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 №7

Технології розроблення програмного забезпечення

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

Варіант 30

Виконав
студент групи ІА-13
Якін С. О.

Перевірив:
Мягкий М.Ю.

Київ 2023

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

Завдання:

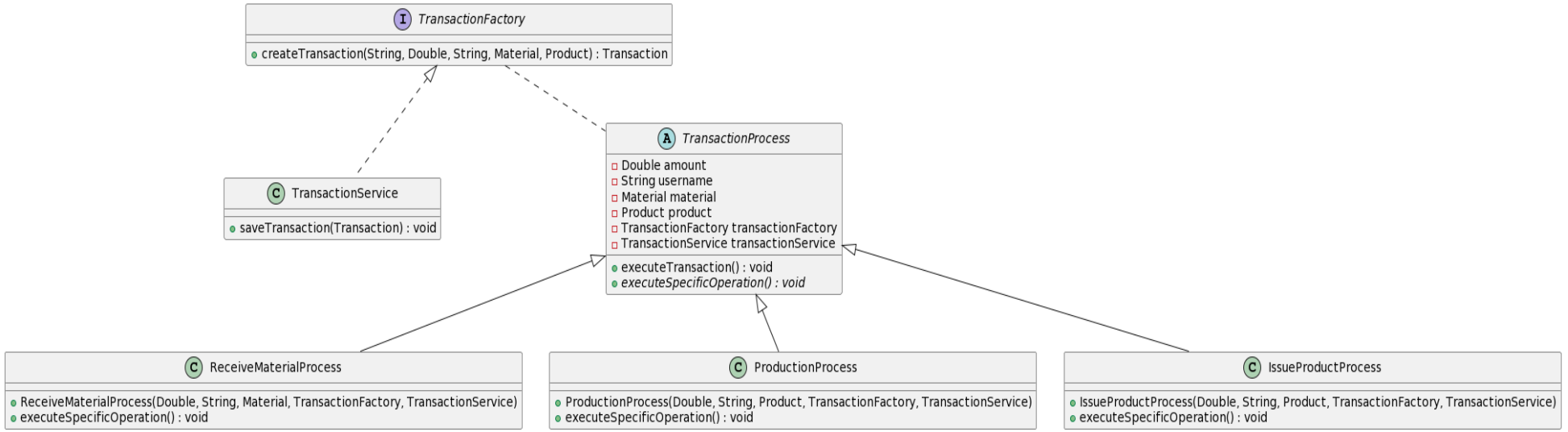
1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми.

Хід роботи:

Тема« Система складського обліку виробництва »

Згідно мого варіанту, було виконано шаблон проектування «Template Method». Шаблон проектування "Template Method" реалізований у абстрактному класі TransactionProcess. Цей клас визначає кістяк алгоритму виконання транзакції, задаючи послідовність кроків: ініціалізація, виконання операції (час виготовлення, видачі та отримання на складі), і її завершення.

Основний принцип патерну полягає в тому, щоб винести логіку примітивної дії в окремий абстрактний метод, і реалізовувати саму логіку в різних класах: на виході ми завжди отримуватимемо однаковий результат, але різними способами.



Метод executeTransaction() є "шаблонним методом", який викликає інші кроки, деякі з яких визначені як абстрактні, наприклад executeSpecificOperation(). Цей абстрактний метод потім перевизначається у підкласах, таких як ReceiveMaterialProcess, ProductionProcess, і IssueProductProcess, для надання конкретної реалізації кроку виконання. На діаграмі представлено ієрархію класів, де TransactionProcess є базовим класом, що визначає загальну структуру алгоритму (template). Від нього успадковані конкретні процеси, кожен з яких реалізує спеціалізовану частину алгоритму. Це дозволяє уніфікувати та стандартизувати процес виконання транзакцій у системі, забезпечуючи одночасно й гнучкість у реалізації специфічних дій. TransactionFactory та TransactionService є допоміжними компонентами, які використовуються в алгоритмі для створення та зберігання транзакцій відповідно. Використання "Template Method" допомагає забезпечити, що всі транзакції виконуються послідовно та єдинообразно, в той час як деталі кожної конкретної транзакції можуть варіюватися залежно від потреб складської системи.

```
2024-01-18T08:23:25.409-05:00  INFO 26492 --- [nio-8080-exec-8] c.e.warehouse.visitors.ReportingVisitor  : User: Sofia is managing warehouse operations.
2024-01-18T08:23:32.084-05:00  INFO 26492 --- [nio-8080-exec-5] c.e.w.process.TransactionProcess         : Starting Transaction
2024-01-18T08:23:37.105-05:00  INFO 26492 --- [nio-8080-exec-5] c.e.w.process.TransactionProcess         : Ending Transaction
2024-01-18T08:24:03.742-05:00  INFO 26492 --- [nio-8080-exec-2] c.e.w.process.TransactionProcess         : Starting Transaction
2024-01-18T08:24:08.745-05:00  INFO 26492 --- [nio-8080-exec-2] c.e.w.process.TransactionProcess         : Ending Transaction
```

Результат обробки дії у системі

Як ми бачимо час для кожної транзакції займає 5 секунд, тим самим позначає час виготовлення/отримання/видачі продукції на складі.

Код:

abstract class TransactionProcess:

```java
public abstract class TransactionProcess {
    3 usages
    private static final Logger logger = LoggerFactory.getLogger(TransactionProcess.class);

    protected Double amount;
    protected String username;
    protected Material material;
    protected Product product;
    4 usages
    protected TransactionFactory transactionFactory;
    4 usages
    protected TransactionService transactionService;
    3 usages
    public TransactionProcess(Double amount, String username, Material material, Product product,
                              TransactionFactory transactionFactory, TransactionService transactionService) {
        this.amount = amount;
        this.username = username;
        this.material = material;
        this.product = product;
        this.transactionFactory = transactionFactory;
        this.transactionService = transactionService;
    }
    3 usages
    public final void executeTransaction() {
        try {
            startTransaction();
            Thread.sleep( millis: 5000);
            executeSpecificOperation();
            endTransaction();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            logger.error("Transaction process interrupted", e);
        }
    }
    1 usage
    private void startTransaction() { logger.info("Starting Transaction"); }
    1 usage  3 implementations
    protected abstract void executeSpecificOperation();
    1 usage
    private void endTransaction() { logger.info("Ending Transaction"); }
}
```

IssueProductProcess:

```java
public class IssueProductProcess extends TransactionProcess {
    1 usage
    public IssueProductProcess(Double amount, String username, Product product,
                               TransactionFactory transactionFactory,
                               TransactionService transactionService) {
        super(amount, username, material: null, product, transactionFactory, transactionService);
    }

    1 usage
    @Override
    protected void executeSpecificOperation() {
        Transaction transaction = transactionFactory.createTransaction( type: "ISSUE", amount, username, material: null, product);
        transactionService.saveTransaction(transaction); // Use transactionService to save the transaction
    }
}
```

ProductionProcess:

```java
public class ProductionProcess extends TransactionProcess {
    1 usage
    public ProductionProcess(Double amount, String username, Product product,
                             TransactionFactory transactionFactory,
                             TransactionService transactionService) {
        super(amount, username, material: null, product, transactionFactory, transactionService);
    }

    1 usage
    @Override
    protected void executeSpecificOperation() {
        Transaction transaction = transactionFactory.createTransaction( type: "PRODUCTION", amount, username, material: null, product)
        transactionService.saveTransaction(transaction); // Use transactionService to save the transaction
    }
}
```

ReceiveMaterialProcess:

```java
public class ReceiveMaterialProcess extends TransactionProcess {

    1 usage
    public ReceiveMaterialProcess(Double amount, String username, Material material,
                                  TransactionFactory transactionFactory,
                                  TransactionService transactionService) {
        super(amount, username, material, product: null, transactionFactory, transactionService);
    }

    1 usage
    @Override
    protected void executeSpecificOperation() {
        Transaction transaction = transactionFactory.createTransaction( type: "RECEIVE", amount, username, material, product: null);
        transactionService.saveTransaction(transaction);
    }
}
```

Висновок: на цій лабораторній роботі я познайомився з такими паттернами, як «Mediator», «Facade», «Bridge», «Template Method», а також розібрався у принципі їх роботи та сфері використання. На практиці спроектував та реалізував паттерн «Template Method» у своєму проекті.