* 1. Означення шаблонів проєктування

Шаблони проєктування (надалі шаблони /патерни) – **це рішення проблем, що часто виникають при проєктуванні** (надалі паралельно дизайні) **програмного забезпечення**. Немовби готові креслення, які можна налаштували для вирішення повторюваних проблем проєктування в коді.

Патерни не можна просто скопіювати в код програми як бібліотеки чи готові функції. Це не фрагменти коду, а **загальні** **концепції** вирішення конкретних проблем, яких можна дотримуватись для реалізації відповідного для конкретного програмного застосунку рішення.

Шаблони часто плутають з алгоритмами, оскільки вони обоє описують типові рішення відомих проблем. Однак алгоритми визначають **чіткий набір дії, що ведуть до певного результату**, а патерни містять більш **високорівневий опис рішення, код реалізації якого може сильно відрізнятись** у різних програмах.

[1]

* 1. Складові шаблонів проєктування

Більшість патернів містять наступні секції:

* **Мета**: короткий опис проблеми та її рішення
* **Мотивація**: додаткове пояснення проблеми й рішення, яке робить можливим шаблон
* **Структура** класів, яка показує кожну частину патерну й зв’язок між ними.
* Приклад **коду** однією з популярних мов програмування полегшує розуміння ідеї, що лежить в основі патерну.

[1]

* 1. Класифікація шаблонів проєктування?

Патерни класифікують за **складністю**, **деталізацією** та **масштабом** застосування відносно всієї системи.

Найпростіші патерни найнижчого рівня часто називають **ідіомами**. Зазвичай їх можна застосовувати лише в одній мові програмування.

На противагу їм, найбільш складні та високорівневі шаблони називають **архітектурними**. Вони підходять для дизайну всього програмного забезпечення.

Крім того всі шаблони можна поділити на 3 категорією залежно від їх мети: на **породжувальний**, **структурні** та **поведінкові** патерни.

[2]

* 1. Породжувальні шаблони

Цей тип патернів надає різноманітні **механізми створення об’єктів**, які підвищують гнучкість та можливість повторного використання коду.

До них належать: **Одинак**, **Будівельник**, **Прототип**, **Фабричний метод** та **Абстрактна фабрика**.

[1]

* + 1. Одинак

Одинак (Синглтон) – це породжувальний патерн, який гарантує, **наявність лише одного екземпляру класу**, забезпечуючи при цьому глобальну точку доступу до нього.

**Проблема**

**Гарантія єдиного екземпляру класу**. Найпоширенішою причиною потреби в єдиному екземплярі класу є контроль доступу до деяких спільних ресурсів, наприклад: бази даних або файлу.

**Глобальна точку доступу до цього екземпляра**. На відміну від звичайної глобальної (статичної) змінної, Синглтон забезпечує захист глобальної точки доступу від перезапису із зовні, при цьому вся логіка міститься всередині самого класу.

**Рішення**

Всі реалізації одинака містять **2** наступні **елементи**:

1. **Приватний конструктор** за замовчуванням, який не дає іншим об’єктам використовувати оператор **new** з класом.
2. **Статичний метод, що заміняє конструктор**. Цей метод викликає приватний конструктор для створення об'єкта і зберігає його в статичному полі. Усі наступні виклики цього методу повертають вже створений об’єкт.

[3]

Рис. 1.4.1.1 – Структура класів реалізації патерна Одинак

Зображення, що містить текст, ряд, знімок екрана, Шрифт

Автоматично згенерований опис

* + 1. Будівельник

**Будівельник** – це породжувальний патерн, який дозволяє **поділяти створення складних об’єктів на етапи**. Шаблон дозволяє створювати **різні типи й представлення об'єкта**, використовуючи **один і той самий код побудови**.

**Проблема**

Потрібно **створити складний об'єкт, який вимагає трудомісткої, покрокової ініціалізації багатьох полів і вкладених об'єктів**. Зазвичай такий **код ініціалізації** ховається **всередині монструозного конструктора** з великою кількістю параметрів. Або ще гірше: **розкиданий по клієнтському коду**.

**Рішення**

Шаблон Будівельник пропонує **витягти код побудови об'єкта** з його власного класу **в** окремі **об'єкти, які називаються будівельниками** або конструкторами.

Шаблон організовує побудову об'єкта у вигляді набору кроків. Важливо те, що **не обов’язково викликати всі кроки**, а лише необхідні для створення певної конфігурації об'єкта.

Деякі з кроків побудови можуть вимагати **різної реалізації**, коли потрібно створити різні представлення продукту. У цьому випадку можна створити **кілька різних класів-конструкторів**, які реалізують той самий набір кроків побудови, але у різний спосіб. Потім можна використовувати цих будівельників для створення різних типів об'єктів.

Кріт того можна зберегти **послідовність** виконання різних **кроків** побудови **в окремий клас – директор**. При цьому можна використовувати **різних будівельників для одного директора**.

[4]

Рис. 1.4.2.1 – Структура класів реалізації патерна Будівельник

Зображення, що містить текст, схема, Паралель, знімок екрана

Автоматично згенерований опис

* + 1. Прототип

Прототип – це породжувальний патерн, який дозволяє **копіювати** існуючі **об'єкти**, **не роблячи** ваш **код** **залежним** від їхніх **класів**.

**Проблема**

Потрібно **створити точну копію існуючого об’єкта**. Найпростіший спосіб – створити новий об'єкт того ж класу, пройтись по всіх полях оригінального об'єкта і скопіювати їх значення до нового об'єкта.

Однак далеко **не всі поля можуть бути публічними**. Крім того такий **код залежний від класу** об’єкта. Який, своєю чергою далеко не завжди відомий, наприклад при роботі з інтерфейсом, який реалізує цей клас.

**Рішення**

Шаблон Прототип **делегує процес клонування об'єктам, які власне клонуються**. Патерн використовує спільний **інтерфейс** для всіх об'єктів, які підтримують клонування. Цей інтерфейс, який зазвичай містить лише один метод, **дозволяє клонувати об'єкт без зв'язування коду з їхнім класом**. Крім того, при такому клонуванні, **об’єкт, що копіюється має доступ до приватних полів копії**.

[5]

Рис. 1.4.3.1 – Структура класів реалізації патерна Прототип

Зображення, що містить текст, квитанція, схема, ряд

Автоматично згенерований опис

* + 1. Фабричний метод

Фабричний метод - це породжувальний патерн, який надає **інтерфейс для створення об'єктів у суперкласі**, при цьому дозволяючи підкласам **змінювати тип об'єктів**, які будуть. створені.

**Проблема**

Розглянемо приклад. Додаток для **управління логістикою**, який обробляє **вантажні перевезення**. часом виникає потреба **реалізувати й морські перевезення**.

Доведеться внести **значні зміни** до всього **коду** – у великій мірі його доведеться переписати. Ба більше, **додавання інших видів перевезень зідйсснення вимагатиме подібних дій**.

**Рішення**

Патерн Фабричний метод пропонує **замінити прямі виклики створення об'єктів** (тут вантажівок та кораблів) на **виклики спеціального фабричного методу**.

Тепер можна **перевизначити фабричний метод у підкласі** і змінити клас продуктів, що створюються цим методом.

**Клієнтський код не бачить різниці між продуктами**, що повертаються різними підкласами, **знаючи лише, що це транспорт**, який може здійснювати перевезення.

[6]

Рис. 1.4.4.1 – Структура класів реалізації патерна Фабричний метод

Зображення, що містить текст, квитанція, схема, знімок екрана

Автоматично згенерований опис

* + 1. Абстрактна фабрика

Абстрактна фабрика - це породжувальний шаблон, який дозволяє створювати **сім'ї пов'язаних об'єктів**, не вказуючи їх конкретних класів.

**Проблема**

Розглянемо приклад. **Симулятор магазину меблів**. Код містить **набір меблів різного типу і стилі кожного набору**. Потрібно створювати різні **меблі, які будуть підходити** одне одному **за стилем**. Крім того слід максимально **спростити додавання** нових **стилів меблів**, адже це доволі частий процес.

**Рішення**

Перше, що пропонує Абстрактна фабрика – оголосити **інтерфейси для кожного окремого виду меблів** (стільця, дивана…). Всі конкретні **продукти повинні їх реалізовувати**, наприклад інтерфейс Chair для кожного стільця.

Наступним кроком буде **оголошення** AbstractFamily– **інтерфейсу зі списком методів створення всіх видів меблів** (CreateChair, CreateSofa…). Ці методи повинні повертати абстрактні типи продуктів, представлені інтерфейсами, створеними раніше.

**Для кожного стилю** продуктів ми створюємо **окремий клас фабрики** на основі інтерфейсу AbstractFactory. **Фабрика - це клас, який повертає продукти певного типу**. Наприклад, ModernFurnitureFactory може створювати лише об'єкти ModernChair, ModernSofa…

**Клієнтський код працює** як з фабриками, так і з продуктами **через відповідні інтерфейси** тож конкретні їх **реалізації** ніяк **на** цей **код не впливають**.

Що стосується **створення самих фабрик** (тобто справжніх конкретних об’єктів), то зазвичай **застосунок створює** їх **на етапі ініціалізації** **залежно від конфігурації середовища** (наприклад обраного користувачем стилю меблів).

[7]

Рис. 1.4.5.1 – Структура класів реалізації патерна Абстрактна фабрика

Зображення, що містить текст, квитанція, схема, Паралель

Автоматично згенерований опис

* + 1. Перехідник
    2. Міст
    3. Компонувальник
    4. Декоратор
    5. Фасад
    6. Легковаговик
    7. Замісник
  1. Поведінкові шаблони

Цей набір патернів стосується **алгоритмів** та **розподілу обов’язків** між об’єктами.

З них виділяють наступні: **Ланцюг відповідальності**, **Команда**, **Ітератор**, **Посередник**, **Знімок**, **Стан**, **Стратегія**, **Шаблонний метод** та **Відвідувач**.

[2]

* + 1. Ланцюг відповідальності
    2. Команда
    3. Ітератор
    4. Посередник
    5. Знімок
    6. Стан
    7. Стратегія
    8. Шаблонний метод
    9. Відвідувач

Список використаних джерел

1. Refactoring Guru. Design Patterns. What's a Design Pattern? URL: [https://refactoring.guru/design–patterns/what–is–pattern](https://refactoring.guru/design-patterns/what-is-pattern) (дата звернення: 01.06.2024)
2. Refactoring Guru. Design Patterns. The Catalog of Design Patterns. URL: [https://refactoring.guru/design–patterns/catalog](https://refactoring.guru/design-patterns/catalog) (дата звернення: 01.06.2024)
3. Refactoring Guru. Design Patterns. Singleton. URL: [https://refactoring.guru/design–patterns/singleton](https://refactoring.guru/design-patterns/singleton) (дата звернення: 01.06.2024)