

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І. І. МЕЧНИКОВА
ФАКУЛЬТЕТ МАТЕМАТИКИ, ФІЗИКИ ТА ІНФОРМАЦІЙНИЙ ТЕХНОЛОГІЙ
КАФЕДРА МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

Лабораторна робота №7
з дисципліни: «Інженерія програмного забезпечення»
за темою: «Патерн проектування Template Method»

Виконав:
студент 3 курсу
Владислав Краковський

Перевірив:
викладач
Пенко В.Г.

Зміст

| | |
|------------------------|----|
| 1 ЗАВДАННЯ РОБОТИ..... | 3 |
| 2 ХІД РОБОТИ..... | 4 |
| Висновок..... | 10 |

1 ЗАВДАННЯ РОБОТИ

Мета роботи: Ознайомитися з патерном проєктування “Шаблонний метод” (Template Method), навчитися реалізовувати його на мові програмування C++, зрозуміти переваги, недоліки та сфери застосування даного патерну.

Умова роботи:

Приклад – обчислення інтегралів.

Використовуючи архітектуру патерна Template method реалізувати наближене обчислення інтегралів методом прямокутників. Математичні основи можна знайти за посиланням: https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BF%D1%80%D1%8F%D0%BC%D0%BE%D0%BA%D1%83%D1%82%D0%BD%D0%B8%D0%BA%D1%96%D0%B2

Рисунок 1.1 - Умова

Варіанти самостійних завдань

1. Розширити попередній приклад за допомогою методу трапецій.

Рисунок 1.2 - Варіант задачі

Система контролю версій Git та рефакторинг

Варіанти самостійних завдань

У наведеному вище викладі кожного патерну наводяться структурний та реальний приклад. Реалізувати структурний приклад та зафіксувати його в системі контролю версій. Далі здійснити серію фіксацій, яка трансформує структурний приклад у реальний. Деякі з проміжних фіксацій повинні полягати у застосуванні того чи іншого прийому рефакторингу.

Рисунок 1.3 - Завдання з GitHub

2 ХІД РОБОТИ

Опис патерну Template Method:

Template Method (Шаблонний метод) — це поведінковий патерн проєктування, який визначає скелет (шаблон) алгоритму в методі базового класу, залишаючи реалізацію деяких кроків цього алгоритму для підкласів. Це дозволяє підкласам змінювати певні етапи алгоритму, не змінюючи його структури загалом.

Як реалізується патерн Template Method?

1. У базовому (абстрактному) класі оголошується шаблонний метод, який описує послідовність кроків алгоритму. Деякі з цих кроків є абстрактними (чистими віртуальними), інші — можуть мати стандартну реалізацію.
2. Підкласи наслідують цей базовий клас і перевизначають конкретні кроки алгоритму (реалізують абстрактні методи або перевизначають віртуальні).
3. Виклик шаблонного методу гарантує, що послідовність виконання алгоритму незмінна, а варіативність реалізації можлива лише в визначених підкласом місцях.

У C++ це означає створення базового класу з методом, який викликає інші (віртуальні) методи, а підкласи реалізують конкретну поведінку.

Для чого використовується патерн Template Method?

- Коли потрібно забезпечити незмінність загальної структури алгоритму, але дозволити деталям змінюватися.

- Для повторного використання коду, коли алгоритми мають спільну частину, а відмінності — лише в окремих кроках.
- Для забезпечення контролю над порядком виконання операцій.

Типові приклади:

- Класичні приклади в ООР: алгоритми сортування, обробки даних, побудова складних об'єктів крок за кроком.
- Відкриття, читання, обробка й закриття файлу з різними форматами.
- Шаблони роботи з базою даних, коли відкриття і закриття однакові, а запит — різний.

Переваги патерну Template Method

- Гарантує незмінність структури алгоритму — загальний хід завжди однаковий.
- Полегшує повторне використання коду — спільна логіка винесена в базовий клас.
- Дозволяє легко розширювати поведінку алгоритму через підкласи.

Недоліки патерну Template Method

- Може призводити до надмірної кількості підкласів, якщо потрібно багато варіантів поведінки.
- Порушення принципу інкапсуляції — підклас може "бачити" деталі алгоритму базового класу.
- Може бути менш гнучким у порівнянні зі стратегією (Strategy), якщо потрібно міняти поведінку динамічно.

Запишемо розв'язок поставленої задачі з використання патерну Template

Method:

```
#include <iostream>
#include <cmath>

// Интерфейс произвольной функции
class AnyFunction {
public:
    virtual double Y(double x) const = 0;
    virtual ~AnyFunction() = default;
};

// Пример: функция гиперболола ( $a * x^2 + b$ )
class Hyperbola : public AnyFunction {
    double a, b;
public:
    Hyperbola(double a, double b) : a(a), b(b) {}
    double Y(double x) const override {
        return a * x * x + b;
    }
};

// Абстрактный класс интеграла (Template Method)
class CommonIntegral {
protected:
    double start, finish;
    int stepNumber;
    const AnyFunction* func;
public:
    CommonIntegral(double start, double finish, int stepNumber,
const AnyFunction* func)
```

```

        : start(start), finish(finish), stepNumber(stepNumber),
func(func) {}

    // Часть интеграла – абстрактный метод
    virtual double IntegralPart(double left, double right) const =
0;

    // Шаблонный метод вычисления интеграла
    double Calculate() const {
        double step = (finish - start) / stepNumber;
        double s = 0.0;
        for (int i = 0; i < stepNumber; ++i)
            s += IntegralPart(start + step * i, start + step * (i
+ 1));
        return s;
    }

    virtual ~CommonIntegral() = default;
};

// Класс метода левых прямоугольников
class LeftRectangle : public CommonIntegral {
public:
    LeftRectangle(double start, double finish, int stepNumber,
const AnyFunction* func)
        : CommonIntegral(start, finish, stepNumber, func) {}

    double IntegralPart(double left, double right) const override
{
        return func->Y(left) * (right - left);
    }
};

// --- ДОБАВЛЯЕМ МЕТОД ТРАПЕЦИЙ ---

```

```

class Trapezoid : public CommonIntegral {
public:
    Trapezoid(double start, double finish, int stepNumber, const
AnyFunction* func)
        : CommonIntegral(start, finish, stepNumber, func) {}

    double IntegralPart(double left, double right) const override
    {
        //  $S = (f(a) + f(b)) / 2 * (b - a)$ 
        return (func->Y(left) + func->Y(right)) * (right - left) /
2.0;
    }
};

// --- Пример использования ---
int main() {
    Hyperbola f(2.0, 3.0);

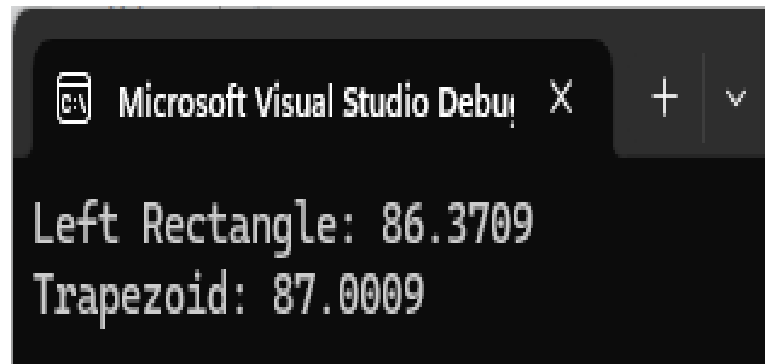
    CommonIntegral* integrall = new LeftRectangle(2.0, 5.0, 100,
&f);
    std::cout << "Left Rectangle: " << integrall->Calculate() <<
std::endl;
    delete integrall;

    CommonIntegral* integral2 = new Trapezoid(2.0, 5.0, 100, &f);
    std::cout << "Trapezoid: " << integral2->Calculate() <<
std::endl;
    delete integral2;

    return 0;
}

```


Подивимось скріншот з результатом на ілюстрації 2.1:



```
Microsoft Visual Studio Debug Console X + v  
Left Rectangle: 86.3709  
Trapezoid: 87.0009
```

Рисунок 2.1 — Результат роботи програми

Програма була завантажена на GitHub згідно з завданням, ілюстрація 2.2:













| | | | |
|--|---------|---|---|
| Template Method: integral by left rectangles and add trapezoid method for integral | 09d4365 |  |  |
|  oouoz442 committed 2 hours ago | | | |
| Template Method: integral by left rectangles | cba336c |  |  |
|  oouoz442 committed 2 hours ago | | | |
| Template Method: real-world example in C++ | 19abe06 |  |  |
|  oouoz442 committed 2 hours ago | | | |
| Template Method: structural example in C++ | 965ff65 |  |  |
|  oouoz442 committed 2 hours ago | | | |

Рисунок 2.2 - GitHub

Висновок

Патерн “Template Method” дозволяє виділити незмінну частину алгоритму в базовий клас, а змінні — делегувати підкласам. Це підвищує гнучкість та повторне використання коду, дозволяє ефективно підтримувати ієрархії алгоритмів із спільною структурою. Проте у великих системах варто бути уважним до кількості підкласів і вибирати цей патерн там, де структура алгоритму дійсно стабільна.