

DECORATOR DESIGN PATTERN

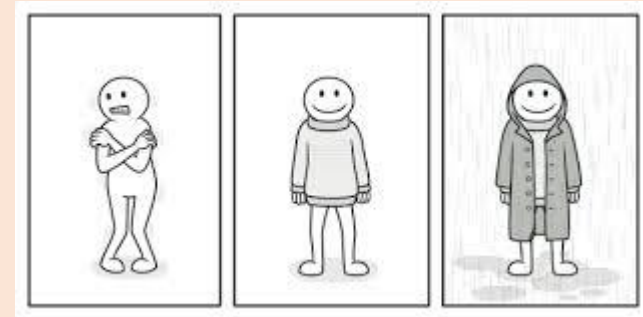
<<Elevate Your Objects: The Decorator Pattern – Adding Features with
Simplicity.>>

Purpose and Motivation Of The Decorator Pattern:

The ***Decorator*** Pattern is designed to extend or augment the behavior of objects in a flexible and reusable way. It allows you to add new functionalities to an object dynamically without altering its structure.

This pattern promotes the principle of open-closed design, enabling you to introduce new features without modifying existing code.

How can you add new functionality to an object dynamically without altering its structure?



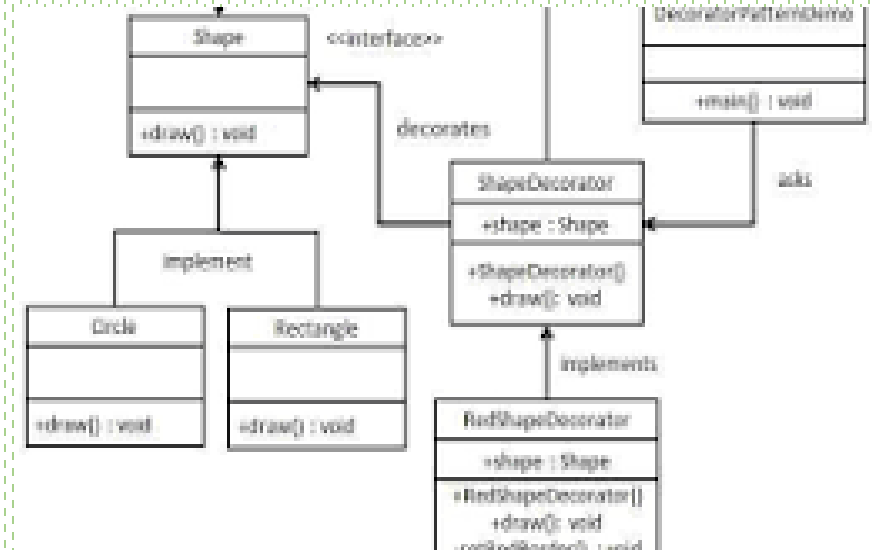
Purpose and Motivation



Decorators provide a flexible alternative to subclassing for extending functionality. You can attach new behaviors to objects at runtime, adapting to changing requirements without modifying existing code.

Purpose and Motivation

Composition over Inheritance: Instead of relying on a complex hierarchy of subclasses, the Decorator Pattern favors composition. By composing objects with different decorators, you can achieve various combinations of functionalities without creating a multitude of subclasses.



Vocabulary

1. Component:

- **Definition:** The common interface or abstract class that defines the base functionality for both concrete components and decorators in the Decorator Pattern.

2. Decorator:

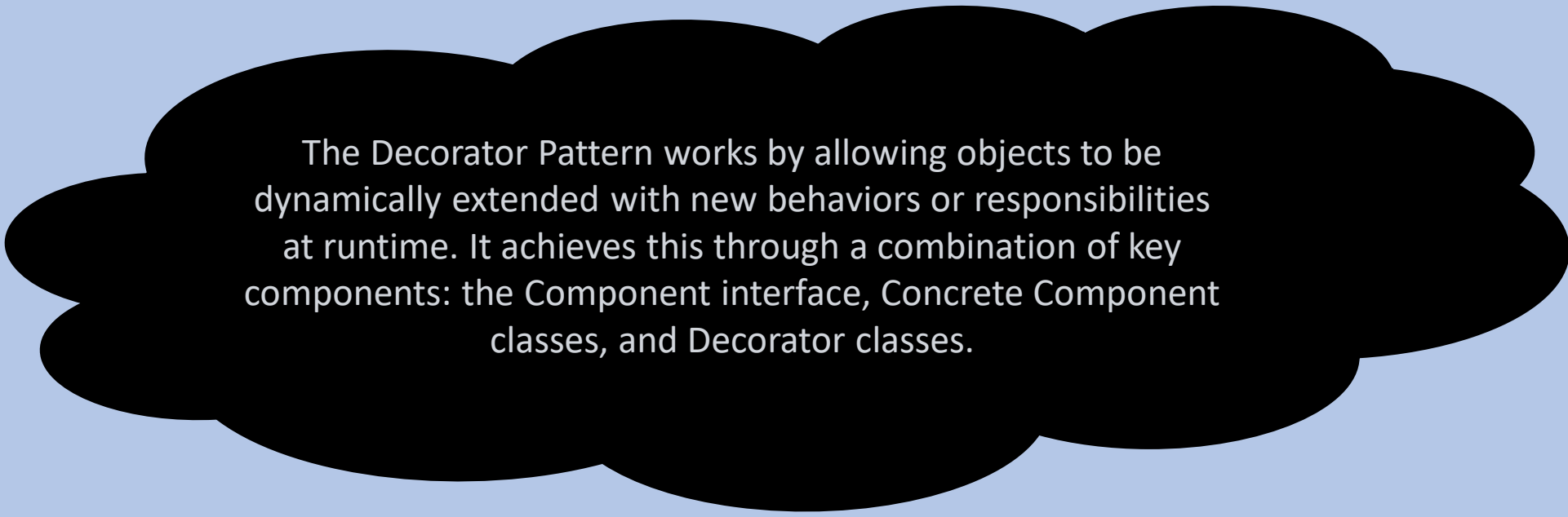
- **Definition :**A structural design pattern element that enhances or extends the behavior of a Component by wrapping itself around it and maintaining a reference to the Component.

3. Concrete Component:

- **Definition:** A class that implements the Component interface, representing the base object to which decorators can be added in the Decorator Pattern.

How does it works?

A Decorator Pattern says that just "attach a flexible additional responsibilities to an object dynamically". In other words, The Decorator Pattern uses composition instead of inheritance to extend the functionality of an object at runtime. The Decorator Pattern is also known as Wrapper.



The Decorator Pattern works by allowing objects to be dynamically extended with new behaviors or responsibilities at runtime. It achieves this through a combination of key components: the Component interface, Concrete Component classes, and Decorator classes.

Advantages

- **Flexibility and Dynamic Behavior:** Enables the dynamic addition of new behaviors or responsibilities to objects at runtime without altering their structure, providing a flexible and adaptable solution.

- **Open-Closed Principle:**

Supports the open-closed principle by allowing the introduction of new decorators without modifying existing code. This promotes code extensibility and minimizes the need for class hierarchy changes.

What can be the advantages?





Did you know?



shutterstock.com - 2261016619

Disadvantages

- **Increased Number of Classes:**

The pattern can lead to an increase in the number of classes in the codebase. While this is mitigated by the use of small, focused classes, it can still impact the overall readability and maintainability.

- **Design Overhead:** The need to create separate classes for each decorator can result in additional design overhead, especially for simple or one-time use cases.

Thank You!!

Thank you all for joining today's presentation. Just as a sandwich requires the right combination of ingredients for a perfect taste, your engagement and participation have enriched our discussion on the MVC design pattern. Your presence has added flavor to this session, and we appreciate your time and attention. If you have any further questions or insights, feel free to reach out. Here's to building well-structured and delightful software together.



THE END