Decorator: Adding behavior without altering the class itself.

1. Want to augment an object with additional functionality
2. Do not want to rewrite or alter existing code (OCP)
3. Want to keep new functionality separate (SRP)
4. Need to be able to interact with existing structures
5. Two options:
   1. Inherit from required objects if possible; some classes are final. (You cannot extend final classes.)
   2. Build a Decorator, which simply references the decorated objects
6. Decorator: Facilitates the addition of behaviors to individual objects without inheriting from them.

String Decorator

1. You take the class in the constructor and delegate the function of the class.

Dynamic Decorator Composition

1. I have a shape interface and 2 shapes that implement it. I want to also add additional information to the shapes like color and transparency.
2. So I create two new classes for color and transparency that implement shape as well and take a shape in the constructor.
3. With this way I wont have access to underlying methods like methods specific to a circle or square.

Static Decorator Composition

1. The goal is the same with dynamic but the only difference is in this one you can determine the types in compile time and in the other one you determine in runtime.
2. Determine the type int he class declaration. → example code

Adapter Decorator

1. Your decorator doesn't have to be just a decorator. It can also be an adapter.
2. Example written code.

Summary

1. A decorator keeps the reference to the decorated objects
2. May or may not forward calls
   1. IDE can Generate Delegated Members
3. Exists in a static variation
   1. Unpleasant constructor argos
   2. Very limited due to
      1. Type erasure
      2. Inability to inherit from type parameters