Software Design & Architecture
Design Patterns/
Structural Design Patterns

Pengyu Nie

Acknowledgements: slides adapted from previous versions by Mei Nagappan and Shane McIntosh, which are adapted from previous versions by Zhen Ming Jiang, Ahmed E. Hassan, Reid Holmes.

Design Patterns Categories

- Creational: concern the process of object creation
  - Singleton, Factory Method, Abstract Factory, *design patterns 1* Builder, Prototype, Object Pool
- Structural: concern the process of assembling objects and classes
  - Adapter, Composite, Decorator, *today* Façade, Bridge, Flyweight, Proxy
- Behavioral: concern the interaction between classes or objects
  - Observer, Strategy, Template Method, *design patterns 3* Iterator, State, Chain of Responsibility, Command, Mediator, Memento

design patterns 4 – your pick from the remaining ones

Adapter

Adapter: Motivation and Intent

- Motivation:
  - we need to interact with objects of a particular class that is incompatible with the current class defined in the client code
  - we do not want to (or more often, cannot) change the class to be used (e.g., they are defined in third-party libraries)
- Intent: convert the interface of a class into another interface

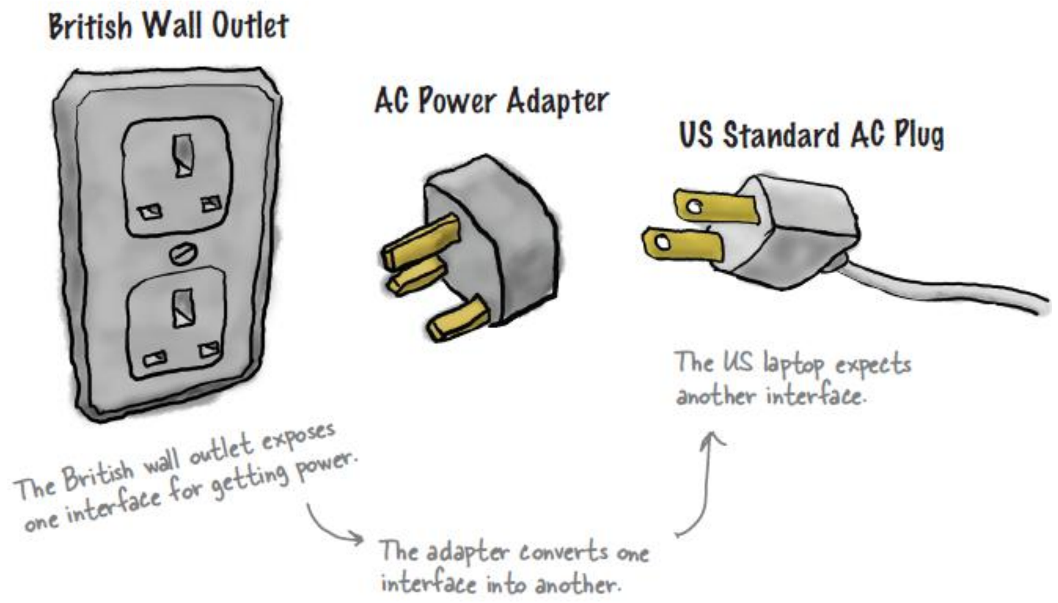
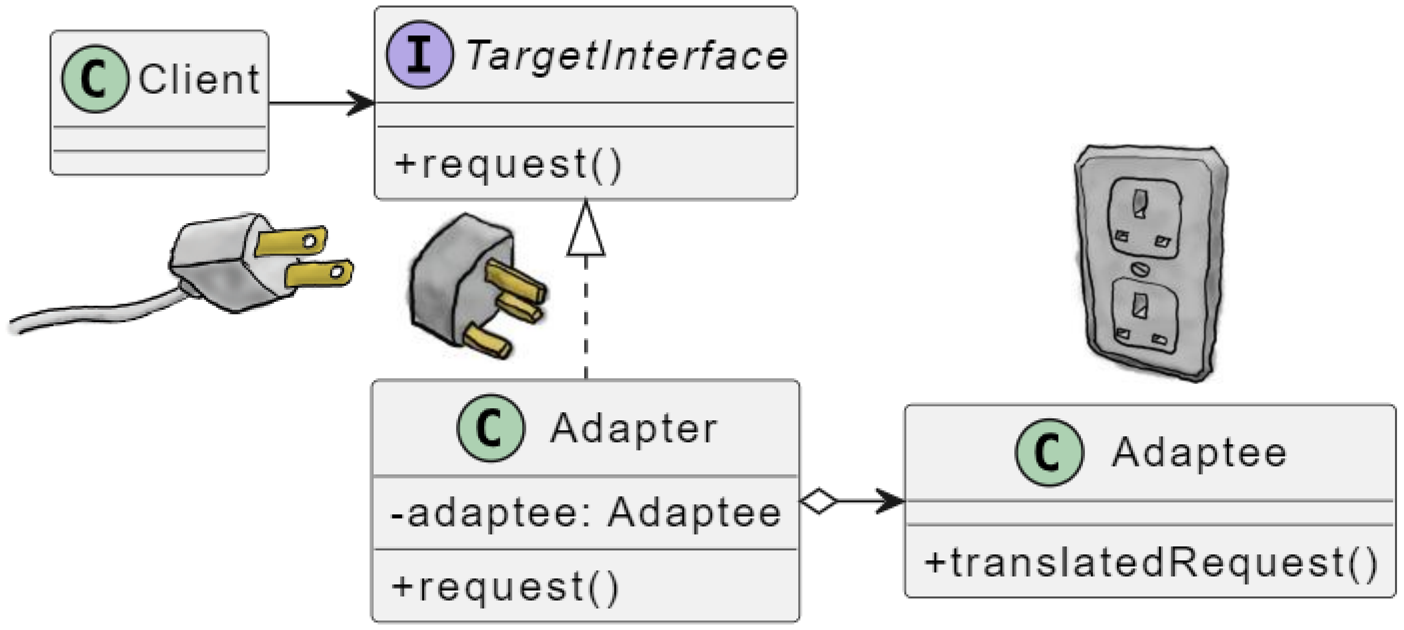

British Wall Outlet
AC Power Adapter
US Standard AC Plug
The US laptop expects another interface.
The British wall outlet exposes one interface for getting power.
The adapter converts one interface into another.

Image source: Eric Freeman and Elisabeth Robson. Head First Design Patterns.

Adapter: Solution



aggregation relationship

✓ Open-closed principle

Demo: https://github.com/pengyunie/CS446Demo1251/tree/main/app/src/main/java/ca/uwaterloo/cs446/dp/adapter

Composite

Composite: Motivation and Intent

- Motivation:
  - applications that have recursive groupings of primitives and containers
    (e.g., basic shapes (lines, circle, text) and compound shapes)
    (e.g., directories and files)
  - client treat containers and primitives in the same way
- Intent: compose objects into tree structures; define a shared interface

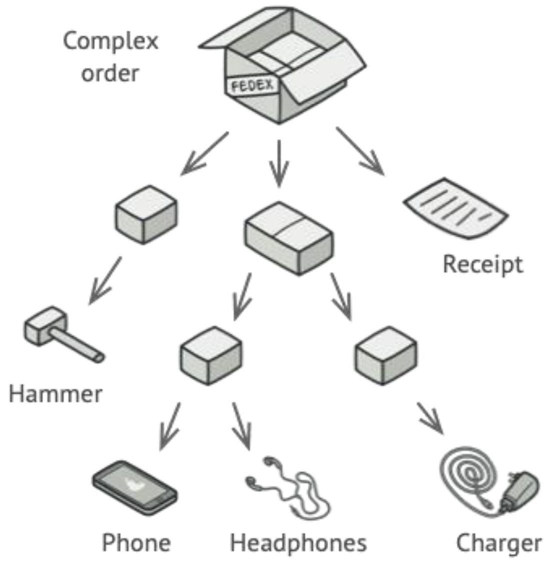
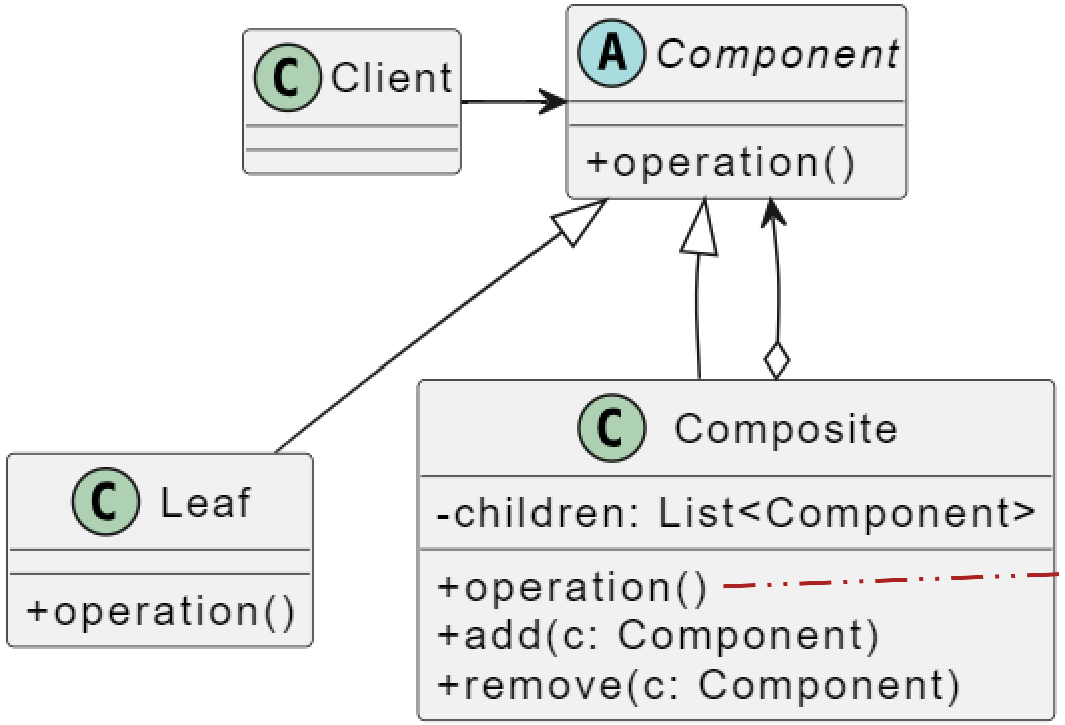


Image source: https://refactoring.guru/design-patterns/composite

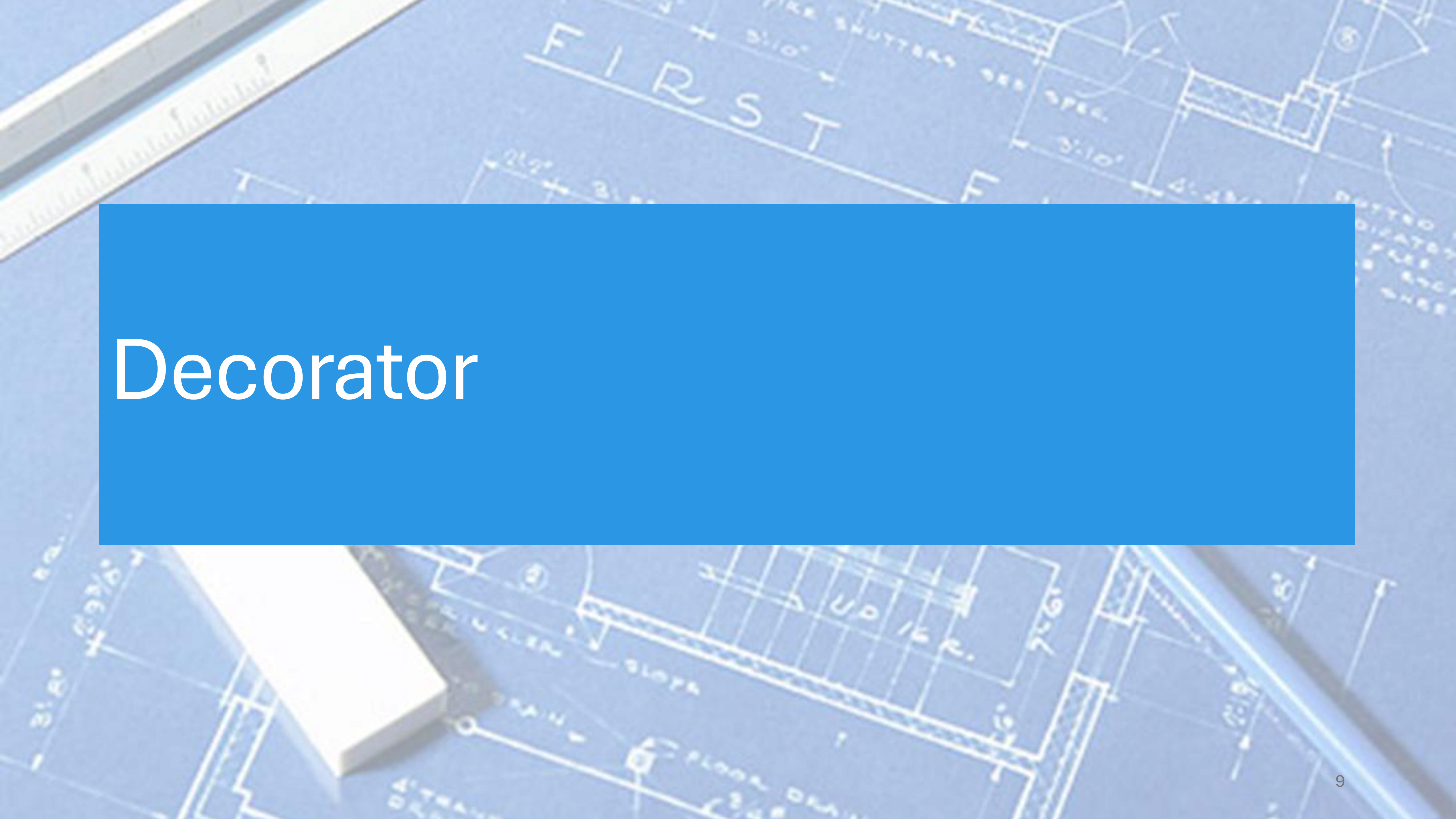
Composite: Solution



```
for (child in children) {
  child.operation()
}
```

✓ Liskov substitution principle

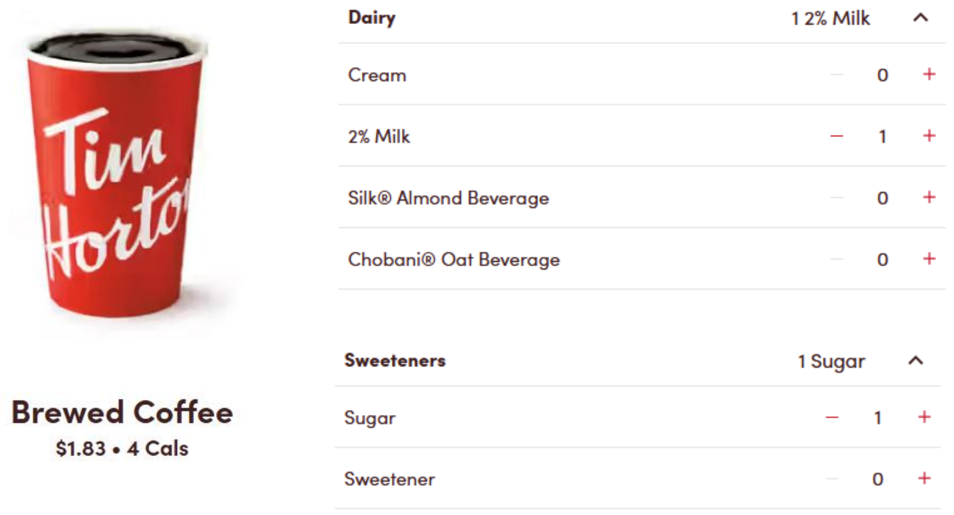
Demo: https://github.com/pengyunie/CS446Demo1251/tree/main/app/src/main/java/ca/uwaterloo/cs446/dp/composite



Decorator

Decorator: Motivation and Intent

- Motivation: extend an object's functionality dynamically at runtime (vs. inheritance that extend functionality statically at compile time)
- Intent: provide a flexible alternative to inheritance; add additional responsibilities dynamically to an object

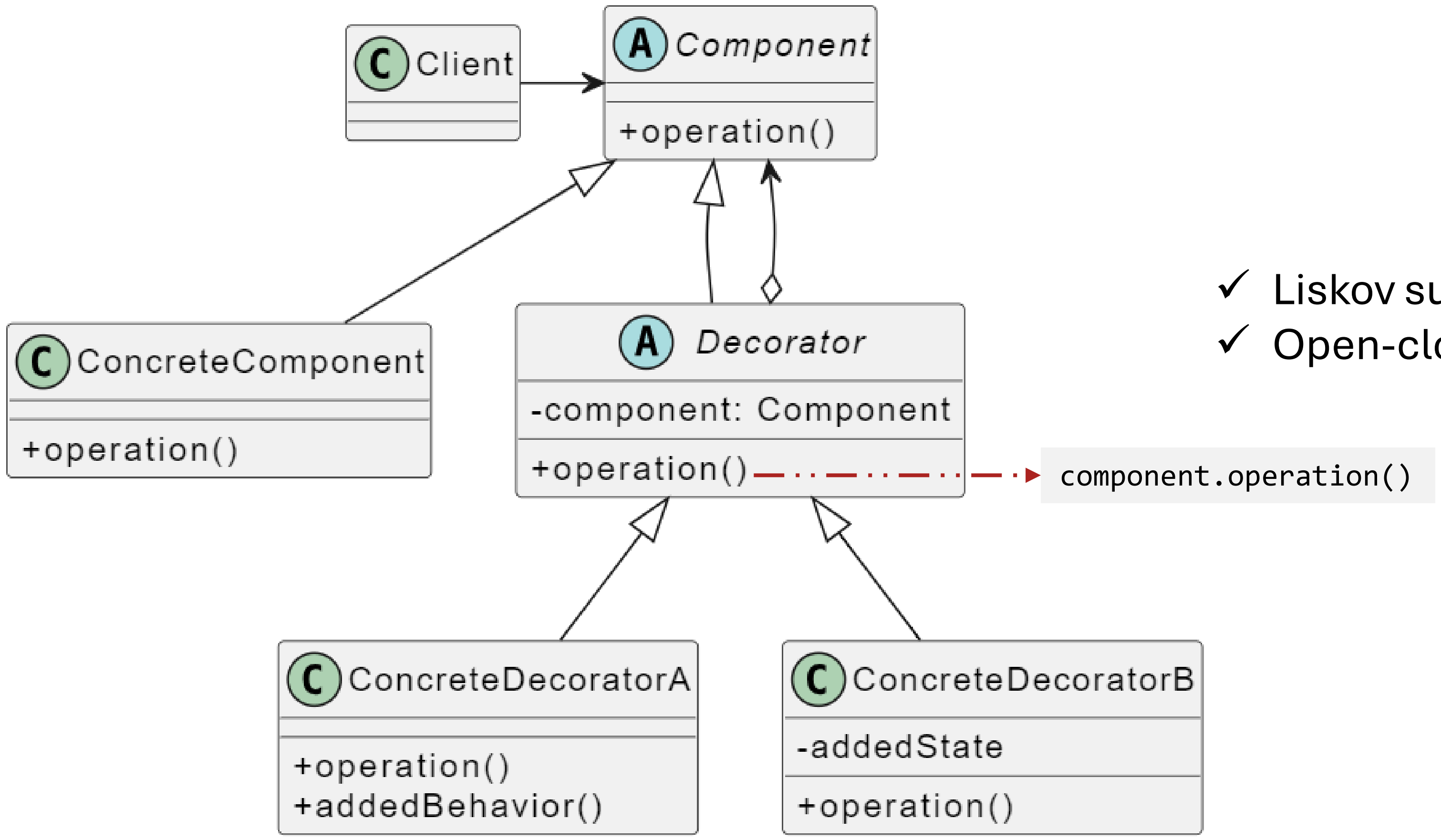


Brewed Coffee
$1.83 • 4 Cals

Dairy	1 2% Milk	^
Cream	0	− +
2% Milk	1	− +
Silk® Almond Beverage	0	− +
Chobani® Oat Beverage	0	− +
Sweeteners	1 Sugar	^
Sugar	1	− +
Sweetener	0	− +

Image source: https://www.timhortons.ca/menu

Decorator: Solution



- ✓ Liskov substitution principle
- ✓ Open-closed principle

Demo: https://github.com/pengyunie/CS446Demo1251/tree/main/app/src/main/java/ca/uwaterloo/cs446/dp/decorator

Agenda (recap)

- Structural design patterns
  - Adapter
  - Composite
  - Decorator