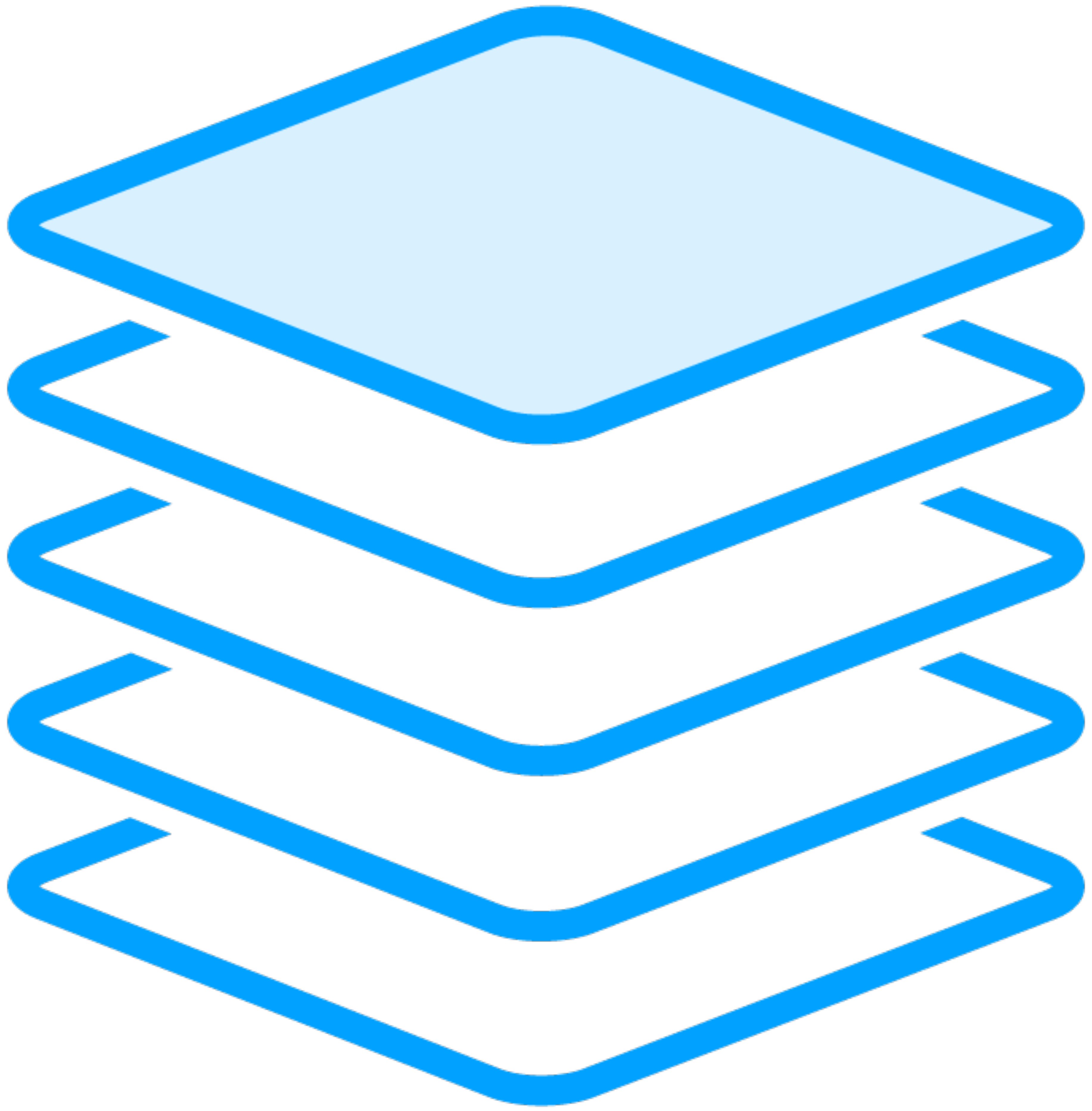
Decorator Design Pattern

Bryan Hansen

@bh5k

Decorator



Concepts

Also called a wrapper

Add behavior without affecting others

More than just inheritance

Single Responsibility Principle

Compose behavior dynamically

Examples:

java.io.InputStream

java.util.Collections#checkedList

Design

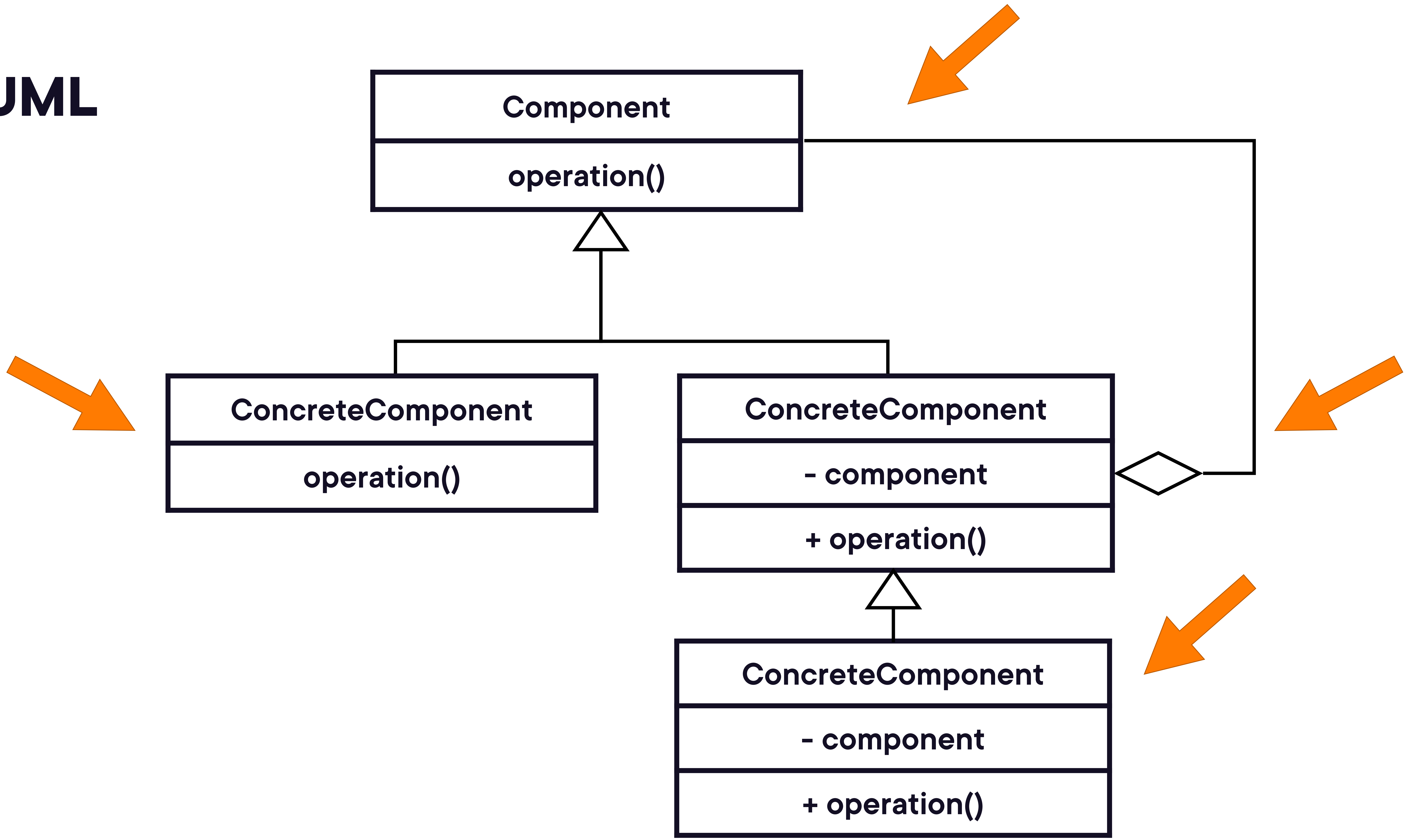
Inheritance based

Utilizes composition and inheritance (is-a, has-a)

Alternative to subclassing

Constructor requires instance from hierarchy

UML


Component
operation()
ConcreteComponent
operation()
ConcreteComponent
- component
+ operation()
ConcreteComponent
- component
+ operation()

Everyday Example - InputStream

```
File file = new File("./output.txt");
file.createNewFile();

OutputStream oStream = new FileOutputStream(file);

DataOutputStream doStream = new DataOutputStream(oStream);
doStream.writeChars("text");
```

Exercise Decorator

Component, ConcreteComponent, Decorator, ConcreteDecorator

Create Decorator

Implement another Decorator

Not a Creational Pattern

Pitfalls

New class for every feature added

Multiple little objects

Often confused with simple inheritance

Contrast

Decorator	VS	Composite
Contains another entity		Tree structure
Modifies behavior (adds)		Leaf and Composite same interface
Doesn’t change underlying object		Unity between objects

Decorator Summary

Original object can stay the same

Unique way to add functionality

Confused with inheritance

Can be more complex for clients