# Decorator Design Pattern

Decorator design pattern is a structural design pattern that is used to modify the behavior of individual objects at run time, for a specific instance, so other instances will not be modified or may have different modifications, meaning individual instances are independent form each other in the modification part. We use [inheritance](https://www.digitalocean.com/community/tutorials/inheritance-java-example) to extend the behavior of an object.

**When to Use the Decorator Pattern?**

1. **Adding New Features:**You want to add extra features to objects without changing their core structure. It’s like putting toppings on a pizza without changing the pizza itself.
2. **Avoiding Messy Code:** You want to avoid having too many different classes for all possible combinations of features. Instead, you can mix, and match decorators as needed.
3. **Open for Extension, Closed for Modification:**You want to make your code ready for future changes by allowing new features to be added without messing up existing code. This aligns with the Open/Closed Principle.

**Common Things in all implementations of the decorator design pattern**

1. Component Interface (defines the common methods for both concrete components and decorators)
2. Concrete Class (it's the base implementation that decorators will modify)
3. Decorator itself (it's an abstract class that implements the component interface)
4. Other classes that interact with the decorator (these are concrete decorators that add specific behaviors)