Лантух-Лященко А.І., Шеветовський В.В., Шимановський О.В., Шмуклер В.С. Чисельні та експериментальні методи раціонального проектування та зведення конструктивних систем. Київ, «Сталь». 2017. 404 с.
3. В.С. Шмуклер, Ю.А. Клімов, Н.П. Буряк. Каркасні системи полегшеного типу. Харків: Золотісторінки, 2008, 336 с.
4. KalmykovO., Gaponova L, Reznikand P, Grebenchuk S. Use of information technologies for energetic portrait construction of cylindrical reinforced concrete shells. 6th International Scientific Conference «Reliability and Durability of Railway Transport Engineering Structures and Buildings MATEC WebConf. Volume 116, 2017. P. 1-6.

Штогрин Д.А.,
асистент кафедри інформаційних технологій,
ЗВО «Університет Короля Данила»,
м. Івано-Франківськ, Україна

РОЛЬ ПАТЕРНІВ ПРОЕКТУВАННЯ В РОЗРОБЦІ АРХІТЕКТУРИ ПЗ

Написання коду вимагає знань та досвіду, щоб мати змогу вирішувати поставлені завдання. Коли розробник самостійно знаходить рішення - він витрачає багато часу, що в свою чергу відображається на його продуктивності та кінцевій вартості продукту. Але реальність полягає в тому, що розробник повинен не стільки самостійно знаходити рішення, як вміти застосувати вже існуючі.

Розробка коду починається з побудови архітектури програми і від її якості залежить не стільки поточне функціонування програми, а її модифікація в майбутньому. Зазвичай, не виникає проблем з архітектурою програми відразу після створення продукту і, якщо ПЗ є сталим, проблеми можуть і не проявлятися. Але в сучасному динамічному світі все дуже

швидко розвивається та змінюється. Як доказ - витіснення класичних моделей розробки ПЗ, які орієнтовані на сталий продукт, гнучкими методологіями, філософія яких побудована на готовності до змін в будь-який момент. Отже, розробляючи програмну архітектуру необхідно зважати, що вона повинна бути гнучкою до змін в майбутньому.

Якщо спробувати самостійно вирішувати всі проблеми з побудовою програмної архітектури, то це призведе до неминучого збільшення витрачених ресурсів на її реалізацію та\або погіршення якості. Набагато оптимальнішим рішенням буде звернутись до накопиченого досвіду попередників, які зіштовхувались з подібними проблемами, успішно їх подолали та поділились використаними способами. Такі рішення в архітектурі програми прийнято називати патернами.

Патерн - це архітектурна конструкція, що представляє собою типове рішення в межах певного контексту, для вирішення певної проблеми, що часто зустрічається. Використання патернів дозволяє покращити якість коду, допомагає зробити код адаптивним до змін та дозволяє зменшити кількість необхідних ресурсів на його розробку. Важливо відмітити, що патерн не має прив'язки до мови програмування, так як це просто підхід(теорія), який дозволяє вирішувати ту чи іншу проблему. Патерни часто плутають з алгоритмами, адже обидва поняття описують типові рішення відомих проблем. Але, якщо алгоритм - це чіткий набір дій, то патерн - це високорівневий опис рішення, реалізація якого може відрізнятися у двох різних програмах.[1,с.26]

Патерни розділяють на три види, кожен з яких відповідає за вирішення завдань з подібною проблематикою:

– Породжуючіпатерни.
– Структурні патерни.
– Патерни поведінки.

Породжувальніпатерни відповідають за створення нових об'єктів. Одним з найбільш популярних є Фабричний метод. Фабричний метод дозволяє використовувати спільний інтер-

фейс, який визначено в суперкласі, для всіх його нащадків. Використання даного патерну позбавляє вас необхідності жорстко прив'язуватись до певного підкласу, натомість ви маєте змогу динамічно створювати необхідний для вас тип.

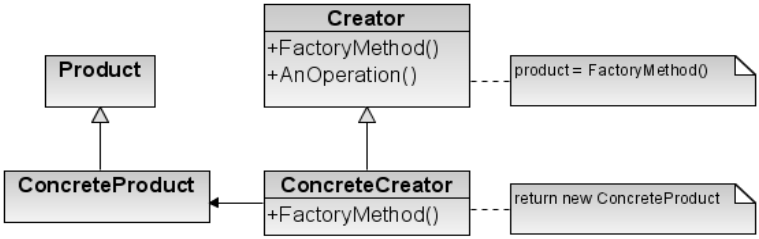

Рис. 1. Схема патерну – Фабричний метод.

Варто зазначити, що для реалізації цього патерну вам необхідно створити паралельну ієрархію. Наприклад, якщо у вас є ієрархія транспортних засобів, то вам у відповідності до кожного класу з ієрархії ТЗ необхідно створити клас, який міститиме метод, що дозволяє створити об'єкт конкретного ТЗ та повертає його під виглядом суперкласу(поліморфізм).

Структурні патерни відповідають за формування взаємодії між класами для виконання необхідних задач та побудову зручних в підтримці ієрархій класів.

Найбільш поширений представник цього типу патернів – Адаптер. Адаптер – структурний патерн проектування, що виконує функції перекладача і дозволяє взаємодіяти об'єктам з різними інтерфейсами. Він отримає запит від одного об'єкту, модифікує його таким чином, щоб його міг отримати інший об'єкт та відправляє йому цей запит. Даний патерн надзвичайно простий, але в той же час корисний і часто використовуваний.

Патерни поведінки відповідають за ефективну та безпечну взаємодію між об'єктами. Одним з найбільш відомих поведінкових патернів є патерн – Стратегія.

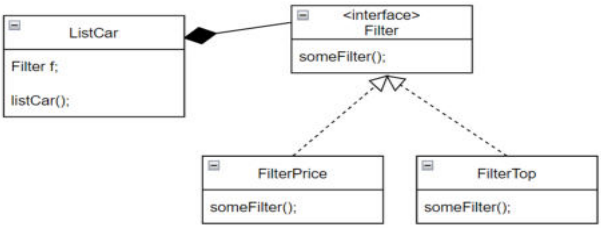

Рис. 2. Схема патерну – Стратегія.

Стратегія – це поведінковий патерн, який визначає сімейство схожих рішень, інкапсулює кожне з них в окремому класі та забезпечує взаємозамінність. Це дозволяє проводити безболісну заміну рішень, ізолює код одного рішення від інших, реалізує принцип відкритості\закритості.

Отже, для уникнення потенційних проблем з архітектурою програми, необхідно використовувати готові перевірені рішення. Це дозволить суттєво зменшити час розробки, покращить розуміння коду та придатність до змін.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ:

1. Олександр Швець. Занурення в патерни проектування. 2021.
2. EricFreeman. ElisabethRobson. HeadFirstDesignPatterns2ndEdition. O`REILLY. 2020.
3. Андрій Будай. Дизайн-патерни – просто, як двері. 2012.