МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА 43

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | К.А. Кочин |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ЛАБОРАТОРНАЯ РАБОТА №8 |
| Описание классов и порождение объектов |
| по дисциплине: Объектно-ориентированное программирование |
|  |
|  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4136 |  |  |  | Бобрович Н.С. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

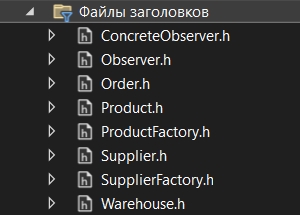
РАБОТУ ВЫПОЛНИЛ

Санкт-Петербург 2023

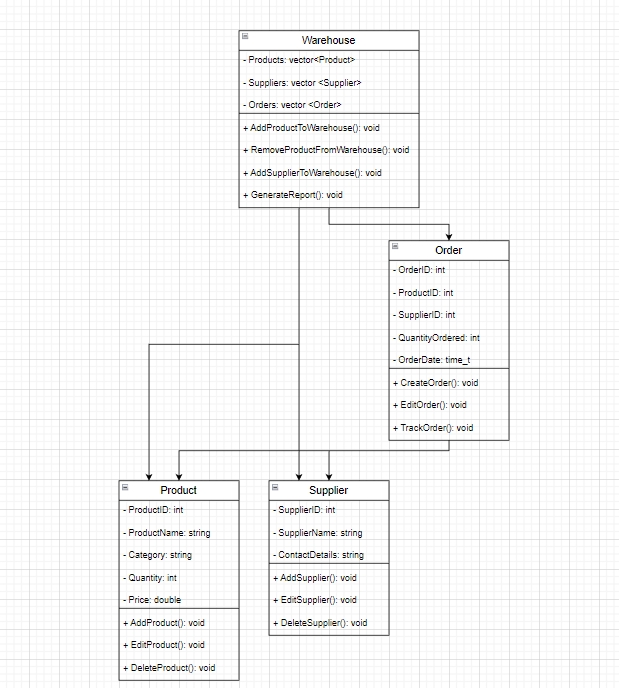
1. Вариант работы: 19 вариант - Склад.
2. В предметной области "Склад" можно выделить следующие сущности:

* Товары (Products): эта сущность будет содержать информацию о товарах, доступных на складе.
* Поставщики (Suppliers): содержит информацию о поставщиках, которые поставляют товары на склад.
* Заказы (Orders): эта сущность будет содержать информацию о заказах, включая товары и их количество.
* Склад (Warehouse): центральная сущность, которая связывает все остальные сущности и управляет ими.

1. Хедэры классов:



1. Диаграммы классов для каждого паттерна в нотации UML. Общая диаграмма, описывающая проект приложения:



1. Функционал

* Добавление/удаление/редактирование информации о товарах.
* Добавление/удаление/редактирование информации о поставщиках.
* Создание/отслеживание/редактирование заказов.
* Просмотр информации о наличии товаров на складе.
* Генерация отчетов о движении товаров и заказов.

1. Проектирование иерархии классов

Class Product

* + - * Fields: ProductID, ProductName, Category, Quantity, Price
      * Methods: AddProduct(), EditProduct(), DeleteProduct()

Class Supplier

* + - Fields: SupplierID, SupplierName, ContactDetails
    - Methods: AddSupplier(), EditSupplier(), DeleteSupplier()

Class Order

* Fields: OrderID, ProductID, SupplierID, QuantityOrdered, OrderDate
* Methods: CreateOrder(), EditOrder(), TrackOrder()

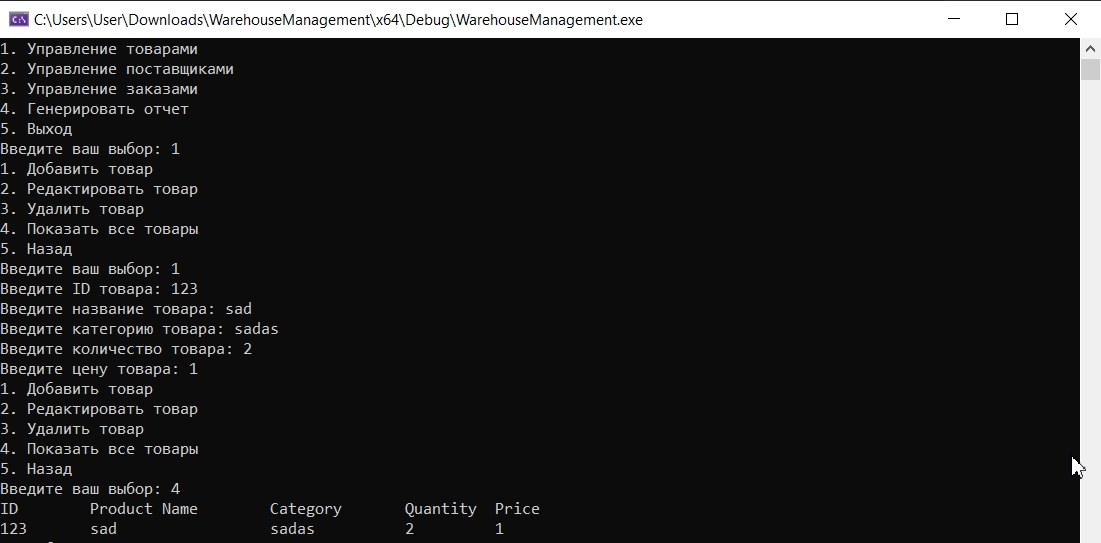
Class Warehouse

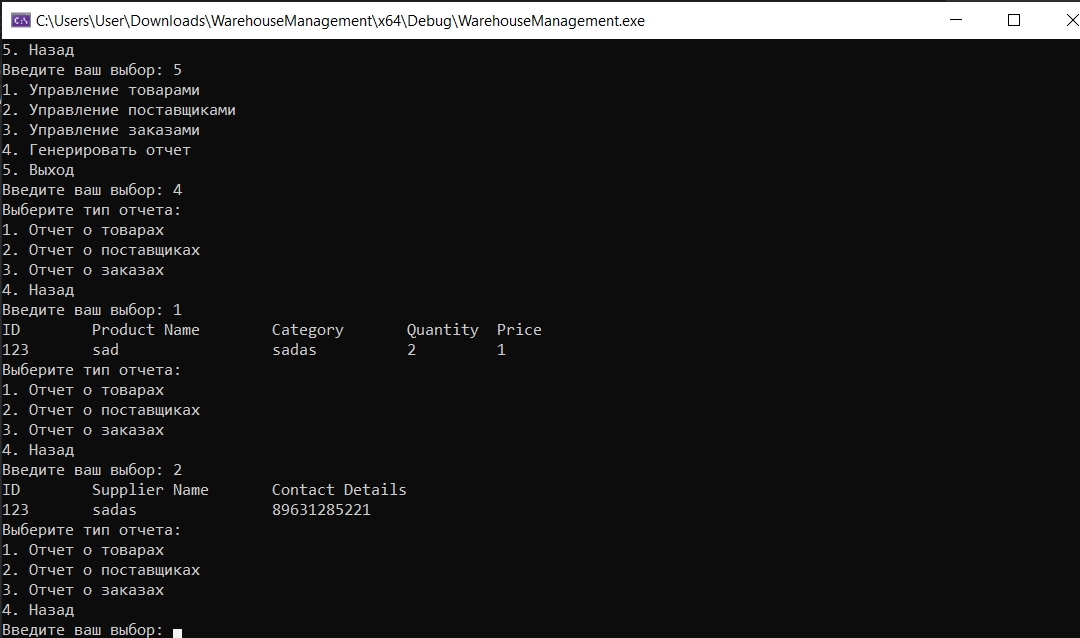
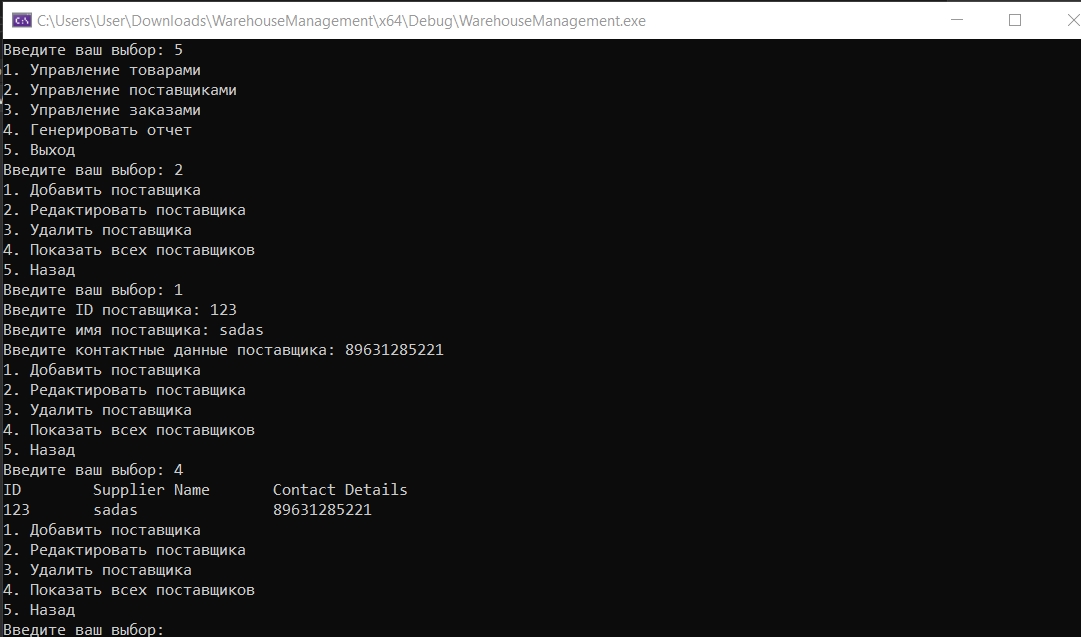
* Fields: Products (list of Product), Suppliers (list of Supplier), Orders (list of Order)
* Methods: AddProductToWarehouse(), RemoveProductFromWarehouse(), AddSupplierToWarehouse(), GenerateReport()

1. Листинг программы:

См Приложение 1.

1. Результат работы программы:





1. Вывод:

В ходе выполнения данной лабораторной работы я научился описанию классов и порождению объектов.

1. Приложения:

Приложение 1:

Order.cpp:

#include "Order.h"

Order::Order(int orderId, int productId, int supplierId, int quantityOrdered, const std::time\_t& orderDate)

: m\_orderId(orderId), m\_productId(productId), m\_supplierId(supplierId), m\_quantityOrdered(quantityOrdered), m\_orderDate(orderDate)

{

}

// Getters

int Order::getOrderId() const { return m\_orderId; }

int Order::getProductId() const { return m\_productId; }

int Order::getSupplierId() const { return m\_supplierId; }

int Order::getQuantityOrdered() const { return m\_quantityOrdered; }

time\_t Order::getOrderDate() const { return m\_orderDate; }

// Setters

void Order::setOrderId(int orderId) { m\_orderId = orderId; }

void Order::setProductId(int productId) { m\_productId = productId; }

void Order::setSupplierId(int supplierId) { m\_supplierId = supplierId; }

void Order::setQuantityOrdered(int quantityOrdered) { m\_quantityOrdered = quantityOrdered; }

void Order::setOrderDate(const time\_t& orderDate) { m\_orderDate = orderDate; }

Product.cpp:

#include "Product.h"

Product::Product(int id, const string& name, const string& category, int quantity, double price)

: m\_id(id), m\_name(name), m\_category(category), m\_quantity(quantity), m\_price(price)

{

}

// Getters

int Product::getId() const { return m\_id; }

string Product::getName() const { return m\_name; }

string Product::getCategory() const { return m\_category; }

int Product::getQuantity() const { return m\_quantity; }

double Product::getPrice() const { return m\_price; }

// Setters

void Product::setId(int id) { m\_id = id; }

void Product::setName(const string& name) { m\_name = name; }

void Product::setCategory(const string& category) { m\_category = category; }

void Product::setQuantity(int quantity) { m\_quantity = quantity; }

void Product::setPrice(double price) { m\_price = price; }

ProductFactory.cpp:

#include "ProductFactory.h"

Product ProductFactory::createProduct(int id, const string& name, const std::string& category, int quantity, double price) {

return Product(id, name, category, quantity, price);

}

Supplier.cpp:

#include "Supplier.h"

Supplier::Supplier(int id, const string& name, const string& contactDetails)

: m\_id(id), m\_name(name), m\_contactDetails(contactDetails)

{

}

// Getters

int Supplier::getId() const { return m\_id; }

string Supplier::getName() const { return m\_name; }

string Supplier::getContactDetails() const { return m\_contactDetails; }

// Setters

void Supplier::setId(int id) { m\_id = id; }

void Supplier::setName(const string& name) { m\_name = name; }

void Supplier::setContactDetails(const string& contactDetails) { m\_contactDetails = contactDetails; }

SupplierFactory.cpp:

#include "SupplierFactory.h"

Supplier SupplierFactory::createSupplier(int id, const string& name, const string& contactDetails) {

return Supplier(id, name, contactDetails);

}

Warehouse.cpp:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "Warehouse.h"

#include <iostream>

void Warehouse::addProduct(const Product& product) {

products.push\_back(product);

}

void Warehouse::removeProductById(int id) {

products.erase(std::remove\_if(products.begin(), products.end(), [id](Product& product) { return product.getId() == id; }), products.end());

}

void Warehouse::editProductById(int id, const Product& newProduct) {

for (auto& product : products) {

if (product.getId() == id) {

product = newProduct;

break;

}

}

}

void Warehouse::addSupplier(const Supplier& supplier) {

suppliers.push\_back(supplier);

}

void Warehouse::editSupplierById(int id, const Supplier& newSupplier) {

for (auto& supplier : suppliers) {

if (supplier.getId() == id) {

supplier = newSupplier;

break;

}

}

}

void Warehouse::createOrder(const Order& order) {

orders.push\_back(order);

notifyObservers();

}

void Warehouse::editOrderById(int id, const Order& newOrder) {

for (auto& order : orders) {

if (order.getOrderId() == id) {

order = newOrder;

notifyObservers();

break;

}

}

}

void Warehouse::registerObserver(Observer\* observer) {

observers.push\_back(observer);

}

void Warehouse::removeObserver(Observer\* observer) {

observers.erase(std::remove(observers.begin(), observers.end(), observer), observers.end());

}

void Warehouse::notifyObservers() {

for (auto observer : observers) {

observer->update();

}

}

// Метод для удаления поставщика по ID

void Warehouse::removeSupplierById(int id) {

suppliers.erase(std::remove\_if(suppliers.begin(), suppliers.end(), [id](Supplier& supplier) { return supplier.getId() == id; }), suppliers.end());

}

// Метод для удаления заказа по ID

void Warehouse::removeOrderById(int id) {

orders.erase(std::remove\_if(orders.begin(), orders.end(), [id](Order& order) { return order.getOrderId() == id; }), orders.end());

}

void Warehouse::displayProducts() const {

cout << std::left << std::setw(10) << "ID"

<< std::setw(20) << "Product Name"

<< std::setw(15) << "Category"

<< std::setw(10) << "Quantity"

<< std::setw(10) << "Price"

<< std::endl;

for (const auto& product : products) {

cout << std::left << std::setw(10) << product.getId()

<< setw(20) << product.getName()

<< setw(15) << product.getCategory()

<< setw(10) << product.getQuantity()

<< setw(10) << product.getPrice()

<< endl;

}

}

void Warehouse::displaySuppliers() const {

cout << left << std::setw(10) << "ID"

<< std::setw(20) << "Supplier Name"

<< std::setw(30) << "Contact Details"

<< std::endl;

for (const auto& supplier : suppliers) {

std::cout << std::left << std::setw(10) << supplier.getId()

<< std::setw(20) << supplier.getName()

<< std::setw(30) << supplier.getContactDetails()

<< std::endl;

}

}

void Warehouse::displayOrders() const {

std::cout << std::left << std::setw(10) << "Order ID"

<< std::setw(10) << "Product ID"

<< std::setw(10) << "Supplier ID"

<< std::setw(10) << "Quantity"

<< std::setw(20) << "Order Date"

<< std::endl;

for (const auto& order : orders) {

time\_t orderDate = order.getOrderDate();

std::cout << std::left << std::setw(10) << order.getOrderId()

<< std::setw(10) << order.getProductId()

<< std::setw(10) << order.getSupplierId()

<< std::setw(10) << order.getQuantityOrdered()

<< std::setw(20) << ctime(&orderDate)

<< std::endl;

}

}

WarehouseManagment.cpp:

#include <iostream>

#include "Warehouse.h"

#include "ConcreteObserver.h"

#include "ProductFactory.h"

using namespace std;

void manageProducts(Warehouse& warehouse) {

while (true) {

int choice;

cout << "1. Добавить товар\n";

cout << "2. Редактировать товар\n";

cout << "3. Удалить товар\n";

cout << "4. Показать все товары\n";

cout << "5. Назад\n";

cout << "Введите ваш выбор: ";

cin >> choice;

if (cin.fail() || choice < 1 || choice > 5) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, попробуйте снова.\n";

continue;

}

if (choice == 5) {

break;

}

int id;

string name;

string category;

int quantity;

double price;

switch (choice) {

case 1: // Добавить товар

cout << "Введите ID товара: ";

while (!(cin >> id)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

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

cout << "Введите название товара: ";

getline(cin, name);

cout << "Введите категорию товара: ";

getline(cin, category);

cout << "Введите количество товара: ";

while (!(cin >> quantity)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

cout << "Введите цену товара: ";

while (!(cin >> price)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

warehouse.addProduct(ProductFactory::createProduct(id, name, category, quantity, price));

break;

case 2: // Редактировать товар

cout << "Введите ID товара для редактирования: ";

while (!(cin >> id)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

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

cout << "Введите новое название товара: ";

getline(cin, name);

cout << "Введите новую категорию товара: ";

getline(cin, category);

cout << "Введите новое количество товара: ";

while (!(cin >> quantity)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

cout << "Введите новую цену товара: ";

while (!(cin >> price)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

warehouse.editProductById(id, Product(id, name, category, quantity, price));

break;

case 3: // Удалить товар

cout << "Введите ID товара для удаления: ";

while (!(cin >> id)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

warehouse.removeProductById(id);

break;

case 4: // Показать все товары

warehouse.displayProducts();

break;

default:

break;

}

}

}

void manageSuppliers(Warehouse& warehouse) {

while (true) {

int choice;

cout << "1. Добавить поставщика\n";

cout << "2. Редактировать поставщика\n";

cout << "3. Удалить поставщика\n";

cout << "4. Показать всех поставщиков\n";

cout << "5. Назад\n";

cout << "Введите ваш выбор: ";

cin >> choice;

if (cin.fail() || choice < 1 || choice > 5) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, попробуйте снова.\n";

continue;

}

if (choice == 5) {

break;

}

int id;

string name;

string contactDetails;

Supplier newSupplier;

switch (choice) {

case 1: // Добавить поставщика

cout << "Введите ID поставщика: ";

while (!(cin >> id)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

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

cout << "Введите имя поставщика: ";

getline(cin, name);

cout << "Введите контактные данные поставщика: ";

getline(cin, contactDetails);

newSupplier = Supplier(id, name, contactDetails);

warehouse.addSupplier(newSupplier);

break;

case 2: // Редактировать поставщика

cout << "Введите ID поставщика для редактирования: ";

while (!(cin >> id)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

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

cout << "Введите новое имя поставщика: ";

getline(cin, name);

cout << "Введите новые контактные данные поставщика: ";

getline(cin, contactDetails);

warehouse.editSupplierById(id, Supplier(id, name, contactDetails));

break;

case 3: // Удалить поставщика

cout << "Введите ID поставщика для удаления: ";

while (!(cin >> id)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

warehouse.removeSupplierById(id);

break;

case 4: // Показать всех поставщиков

warehouse.displaySuppliers();

break;

default:

break;

}

}

}

void manageOrders(Warehouse& warehouse) {

while (true) {

int choice;

cout << "1. Создать заказ\n";

cout << "2. Редактировать заказ\n";

cout << "3. Показать все заказы\n";

cout << "4. Назад\n";

cout << "Введите ваш выбор: ";

cin >> choice;

if (cin.fail() || choice < 1 || choice > 4) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, попробуйте снова.\n";

continue;

}

if (choice == 4) {

break;

}

int orderId, productId, supplierId, quantityOrdered;

time\_t orderDate;

Order newOrder;

switch (choice) {

case 1: // Создать заказ

cout << "Введите ID заказа: ";

while (!(cin >> orderId)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

cout << "Введите ID продукта: ";

while (!(cin >> productId)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

cout << "Введите ID поставщика: ";

while (!(cin >> supplierId)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

cout << "Введите количество заказанного: ";

while (!(cin >> quantityOrdered)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

orderDate = time(nullptr);

newOrder = Order(orderId, productId, supplierId, quantityOrdered, orderDate);

warehouse.createOrder(newOrder);

break;

case 2: // Редактировать заказ

cout << "Введите ID заказа для редактирования: ";

while (!(cin >> orderId)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

cout << "Введите новый ID продукта: ";

while (!(cin >> productId)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

cout << "Введите новый ID поставщика: ";

while (!(cin >> supplierId)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

cout << "Введите новое количество заказанного: ";

while (!(cin >> quantityOrdered)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

orderDate = time(nullptr);

warehouse.editOrderById(orderId, Order(orderId, productId, supplierId, quantityOrdered, orderDate));

break;

case 3: // Удалить заказ

cout << "Введите ID заказа для удаления: ";

while (!(cin >> orderId)) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, введите число: ";

}

warehouse.removeOrderById(orderId);

break;

default:

break;

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

Warehouse warehouse;

ConcreteObserver observer;

warehouse.registerObserver(&observer);

while (true) {

int choice;

cout << "1. Управление товарами\n";

cout << "2. Управление поставщиками\n";

cout << "3. Управление заказами\n";

cout << "4. Генерировать отчет\n";

cout << "5. Выход\n";

cout << "Введите ваш выбор: ";

cin >> choice;

if (cin.fail() || choice < 1 || choice > 5) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, попробуйте снова.\n";

continue;

}

if (choice == 5) {

break;

}

switch (choice) {

case 1:

manageProducts(warehouse);

break;

case 2:

manageSuppliers(warehouse);

break;

case 3:

manageOrders(warehouse);

break;

case 4:

while (true) {

int reportChoice;

cout << "Выберите тип отчета:\n";

cout << "1. Отчет о товарах\n";

cout << "2. Отчет о поставщиках\n";

cout << "3. Отчет о заказах\n";

cout << "4. Назад\n";

cout << "Введите ваш выбор: ";

cin >> reportChoice;

if (cin.fail() || reportChoice < 1 || reportChoice > 4) {

cin.clear();

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

cout << "Неверный ввод. Пожалуйста, попробуйте снова.\n";

continue;

}

if (reportChoice == 4) {

break;

}

switch (reportChoice) {

case 1:

warehouse.displayProducts();

break;

case 2:

warehouse.displaySuppliers();

break;

case 3:

warehouse.displayOrders();

break;

default:

break;

}

}

break;

default:

break;

}

}

return 0;

}

ConcreteObserver.h:

#pragma once

#include "Observer.h"

#include <iostream>

using namespace std;

class ConcreteObserver : public Observer {

public:

void update() override {

cout << "Observer: Detected changes in orders.\n";

}

};

Observer.h:

#pragma once

class Observer {

public:

virtual ~Observer() = default;

virtual void update() = 0;

};

Order.h:

#pragma once

#include <string>

#include <ctime>

using namespace std;

class Order {

public:

Order() = default;

Order(int orderId, int productId, int supplierId, int quantityOrdered, const time\_t& orderDate);

// Getters

int getOrderId() const;

int getProductId() const;

int getSupplierId() const;

int getQuantityOrdered() const;

time\_t getOrderDate() const;

// Setters

void setOrderId(int orderId);

void setProductId(int productId);

void setSupplierId(int supplierId);

void setQuantityOrdered(int quantityOrdered);

void setOrderDate(const time\_t& orderDate);

private:

int m\_orderId;

int m\_productId;

int m\_supplierId;

int m\_quantityOrdered;

time\_t m\_orderDate;

};

Product.h:

#pragma once

#include <string>

using namespace std;

class Product {

public:

Product(int id, const string& name, const string& category, int quantity, double price);

// Getters

int getId() const;

string getName() const;

string getCategory() const;

int getQuantity() const;

double getPrice() const;

// Setters

void setId(int id);

void setName(const string& name);

void setCategory(const string& category);

void setQuantity(int quantity);

void setPrice(double price);

private:

int m\_id;

string m\_name;

string m\_category;

int m\_quantity;

double m\_price;

};

ProductFactory.h:

#pragma once

#include "Product.h"

class ProductFactory {

public:

static Product createProduct(int id, const std::string& name, const std::string& category, int quantity, double price);

};

Supplier.h:

#pragma once

#include <string>

using namespace std;

class Supplier {

public:

Supplier() = default;

Supplier(int id, const string& name, const string& contactDetails);

// Getters

int getId() const;

string getName() const;

string getContactDetails() const;

// Setters

void setId(int id);

void setName(const string& name);

void setContactDetails(const string& contactDetails);

private:

int m\_id;

string m\_name;

string m\_contactDetails;

};

SupplierFactory.h:

#pragma once

#include "Supplier.h"

class SupplierFactory {

public:

static Supplier createSupplier(int id, const std::string& name, const std::string& contactDetails);

};

Warehouse.h:

#pragma once

#include <vector>

#include <iomanip>

#include "Product.h"

#include "Supplier.h"

#include "Order.h"

#include "Observer.h"

class Warehouse {

public:

void addProduct(const Product& product);

void removeProductById(int id);

void editProductById(int id, const Product& newProduct);

void addSupplier(const Supplier& supplier);

void editSupplierById(int id, const Supplier& newSupplier);

void createOrder(const Order& order);

void editOrderById(int id, const Order& newOrder);

void registerObserver(Observer\* observer);

void removeObserver(Observer\* observer);

void notifyObservers();

void removeSupplierById(int id);

void removeOrderById(int id);

void displayProducts() const;

void displaySuppliers() const;

void displayOrders() const;

private:

vector<Product> products;

vector<Supplier> suppliers;

vector<Order> orders;

vector<Observer\*> observers;

};