**Аннотация**

В данной статье рассматриваются преимущества и недостатки применения паттернов проектирования при разработке клиент-серверного приложения на фреймворке SpringBoot.

/**/

**Ключевые слова:** паттерн проектирования, Java, строитель, Шаблонный метод

## Введение

В настоящее время многие области жизни человека переходят в цифру, в частности в клиент-серверное приложение, и для того, чтобы этот переход был быстрее, менее трудозатратным и более эффективным программисты создали паттерны проектирования.

Паттерны (шаблоны) проектирования - проверенные и готовые к использованию решения часто возникающих в повседневном программировании задач. [1]

Клиент-серверное приложение – приложение, которое состоит из 2-х частей: клиентская часть, иначе пользовательский интерфейс, и серверная часть, которая выполняет основную логику обработки данных.

## Основная часть

При разработке клиент-серверного приложения, в данной статье будет рассматриваться WEB-приложение, созданное с помощью стека технологий SpringBoot, язык Java, Node.js, язык JavaScript, PostgreSQL – хранение данных.

Сначала рассмотрим виды паттернов проектирования. Всего существует на данный момент 3 вида: Структурные, порождающие и поведенческие. Порождающие паттерны – используются для гибкого создания объектов, без внесения в программу лишних зависимостей. Структурные паттерны требуются для построения связей между объектами. Поведенческие паттерны помогают создать эффективную коммуникацию между объектами. [2]

Паттерны проектирования применяются вне зависимости от языка программирования, в данной статье взят для примера язык Java. Во время разработки системы может возникнуть ситуация, когда требуется постоянно создавать сложный объект, состоящий из нескольких полей, к примеру кастомный запрос, в котором может быть n-ое количество параметров, фильтров. Создание обычным способом, через оператор new будет трудозатратно, так как нужно неопределенное количество конструкторов на разное количество входных данных. Для решения данной проблемы необходимо использовать паттерн строитель, или же Builder [3]. Для реализации данного паттерна необходимо учесть несколько условий:

1. Должен быть класс, к примеру *Query*, содержащий n полей. В данном классе необходим вложенный статический класс, названный следующим образом: *имя внешнего класса*Builder, в данном случае *QueryBuilder*, и поля вложенного класса точно такие же, как и поля внешнего класса;
2. Все методы вложенного класса, кроме метода *build*, должны возвращать этот экземпляр класса, *this*;
3. Во вложенном классе должен быть метод *build*, вызывающийся после выполнения всех остальных методов, то есть после настройки данного объекта, который возвращает готовый экземпляр внешнего класса;
4. Во внешнем классе необходим конструктор, который принимает входным параметром вложенный класс Builder в качестве аргумента. [4]

На рисунке 1.1 видно, что создан класс Query с четырмя полями, а именно cls, sql, params, replace и двумя методами execute и executeOne, что для паттерна не так важно, так как это функциональность самого класса. В классе Query есть статический класс QueryBuilder, с теми же полями и установкой значений в эти поля, в конце метод build, который возвращает класс Query.

```
public class Query {
    private Class cls;
    private String sql;
    private Map<String, Object> params;
    private Map<String, String> replace;
    public Query(QueryBuilder queryBuilder){...}
    public List<Object> execute(){...}
    public Object executeOne(Integer id){...}
    public static class QueryBuilder{
        private Class cls;
        private String sql;
        private Map<String, Object> params = new HashMap<>();
        private Map<String, String> replace = new HashMap<>();
        public QueryBuilder(String sql) { this.sql = sql; }
        public QueryBuilder forClass(Class cls){...}
        public QueryBuilder setParams(Map<String, Object> params){...}
        public QueryBuilder setParam(String paramName, Object paramValue){...}
        public QueryBuilder injectSql(String placeholder, String sql){...}
        public QueryBuilder injectSqlIf(boolean condition, String placeholder, String sql){...}
        public Query build() { return new Query( queryBuilder: this); }
    }
}
```

Рисунок 1.1. Паттерн строитель. Создание класса и вложенного статического класса

Для применения данного паттерна необходимо вызвать конструктор со статическим классом, методы для настройки строителя и в конце метод build.

```
public class Main {
    public static void main(String[] args) {
        Query query = new Query.QueryBuilder( sql: "SELECT * FROM table WHERE 1=1 /*PLACEHOLDER*/ /*PLACEHOLDER_IF*/")
                .forClass(Main.class)
                .injectSql( placeholder: "/*PLACEHOLDER*/", sql: "AND column_1 = :value")
                .injectSqlIf( condition: true, placeholder: "/*PLACEHOLDER_IF*/", sql: "AND column_2 = :value_2")
                .setParam( paramName: "value", paramValue: 1)
                .setParam( paramName: "value_2", paramValue: 1)
                .build();
    }
}
```

Так-же, во время разработки встречается проблема создания однотипных объектов, но с разной логикой в отдельных случаях. К примеру, есть задача создать n-ое количество репозиториев, в которых будет множество одинаковых методов, к примеру insert, update, delete, отвечающие за вставку, обновление и удаление записи соответственно, и 3 отличающихся метода, а именно load, drop, create, отвечающие за загрузку, удаление и создание таблицы соответственно. Для решения данной задачи необходимо использовать паттерн «Шаблонный метод» [5], который требует создания одного класса родительского, в данном случае TableRepository с 3-мя определенными методами (insert, update, delete) и тремя абстрактными методами (load, drop, create). Абстрактные методы переопределяются в наследниках класса.

```
public abstract class TableRepository {
    public void insert(Object obj){...}

    public void update(Object obj){...}

    public void delete(Object obj){...}

    public abstract void load();
    public abstract void drop();
    public abstract void create();
}
```

Рисунок 2.1. Абстрактный класс-родитель TableRepository

```
public class TableRepositoryFirstChild extends TableRepository{
    @Override
    public void load() {...}

    @Override
    public void drop() {...}

    @Override
    public void create() {...}
}
```

Рисунок 2.2. Первый класс наследуемый от TableRepository

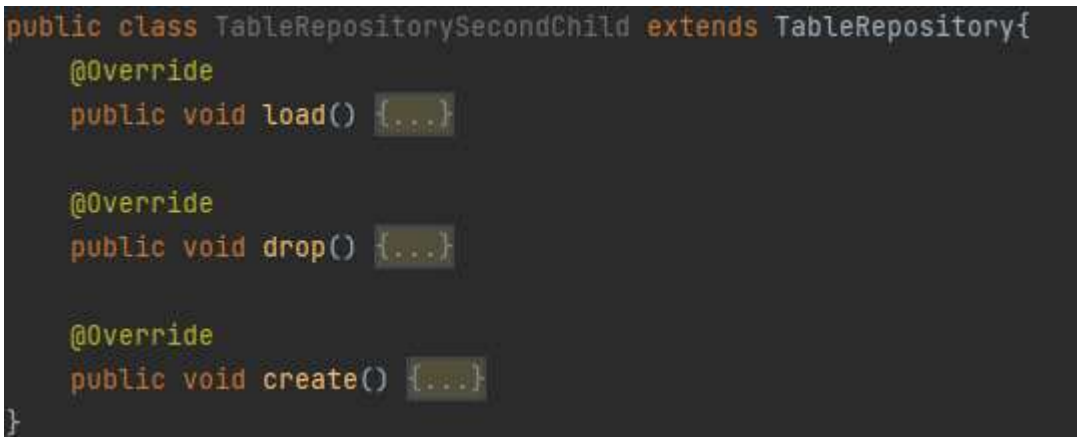


Рисунок 2.3. Второй класс, наследуемый от TableRepository


Рисунок 2.4. Применение паттерна проектирования «Шаблонный метод»

Как видно из примера выше, шаблонный метод позволяет создать массив объектов TableRepository и вызвать методы дочерних классов drop, create, load.

**Вывод**

В разработке клиент-серверного приложения часто возникают проблемы создания объектов, связей объектов, и для решения практически любой задачи можно найти подходящий паттерн проектирования. Паттерны, или как их еще называют, шаблоны, помогают в написании кода, ведь это своего рода инструкция, которая позволяет решить поставленную задачу наименьшими усилиями.

**Список литературы.**


1. https://tproger.ru/translations/design-patterns-for-beginners/
2. https://javarush.com/quests/lectures/questservlets.level16.lecture00
3. https://javarush.com/groups/posts/2267-patternih-proektirovanija-v-java
4. https://javarush.com/groups/posts/3822-kofe-breyk-124-pattern-proektirovanija-builder-kak-rabotaet-serializacija-i-deserializacija-v-j
5. https://topjava.ru/blog/pattern-shablonnyy-metod-v-java