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

ЛАБОРАТОРНА РОБОТА № 8

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 патерну «Декоратор» та патерну «Адаптер».

Отже маємо програму, яка моделює логістичну компанію, що використовує «Декоратор» для додавання відстеження вантажу, страхування, (що дозволяє динамічно додавати нову поведінку об'єктам, не змінюючи їхній код) та «Адаптер» для адаптації нового стандарту на зміну старому модулю трансопрту, який має інший інтерфейс. (що дозволяє об'єктам з несумісними інтерфейсами працювати разом)

```
#include <iostream>
#include <memory>
#include <string>

//
// ==== 1. Базовий інтерфейс доставки ===
//
class Delivery {
public:
    virtual void deliver() = 0;
    virtual ~Delivery() = default;
};

//
// ==== 2. Конкретна реалізація доставки ===
//
class BasicDelivery : public Delivery {
public:
    void deliver() override {
        std::cout << "Доставка вантажу звичайним способом." << std::endl;
    }
};

//
// ==== 3. Декоратор: базовий клас ===
//
class DeliveryDecorator : public Delivery {
protected:
    std::unique_ptr<Delivery> wrappee;
public:
    DeliveryDecorator(std::unique_ptr<Delivery> d) : wrappee(std::move(d)) {}
};

//
// ==== 4. Декоратори конкретних можливостей ===
```

```cpp
//
class TrackingDecorator : public DeliveryDecorator {
public:
    TrackingDecorator(std::unique_ptr<Delivery> d) :
DeliveryDecorator(std::move(d)) {}

    void deliver() override {
        wrappee->deliver();
        std::cout << ">> Відстеження доставки увімкнено." << std::endl;
    }
};

class InsuranceDecorator : public DeliveryDecorator {
public:
    InsuranceDecorator(std::unique_ptr<Delivery> d) :
DeliveryDecorator(std::move(d)) {}

    void deliver() override {
        wrappee->deliver();
        std::cout << ">> Вантаж застрахований." << std::endl;
    }
};


//
// ==== 5. Адаптер: Старий транспорт з іншим інтерфейсом ===
//
class OldTransportSystem {
public:
    void oldTransportMethod(const std::string& cargo) {
        std::cout << "Старий транспорт доставив: " << cargo << std::endl;
    }
};

//
// ==== 6. Адаптер: пристосовує до нового інтерфейсу Delivery ===
//
class TransportAdapter : public Delivery {
private:
    OldTransportSystem* oldSystem;
    std::string cargo;
public:
    TransportAdapter(OldTransportSystem* system, const std::string& c)
        : oldSystem(system), cargo(c) {}

    void deliver() override {
        oldSystem->oldTransportMethod(cargo);
    }
};

//
```

```
// ==== 7. Демонстрація ===
//
int main() {
    std::cout << "=== Приклад з декоратором ===" << std::endl;
    std::unique_ptr<Delivery> delivery = std::make_unique<BasicDelivery>();
    delivery = std::make_unique<TrackingDecorator>(std::move(delivery));
    delivery = std::make_unique<InsuranceDecorator>(std::move(delivery));
    delivery->deliver();

    std::cout << "\n=== Приклад з адаптером ===" << std::endl;
    OldTransportSystem legacySystem;
    std::unique_ptr<Delivery> adaptedDelivery =
std::make_unique<TransportAdapter>(&legacySystem, "Електроніка");
    adaptedDelivery->deliver();

    return 0;
}
```

Патерн **«Декоратор»** дозволяє динамічно додавати нову поведінку об'єктам, не змінюючи їхній код. У коді це реалізовано як:

- BasicDelivery – базовий тип доставки;
- TrackingDecorator додає можливість відстеження;
- InsuranceDecorator – додає страхування;

«Декоратор» дозволяє динамічно додавати функціональність до доставки (відстеження, страхування) без зміни існуючих класів.

Патерн **«Адаптер»** дозволяє об'єктам з несумісними інтерфейсами працювати разом. У коді це реалізовано як:

- OldTransportSystem – старий клас транспорту з несумісним методом oldTransportMethod();
- TransportAdapter — реалізує інтерфейс Delivery і всередині викликає старий метод.

Патерн «Адаптер» забезпечує сумісність між новим інтерфейсом логістичної системи та старим кодом транспорту, даючи змогу уникнути дублювання та спростити інтеграцію.

Перевагами такого підходу є швидке створення нових екземплярів, особливо якщо створення з нуля є ресурсомістким, а також збереження стану об'єкта при його копіюванні.