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 №6

Теорія Розробки Програмного Забезпечення

ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Варіант 23

Виконав

студент групи ІА-14:

Фіалківський І.О.

Перевірив:

Мягкий М.Ю.

Київ 2023

Тема: ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

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

Хід роботи:

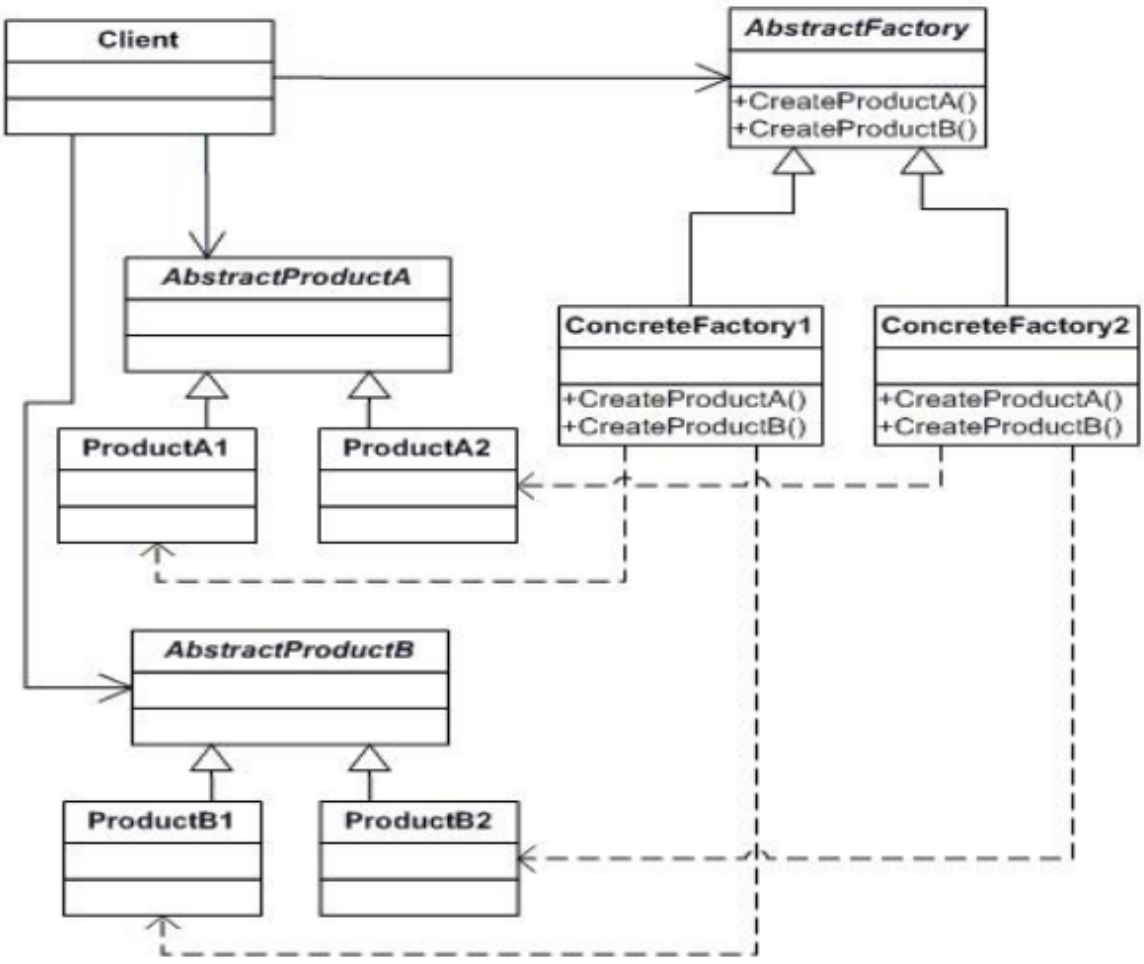
Тема: Згідно мого варіанту, було виконано шаблон проектування «Abstract Factory».

Project Management software (proxy, chain of responsibility, abstract factory, bridge, flyweight, client-server)

Абстрактна фабрика, як і всі фабричні патерни, допомагає нам правильно організувати створення нових об'єктів. З її допомогою ми керуємо "випуском" різних сімейств взаємозалежних об'єктів.

Шаблон «Abstract Factory»

Структура:



Цей шаблон передусім структурує знання про схожі об'єкти і створює можливість взаємозаміни різних сімейств (робота з Oracle ведеться також, як і робота з SQL Server). Проте, при використанні такої схеми украй незручно розширювати фабрику - для додавання нового методу у фабрику необхідно додати його в усіх фабриках і створити відповідні класи, що створюються цим методом.

Інтерфейс `ValidationFactory` є абстрактним інтерфейсом factory.

В ньому задекларовний метод `createValidationChain` який створює `ValidationChain` та повертає перший у списку валідатор.

```
package com.example.util.validation.factory;

import com.example.util.validation.ValidationChain;

public interface ValidationFactory<T> {
    ValidationChain<T> createValidationChain();
}
Інтерфейс ValidationFactory визначає обов'язковий метод для створення ланцюжка валідації, і реалізації цього інтерфейсу будуть відповідати за конкретну реалізацію створення ланцюжка валідації для певного типу об'єкта.
```

Реалізація абстрактної фабрики валідації а конкретно `TaskValidationFactory`. Прописана реалізація `createValidationChain` для task. В цій реалізації викликається статичний метод на інтерфейсі `ChainElement` який створює чергу з наявних валідаторів. Після цього ми створюємо `TaskValidationChain` передаючи в нього список перший `validationStep`

```
package com.example.util.validation.factory;

import com.example.entity.Task;
import com.example.util.validation.validator.AbstractValidationStep;
import com.example.util.validation.validator.ChainElement;
import com.example.util.validation.TaskValidationChain;
import com.example.util.validation.ValidationChain;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.List;

@Component
```

```
public class TaskValidationFactory implements ValidationFactory<Task> {

    private final List<AbstractValidationStep<Task>> validationSteps;

    @Autowired
    public TaskValidationFactory(List<AbstractValidationStep<Task>>
validationSteps) {
        this.validationSteps = validationSteps;
    }

    @Override
    public ValidationChain<Task> createValidationChain() {
        AbstractValidationStep<Task> chainHead =
ChainElement.buildChain(validationSteps);
        return new TaskValidationChain(chainHead);
    }
}
```

Отже, ця реалізація фабрики валідації для об'єктів типу Task дозволяє автоматично створювати ланцюжок валідаторів з переданих валідаторів та використовувати його для валідації об'єктів типу Task.

Використання цього фактрі клієнтом

```
private final ValidationChain<Task> validation;

@Autowired
public TaskController(ModelMapper modelMapper, TaskService taskService,
ValidationFactory<Task> validationFactory) {
    this.modelMapper = modelMapper;
    this.taskService = taskService;
    this.validation = validationFactory.createValidationChain();
}
```

Отже, конструктор TaskController автоматично ініціалізує валідаційний ланцюжок для об'єктів типу Task за допомогою фабрики валідації (validationFactory). Це дозволяє легко використовувати ланцюжок для валідації об'єктів у методах контролера.

Переглянути весь код можна за посиланням:
https://github.com/vergovters/trpz-spring/tree/master

Конкретні частини реалізаці:

https://github.com/vergovters/trpz-spring/blob/master/trpz/demo/src/main/java/com/example/util/validation/factory/ValidationFactory.java

https://github.com/vergovters/trpz-spring/blob/master/trpz/demo/src/main/java/com/example/util/validation/factory/TaskValidationFactory.java

Висновок: шаблон абстрактної фабрики виявився потужним інструментом для організації створення взаємозалежних об'єктів в системі. Його використання дозволило ефективно керувати "випуском" різних сімейств об'єктів, забезпечуючи гнучкість і можливість легкої заміни об'єктів в системі.Загалом, використання абстрактної фабрики у поєднанні із шаблоном "ланцюжок відповідальності" сприяє покращенню структури системи, зробивши її більш гнучкою, розширюваною та легко зрозумілою.