К. А. Леонова
(ГГУ имени Ф. Скорины, Гомель)
Науч. рук. **О. М. Дерюжкова**, канд. физ.-мат. наук, доцент

ИСПОЛЬЗОВАНИЕ ШАБЛОНА DECORATOR ПРИ ПРОГРАММИРОВАНИИ НА JAVA

В программировании, особенно в языке Java, существует множество шаблонов проектирования, которые помогают разработчикам создавать эффективный и гибкий код. Шаблон Decorator является часто используемым шаблонам в структурном программировании и позволяет упростить сложные системы и создать модульный и легко читаемый код.

Проблема: Есть класс, который выполняет определенные операции, и нужно добавить к нему новую функциональность без изменения существующего кода. Классический подход – наследование от существующего класса и добавление новых методов и свойств. Однако этот подход имеет некоторые недостатки. Во-первых, наследование создает жесткую связь между классами, что может стать проблемой при изменении функциональности. Во-вторых, наследование позволяет только добавлять функциональность, но не удалять или изменять существующую.

Решение: Шаблон Decorator предлагает альтернативный подход к добавлению функциональности объектам. Вместо наследования от существующего класса создается новый класс-декоратор, который оборачивает исходный объект и добавляет к нему новую функциональность. Декоратор имеет тот же интерфейс, что и исходный объект, поэтому клиентский код может использовать и декорированный объект, и исходный объект без изменений.

Пример: Для лучшего понимания шаблона Decorator рассмотрим пример с классом Shape, который представляет геометрическую фигуру, и двумя декораторами – ColorDecorator и BorderDecorator (рисунок 1).

```
interface Shape {
void draw();
}
class Circle implements Shape {
      @Override
public void draw() {
System.out.println("Drawing a circle");
      }
 }
classColorDecorator implements Shape {
private Shape shape;
private String color;
publicColorDecorator(Shape shape, String color) {
this.shape = shape;
this.color = color;
      }
     @Override
public void draw() {
shape.draw();
System.out.println("Coloring the shape with " + color);
      }
 }
classBorderDecorator implements Shape {
private Shape shape;
privateintborderWidth;
publicBorderDecorator(Shape shape, intborderWidth) {
this.shape = shape;
this.borderWidth = borderWidth;
      }
     @Override
public void draw() {
shape.draw();
System.out.println("Drawing border with width " +
borderWidth);
      }
 }
   Shape circle = new Circle();
   circle.draw(); // Output: Drawing a circle
   Shape coloredCircle = new ColorDecorator(circle, "red");
   coloredCircle.draw(); // Output: Drawing a circle\n
Coloring the shape with red
   Shape borderedColoredCircle = new
BorderDecorator(coloredCircle, 2);
   borderedColoredCircle.draw(); // Output: Drawing a circle\n
Coloring the shape with red\n Drawing border with width 2
```

Рисунок 1 – Класс Shape

В данном примере класс Circle представляет собой простую геометрическую фигуру – круг. Декоратор ColorDecorator добавляет кругу возможность изменять его цвет, а декоратор BorderDecorator – возможность рисовать границу с заданной шириной.

В данном примере мы создаем объект Circle, а затем оборачиваем его в декораторы ColorDecorator и BorderDecorator, добавляя к нему новую функциональность. Каждый декоратор передает вызов метода draw() исходному объекту и добавляет свою функциональность.

Шаблон Decorator полезен, когда необходимо добавить новую функциональность к объектам, не изменяя существующий код. Он позволяет создавать модульный и гибкий код, в котором можно комбинировать различные декораторы для достижения нужного результата. Шаблон Decorator также позволяет избежать проблем, связанных с наследованием, такими как жесткая связь между классами и невозможность удаления или изменения существующей функциональности.

Используя шаблон Decorator можно создавать более читабельный, модульный и гибкий код, что в конечном итоге дает возможность разрабатывать более масштабируемые и поддерживаемые приложения.

Литература

1. Декоратор. Суть паттерна [электронный ресурс]. – 2023. – URL: https://refactoring.guru/design-patterns/decorator. – Дата доступа: 08.02.2024.

2. Фасад. Суть паттерна [электронный ресурс]. – 2023. – URL:https://refactoring.guru/design-patterns/facade.. – Дата доступа: 15.02.2024.

Н. В. Лукашевич
(ГГУ имени Ф. Скорины, Гомель)
Науч. рук. **Г. Ю. Тюменков**, канд. физ.-мат. наук, доцент

ПЕРСПЕКТИВЫ РАЗВИТИЯ ВЕБ-ФОРМ

Веб-формы – это электронные формы, которые размещаются на веб-сайтах и заполняются с помощью браузера. Веб-формы могут быть использованы для различных целей, таких как анкетирование, опросы, заявления, заказы, договоры, отчеты и т. д. Они имеют ряд преимуществ перед бумажными документами, таких как экономия времени и ресурсов, уменьшение ошибок и потерь данных, повышение эффективности и качества обработки информации, удобство хранения и доступа к данным, возможность автоматизации и интеграции с другими системами и сервисами.

Однако, веб-формы также сталкиваются с рядом проблем и вызовов, связанных с их созданием, использованием и защитой. Некоторые из этих проблем и вызовов включают в себя: сложность и неоднозначность правового регулирования электронных документов, необходимость обеспечения подлинности, целостности и конфиденциальности данных, риск нарушения персональных данных и интеллектуальной собственности, недостаток универсальных стандартов и форматов для электронных форм, разнообразие и несовместимость технических средств и платформ для работы с электронными формами, низкий уровень информационной грамотности и доверия у пользователей и т. д.

Целью данной статьи является анализ современного состояния и перспектив развития веб-форм как важного элемента цифровой трансформации общества. Для достижения этой цели рассмотрим следующие вопросы: определение и классификация веб-форм, основные виды и области применения веб-форм, преимущества и недостатки веб-форм по сравнению с бумажными документами, проблемы, связанные с веб-формами, а также