Modular design made easy

using decoration and composition

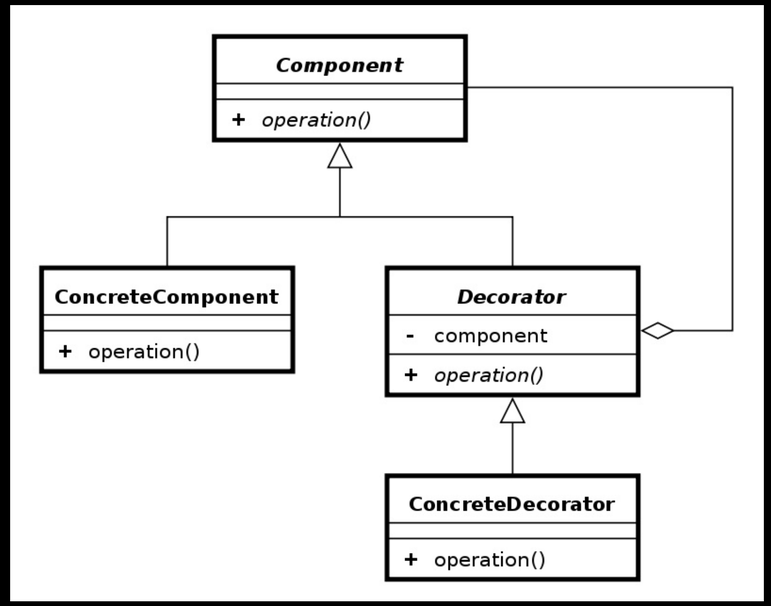
Anderson Costa, Feb 10th 2022

General guidelines

- Program to an interface, not an implementation
  - Clients remain unaware of the specific types of the objects they use
  - Clients remain unaware of the classes that conform to these objects
  - It is a good way to achieve **dynamic binding** and **polymorphism**
- Favor **object composition** over **class inheritance**
  - With inheritance, objects can see the specifications of parent objects
  - With composition, objects are only aware of what is in the interface

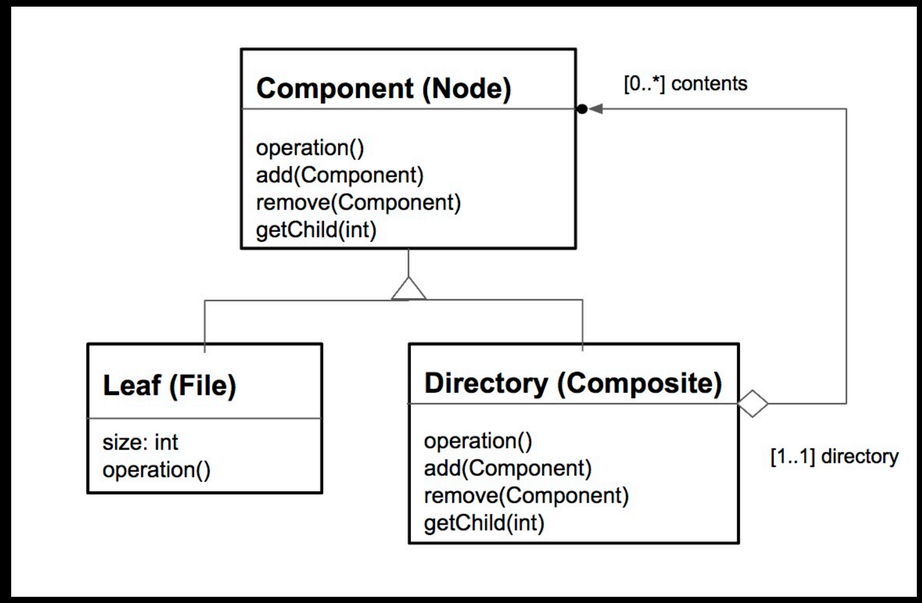
Decorator design pattern

- Allows behaviour to be added to an individual object, dynamically, without affecting the behaviour of other objects from the same class
- Is often useful for adhering to SRP, as functionality can be divided between class with unique areas of concern



Composite design pattern

- Describes a group of objects that are treated the same way as a single instance of the same type of object
- Implementing the composite pattern lets clients treat individual objects and compositions uniformly



Case study

Write code to fetch user's orders

- Fetch orders from a remote endpoint
- Cache successful responses
- Read from cache first, fallback to remote
- Read from remote first, fallback to cache
- Throttle network request to have at least 1 minute between requests
  (This was left as an exercise for the reader)