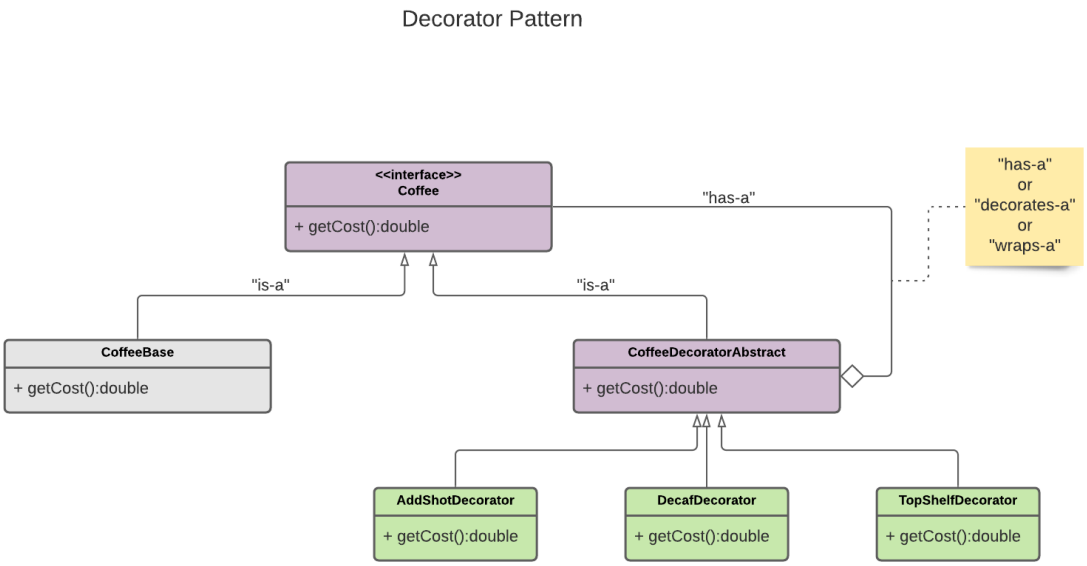
Decorator Pattern

Design Pattern Details

Title	Decorator Pattern
Description	The decorator pattern is a structural design pattern that augments a class or decorates a class with additional behavior. In instances where the interface for this class is predefined, this comes in the form of modified behavior based on the behavior of what it decorates. This might simply transform a value or it might additionally have “side-effects” such as decorating a method with logging behavior.
Anatomy diagram	
Key to anatomy diagram	1. Decorators have both the “is-a” relationship and the “has-a” relationship 2. There is a base concretion that is by itself not decorating anything (like the base case in recursion) 3. Decorator concretions adhere to the same interface as non-decorator concretions

Specifications

Usage	**As a structural design pattern, decorator provides a blueprint for implementing custom behavioral permutations that vary at run time and avoids exponential class explosion. The structure naturally lends itself to mix-ins, where various decorations are mixed in based on a variety of factors. Due to the overhead associated with use, this is better suited for decorating system constructs rather than collections of data or entities.**
Variations	The structure of the Decorator pattern is very similiar to other Design patterns - such as Composite - where both form a tree like structure. An important distinction is the multiplicities - where a decorator decorates one thing, where as Composite has one-to-many things. For more infomation about these differences, please check out the links below. Decorator Pattern: http://www.blackwasp.co.uk/Decorator.aspx Christopher Okravi: https://youtu.be/EECfgFQ44Kg SER316 Design Patterns (time = 41:29): https://youtu.be/NPk95pYbc_s?t=2489
Behavior	Decorators can and should by their nature be treated as if they were what they are decorating. Importantly, what they are decorating is not being changed. Multiple decorators could conceivably decorate the same thing, each returning their respective results without modifying the state of the shared decorated component. While a decorator can have side effects, those side effects should not alter the state of what they are decorating.