Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №7

Технологія розробки програмного забезпечення

Шаблони «Mediator», «Facade», «Bridge», «Template Method»

Виконала студентка
групи ІА-23:
Шрубович Н. С.

Перевірив:
Мягкий М. Ю.

Київ 2024

Тема: Шаблони «Mediator», «Facade», «Bridge», «Template Method».

Мета: Ознайомитися з принципами роботи шаблонів проектування «Mediator», «Facade», «Bridge», «Template Method», їх перевагами та недоліками. Набути практичних навичок у застосуванні шаблону adapter при розробці програмного забезпечення на прикладі реалізації архіватора.

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми

Варіант:

..3 Текстовий редактор (strategy, command, observer, template method, flyweight, SOA)

Текстовий редактор повинен вміти розпізнавати текстові файли в будь-якій кодуванні, мати розширені функції редагування: макроси, сніппети, підказки, закладки, перехід на рядок / сторінку, підсвічування синтаксису (для однієї мови програмування або розмітки на розсуд студента).

ЗМІСТ

Теоретичні відомості

Принцип “Don’t Repeat Yourself” (DRY) – це програмістський принцип, який закликає уникати повторення одного й того ж коду або інформації в програмному забезпеченні. Ідея полягає в тому, щоб написати код один раз і використовувати його в багатьох місцях, замість того щоб дублювати один і той же код у різних частинах програми.

Основні положення DRY:

1. Уникнення дублювання коду: Якщо одна і та ж інформація або функціональність міститься в декількох місцях програми, її слід об’єднати в одному місці. Це підвищує підтримуваність і знижує ймовірність помилок, пов’язаних з оновленням кодової бази.
2. Ефективне повторне використання коду: Завдяки DRY, коди можуть бути повторно використані без внесення змін у декілька місць, що робить їх менш вразливими до помилок і спрощує внесення змін.
3. Краща організація коду: Дотримання DRY допомагає організувати кодову базу більш чітко і логічно, що сприяє кращому розумінню програмного коду та полегшує процес розробки і підтримки програмного забезпечення.

Шаблон «Template Method» визначає загальний алгоритм у базовому класі, дозволяючи підкласам реалізовувати або перевизначати окремі його кроки. Це забезпечує гнучкість і повторне використання коду при збереженні структури алгоритму.

Структура:

1. Базовий клас:
   Містить шаблонний метод, що визначає послідовність кроків алгоритму. Деякі кроки можуть бути абстрактними (вимагають реалізації в підкласах), іншими — мати реалізацію за замовчуванням.
2. Підкласи:
   Реалізують специфічні кроки алгоритму або змінюють стандартну поведінку для конкретного випадку.

Ключові поняття:

- Абстрактні кроки: Метод у базовому класі, який обов'язково має бути реалізований у підкласах.
- Кроки за замовчуванням: Метод із типовою реалізацією в базовому класі, яку можна змінювати у підкласах.
- Хуки: Додаткові кроки, які можна перевизначати у підкласах, але їхнє використання не обов'язкове.

Переваги:

- Полегшує повторне використання коду, спрощуючи реалізацію схожих алгоритмів.
- Дозволяє підкласам змінювати окремі кроки без впливу на загальну структуру алгоритму.
- Сприяє стандартизації алгоритму, забезпечуючи єдину послідовність дій.

Недоліки:

- Збільшується жорсткість коду через фіксовану структуру алгоритму.
- Зміна базової реалізації може порушити принцип підстановки Лісков.
- З ростом кількості кроків шаблонний метод може стати складним для підтримки.

Шаблон «Template Method» ідеально підходить для стандартизації алгоритмів, які мають однакову структуру, але потребують варіативності в окремих етапах.

Хід Роботи

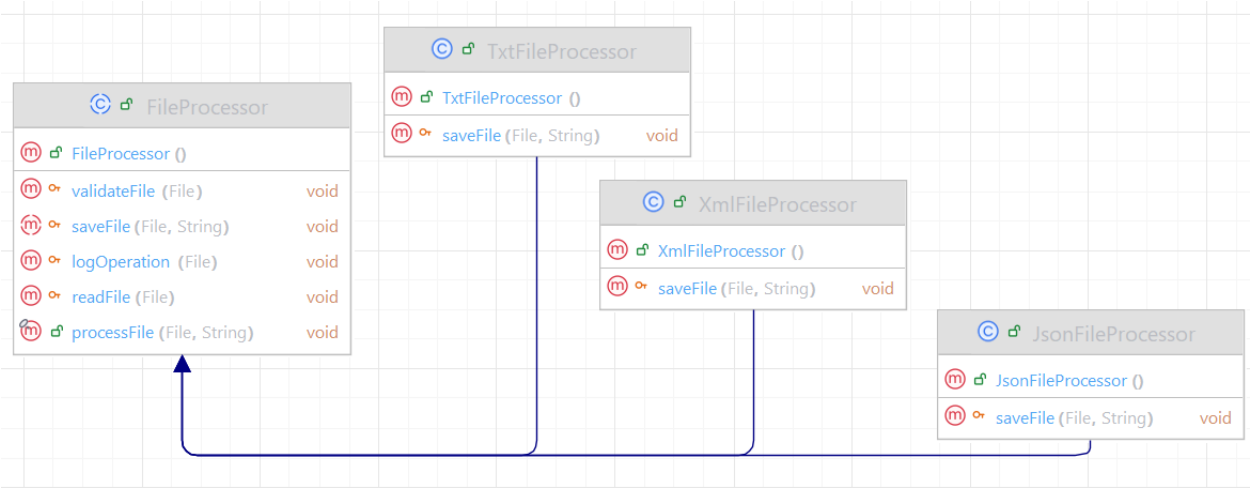

Рис 1. Діаграма класів

Діаграма класів

Діаграма відображає реалізацію шаблону **Template Method**, який забезпечує єдиний алгоритм для обробки файлів різних форматів (TXT, JSON, XML). Базовий клас визначає загальний процес, а підкласи реалізують специфічні деталі для кожного формату.

Базовий клас: FileProcessor

FileProcessor — це абстрактний клас, що визначає шаблонний метод processFile, який керує загальним алгоритмом обробки файлу.

Основні методи:

- processFile(File file, String path) — реалізує основний алгоритм обробки.
- readFile(File file) — читає вміст файлу.
- validateFile(File file) — перевіряє коректність файлу.
- logOperation(File file) — записує лог обробки.
- saveFile(File file, String path) — абстрактний метод для збереження файлу, реалізується в підкласах.

Базовий клас містить загальні кроки алгоритму, які є спільними для всіх форматів файлів.

Конкретні підкласи

Кожен підклас реалізує специфічний метод saveFile, відповідно до формату файлу:

- **TxtFileProcessor** — зберігає файл у форматі TXT.
- **JsonFileProcessor** — зберігає файл у форматі JSON.
- **XmlFileProcessor** — зберігає файл у форматі XML.

Ці підкласи доповнюють загальний алгоритм базового класу, додаючи специфічні деталі для кожного типу файлу.

Призначення патерну Template Method

1. Уніфікація алгоритму
   Визначає єдиний процес обробки файлів із послідовністю кроків: читання, перевірка, збереження та логування.
2. Розширюваність
   Легке додавання підтримки нових форматів файлів шляхом створення нових підкласів, наприклад, YamlFileProcessor.
3. Повторне використання коду
   Загальні кроки, такі як читання чи логування, реалізовані у базовому класі, що усуває дублювання коду.
4. Гнучкість
   Підкласи можуть перевизначати лише необхідні кроки, зберігаючи спільну структуру алгоритму.

Взаємодія компонентів

Клієнт викликає метод processFile через об’єкт конкретного підкласу.
Базовий клас керує виконанням загальних кроків, тоді як специфічні дії делегуються підкласам.
Ця архітектура дозволяє стандартизувати процес обробки файлів і легко адаптувати його для нових форматів, зберігаючи чистоту та масштабованість коду.

Робота патерну

Опис роботи патерну Template Method

Обробка файлу у форматі TXT
Роль базового класу: базовий клас FileProcessor визначає загальний алгоритм обробки файлу через метод processFile, включаючи читання, перевірку валідності, логування та збереження.
Реалізація специфіки: клас TxtFileProcessor реалізує метод saveFile, що зберігає файл у форматі TXT.
Спрощення: клієнтський код викликає лише processFile, не знаючи, як саме виконується збереження файлу у форматі TXT.

Обробка файлу у форматі JSON
Роль базового класу: FileProcessor виконує спільні дії, такі як перевірка валідності та логування, у методі processFile.
Реалізація специфіки: клас JsonFileProcessor реалізує метод saveFile, що зберігає файл у форматі JSON.
Спрощення: клієнт працює з абстрактним алгоритмом, не заглиблюючись у деталі збереження у форматі JSON.

Обробка файлу у форматі XML
Роль базового класу: метод processFile виконує типові операції, які однакові для всіх форматів файлів.
Реалізація специфіки: клас XmlFileProcessor реалізує метод saveFile, що забезпечує збереження файлу у форматі XML.
Спрощення: клієнтський код викликає processFile, не вникаючи у деталі роботи з XML.

Спрощення завдяки Template Method

1. Уніфікація процесу: усі формати файлів обробляються через один загальний алгоритм processFile, визначений у базовому класі.
2. Ізоляція специфічної логіки: збереження у форматах TXT, JSON, XML реалізується у відповідних підкласах, не дублюючи загальний код.
3. Простота для клієнта: клієнтський код працює лише з базовим методом processFile, не залежачи від деталей обробки кожного формату.

Приклад використання
Клієнт створює екземпляр відповідного підкласу (TxtFileProcessor, JsonFileProcessor, XmlFileProcessor) і викликає метод processFile. Базовий клас

автоматично керує виконанням загального алгоритму, тоді як специфічні дії делегуються підкласам.

Шаблон Template Method дозволяє стандартизувати процес обробки файлів різних форматів, розділяючи спільний алгоритм на рівні базового класу та специфічну логіку на рівні підкласів. Це зменшує дублювання коду, підвищує гнучкість і забезпечує легкість додавання нових форматів файлів у майбутньому.

Переваги використання шаблону Template Method

1. Спрощення клієнтського коду
   Завдяки шаблону Template Method клієнтський код викликає лише загальний метод processFile, не вникаючи в деталі роботи з файлами різних форматів. Це забезпечує єдиний інтерфейс для обробки файлів, роблячи код зрозумілим і легким для читання.

2. Інкапсуляція загальної логіки
   Загальний алгоритм обробки файлів (зчитування, перевірка валідності, логування, збереження) визначено в базовому класі FileProcessor. Деталі, які відрізняються для форматів TXT, JSON або XML, реалізовані у підкласах. Це дозволяє уникнути дублювання коду.

3. Гнучкість і розширюваність
   - Додавання нового формату файлу (наприклад, YAML) здійснюється шляхом створення нового підкласу, що реалізує специфічний метод saveFile.
   - Базовий алгоритм у класі FileProcessor залишається незмінним, що відповідає принципу відкритості/закритості (OCP).

4. Зниження залежностей
   Клієнтський код працює лише з базовим класом FileProcessor, не залежачи від конкретних реалізацій підкласів. Це забезпечує низьке зв’язування системи та спрощує її супровід.

5. Покращення підтримки та тестування
   Оскільки загальна логіка обробки файлів зосереджена в одному місці (базовий клас), тестування та підтримка алгоритму стають простішими. Заміна або модифікація специфічної логіки в підкласах не впливає на загальний алгоритм.

6. Єдиний контроль точок виконання
   Всі операції обробки файлів централізовано виконуються через базовий метод processFile. Це забезпечує узгодженість виконання операцій і зменшує ризик помилок.

7. Повторне використання коду
   Загальна частина алгоритму, така як логування чи перевірка валідності, реалізована лише один раз у базовому класі, що значно знижує кількість дубльованого коду.

8. Зручність у масштабуванні
   Додавання нового формату файлу або оновлення специфічної логіки підкласу не вимагає змін у базовому класі або клієнтському коді, що підвищує масштабованість системи.

9. Гарантована послідовність виконання
   Завдяки Template Method структура алгоритму залишається незмінною, незалежно від реалізації підкласів. Це гарантує стабільність виконання операцій для всіх форматів файлів.

Висновок: Отже, у ході виконання лабораторної роботи було реалізовано шаблон проектування Template Method, який забезпечив уніфікований підхід до обробки файлів різних форматів (TXT, JSON, XML). Завдяки використанню цього шаблону вдалося виділити загальні кроки алгоритму в базовому класі FileProcessor, залишивши реалізацію специфічних дій для підкласів. Це спростило клієнтський код, зробило його чистим і зрозумілим, а також забезпечило гнучкість і масштабованість системи. Клієнтський код працює лише з базовим інтерфейсом, не залежачи від конкретних реалізацій, що дозволяє легко додавати підтримку нових форматів файлів або модифікувати існуючі, не порушуючи цілісності системи.

Посилання на код: https://github.com/Nata198111/TRPZ/tree/master