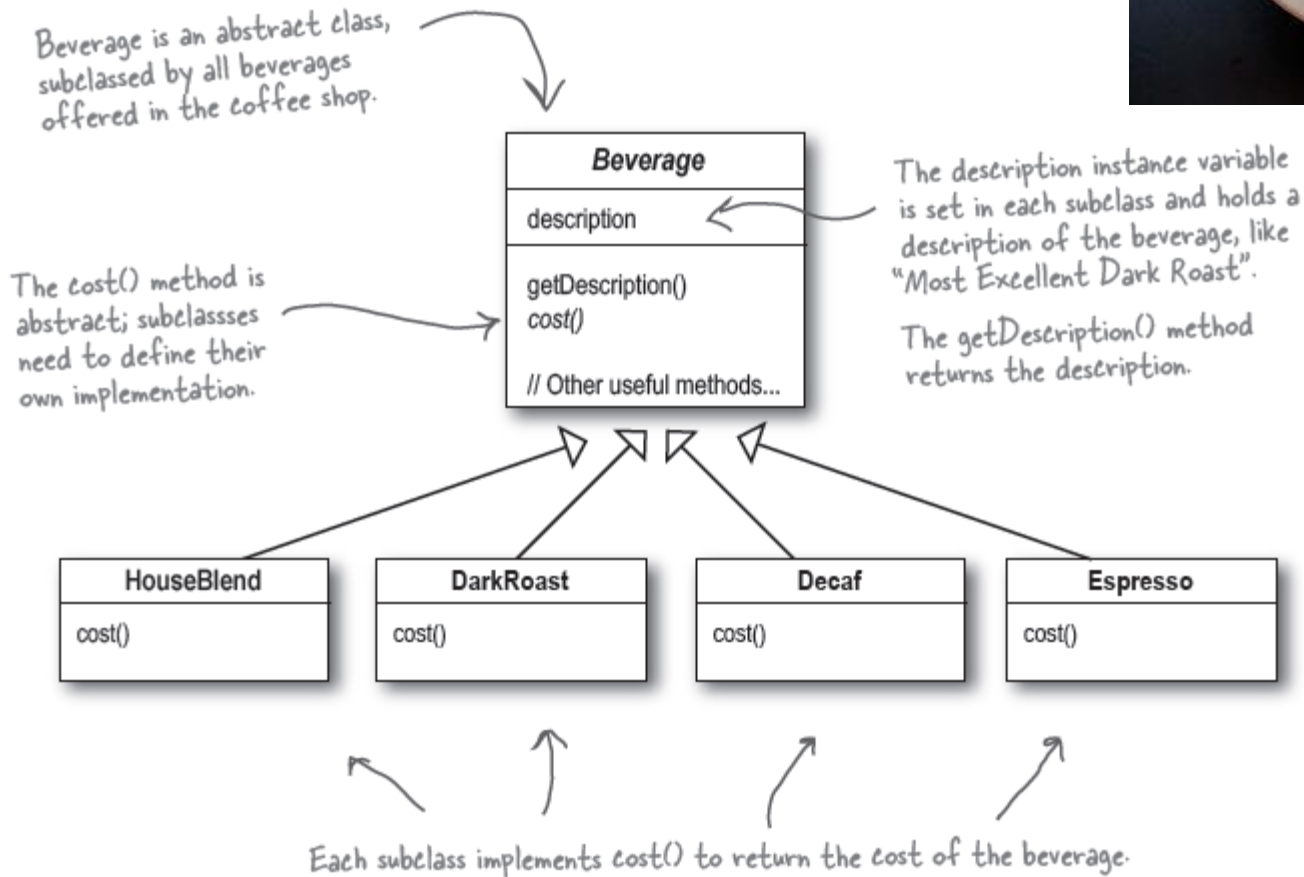


Design Patterns, Decorating Pattern

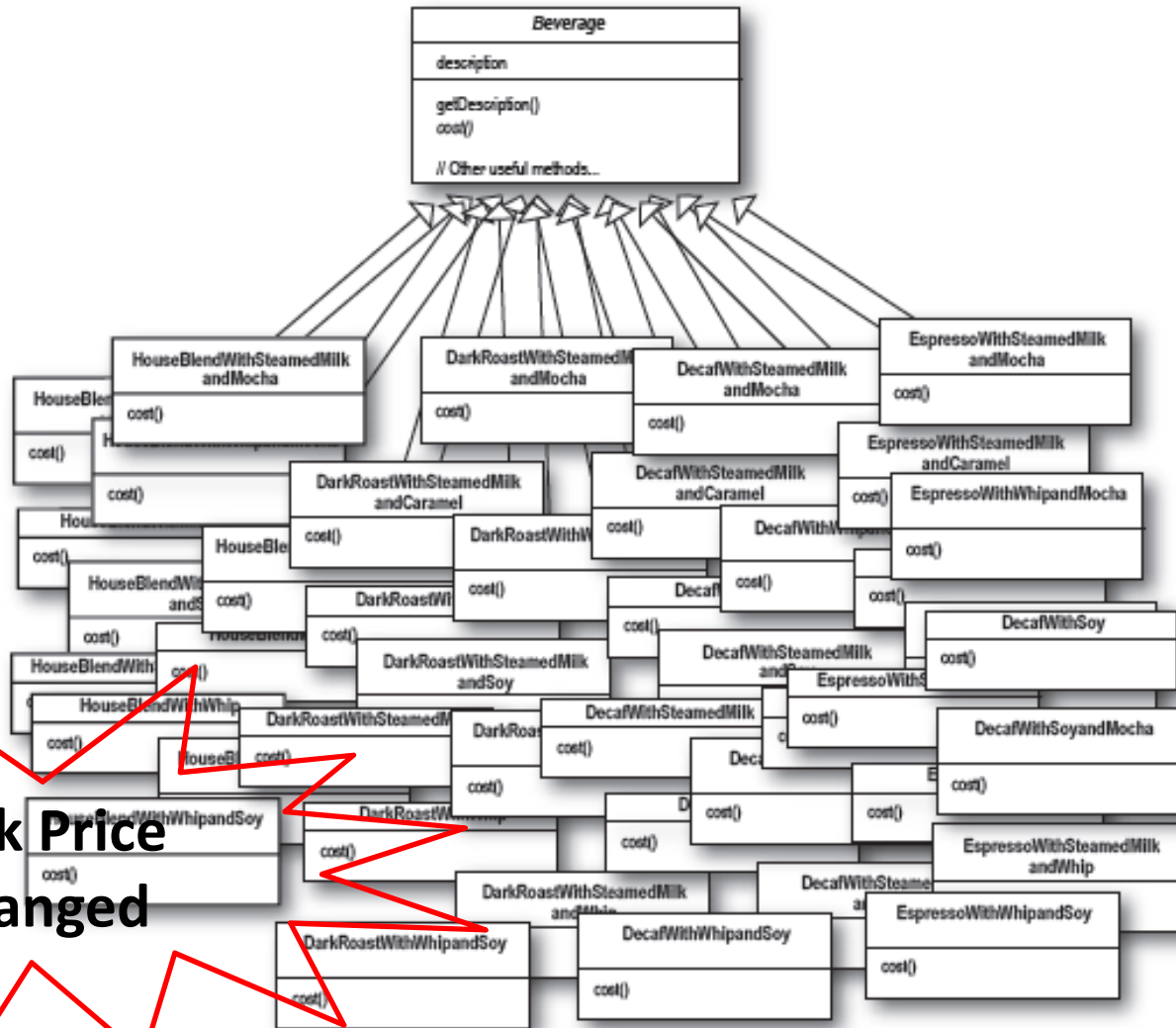
Dr, Ayman Ezzat

Decorating Pattern Design Eye for the Inheritance Guy.”

- Coffee



Class Explosion



Milk Price Changed


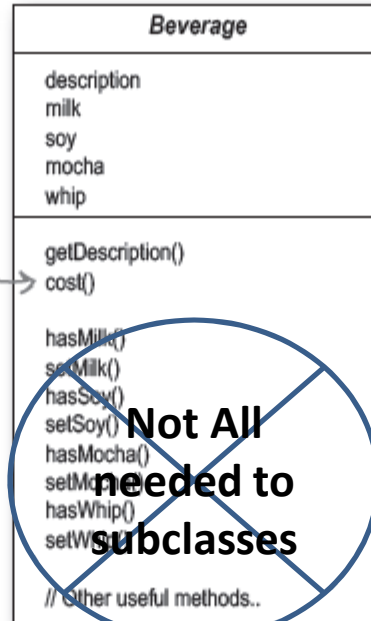


I got the idea, one function do every thing !!!

Now let's add in the subclasses, one for each beverage on the menu:


The superclass `cost()` will calculate the costs for all of the condiments, while the overridden `cost()` in the subclasses will extend that functionality to include costs for that specific beverage type.

Each `cost()` method needs to compute the cost of the beverage and then add in the condiments by calling the superclass implementation of `cost()`.

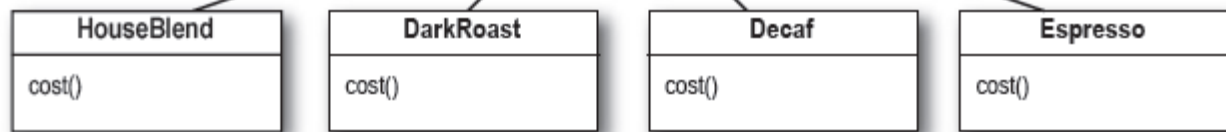


Design Principle

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



Sorry, we're closed.
That's right, we spent a lot of time getting this code correct and bug free, so we can't let you alter the existing code. It must remain closed to modification. If you don't like it, you can speak to the manager.



Decorate

- 1 **Take a DarkRoast object**
- 2 **Decorate it with a Mocha object**
- 3 **Decorate it with a Whip object**
- 4 **Call the cost() method and rely on delegation to add on the condiment costs**



Example of wrapping

1 We start with our DarkRoast object.



Remember that DarkRoast inherits from Beverage and has a cost() method that computes the cost of the drink.

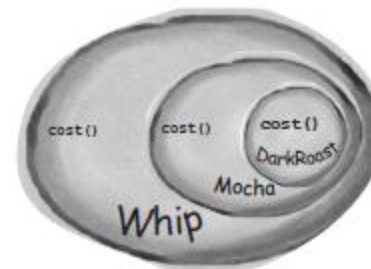
2 The customer wants Mocha, so we create a Mocha object and wrap it around the DarkRoast.



The Mocha object is a decorator. Its type mirrors the object it is decorating in this case, a Beverage. (By "mirror", we mean it is the same type.)

So, Mocha has a cost() method too, and through polymorphism we can treat any Beverage wrapped in Mocha as a Beverage, too (because Mocha is a subtype of Beverage).

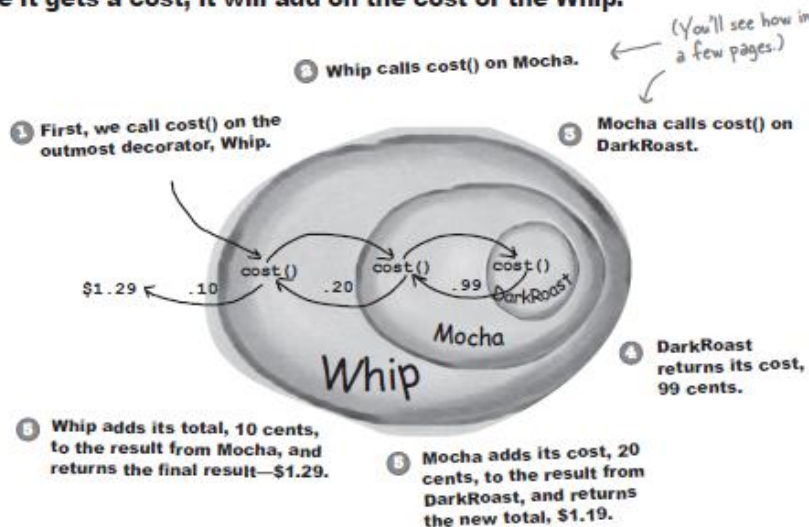
The customer also wants Whip, so we create a Whip decorator and wrap Mocha with it.



Whip is a decorator, so it also mirrors DarkRoast's type and includes a cost() method.

So, a DarkRoast wrapped in Mocha and Whip is still a Beverage and we can do anything with it we can do with a DarkRoast, including call its cost() method.

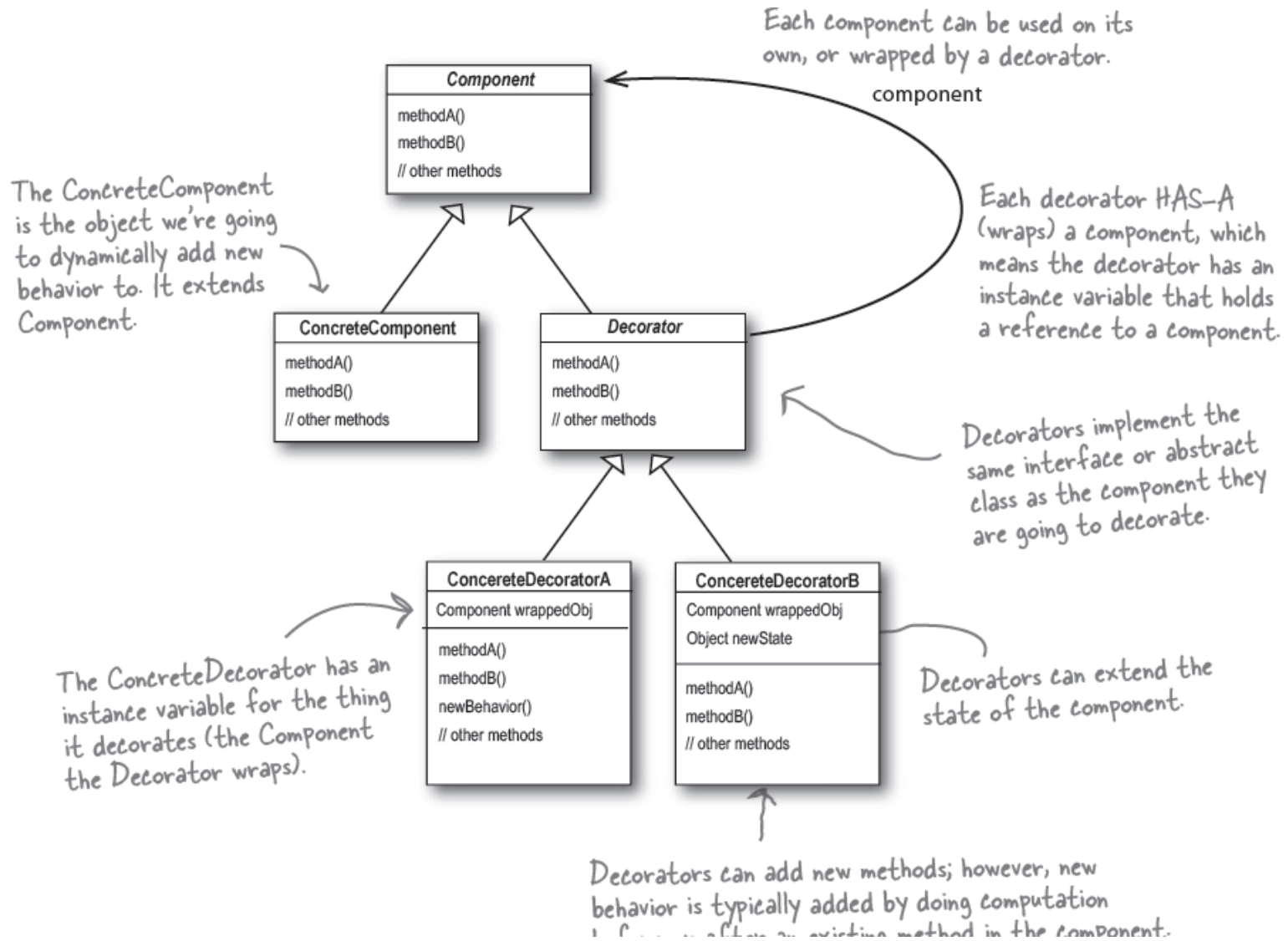
4 Now it's time to compute the cost for the customer. We do this by calling cost() on the outermost decorator, Whip, and Whip is going to delegate computing the cost to the objects it decorates. Once it gets a cost, it will add on the cost of the Whip.



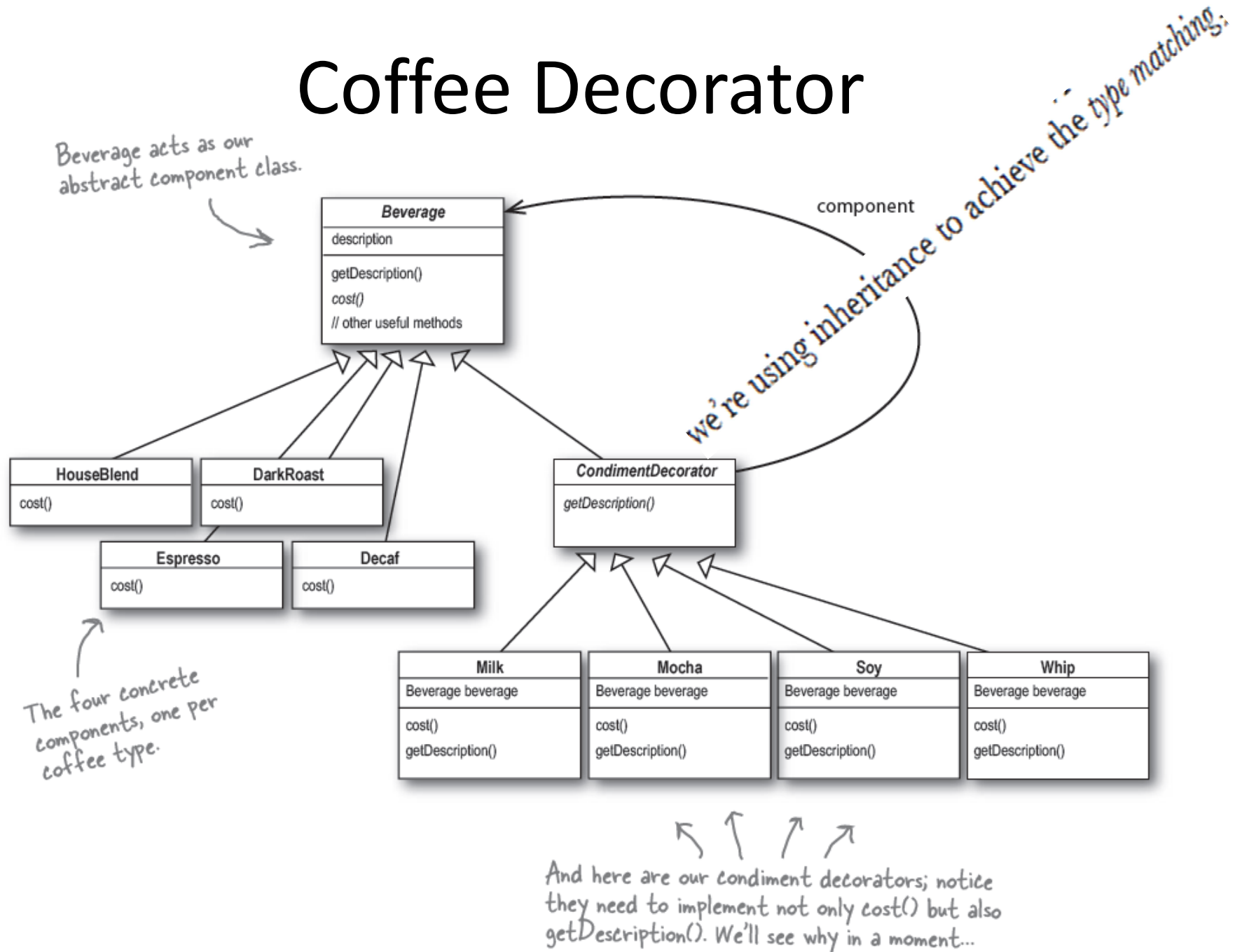
Definition to decoration pattern

The Decorator Pattern attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

Decorator Diagram



Coffee Decorator



Coding

```
public abstract class Beverage {  
    String description = "Unknown Beverage";  
  
    public String getDescription() {  
        return description;  
    }  
  
    public abstract double cost();  
}
```

Beverage is an abstract class with the two methods `getDescription()` and `cost()`.

`getDescription` is already implemented for us, but we need to implement `cost()` in the subclasses.

```
public abstract class CondimentDecorator extends Beverage {  
    public abstract String getDescription();  
}
```

First, we need to be interchangeable with a `Beverage`, so we extend the `Beverage` class.

We're also going to require that the condiment decorators all reimplement the `getDescription()` method. Again, we'll see why in a sec...

Coding

```
public class Espresso extends Beverage {  
  
    public Espresso() {  
        description = "Espresso";  
    }  
  
    public double cost() {  
        return 1.99;  
    }  
}
```

First we extend the Beverage class, since this is a beverage.

To take care of the description, we set this in the constructor for the class. Remember the description instance variable is inherited from Beverage.

Finally, we need to compute the cost of an Espresso. We don't need to worry about adding in condiments in this class, we just need to return the price of an Espresso: \$1.99.

Coding

Mocha is a decorator, so we extend `CondimentDecorator`.

Remember, `CondimentDecorator` extends `Beverage`.

We're going to instantiate Mocha with a reference to a `Beverage` using:

(1) An instance variable to hold the beverage we are wrapping.

(2) A way to set this instance variable to the object we are wrapping. Here, we're going to pass the beverage we're wrapping to the decorator's constructor.

```
public class Mocha extends CondimentDecorator {
    Beverage beverage;

    public Mocha(Beverage beverage) {
        this.beverage = beverage;
    }

    public String getDescription() {
        return beverage.getDescription() + ", Mocha";
    }

    public double cost() {
        return .20 + beverage.cost();
    }
}
```

Now we need to compute the cost of our beverage with Mocha. First, we delegate the call to the object we're decorating, so that it can compute the cost; then, we add the cost of Mocha to the result.

We want our description to not only include the beverage – say “Dark Roast” – but also to include each item decorating the beverage, for instance, “Dark Roast, Mocha”. So we first delegate to the object we are decorating to get its description, then append “, Mocha” to that description.

Main Program to test

```
public class StarbuzzCoffee {  
  
    public static void main(String args[]) {  
        Beverage beverage = new Espresso();  
        System.out.println(beverage.getDescription()  
            + " $" + beverage.cost());  
  
        Beverage beverage2 = new DarkRoast();  
        beverage2 = new Mocha(beverage2);  
        beverage2 = new Mocha(beverage2);  
        beverage2 = new Whip(beverage2);  
        System.out.println(beverage2.getDescription()  
            + " $" + beverage2.cost());  
  
        Beverage beverage3 = new HouseBlend();  
        beverage3 = new Soy(beverage3);  
        beverage3 = new Mocha(beverage3);  
        beverage3 = new Whip(beverage3);  
        System.out.println(beverage3.getDescription()  
            + " $" + beverage3.cost());  
    }  
}
```

Order up an espresso, no condiments
and print its description and cost.

Make a DarkRoast object.
Wrap it with a Mocha.

Wrap it in a second Mocha.

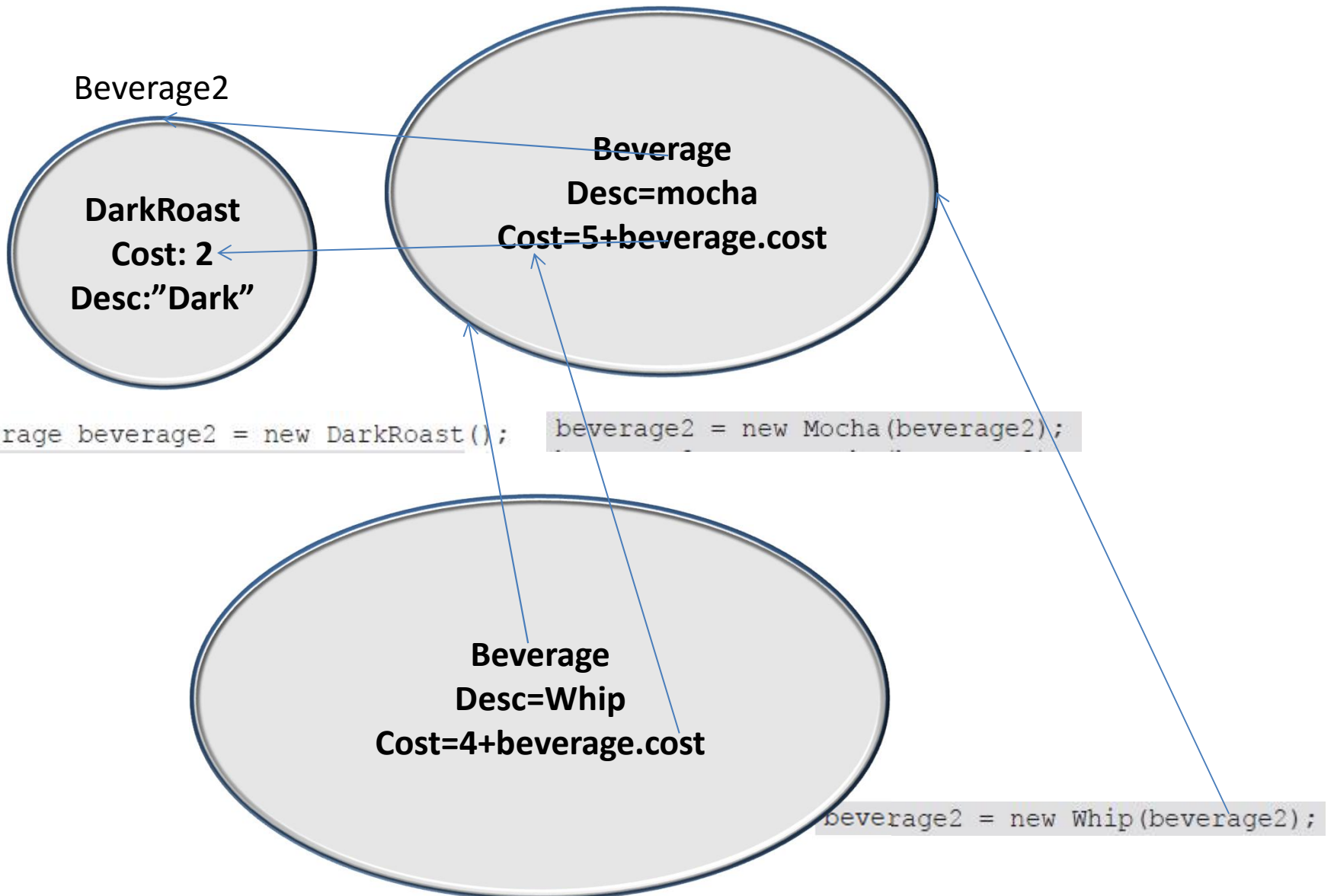
Wrap it in a Whip.

Finally, give us a HouseBlend
with Soy, Mocha, and Whip.

```
File Edit Window Help CloudsInMyCoffee
```

```
% java StarbuzzCoffee  
Espresso $1.99  
Dark Roast Coffee, Mocha, Mocha, Whip $1.49  
House Blend Coffee, Soy, Mocha, Whip $1.34  
%
```

Code Anatomy



New barista training

"double mocha soy lotte with whip"

