Decorator Design Pattern

David Nielsen

Decorator

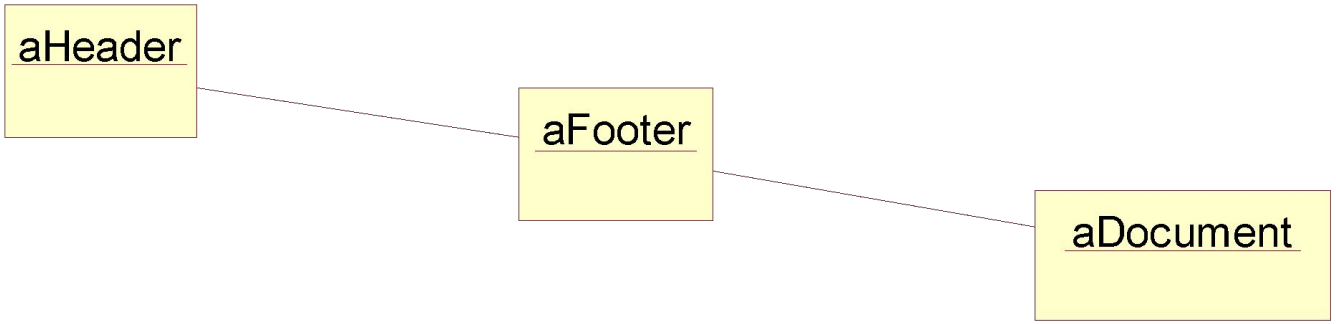
- Object structural pattern
  - Concerned with object composition
- Also known as Wrapper

Motivation

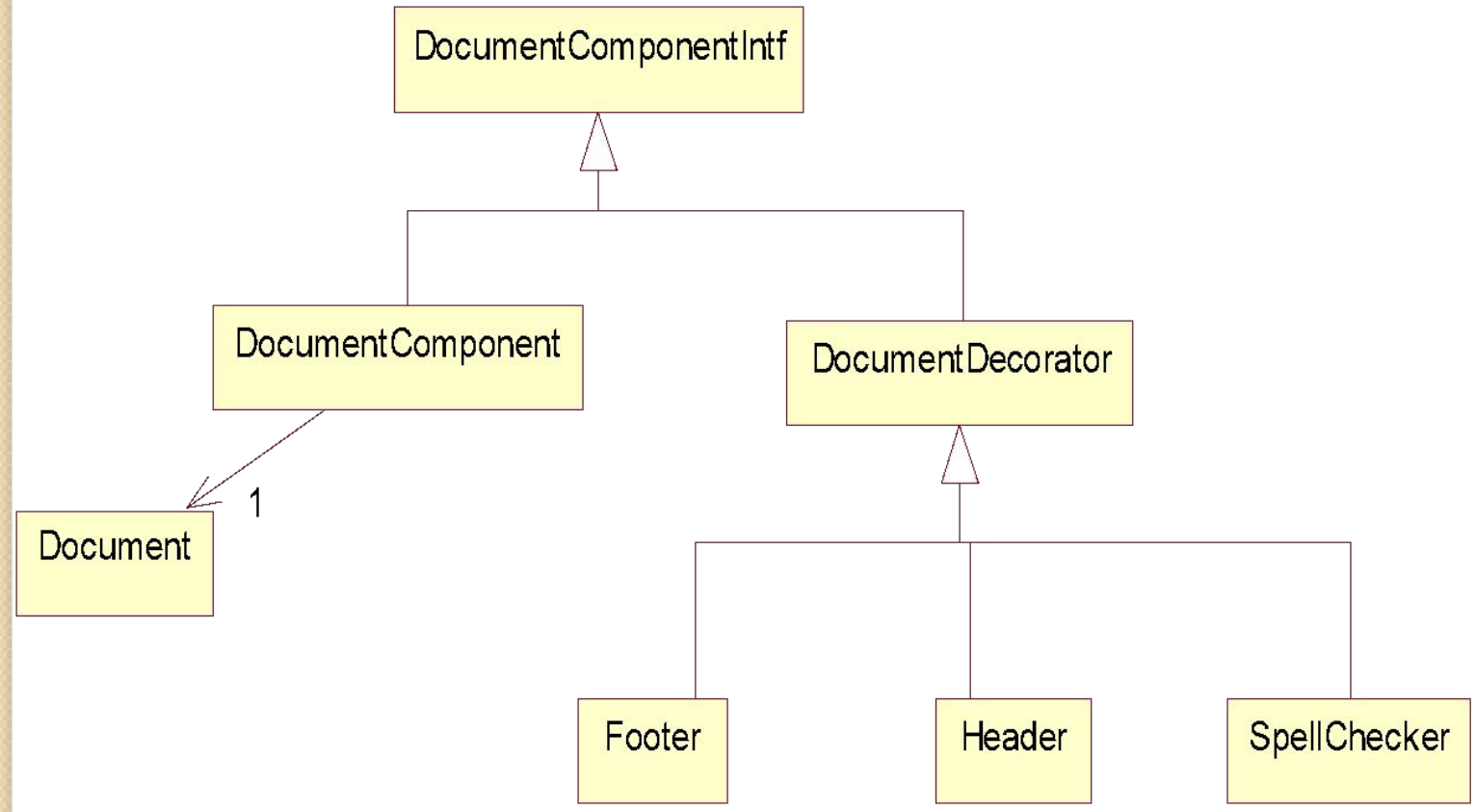
- Add responsibilities to an object
- Can add responsibilities via inheritance
  - Choice made at compile time
- Decorator supports
  - Capabilities to be added dynamically
  - No changes to existing class hierarchy

Composition

- Decorator chains objects together
  - Each object in the chain adds a capability
  - The last object in the chain is the original object



Example: class diagram



Participants

- Component
  - Defines the interface for objects that can have responsibilities added dynamically
- Concrete Component
  - Defines an object to which additional responsibilities can be attached
- Decorator
  - Maintains a reference to a Component object and defines an interface that conforms to the component's interface

Consequences

- More flexibility than static inheritance
- Avoids feature-laden classes high up in the hierarchy
- A decorator and its component are not identical
- Lots of little objects

Implementation

- Interface conformance
- Omitting the abstract decorator class
  - Avoids needless complexity if you only need to add one responsibility
- Keep component classes lightweight
  - It should define an interface – not store data