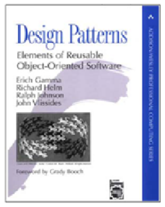


Design Pattern Definitions from the GoF Book

The Decorator Pattern

Attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

Creational Patterns

- The Factory Method Pattern
- The Abstract Factory Pattern
- The Singleton Pattern
- The Builder Pattern
- The Prototype Pattern

Structural Patterns

- The Decorator Pattern
- The Adapter Pattern
- The Facade Pattern
- The Composite Pattern
- The Proxy Pattern
- The Bridge Pattern
- The Flyweight Pattern

Behavioral Patterns

- The Strategy Pattern
  Defines a family of algorithms, encapsulates each one, and makes them interchangeable.
- The Observer Pattern
  Defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically.
- The Command Pattern
- The Template Method Pattern
- The Iterator Pattern
- The State Pattern
- The Chain of Responsibility Pattern
- The Interpreter Pattern
- The Mediator Pattern
- The Memento Pattern
- The Visitor Pattern

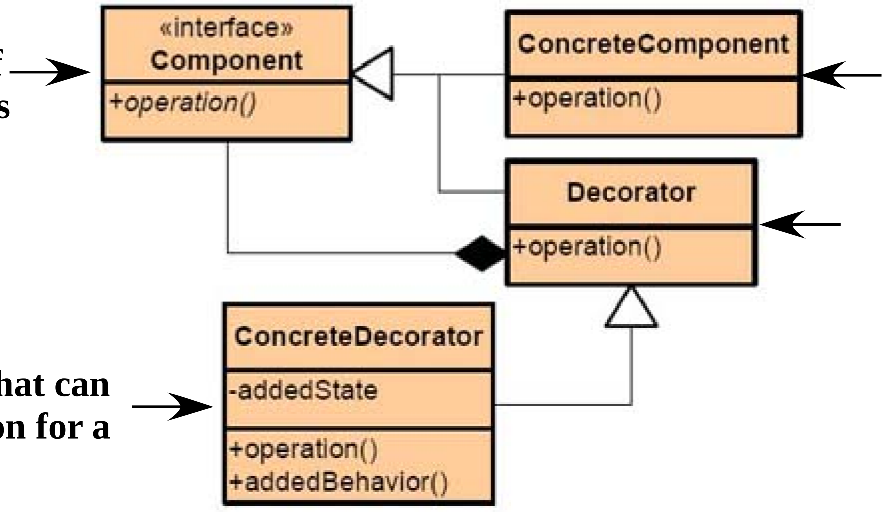
Design Patterns: Decorator

Quick Overview

Attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

Defines the interface of an object that performs certain actions.

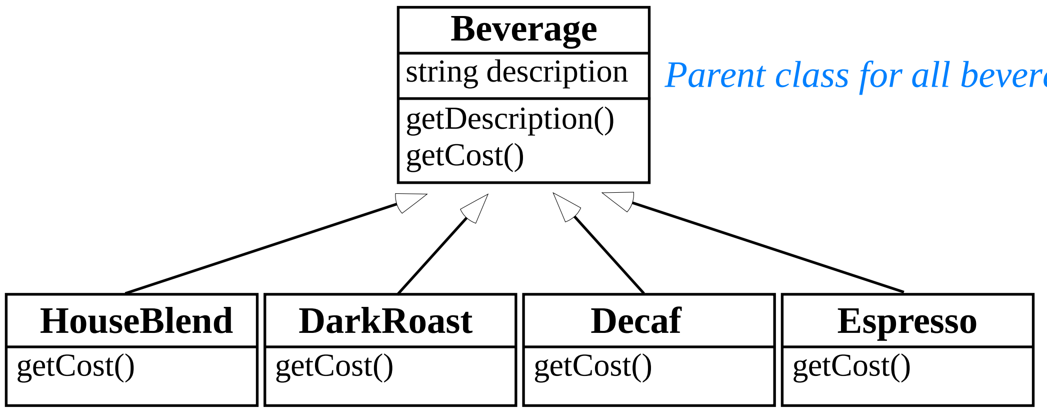
Defines an object that performs those actions.



Defines an object that can perform some action for a component.

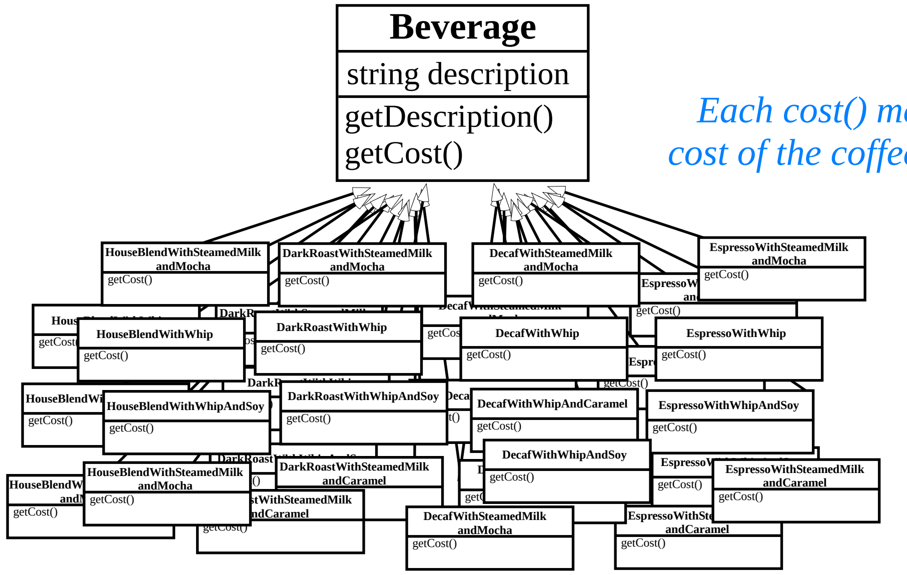
Design Patterns: Decorator

Welcome to Starbuzz Coffee



Parent class for all beverages

In the beginning it was simple. But, then they expanded and things changed...



Beverage
string description
getDescription()
getCost()
HouseBlendWithSteamedMilk andMocha
DarkRoastWithSteamedMilk andMocha
DecafWithSteamedMilk andMocha
EspressoWithSteamedMilk andMocha
HouseBlendWithWhip
DarkRoastWithWhip
DecafWithWhip
EspressoWithWhip
HouseBlendWithWhipAndSoy
DarkRoastWithWhipAndSoy
DecafWithWhipAndCaramel
EspressoWithWhipAndSoy
DecafWithWhipAndSoy
HouseBlendWithSteamedMilk andMocha
DarkRoastWithSteamedMilk andCaramel
EspressoWithSteamedMilk andCaramel
DecafWithSteamedMilk andMocha
getCost()

*Each cost() method calculates the cost of the coffee **and** the condiments.*

Design Patterns: Decorator

What happens when things change?

We add a new beverage, like tea

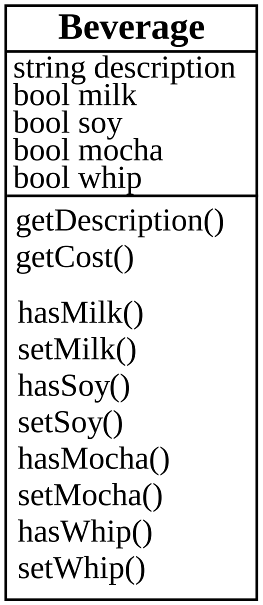
The price of milk goes up

*The customer wants a **double** mocha*

We add a new condiment

One Possible Solution

The parent class now handles the cost of the condiments and subclasses take care of their special type.



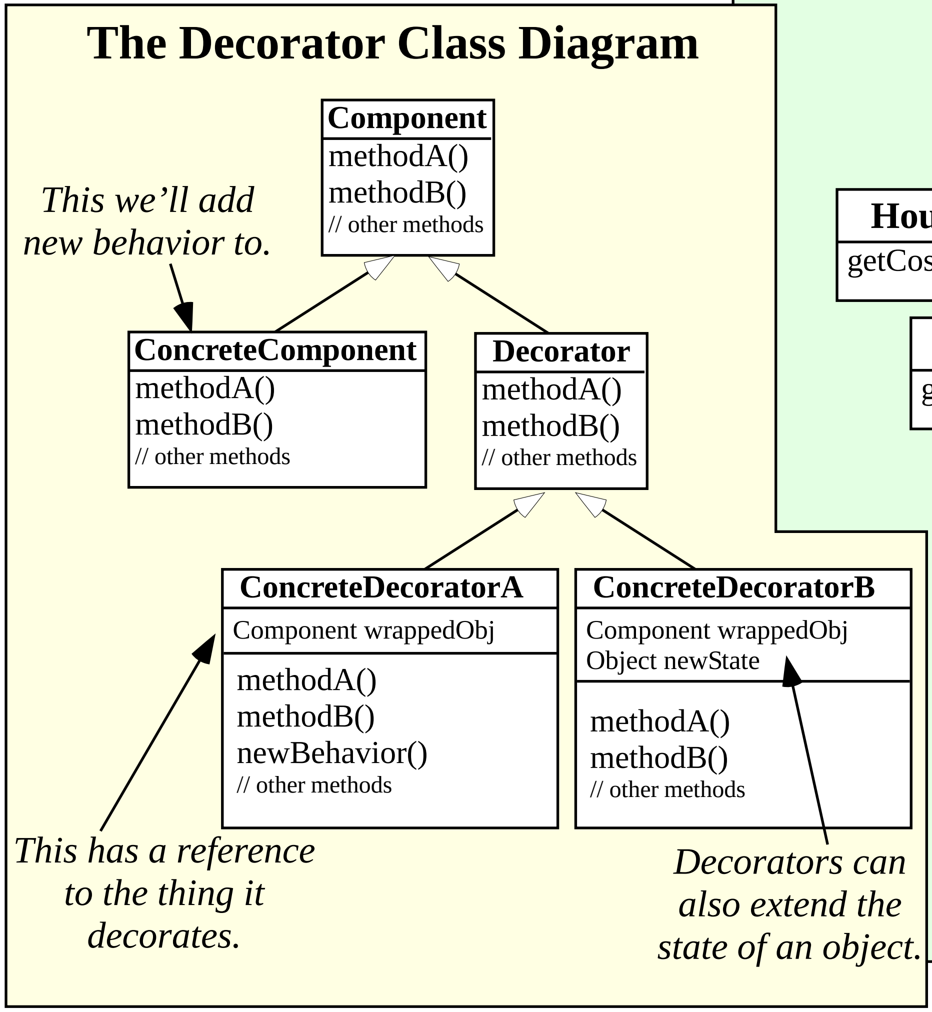
Beverage
string description bool milk bool soy bool mocha bool whip

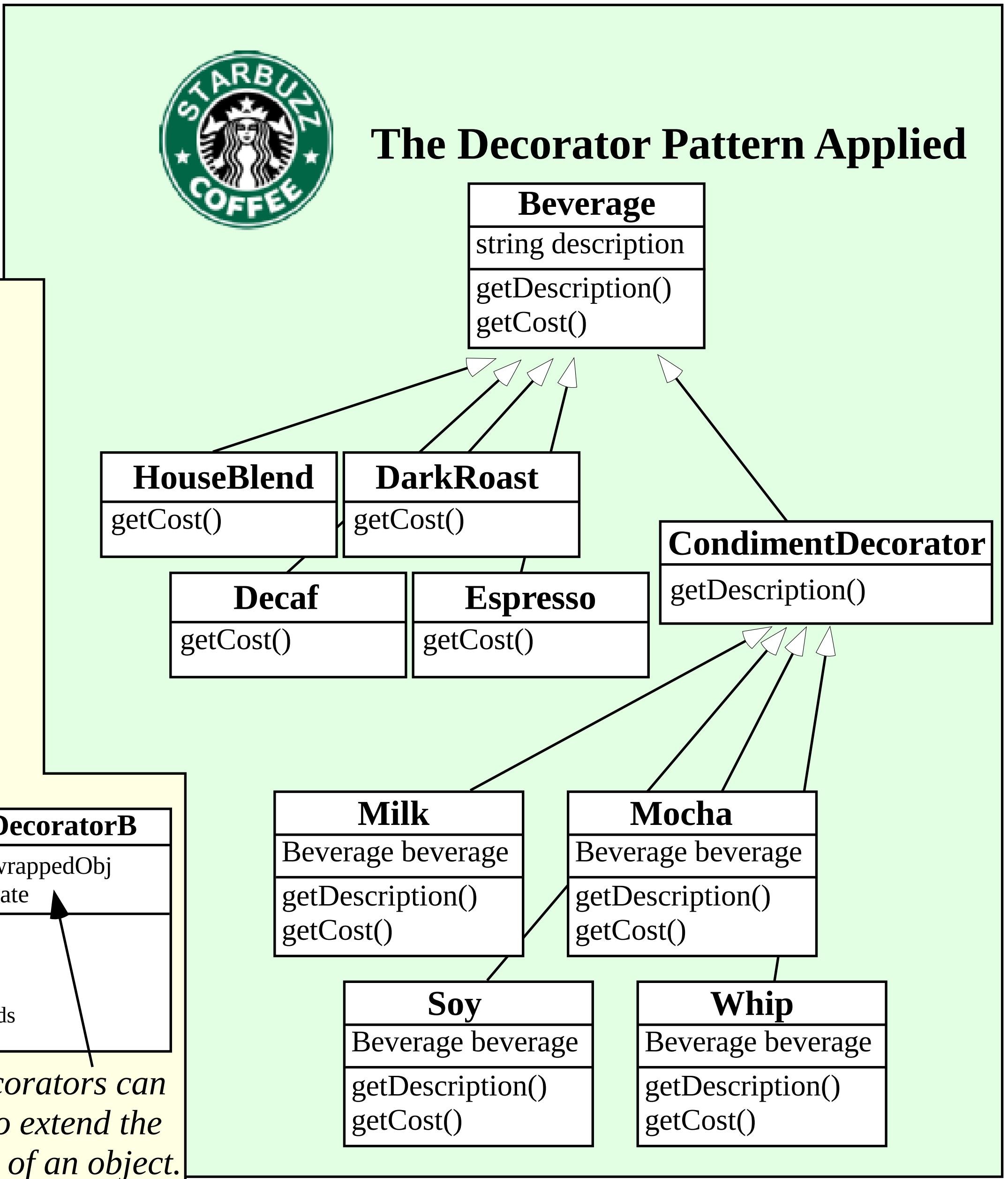
getDescription()
getCost()
 	
hasMilk()
setMilk()
hasSoy()
setSoy()
hasMocha()
setMocha()
hasWhip()
setWhip()

Design Principles

- ***Classes should be open for extension, but closed for modification.***

Design Patterns: Decorator



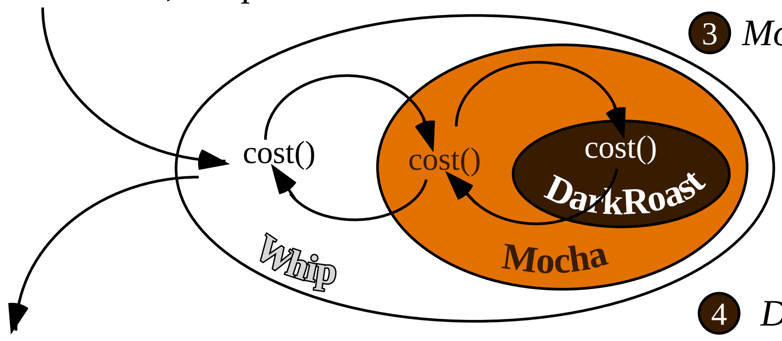


Design Patterns: Decorator

1 Call cost() on the outer most decorator, Whip

2 Whip calls cost() on Mocha

3 Mocha calls cost() on DarkRoast

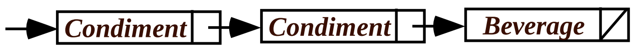


4 DarkRoast returns its' cost, $1.99 to Mocha*

5 Mocha adds its' cost, $.40 and returns the new total, $2.39* to Whip*

6 Whip adds its' cost, $.50 and returns the final total, $2.89**

Sure looks a lot like a linked list, or a stack doesn't it?

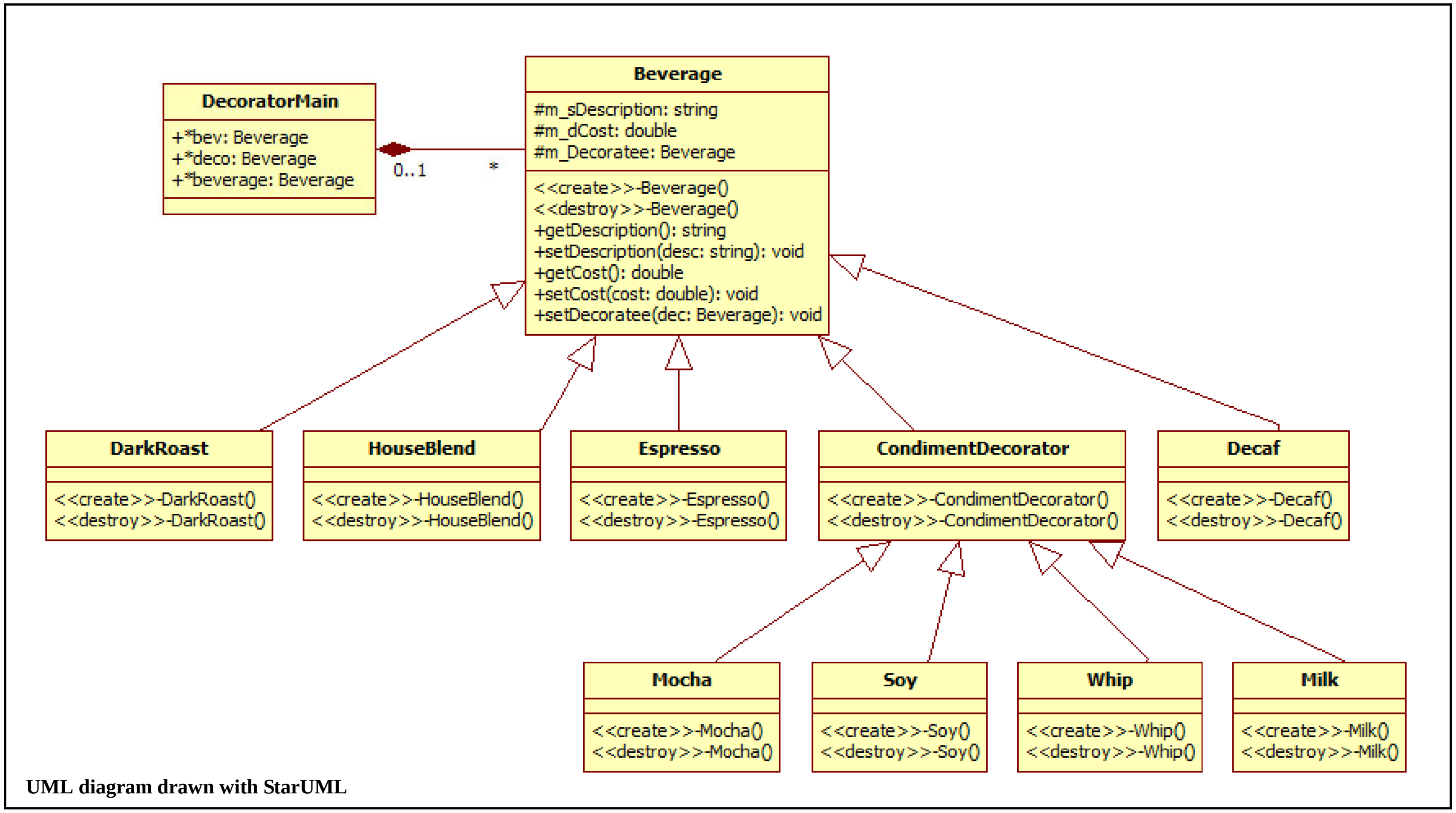


- **Decorators have the same parent type as the objects they decorate.**
- **One or more decorators can wrap an object.**
- **Because both share the same parent type, a decorator can be passed in place of the object it wraps.**
- **A Decorator adds its' own behavior before and/or after delegating to the object it wraps to do the rest of the job.**
- **Objects can be decorated dynamically at run time.**

** Yeah, I know these prices are probably way too low for reality!*

Design Patterns: Decorator

Code Sample



DecoratorMain

Instantiates instances of Beverage
Stacks each with a variety of CondimentDecorators
Calls getCost on the outermost decorator

Let's look at the code and run the demonstration.