

ШАБЛОНЫ ПРОЕКТИРОВАНИЯ BUILDER И SINGLETONE В JAVA

Шаблон проектирования **Builder** используется для создания объектов с комплексной конфигурацией, позволяя создавать объекты пошагово или с различными вариациями, сохраняя при этом читаемость кода и гибкость настроек [1].

Цели шаблона Builder:

1. Упрощение создания сложных объектов: когда объект имеет множество параметров или настроек, использование многочисленных конструкторов может быть неудобным или запутанным. Builder позволяет создавать объекты пошагово, добавляя нужные компоненты по мере необходимости.

2. Изоляция процесса конструирования объекта: Builder помогает скрыть сложность создания объекта от клиентского кода, позволяя ему использовать простой и интуитивно понятный интерфейс.

Ключевые элементы шаблона Builder:

Director (Директор): управляющий объект, который определяет порядок шагов конструирования объекта через Builder.

Builder (Строитель): интерфейс или абстрактный класс, определяющий методы для создания частей объекта.

ConcreteBuilder (Конкретный строитель): интерфейс Builder, предоставляющий конкретную реализацию методов для создания и сборки частей объекта.

Product (Продукт): итоговый объект, который создается с использованием Builder.

На рисунках 1 и 2 представлен пример простой реализации шаблона Builder на JavaScript для создания объекта Car. Класс CarBuilder используется для пошагового создания объекта Car с помощью методов установки значений и метода build(), который фактически создает объект Car с помощью передачи самого себя (экземпляра CarBuilder) в конструктор Car.

```
1 class Car {
2   constructor(builder) {
3     this.make = builder.make;
4     this.model = builder.model;
5     this.year = builder.year;
6   }
7 }
8
9 class CarBuilder {
10  constructor(make, model) {
11    this.make = make;
12    this.model = model;
13  }
14
15  setYear(year) {
16    this.year = year;
17    return this;
18  }
19
20  build() {
21    return new Car(this);
22  }
23 }
24
25 // Использование Builder для создания объекта Car
26 const myCar = new CarBuilder('Toyota', 'Corolla').setYear(2023).build();
27 console.log(myCar);
```

Рисунок 1 – Пример шаблона Builder

```
Car { make: 'Toyota', model: 'Corolla', year: 2023 }
```

Рисунок 2 – Конструктор Car

В данном случае, создается экземпляр myCar с помощью Builder, указывается марка и модель, а также год выпуска.

Шаблон проектирования **Singleton** относится к порождающим шаблонам и используется для создания класса, который гарантирует наличие только одного экземпляра этого класса во всем приложении. Шаблон Singleton предоставляет глобальную точку доступа к этому экземпляру [2].

Особенности шаблона Singleton:

Одиночка (Singleton): это класс, который имеет статическое поле для хранения единственного экземпляра самого себя и метод для получения этого экземпляра.

Приватный конструктор: Singleton обычно имеет приватный конструктор, чтобы предотвратить создание объектов через оператор new.

Статический метод доступа: обычно есть статический метод, который возвращает экземпляр Singleton, создавая его при первом вызове и возвращая сохраненный экземпляр при последующих вызовах.

Singleton гарантирует, что всегда будет существовать только один экземпляр класса в приложении. Это полезно, когда требуется обеспечить доступ к общему ресурсу из разных частей программы или когда нужен объект с глобальной областью видимости. Однако использовать Singleton следует осторожно, чтобы избежать нежелательных связей и затруднений в тестировании кода.

На рисунках 3 и 4 представлен пример, в котором конструктор Singleton проверяет наличие существующего экземпляра класса. Если экземпляр еще не создан, создается новый и сохраняется в статической переменной Singleton.instance. Последующие вызовы конструктора возвращают уже существующий экземпляр.

```
1 class Singleton {
2   constructor() {
3     if (!Singleton.instance) {
4       Singleton.instance = this;
5       this.data = Math.random(); // Для демонстрации добавим случайное значение
6     }
7
8     return Singleton.instance;
9   }
10
11  getData() {
12    return this.data;
13  }
14 }
15
16 // Использование Singleton
17 const instance1 = new Singleton();
18 const instance2 = new Singleton();
19
20 console.log(instance1.getData()); // Выводим данные из Singleton
21 console.log(instance2.getData()); // Выводим те же данные, так как это один и тот же экземпляр Singleton
22 console.log(instance1 === instance2); // true - это один и тот же экземпляр
```

Рисунок 3 – Пример шаблона Singleton

```
0.7404373355133269
0.7404373355133269
true
```

Рисунок 4 – Вывод результата

Это простой способ создания Singleton в JavaScript гарантирует, что всегда будет существовать только один экземпляр класса Singleton и обеспечить доступ к этому экземпляру из любой точки программы.

Таким образом, использование шаблонов проектирования Builder и Singleton упрощает решение типовых задач, возникающих при разработке сложных приложений.

Литература

1. Строитель [электронный ресурс]. – 2023. – URL: <https://refactoring.guru/ru/design-patterns/builder>. – Дата доступа: 21.02.2024.
2. Одиночка (Singleton) | Паттерны в C# и .NET [электронный ресурс]. – 2023. – URL: <https://metanit.com/sharp/patterns/2.3.php>. – Дата доступа: 10.01.2024.

С. В. Кацубо

(ГГУ имени Ф. Скорины, Гомель)

Науч. рук. **В. Н. Кулинченко**, ст. преподаватель

РАЗРАБОТКА ПОДСИСТЕМЫ РЕГИСТРАЦИИ ПЕРЕСЕЧЕНИЯ ЗОНЫ ПОКРЫТИЯ WI-FI НЕСУЩИХ СТЕН ЗДАНИЯ

Подсистема регистрации разработана для промышленных объектов, которым нужна безопасная Wi-Fi сеть, ограниченная по периметру несущих стен здания. Приложение способно отображать Wi-Fi сеть на чертеже плана этажа, определять критические зоны падения полезного сигнала и его отсутствие. Безопасной сетью считается та сеть, которая не выходит за периметр несущих стен этажа.

Веб-приложение состоит из трех частей: front-end, back-end и база данных. Все эти компоненты взаимодействуют между собой. Для взаимодействия между front-end и back-end используется протокол HTTP. Для взаимодействия между back-end и базой данных используется специальный фреймворк Beego.

Расчет дальности Wi-Fi сигнала происходит на back-end уровне. Изначально сервер принимает запрос в виде JSON с front-end уровнем. В этот запрос входят координаты роутеров, которые указываются пользователем, а также загружается план этажа. На основе этих начальных данных и используя соответствующие формулы происходит расчет дальности Wi-Fi сигнала. Необходимо учитывать то обстоятельство, что при удалении от роутера полезный сигнал все больше затухает. На основании расчета дальности Wi-Fi сигнала происходит прорисовка полученного результата на плане этажа пользователя. Для прорисовки используется восемь цветов, от зеленого, который обозначает, что в этой области наилучший Wi-Fi сигнал, до красного, где сигнал наихудший. Также при прорисовке учитываются физическое расположение точек доступа, а также конечные координаты, до каких пор будет распространяться полезный сигнал, и анализируется изображение для поиска препятствий между двумя точками. Из-за различных препятствий с различным коэффициентом радиопроницаемости происходит затухание сигнала полезного сигнала Wi-Fi. Тип препятствий также важен, так как это прямо влияет на коэффициент затухания сигнала. После того, как создана карта покрытия Wi-Fi сигнала, она отображается на экране пользователя.