Международный журнал прикладных наук и технологий "Integral"

Научная статья

Original article

DOI 10.55186/27131424_2023_5_6_2

ПАТТЕРН ПРОЕКТИРОВАНИЯ МОСТ В JAVA РАЗРАБОТКЕ

BRIDGE DESIGN PATTERN IN JAVA DEVELOPMENT

Михиенков Кирилл Сергеевич, студент, федеральное государственное бюджетное образовательное учреждение высшего образования "МИРЭА - Российский технологический университет" РТУ МИРЭА (119454 Россия, г. Москва, ул. проспект Вернадского, д. 78, стр. 4), тел. +7(495)999-51-61, ORCID: http://orcid.org/, mikheenkov@mirea.ru

Узоров Кирилл Александрович, студент, федеральное государственное бюджетное образовательное учреждение высшего образования "МИРЭА - Российский технологический университет" РТУ МИРЭА (119454 Россия, г. Москва, ул. проспект Вернадского, д. 78, стр. 4), тел. +7(495)471-14-47, ORCID: http://orcid.org/, uzorov@mirea.ru

Бузыкова Юлия Сергеевна, доцент, федеральное государственное бюджетное образовательное учреждение высшего образования "МИРЭА - Российский технологический университет" РТУ МИРЭА (119454 Россия, г. Москва, ул. проспект Вернадского, д. 78, стр. 4), тел. +7(495)681-87-90, ORCID: http://orcid.org/, juliaserg_buz@mail.ru

Kirill M. Sergeevich, student, Federal State Budget Educational Institution of Higher Education «MIREA - Russian Technological University» (78 prospekt Vernadskogo st., p. 4, Moscow, 119454 Russia), tel. +7(495)999-51-61, ORCID: http://orcid.org/, mikheenkov@mirea.ru

Kirill A. Uzorov, student, Federal State Budget Educational Institution of Higher Education «MIREA - Russian Technological University» (78 prospekt Vernadskogo st., p. 4, Moscow, 119454 Russia), tel. +7(495)471-14-47, ORCID: http://orcid.org/, uzorov@mirea.ru

Yulia S. Buzykova, docent, Federal State Budget Educational Institution of Higher Education «MIREA - Russian Technological University» (78 prospekt Vernadskogo st., p. 4, Moscow, 119454 Russia), tel. +7(495)681-87-90, ORCID: http://orcid.org/, juliaserg_buz@mail.ru

Аннотация. Статья посвящена демонстрации эффективности паттерна проектирования Bridge для решения задачи разработчика, возникающей при проектировании информационных систем для построения процесса разработки и систематизации принимаемых решений для выпускаемой продукции, а именно, в части отделения абстракции от реализации в рамках разрабатываемого проекта. Обозначены плюсы и минусы паттерна проектирования Bridge. Приведены листинги реализации паттерна Bridge.

Abstract. The modern approach to the design of information systems involves the use of certain patterns to build the development process and systematize the decisions made for products. Most developers are faced with the task of separating abstraction from implementation within their own project. The example demonstrated the effectiveness of the Bridge design pattern for solving such problems.

Ключевые слова: *Паттерн проектирования мост, Java разработка, проектирования информационных систем*

Keywords: *Bridge design pattern, Java development, information systems design*

Современный подход к проектирования информационных систем предполагает использование определённых паттернов для построения процесса разработки и систематизации принимаемых решений для выпускаемой продукции.

Паттерны проектирования имеют высокую актуальность и используются в настоящее время в различных языках программирования, в том числе и на Java.

Это связано с тем, что они позволяют повысить качество и эффективность разработки программного обеспечения, а также упростить ее сопровождение и расширение.

Перед большинством разработчиков встаёт необходимость разделять абстракцию и реализацию в рамках своего проекта, не перегружая себя работой по созданию тысяч дополнительных вытекающих классов. В такой ситуации очень полезным оказывается паттерн проектирования Bridge.

Особенностью паттерна проектирования Bridge является его эффективность и удобство его использования при необходимости отделять абстракцию от реализации и вносить изменения в одну ветку разработки, не ломая другую.

Рассматриваемый паттерн Bridge используется для уменьшения связанности классов программы. Данный паттерн применяется когда:

Необходимо расширить количество сущностей в две стороны (геометрические фигуры и цвета как будет продемонстрированно в примере ниже).

Если есть желание разделить большой класс, который не отвечает принципу Single responsibility, на более маленькие классы с узкопрофильным функционалом.

При необходимости вносить изменения в логику работы неких сущностей во время работы программы.

При необходимости спрятать реализацию от клиентов класса (библиотеки).

Обозначим плюсы и минусы паттерна проектирования Bridge.

Плюсы:

1.Разделение абстракции и реализации: Это позволяет разработчикам изменять абстракцию и реализацию независимо друг от друга, что делает архитектуру более гибкой и расширяемой.

2.Уменьшение связности: Bridge уменьшает связность между классами, что облегчает тестирование и поддержку кода.

3.Упрощение кода: Bridge позволяет использовать один и тот же код реализации для нескольких абстрактных классов, что упрощает код и снижает его объем.

4.Возможность замены реализации: Bridge позволяет заменять реализацию без изменения кода абстракции.

Минусы:

1.Дополнительная сложность: Разделение абстракции и реализации может привести к дополнительной сложности в коде, особенно если вы неопытны в использовании паттернов проектирования.

2.Дополнительный код: Bridge требует написания дополнительного кода, чтобы разделить абстракцию и реализацию, что может увеличить объем кода.

3.Усложнение отладки: Если не правильно реализовать паттерн, то может быть трудно понять, где возникла проблема, что усложнит отладку.

4.Усложнение архитектуры: Использование Bridge может привести к усложнению архитектуры, особенно если проект не подразумевает использование этого паттерна.

Смоделируем пример применения паттерна Bridge.

Для этого в интегрированной среде разработки IntelliJ IDEA создадим новый проект. В нём создадим абстрактный класс Shape, который обобщённо описывает машину. Код представлен в листинге 1.

Листинг 1 – Класс Shape

```
package org.example;
```

Листинг 1 – Класс Shape(продолжение)

```
public abstract class Shape {
    public abstract void draw();
}
```

При создании новых фигур, классы этих фигур будут наследоваться от класса Shape. Пример создания класса Square приведён в листинге 2.

Листинг 2 – Класс Square

```
package org.example;
```

```
public class Square extends Shape{
    @Override
    public void draw() {
        System.out.println("Drawing Square");
    }
}
```

Однако при добавлении понятия «цвет», от которого будет зависеть функционал метода draw(), для каждой фигуры каждого цвета будет требоваться создать новый класс. Например: Если у есть 2 класса определяющие фигуры (Square и Triangle), и вводятся три цвета (Black, Red, Blue), то понадобится создать 6 новых классов: TriangleBlack, TriangleRed, TriangleBlue, SquareBlack, SquareRed и SquareBlue.

Соответственно количество классов при создании нового цвета или фигуры растёт в геометрической прогрессии. Паттерн Bridge диктует, как вносить изменения в функционал классов одной ветки, не ломая логику другой.

В случае вышеописанного примера реализация паттерна Bridge заключается в выводе цвета в отдельный интерфейс. Листинги интерфейса Color и его трёх имплементаций показаны в листингах 3-6.

Листинг 3 – Интерфейс Color

```
package org.example;

public interface Color {
    void fillColor();}
```

Листинг 4 – Класс BlackColor

```
package org.example;

public class BlackColor implements Color{
    @Override
    public void fillColor() {
        System.out.println("Filling in black color");
    }
}
```

Листинг 5 – Класс RedColor

```
package org.example;
```

```
public class RedColor implements Color{
    @Override
    public void fillColor() {
        System.out.println("Filling in red color");
    }
}
```

Листинг 6 – Класс BLueColor

```
package org.example;

public class BlueColor implements Color{
    @Override
    public void fillColor() {
        System.out.println("Filing in blue color");
    }
}
```

Остаётся только добавить поле типа Color в класс Shape, задать получение его значения в конструкторе и задать фигурам использование функционала интерфейса Color. Доработанная версия класса Shape и Square продемонстрированы в листингах 7-8.

Листинг 7 – Класс Shape

```
package org.example;


public abstract class Shape {
    protected Color color;

    public Shape(Color color) {
        this.color = color;
    }

    public abstract void draw();
}
```

Листинг 8 – Класс Square

```
package org.example;

public class Square extends Shape{
    public Square(Color color) {
        super(color);
    }
```

```
        @Override
        public void draw() {
            System.out.println("Drawing Square");
            color.fillColor();
        }
    }
```

На приведённом примере легко обозначить основные понятия паттерна Bridge:

Абстракция (Abstraction) - определяет интерфейс высокоуровневых операций, которые должны быть выполнены клиентом - класс Shape.

Реализация (Implementation) - определяет интерфейс низкоуровневых операций, которые должны быть выполнены конкретной реализацией - интерфейс Color.

Реализация абстракции (RefinedAbstraction) - определяет интерфейс для всех классов реализации – класс Square.

Конкретная реализация (Concrete Implementation) - реализует интерфейс низкоуровневых операций и связывается с классом абстракции через интерфейс реализации абстракции – классы BlackColor. RedColor и BlueColor.

Таким образом на практическом примере было продемонстрирована польза эффективность использования Bridge. При разумном интегрировании данного паттерна в процесс разработки системы можно значительно упростить код, повысить его устойчивость, облегчить тестирование и поддержку кода.

Литература

1. Bridge Design Pattern in Java // DigitalOcean. URL: : https://www.digitalocean.com/community/tutorials/bridge-design-pattern-java (дата обращения 14.04.2023)
2. Design Patterns - Bridge Pattern // tutorialspoint URL: https://www.tutorialspoint.com/design_pattern/bridge_pattern.htm (дата обращения: 14.04.2023).
3. Miroslav Wengner Practical Design Patterns for Java Developers. – 1е изд. - packt, 2023. - 226 с.

4. The Bridge Pattern in Java // Baeldung(2022) URL: https://www.baeldung.com/java-bridge-pattern (дата обращения 13.04.2023)
5. Паттерн проектирования Мост (Bridge Pattern) // JavaRush URL: https://javarush.com/groups/posts/2570-znakomstvo-s-patternom-proektirovanija-bridge

References

1. Bridge Design Pattern in Java (2022), Available at : https://www.digitalocean.com/community/tutorials/bridge-design-pattern-java (accessed 14 April 2023).
2. Design Patterns - Bridge Pattern (2023) Available at : https://www.tutorialspoint.com/design_pattern/bridge_pattern.htm (accessed 14 April 2023).
3. Miroslav Wengner Practical Design Patterns for Java Developers. – Vol 1. - packt, 2023. 226 p.
4. The Bridge Pattern in Java (2022) Availzble at : https://www.baeldung.com/java-bridge-pattern (accessed 13 April 2023).
5. Bridge design pattern (2020) Available at : https://javarush.com/groups/posts/2570-znakomstvo-s-patternom-proektirovanija-bridge (accessed 13 April 2023).

Для цитирования: Михиенков К.С., Узоров К.А., Бузыкова Ю.С. ПАТТЕРН ПРОЕКТИРОВАНИЯ МОСТ В JAVA РАЗРАБОТКЕ // Международный журнал прикладных наук и технологий "Integral" №6/2023