Structural Design pattern : Decorator Design pattern

**Definition:**

The decorator pattern attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

In an object-oriented paradigm, decorators are one of the largely used [structural design patterns](https://refactoring.guru/design-patterns/structural-patterns). This pattern changes the functionality of objects at runtime without impacting its blueprint or the existing functionality.

While working with these patterns, we might confuse decorators with an inheritance. However, there is a difference between the two. Decorators are used when we have to add and remove responsibilities from an object dynamically. Whereas, inheritance can do the same, but not at run time.

**Decorator patterns simply allow you to dynamically add functionality to a specific object instead of an entire class of objects.**

How can responsibilities be added to an object dynamically?

How can the functionality of an object be extended at run-time?

References :

[**https://dzone.com/articles/is-inheritance-dead**](https://dzone.com/articles/is-inheritance-dead) **(detailed inheritance and decorator (i.e. composition or aggregation ))**

<https://www.youtube.com/watch?v=XihyLU6MhzY> ( good one)

<https://dzone.com/articles/decorator-design-patterns>

<https://www.geeksforgeeks.org/the-decorator-pattern-set-2-introduction-and-design/>

<https://www.journaldev.com/1540/decorator-design-pattern-in-java-example>