Decorator Pattern

> Attach additional responsibility to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

- The solution to the Decorator pattern mainly relies upon association rather than inheritance.
- Decorator leans toward association to enhance the capabilities or responsibilities of an object.
- The relationship used in the Decorator pattern is recursive.

When to use Decorator Pattern?

- To add responsibilities to individual objects dynamically and transparently, without affecting other objects.
- Responsibilities may be withdrawn.
- Extension by subclassing is impractical.

Consequences of using Decorator Pattern

- Flexibility compared to static inheritance.
- Functionality may be added/removed at runtime.
- You only get features you ask for.
- Decorator acts as transparent enclosure.
- Demerits →
  - Can’t rely on Object identity.
  - Lots of little objects.

Decorator vs. Other Patterns

- Adapter changes objects interface. Decorator changes only its responsibilities.
- Decorator is not intended for object aggregation like Composite.
- Strategy and Decorator are used to change an object — Strategy lets you change the guts of an object, Decorator its skin.