Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №**5**
з предмету «Технології розроблення програмного забезпечення»
на тему «**Шаблони “Adapter”, “Builder”, “Command”, “Chain of Responsibility”, “Prototype”**»

Виконав
студент групи ІА-23:
Воронюк Є. В.

Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Мета: реалізувати частину функціоналу робочої програми у вигляді класів із застосуванням одного із розглянутих шаблонів проектування.

Завдання 1: ознайомитися з короткими теоретичними відомостями.
Завдання 2: реалізувати частину функціоналу робочої програми у вигляді класів та їх взаємодій для досягнення конкретних функціональних можливостей.
Завдання 3: застосування одного з розглянутих шаблонів при реалізації програми.

Тема для виконання:

22 - FTP-server (state, builder, memento, template method, visitor, client- server)

FTP-сервер повинен вміти коректно обробляти і відправляти відповіді по протоколу FTP, з можливістю створення користувачів (з паролями) і доступних їм папок, розподілу прав за стандартною схемою (rwe), ведення статистики з'єднань, обмеження максимальної кількості підключень і максимальної швидкості поширення глобально і окремо для кожного облікового запису.

Короткі теоретичні відомості:

«Анти-шаблони» проектування

Анти-патерни (anti-patterns), також відомі як пастки (pitfalls) - це класи найбільш часто впроваджуваних поганих рішень проблем. Вони вивчаються, як категорія, в разі коли їх хочуть уникнути в майбутньому, і деякі їхні окремі випадки можуть бути розпізнані при вивченні непрацюючих систем.

Термін походить з інформатики, від авторів «Банди чотирьох» книги «Шаблони проектування», яка заклала приклади практики хорошого програмування. Автори назвали ці хороші методи «шаблонами проектування», і протилежними їм є «анти-патерни». Частиною хоршої практики програмування є уникнення анти-патернів.

Анти-патерни в управлінні розробки ПЗ:

- Дим і дзеркала (Smoke and mirrors): Демонстрація того, як будуть виглядати ненаписані функції (назва походить від двох улюблених способів, якими фокусники приховують свої секрети);
- Роздування ПЗ (Software bloat): Дозвіл наступним версіям системи вимагати все більше і більше ресурсів;
- Функції для галочки: Перетворення програми в конгломерат погано реалізованих і не пов'язаних між собою функцій (як правило, для того, щоб заявити в рекламі, що функція є);

Анти-патерни в розробці ПЗ:

- Інверсія абстракції (Abstraction inversion): Приховування частини функціоналу від зовнішнього використання, з надією на те, що ніхто не буде його використовувати;
- Невизначена точка зору (Ambiguous viewpoint): Подання моделі без специфікації її точки розгляду;
- Великий клубок бруду (Big ball of mud): Система зі структурою, яку не можна розпізнати;
- Блоб (Blob): див. Божественний об'єкт (God object);
- Бензинова фабрика (Gas factory): Необов'язкова складність дизайну;
- Затичка на введення даних (Input kludge): Забудькуватість в специфікації і виконанні підтримки можливого неправильного введення;
- Роздування інтерфейсу (Interface bloat): Виготовлення інтерфейсу дуже потужним і дуже важким для реалізації;
- Чарівна кнопка (Magic pushbutton): Виконання результатів дій користувача у вигляді невідповідного (недостатньо абстрактного) інтерфейсу. Наприклад, в системах типу Delphi це написання прикладної логіки в обробниках натискань на кнопку;
- Перестановка (комп'ютер) (Re-Coupling): Процес впровадження непотрібної залежності;
- Димохід (Stovepipe system): Рідко підтримувана збірка погано пов'язаних компонентів;
- Гонки (Race hazard, Race condition): Непередбачені можливості настання подій в порядку, відмінному від очікуваного;
- Мишача метушня: Невиправдане створення безлічі дрібних і абстрактних класів для вирішення однієї конкретної задачі більш високого рівня;
- Кінці, що висять: Інтерфейс, більшість методів якого безглузді і реалізуються «пустушками»;
- Туфелька Золушки: Спроба «натягнути» на об'єкт вже наявний малопідходящий за змістом інтерфейс, замість створення нового;

Анти-патерни в об'єктно-орієнтованому програмуванні:

- Базовий клас-утиліта (BaseBean): Спадкування функціональності з класу-утиліти замість делегування до нього;
- Виклик предка (CallSuper): Для реалізації прикладної функціональності методу класу-нащадка потрібно в обов'язковому порядку викликати ті ж методи класу-предка;
- Помилка порожнього підкласу (Empty subclass failure): Створення класу (в Perl), який не проходить «перевірку порожнечі підкласу» («Empty Subclass Test») через різницю у поведінці в порівнянні з класом, який успадковується від нього без змін;
- Божественний об'єкт (God object): Концентрація занадто великої кількості функцій в одній частині системи (класі);
- Об'єктна клоака (Object cesspool): перевикористання об'єктів, що знаходяться в непридатному для перевикористання стані;

• Полтергейст (комп'ютер) (Poltergeist): Об'єкти, чиє єдине призначення - передавати інформацію іншим об’єктам;

• Проблема йо-йо (Yo-yo problem): Надмірна розмитість сильно пов'язаного коду (наприклад, виконуваного по порядку) по ієрархії класів;

• Самотність (антипаттерн) (Singletonitis): Недоречне використання патерну «одинак»;

• Приватизація (антипаттерн) (Privatisation): Приховування функціоналу в приватній секції (private), що ускладнює його розширення в класах-нащадках ;

Анти-патерни в програмуванні:

• Непотрібна складність (Accidental complexity): Внесення непотрібної складності в рішення;

• Дія на відстані (Action at a distance): Несподівана взаємодія між широко розділеними частинами системи;

• Накопичити і запустити (Accumulate and fire): Установка параметрів підпрограм в наборі глобальних змінних;

• Сліпа віра (Blind faith): Недостатня перевірка (a) коректності виправлення помилки або (b) результату роботи підпрограми;

• Активне очікування (Busy spin): Споживання ресурсів ЦПУ (процесорного часу) під час очікування події, зазвичай за допомогою постійно повторюваної перевірки, замість того, щоб використовувати асинхронне програмування (наприклад, систему повідомлень або подій);

• Кешування помилки (Caching failure): Забувати скинути прапор помилки після її обробки;

• Смердючий підгузник (The Diaper Pattern Stinks): Скидання прапора помилки без її обробки або передачі вищестоящому оброблювачу;

• Перевірка типу замість інтерфейсу (Checking type instead of membership, Checking type instead of interface): Перевірка того, що об'єкт має специфічний тип в той час, коли потрібно тільки певний інтерфейс;

• Кодування шляхом виключення (Coding by exception): Додавання нового коду для підтримки кожного спеціального розпізнаного випадку;

• Таємничий код (Cryptic code): Використання абревіатур замість мнемонічних імен;

• Жорстке кодування (Hard code): Впровадження припущень про оточення системи в занадто великій кількості точок її реалізації;

• Потік лави (Lava flow): Збереження небажаного (зайвого або низькоякісного) коду через те, що його видалення занадто дороге або буде мати непередбачувані наслідки;

• Спагеті-код (Spaghetti code, іноді «макарони»): Код з надмірно заплутаним порядком виконання;

• Мильна бульбашка (Soap bubble): Клас, ініціалізований сміттям, максимально довго прикидається, що містить якісь дані;

• Мютексне пекло (Mutex hell): Впровадження занадто великої кількості об'єктів синхронізації між потоками;

Методологічні анти-патерни:

• Програмування методом копіювання-вставки (Copy and paste programming): Копіювання (і легка модифікація) існуючого коду замість створення спільних рішень;

• Дефакторінг (De-Factoring): Процес знищення функціональності і заміни її документацією;

• Золотий молоток (Golden hammer): Сильна впевненість в тому, що улюблене рішення універсально застосовується. Назва походить від приказки «коли в руках молоток, всі проблеми здаються цвяхами»;

• Фактор неймовірності (Improbability factor): Припущення про неможливість того, що спрацює відома помилка;

• Передчасна оптимізація (Premature optimization): Оптимізація на основі недостатньої інформації;

• Винахід колеса (Reinventing the wheel): Створення з нуля, замість використання готового рішення;

• Винахід квадратного колеса (Reinventing the square wheel): Створення поганого рішення, коли існує гарне відоме рішення;

• Самознищення (Self-destruction): Фатальна помилка або нестандартна поведінка програми, яка веде до відмови в обслуговуванні, що виникла внаслідок іншої менш серйозної помилки. Наприклад, при виникненні помилки, додаток починає дуже швидко і багато писати в лог, внаслідок чого закінчується місце на жорсткому диску швидше, ніж це виявить моніторинг;

• Два тунеля: Винесення нового функціоналу в окремий додаток замість розширення вже наявного. Найчастіше застосовується, коли з яких-небудь причин (в основному, при нестачі часу або небажанні менеджменту) внесення змін у вже наявний код вимагає великих витрат, ніж створення нового. При цьому у клієнта в кінцевому підсумку працюють два додатки, запускаючись одночасно або попеременно один з одного;

Анти-патерни управління конфігурацією:

• DLL-пекло (DLL hell): Проблеми з версіями, доступністю і збільшенням кількості DLL, особливо в Microsoft Windows;

• Пекло залежностей (Dependency hell): Проблеми з версіями потрібних продуктів, особливо в системах UNIX / GNU / Linux;

Організаційні анти-патерни:

• Аналітичний параліч (Analysis paralysis): невиправдано великі витрати на аналіз і проектування. Часто призводить до закриття проекту до початку його реалізації;

• Дійна корова (Cash cow): коли при наявності продукту, що приносить вигоду без істотних вкладень, не вкладаються кошти в розвиток і розробку нових продуктів;

• Тривале старіння (Continuous obsolescence): виділення непропорційно великих зусиль на портування системи в нові оточення;

• Скидування витрат (Cost migration): перенесення витрат на проект уразливому відділу або бізнес-партнеру;

• Повзуче покращення (Creeping featurism): додавання нових поліпшень на шкоду сумарному якості системи;

• Розробка комітетом (Design by committee): розробка проекту без централізованого управління, або при некомпетентному керівництві;

• Ескалація зобов'язань (Escalation of commitment): продовження реалізації рішення після того, як доведено його хибність;

• Я тобі це казав (I told you so): ігнорування думки експерта;

• Управління засноване на числах (Management by numbers): зайва увага до чисельних показників, або наявність опосередкованого відношення до керованої системи, або складним для отримання;

• Драконівські заходи (Management by perkele): невиправдано жорсткий стиль управління;

• Управління грибами (Mushroom management): недостатнє інформування працівників про виконувану роботу;

• Розповзання рамок (Scope creep): втрата контролю над розростанням проекту;

• Замикання на продавці (Vendor lock-in): жорстка прив'язка до постачальника;

• Тепле тіло (Warm body): людина, чий внесок в проект під сумнівом;

• Єдина обізнана людина (Single head of knowledge, SHOK): коли життєво важливими для проекту відомостями або навичками володіє тільки одна людина в команді; без неї робота зупиняється;

• Лицар на білому коні (Knight in shining armor, KISA): коли на сцені з'являється людина, яка намагається полагодити все, не повідомляючи нікому, що він зробив і чому;

Хід роботи::

Реалізація патерну:

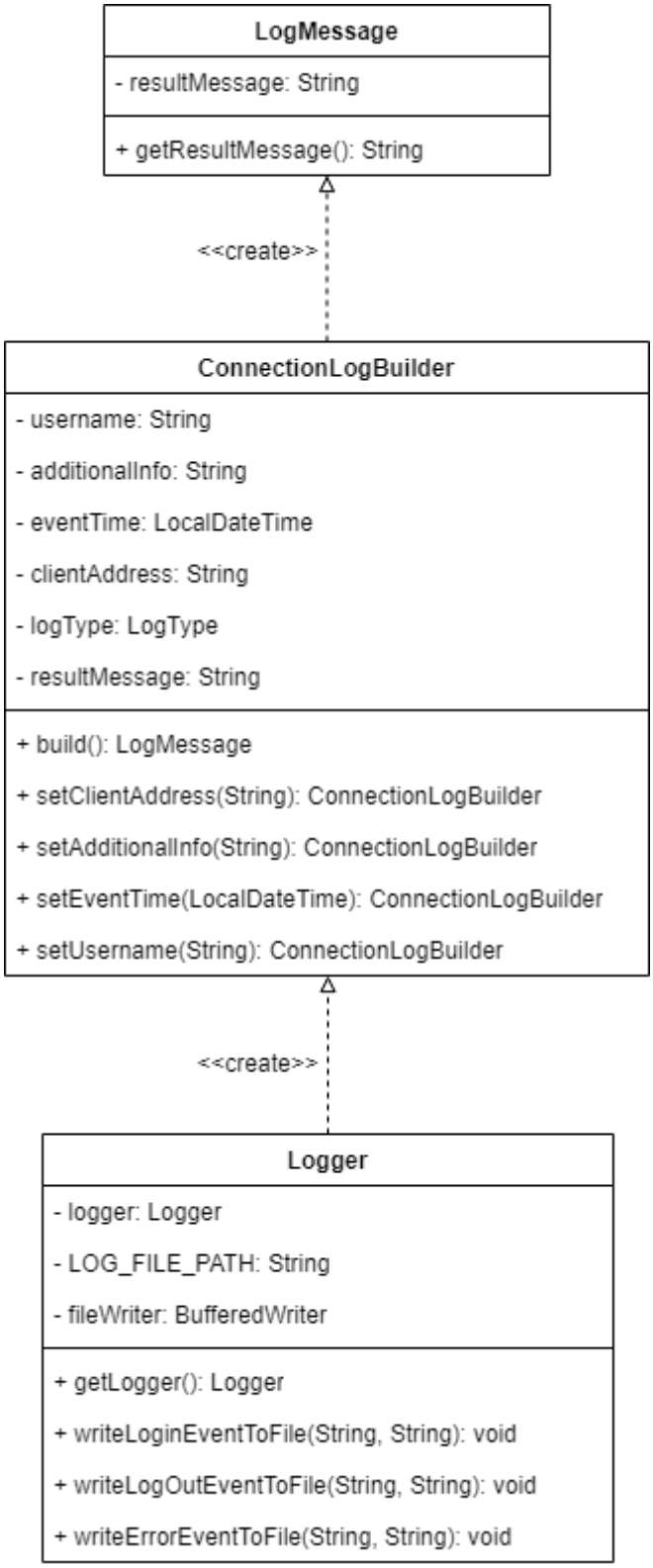

Рисунок 1 – реалізація патерну Builder

Опис реалізації патерну:

Шаблон Builder використовується для створення об'єктів, які представляють записи в лог-файлі. Оскільки лог-файл може містити різноманітну інформацію, використання білдера дозволяє послідовно та гнучко створювати об'єкти логів і додавати до них необхідні атрибути.

Наприклад, для кожної події, що відбувається на сервері (спроба входу або виходу), білдер може допомагати створювати об'єкт логу з необхідною інформацією (час, користувач, дія). Використання білдера полегшує процес формування логів та забезпечує їхню структурованість.

Логіка роботи:

Покрокове створення об'єкта LogMessage:

- Клас ConnectionLogBuilder дозволяє встановлювати параметри (username, eventTime, clientAddress, тощо) поступово, через методи setClientAddress, setAdditionalInfom, setEventTime, setUsername.
- Після заповнення всіх необхідних параметрів викликається метод build(), який створює об'єкт LogMessage.

Використання у класі Logger:

- Клас Logger викликає ConnectionLogBuilder для створення логів різних типів (наприклад, ERROR, LOGIN, LOGOUT).
- Після створення LogMessage, він записує його текст у файл за допомогою BufferedWriter.

Приклад записів у лог-файлі:

```
LOG IN: Client 0:0:0:0:0:0:0:1 logged in as admin at 2024-11-20T19:05.
LOG IN: Client 0:0:0:0:0:0:0:1 logged in as admin at 2024-11-20T19:05.
LOG IN: Client 0:0:0:0:0:0:0:1 logged in as admin at 2024-11-20T19:10.
LOG IN: Client 0:0:0:0:0:0:0:1 logged in as admin at 2024-11-20T19:17.
```

Бачимо записи входу на сервер: вказано ip-адресу користувача (ipv4 або ipv6 - залежить від способу входу на сервер), юзернейм та час входу.

```
PS D:\Лабораторні роботи\5 семестр\Технології розроблення програмного забезпечення\SDT-labs\ftp-server> ftp 127.0.0.1
Connected to 127.0.0.1.
220 Successfully connected.
502 Command not implemented.
User (127.0.0.1:(none)): admin
331 Username okay, need password.
Password:
230 User successfully logged in.
```

Рисунок 2 – процес входу на сервер

Вихідний код системи:

Вихідний код системи розміщено на GitHub: [посилання]

Висновок: під час виконання даної лабораторної роботи було реалізовано частину функціоналу робочої програми у вигляді класів та їхньої взаємодії із застосуванням одного із розглянутих шаблонів проектування – а саме шаблону Builder. Існуючий код було оновлено для використання відповідного паттерну.