## Overview

The design is based on the classic Decorator Pattern, where:

- **Component**: The abstract `Beverage` class serves as the base component.

- **Concrete Components**: Classes such as `Espresso`, `HouseBlend`, `DarkRoast`, and `Decaf`.

- **Decorator**: The abstract class `condimentDecorator` extends `Beverage`.

- **Concrete Decorators**: Classes like `Mocha`, `Whip`, `Soy`, and `SteamedMilk`.

## Execution Steps

1. **Initialization of Beverages**: Instances of concrete `Beverage` classes are created.

2. **Dynamic Decoration**: Beverages are decorated dynamically using condiment decorators.

3. **Printing the Final Beverage**: The application prints out the full description and final cost.

## How the Decorator Pattern is Dynamically Implemented

- **Wrapping and Extensibility**: Each decorator takes a `Beverage` object as a parameter.

- **Dynamic Composition**: Condiments can be added, removed, or reordered at runtime.

- **Separation of Concerns**: Base beverage classes handle core logic, decorators handle enhancements.

## Benefits of Using the Decorator Pattern in This Example

1. **Flexibility**: Allows dynamic composition without creating multiple subclasses.

2. **Enhanced Readability and Maintainability**: Each class has a single responsibility.

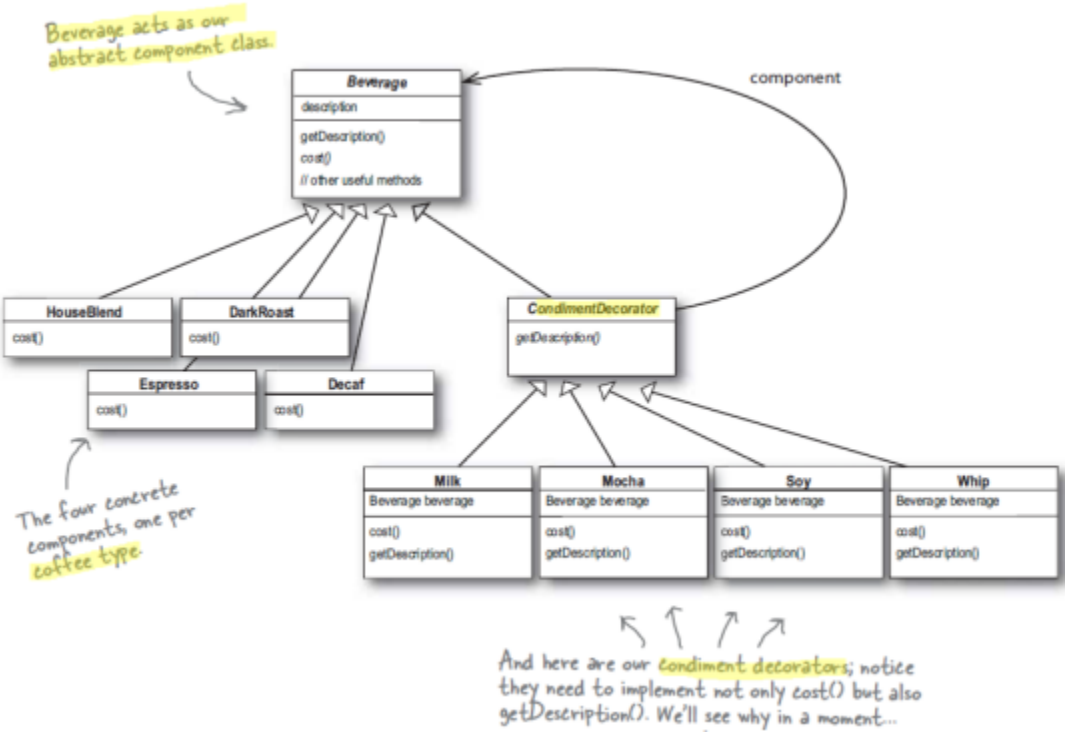
3. **Dynamic Behavior**: Decorators modify objects at runtime.

4. **Open/Closed Principle**: Open to extension without modifying base classes.

# Ouput:

```
Espresso  $1.99
DarkRoast, Mocha, Mocha, Whip  $1.49
House Blend Coffee, Soy, Mocha, Whip  $1.34
```

UML:


Beverage acts as our abstract component class.
Beverage
description
getDescription()
cost()
// other useful methods
component
HouseBlend
cost()
DarkRoast
cost()
Espresso
cost()
Decaf
cost()
CondimentDecorator
getDescription()
The four concrete components, one per coffee type.
Milk
Beverage beverage
cost()
getDescription()
Mocha
Beverage beverage
cost()
getDescription()
Soy
Beverage beverage
cost()
getDescription()
Whip
Beverage beverage
cost()
getDescription()
And here are our condiment decorators; notice they need to implement not only cost() but also getDescription(). We'll see why in a moment...