Лабораторная работа №10. Шаблон проектирования «Абстрактная фабрика»

Теория

Шаблоны (паттерны) проектирования - это готовые к использованию решения часто возникающих в программировании задач. Это не класс и не библиотека, которую можно подключить к проекту, это нечто большее. Паттерны проектирования, подходящий под задачу, реализуется в каждом конкретном случае. Следует, помнить, что такой паттерн, будучи примененным неправильно или к неподходящей задаче, может принести немало проблем. Тем не менее, правильно примененный паттерн поможет решить задачу легко и просто.

Типы паттернов:

- *порождающие*
- *структурные*
- *поведенческие*

Порождающие паттерны предоставляют механизмы инициализации, позволяя создавать объекты удобным способом. Структурные паттерны определяют отношения между классами и объектами, позволяя им работать совместно. Поведенческие паттерны используются для того, чтобы упростить взаимодействие между сущностями.

Порождающие:

- **Singleton** (Одиночка) - ограничивает создание одного экземпляра класса, обеспечивает доступ к его единственному объекту.
- **Factory** (Фабрика) - используется, когда у нас есть суперкласс с несколькими подклассами и на основе ввода, нам нужно вернуть один из подкласса.
- **Abstract Factory** (Абстрактная фабрика) - используем супер фабрику для создания фабрики, затем используем созданную фабрику для создания объектов.
- **Builder** (Строитель) - используется для создания сложного объекта с использованием простых объектов. Постепенно он создает больший объект от малого и простого объекта.
- **Prototype** (Прототип) - помогает создать дублированный объект с лучшей производительностью, вместо нового создается возвращаемый клон существующего объекта.

Структурные:

- **Adapter** (Адаптер) - это конвертер между двумя несовместимыми объектами. Используя паттерн адаптера, мы можем объединить два несовместимых интерфейса.
- **Composite** (Компоновщик) - использует один класс для представления древовидной структуры.
- **Proxy** (Заместитель) - представляет функциональность другого класса.
- **Flyweight** (Легковес) - вместо создания большого количества похожих объектов, объекты используются повторно.
- **Facade** (Фасад) - беспечивает простой интерфейс для клиента, и клиент использует интерфейс для взаимодействия с системой.
- **Bridge** (Мост) - делает конкретные классы независимыми от классов реализации интерфейса.
- **Decorator** (Декоратор) - добавляет новые функциональные возможности существующего объекта без привязки его структуры.

Поведенческие:

- **Template Method** (Шаблонный метод) - определяющий основу алгоритма и позволяющий наследникам переопределять некоторые шаги алгоритма, не изменяя его структуру в целом.
- **Mediator** (Посредник) - предоставляет класс посредника, который обрабатывает все коммуникации между различными классами.
- **Chain of Responsibility** (Цепочка обязанностей) - позволяет избежать жесткой зависимости отправителя запроса от его получателя, при этом запрос может быть обработан несколькими объектами.
- **Observer** (Наблюдатель) - позволяет одним обьектам следить и реагировать на события, происходящие в других объектах.
- **Strategy** (Стратегия) - алгоритм стратегии может быть изменен во время выполнения программы.
- **Command** (Команда) - интерфейс команды объявляет метод для выполнения определенного действия.
- **State** (Состояние) - объект может изменять свое поведение в зависимости от его состояния.
- **Visitor** (Посетитель) - используется для упрощения операций над группировками связанных объектов.
- **Interpreter** (Интерпретатор) - определяет грамматику простого языка для проблемной области.
- **Iterator** (Итератор) - последовательно осуществляет доступ к элементам объекта коллекции, не зная его основного представления.
- **Memento** (Хранитель) - используется для хранения состояния объекта, позже это состояние можно восстановить.

Abstract Factory (Абстрактная фабрика)

Описание:

- Позволяет выбрать конкретную реализацию фабрики из семейства возможных фабрик. Создает семейство связанных объектов. Легко расширяется.

Реализация:

```
interface Lada {
    long getLadaPrice();
}
interface Ferrari {
    long getFerrariPrice();
}
interface Porshe {
    long getPorshePrice();
}
interface InteAbsFactory {
    Lada getLada();
    Ferrari getFerrari();
    Porshe getPorshe();
}
class UaLadaImpl implements Lada {// первая
    public long getLadaPrice() {
        return 1000;
    }
}
class UaFerrariImpl implements Ferrari {
    public long getFerrariPrice() {
        return 3000;
    }
}
class UaPorsheImpl implements Porshe {
    public long getPorshePrice() {
        return 2000;
    }
}
class UaCarPriceAbsFactory implements InteAbsFactory {
    public Lada getLada() {
        return new UaLadaImpl();
    }
    public Ferrari getFerrari() {
        return new UaFerrariImpl();
    }
    public Porshe getPorshe() {
        return new UaPorsheImpl();
    }
}// первая
class RuLadaImpl implements Lada {// вторая
    public long getLadaPrice() {
        return 10000;
    }
}
class RuFerrariImpl implements Ferrari {
    public long getFerrariPrice() {
        return 30000;
    }
}
class RuPorsheImpl implements Porshe {
    public long getPorshePrice() {
```

```
        return 20000;
    }
}
class RuCarPriceAbsFactory implements InteAbsFactory {
    public Lada getLada() {
        return new RuLadaImpl();
    }
    public Ferrari getFerrari() {
        return new RuFerrariImpl();
    }
    public Porshe getPorshe() {
        return new RuPorsheImpl();
    }
}// вторая

public class AbstractFactoryTest {//тест
    public static void main(String[] args) {
        String country = "UA";
        InteAbsFactory ifactory = null;
        if(country.equals("UA")) {
            ifactory = new UaCarPriceAbsFactory();
        } else if(country.equals("RU")) {
            ifactory = new RuCarPriceAbsFactory();
        }

        Lada lada = ifactory.getLada();
        System.out.println(lada.getLadaPrice());
    }
}
```

Задание

Реализовать шаблон «Абстрактная фабрика», взяв за основу свой проект, созданный в последних лабораторных работах.