Міністерство освіти і науки України
Хмельницький національний університет
Кафедра комп’ютерної інженерії та інформаційних систем

ЛАБОРАТОРНА РОБОТА № 3-4

з дисципліни: Об’єктно-орієнтовані технології програмування

Виконав студент: Роюк Р.В.
Група: КІ2м-24-2
Перевірив: Лисенко С.М.

2025

Завдання: написати програмне забезпечення, що описує застосування породжуючого патерну «Стратегія» та патерну «Макрокоманди».

Отже, маємо програму, яка реалізує патерни "Стратегія" та "Макрокоманда" для логістичної компанії.

```
#include <iostream>
#include <vector>
#include <memory>

// Патерн "Стратегія" - визначає родину алгоритмів, інкапсулює їх і забезпечує їхню взаємозамінність.

// Абстрактний клас для стратегії доставки
class DeliveryStrategy {
public:
    virtual ~DeliveryStrategy() = default;
    virtual std::string deliver(const std::string& cargo) = 0;
};

// Конкретні стратегії доставки
class RoadDelivery : public DeliveryStrategy {
public:
    std::string deliver(const std::string& cargo) override {
        return "Доставка " + cargo + " вантажівкою по дорозі.";
    }
};

class SeaDelivery : public DeliveryStrategy {
public:
    std::string deliver(const std::string& cargo) override {
        return "Доставка " + cargo + " кораблем по морю.";
    }
};

class AirDelivery : public DeliveryStrategy {
public:
    std::string deliver(const std::string& cargo) override {
        return "Доставка " + cargo + " літаком по повітрю.";
    }
};

// Клас контексту, що використовує певну стратегію
class DeliveryContext {
private:
    std::unique_ptr<DeliveryStrategy> strategy;
public:
    DeliveryContext(std::unique_ptr<DeliveryStrategy> initStrategy) : strategy(std::move(initStrategy)) {}
```

```
    void setStrategy(std::unique_ptr<DeliveryStrategy> newStrategy) {
        strategy = std::move(newStrategy);
    }

    std::string executeDelivery(const std::string& cargo) {
        return strategy->deliver(cargo);
    }
};

// Патерн "Макрокоманда" - дозволяє групувати кілька команд і виконувати їх як одну операцію.

// Абстрактний клас команди
class Command {
public:
    virtual ~Command() = default;
    virtual std::string execute() = 0;
};

// Конкретні команди
class LoadCargoCommand : public Command {
private:
    std::string cargo;
public:
    explicit LoadCargoCommand(const std::string& cargo) : cargo(cargo) {}

    std::string execute() override {
        return "Завантаження " + cargo + " на транспорт.";
    }
};

class TransportCargoCommand : public Command {
private:
    DeliveryContext& deliveryContext;
    std::string cargo;
public:
    TransportCargoCommand(DeliveryContext& context, const std::string& cargo) : deliveryContext(context), cargo(cargo) {}

    std::string execute() override {
        return deliveryContext.executeDelivery(cargo);
    }
};

class UnloadCargoCommand : public Command {
private:
    std::string cargo;
public:
    explicit UnloadCargoCommand(const std::string& cargo) : cargo(cargo) {}

    std::string execute() override {
```

```
        return "Розвантаження " + cargo + " з транспорту.";
    }
};

// Макрокоманда
class MacroCommand : public Command {
private:
    std::vector<std::unique_ptr<Command>> commands;
public:
    void addCommand(std::unique_ptr<Command> command) {
        commands.push_back(std::move(command));
    }

    std::string execute() override {
        std::string result;
        for (const auto& command : commands) {
            result += command->execute() + "\n";
        }
        return result;
    }
};

// Демонстрація роботи
int main() {
    // Вибираємо початкову стратегію
    DeliveryContext deliveryContext(std::make_unique<RoadDelivery>());
    std::string cargo = "продукти харчування";

    // Створення макрокоманди
    MacroCommand macroCommand;
    macroCommand.addCommand(std::make_unique<LoadCargoCommand>(cargo));
    macroCommand.addCommand(std::make_unique<TransportCargoCommand>(deliveryContext
, cargo));
    macroCommand.addCommand(std::make_unique<UnloadCargoCommand>(cargo));

    // Виконання макрокоманди
    std::cout << macroCommand.execute();

    // Зміна стратегії на повітряну
    deliveryContext.setStrategy(std::make_unique<AirDelivery>());
    std::cout << deliveryContext.executeDelivery("медикаменти") << std::endl;

    return 0;
}
```

Патерн **«Стратегія»** визначає родину алгоритмів, інкапсулює кожен з них і забезпечує їх взаємозамінність. Це дозволяє динамічно змінювати спосіб виконання операцій у програмі без зміни її структури. У коді це реалізовано як:

- Абстрактний клас DeliveryStrategy – визначає інтерфейс для всіх стратегій доставки (deliver(self, cargo)).
- Конкретні стратегії (RoadDelivery, SeaDelivery, AirDelivery) – реалізують метод deliver(self, cargo), що описує спосіб доставки.
- Контекст DeliveryContext – використовує певну стратегію доставки, має метод set_strategy(self, strategy), що дозволяє змінити стратегію, метод execute_delivery(self, cargo) виконує доставку за обраною стратегією.

Перевагами такого підходу є:

- Гнучкість: можна легко змінювати спосіб доставки (наземний, морський, повітряний).
- Зниження залежності від конкретних класів: клієнтський код працює з абстракцією, а не з конкретними реалізаціями.
- Масштабованість: додати нову стратегію просто (наприклад, доставку дронами), не змінюючи існуючий код.

Патерн **«Макрокоманди»** дозволяє об'єднувати кілька команд у єдину, яка виконує всі дії послідовно. У коді це реалізовано як:

- Абстрактний клас Command – визначає метод execute(), який мають реалізувати всі команди.
- Конкретні команди (LoadCargoCommand, TransportCargoCommand, UnloadCargoCommand) – LoadCargoCommand: моделює завантаження товару на транспорт, TransportCargoCommand: викликає стратегію доставки, UnloadCargoCommand: моделює розвантаження товару.
- Клас MacroCommand – зберігає список команд, метод add_command(self, command): додає команду до списку, метод execute(self): виконує всі команди по черзі.
- Демонстрація роботи – створюється макрокоманда (MacroCommand), до неї додаються три команди (завантаження, транспортування, розвантаження), викликається execute(), що виконує всі команди послідовно.

Перевагами такого підходу є:

- Спрощення керування процесами: усі операції групуються в одну команду.
- Легке масштабування: можна додавати нові кроки (наприклад, митний контроль перед відправленням).
- Гнучкість: можна змінювати склад команди без модифікації основного алгоритму.

Висновок: у ході лабораторної роботи було реалізовано програмний код для логістичної компанії, з використанням патернів «Стратегія» та «Макрокоманди». Патерн «Стратегія» дозволяє легко змінювати метод доставки, а патерн «Макрокоманди» дозволяє групувати всі етапи доставки в єдину операцію. Разом ці два патерни створюють гнучку і масштабовану архітектуру, яка може адаптуватись до будь-яких потреб логістичної компанії.