductID() const { return productID; }

void setPrice(double p) { price = p; }

double getPrice() const { return price; }

void setQuantity(int q) { quantity = q; }

int getQuantity() const { return quantity; }

void setCategory(std::string c) { category = c; }

std::string getCategory() const { return category; }

void setBrand(std::string b) { brand = b; }

std::string getBrand() const { return brand; }

void setRating(int r) { rating = r; }

int getRating() const { return rating; }

void setAvailability(bool a) { availability = a; }

bool isAvailable() const { return availability; }

};

#endif

*Sale.h:*

#ifndef SALE\_H

#define SALE\_H

#include <iostream>

#include <vector>

#include "Order.h"

#include "PaymentMethod.h"

class Sale

{

private:

int saleID;

Order order;

float discount;

float finalAmount;

PaymentMethod paymentMethod;

std::string saleDate;

std::string sellerName;

std::string status;

public:

Sale(int saleID, const Order& order, float discount, float finalAmount, const PaymentMethod& paymentMethod,

const std::string saleDate, const std::string sellerName, const std::string status) : saleID(saleID),

order(order), discount(discount), finalAmount(finalAmount), paymentMethod(paymentMethod), saleDate(saleDate),

sellerName(sellerName), status(status){}

~Sale() {}

};

#endif

*Warehouse.h:*

#ifndef WAREHOUSE\_H

#define WAREHOUSE\_H

#include <iostream>

#include <vector>

#include "Product.h"

class Warehouse {

private:

int warehouseID;

std::string location;

std::string managerName;

int capacity;

int currentStock;

int incomingStock;

int outgoingStock;

std::vector<Product> products; // Вектор продуктів на складі

public:

};

#endif

*MyForm.h:*

#pragma once

#include "Product.h"

#include "Client.h"

#include "Order.h"

#include "Cart.h"

#include "Warehouse.h"

#include <iostream>

#include <string>

#include <fstream>

#include <vector>

#include <algorithm>

#include "ClientOrder.h"

#include "CartOrder.h"

#include "OrderHistory.h"

namespace Project {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace std;

/// <summary>

/// Summary for MyForm

/// </summary>

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: Add the constructor code here

//

}

protected:

/// <summary>

/// Clean up any resources being used.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private:

/// <summary>

/// Required designer variable.

/// </summary>

System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

void InitializeComponent(void)

{

this->SuspendLayout();

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(538, 400);

this->Name = L"MyForm";

this->Text = L"MyForm";

this->Load += gcnew System::EventHandler(this, &MyForm::MyForm\_Load);

this->ResumeLayout(false);

}

#pragma endregion

private: System::Void MyForm\_Load(System::Object^ sender, System::EventArgs^ e) {

// 9) ...

// Доступ до публічного поля і методу

Product prod("Laptop", 1, 999.99, 10, "Electronics", "BrandX", 4.5, true);

std::cout << "Public ID: " << prod.getProductID() << std::endl;

// Створення об'єкта PaymentMethod

PaymentMethod paymentMethod(1, "Credit Card", 2.5, true, "USD", 1000, "Visa", 0.05f);

// Створення об'єкта Order

Order order(1, prod, 1, paymentMethod, true, "123 Main St", "Shipped");

// Створення об'єкта Client

Client client("John Doe", 1001, "johndoe@example.com", "123-456-7890", "456 Elm St", paymentMethod, "Gold", 5000.0f);

// 3) Створення об'єкта ClientOrder, використовуючи дані з Client і Order

ClientOrder clientOrder(client.getName(), client.getClientID(), client.getEmail(), client.getPhone(),

client.getAddress(), paymentMethod, client.getMembershipType(), client.getTotalSpent(),

order.getOrderID(), order.getProduct(), order.getQuantity(),

order.getPaymentMethod(), order.getIsPaid(), order.getDeliveryAddress(), order.getOrderStatus());

// 4) Виклик методу для відображення інформації про клієнта і замовлення

clientOrder.displayClientOrderInfo();

// 5) Створення об'єкта OrderHistory

OrderHistory orderHistory1 (client.getName(), client.getClientID(), client.getEmail(),

client.getPhone(), client.getAddress(), paymentMethod,

client.getMembershipType(), client.getTotalSpent());

// 2,5)

OrderHistory orderHistory("Ambrello", "wenoinowe@wegn", 123);

// Додавання замовлення до історії

orderHistory.addOrder(order);

// Відображення історії замовлень

orderHistory.displayOrderHistory();

// 2)

orderHistory.showClientDetails();

// 6) using

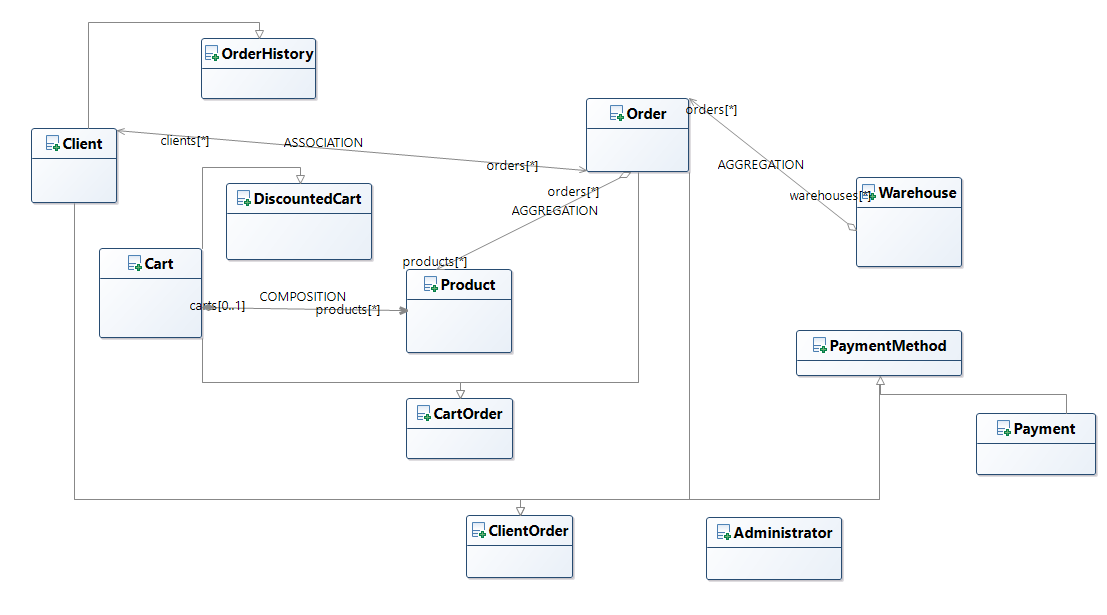
orderHistory.displayInfo();

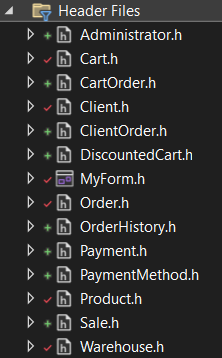
return;

}

};

}





*Посилання на GitHub:*

https://github.com/Annihilation4/OOPLab2