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 №4

## **Технологія розробки програмного забезпечення**

*Шаблони «Singleton», «Iterator», «Proxy», «State», «Strategy»*

Виконала студентка
групи ІА-23:
Кашуб’як С. М.

Перевірив:
Мягкий М. Ю.

Київ 2024

**Тема:** Шаблони «Singleton», «Iterator», «Proxy», «State», «Strategy»

**Мета:** Метою даної лабораторної роботи є ознайомлення з шаблонами проєктування, зокрема з шаблоном "Strategy", та їх практичне застосування при розробці програмного забезпечення.

**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:**

**..14 Архіватор (strategy, adapter, factory method, facade, visitor, p2p)**

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

**ЗМІСТ**

## Теоретичні відомості

**Шаблони проєктування (або патерни)** — це перевірені практикою рішення загальних проблем проєктування програмного забезпечення. Вони описують стандартні підходи, які можна застосувати в різних ситуаціях, щоб вирішити певні проблеми, зберігаючи при цьому структуру системи зрозумілою та підтримуваною. Шаблони проєктування мають загальновживані назви, що дозволяє розробникам легко обговорювати та впроваджувати відповідні підходи.

Основні переваги шаблонів проєктування:

- Зменшення часу та зусиль на створення архітектури.
- Гнучкість та адаптованість системи до змін.
- Полегшення підтримки та розвитку системи.
- Стійкість системи до змін та спрощення інтеграції з іншими системами.

**Шаблон "Стратегія" (Strategy)** — це поведінковий шаблон проектування, який дозволяє змінювати поведінку об'єкта під час виконання програми, вибираючи з кількох варіантів алгоритмів (стратегій). Він дозволяє визначити сімейство алгоритмів, інкапсулювати кожен з них і зробити їх взаємозамінними. Клієнтський код може змінювати стратегію об'єкта, не змінюючи його структуру.

Основні компоненти:

1. Контекст (Context) - клас, що використовує стратегію та делегує виконання конкретної поведінки стратегії.
2. Стратегія (Strategy) - інтерфейс або абстрактний клас, що визначає загальний метод для всіх конкретних стратегій.
3. Конкретні стратегії (Concrete Strategies) - класи, які реалізують конкретні алгоритми або поведінки.

Переваги:

- Легко змінювати алгоритми під час виконання.
- Спрощує додавання нових варіантів поведінки без змін у клієнтському коді.
- Підвищує гнучкість і масштабованість програми.

Недоліки:

- Збільшується кількість класів у програмі.
- Клієнтський код відповідає за вибір стратегії.

# Хід Роботи

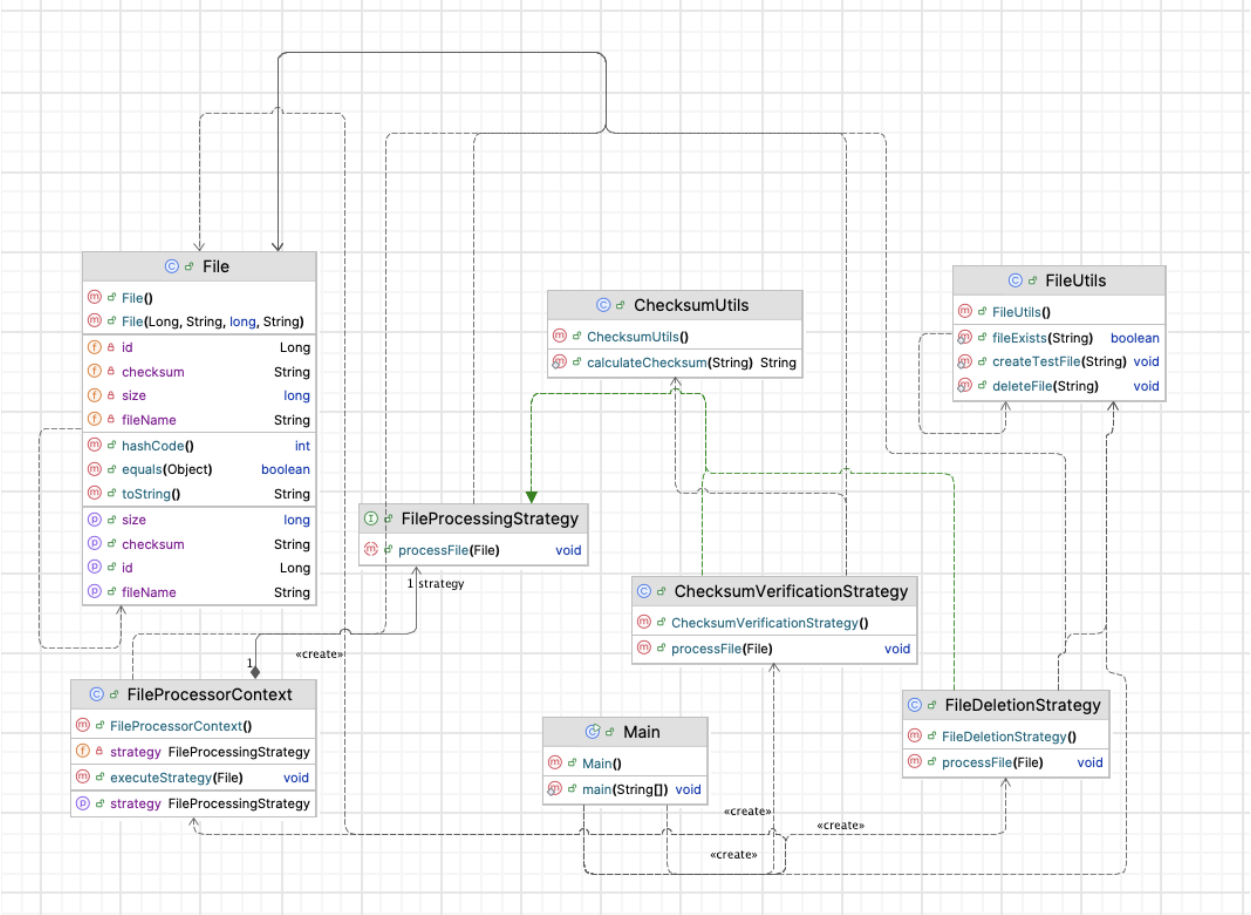


Рис 1. Діаграма класів

## Діаграма класів

1. Контекст – на діаграмі FileProcessorContext пов'язаний із загальним інтерфейсом FileProcessingStrategy через композицію.
2. Загальний інтерфейс – інтерфейс FileProcessingStrategy визначає метод processFile, який реалізують конкретні стратегії.
3. Конкретні стратегії – діаграма показує класи ChecksumVerificationStrategy і FileDeletionStrategy, які реалізують інтерфейс. Це відповідає структурі шаблону.
4. Заміна алгоритмів – на діаграмі видно, що FileProcessorContext може динамічно використовувати різні реалізації FileProcessingStrategy.

```
Calculated checksum: e7cb632359a2be17c1008b50f9ec85691cd5d66834d5fe8f63ef65ceb06682ee
File checksum is valid for: example.txt
File deleted: example.txt
```

Рис 2. Застосування шаблону при реалізації програми

**Робота патерну**

1. Створення тестового файлу – використовується метод createTestFile з класу FileUtils, щоб створити файл “example.txt”, якщо він ще не існує. Це імітує створення файлу, який потім обробляється.
2. Ініціалізація об’єкта файлу - створюється об’єкт класу File і встановлюються його властивості, такі як ім’я файлу та контрольна сума.
3. Контекст та стратегії – створюється об’єкт контексту FileProcessorContext, який буде змінювати стратегію обробки файлу. Спочатку контексту передається стратегія перевірки контрольної суми (ChecksumVerificationStrategy), і викликається метод executeStrategy, що перевіряє, чи збігається контрольна сума файлу з вказаною. Потім контексту передається стратегія видалення файлу (FileDeletionStrategy), і викликається метод executeStrategy, що видаляє файл.
4. Робота стратегій – ChecksumVerificationStrategy: Перевіряє, чи збігається контрольна сума файлу з вказаною в об’єкті File. Для цього використовує метод ChecksumUtils.calculateChecksum(). FileDeletionStrategy: Викликає метод deleteFile з класу FileUtils, щоб видалити файл, якщо він існує.

**Приклад роботи патерну**

1. Програма створює файл “example.txt”, якщо його ще немає, і встановлює його контрольну суму.
2. Спочатку застосовується стратегія ChecksumVerificationStrategy – вона перевіряє контрольну суму файлу. Якщо контрольна сума збігається, виводиться повідомлення “File checksum is valid for: example.txt”. В іншому випадку - повідомлення про невірну контрольну суму.
3. Потім програма застосовує стратегію FileDeletionStrategy - вона видаляє файл “example.txt”, якщо він існує.

**Переваги використання шаблону Strategy у цьому випадку:**

- Гнучкість: Можна легко додавати нові стратегії обробки файлів (наприклад, для архівації або шифрування файлів), не змінюючи клас контексту.
- Розширюваність: Нові стратегії можна додавати без зміни існуючого коду - лише реалізуючи новий клас стратегії, що реалізує інтерфейс.
- Читабельність та тестування: Кожна стратегія є незалежною одиницею, яку можна тестувати окремо. Це покращує підтримку та спрощує тестування.

**Висновок**: отже, у ході виконання лабораторної роботи було реалізовано шаблон strategy, який дозволяє замінити складний набір умов і операцій, що обробляються в одному класі, на набір взаємозамінних стратегій, що значно полегшує управління поведінкою програми.