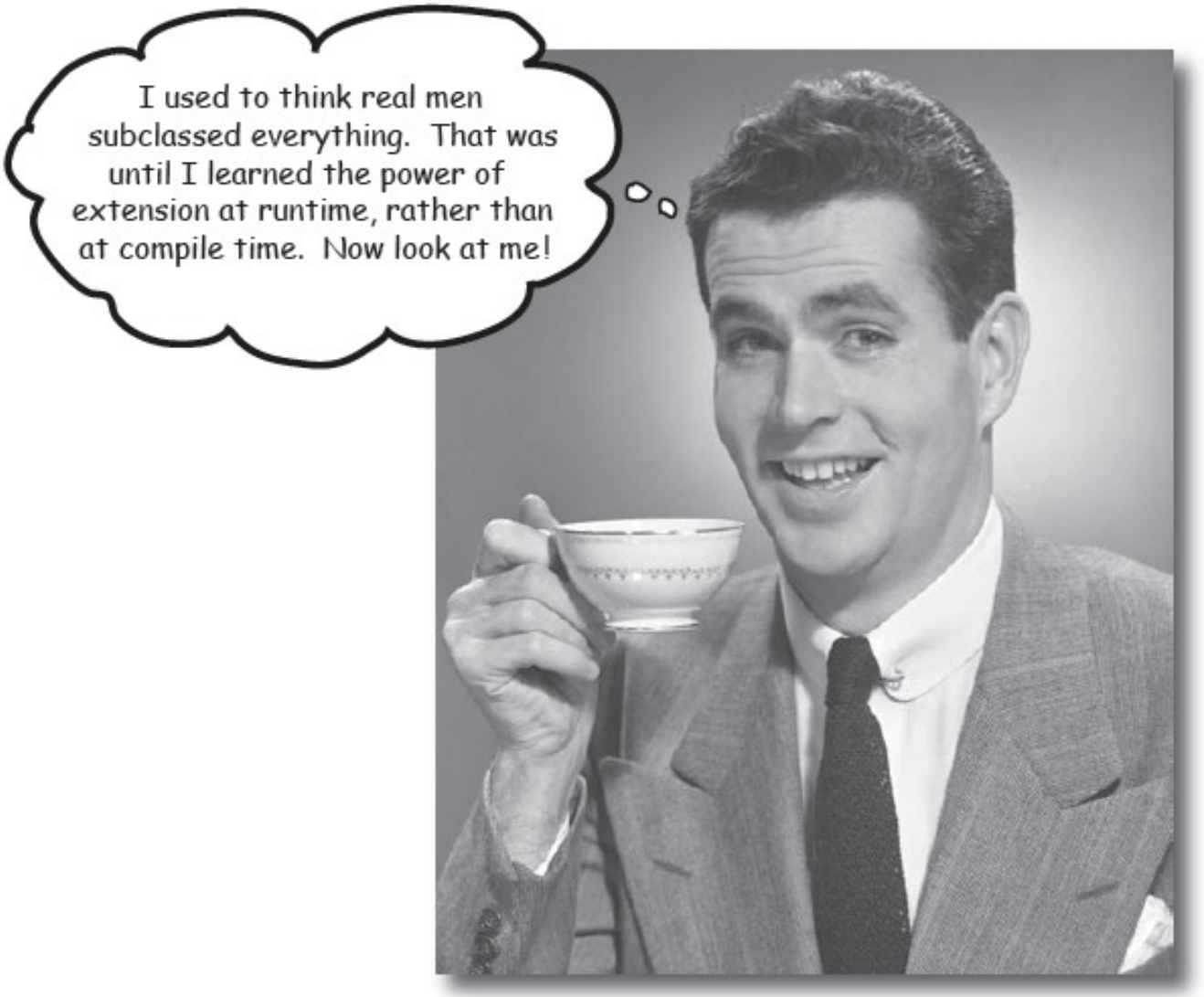
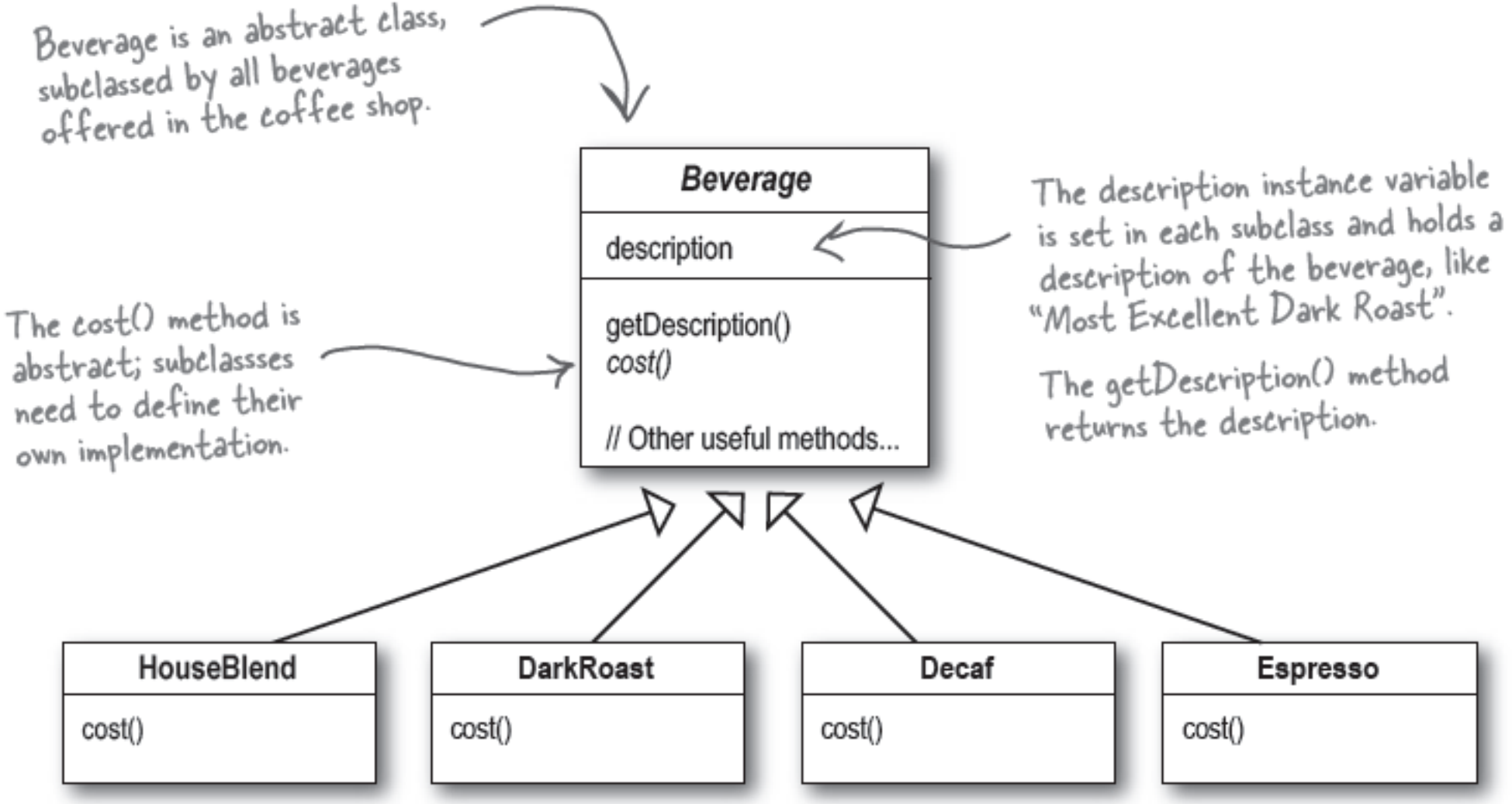
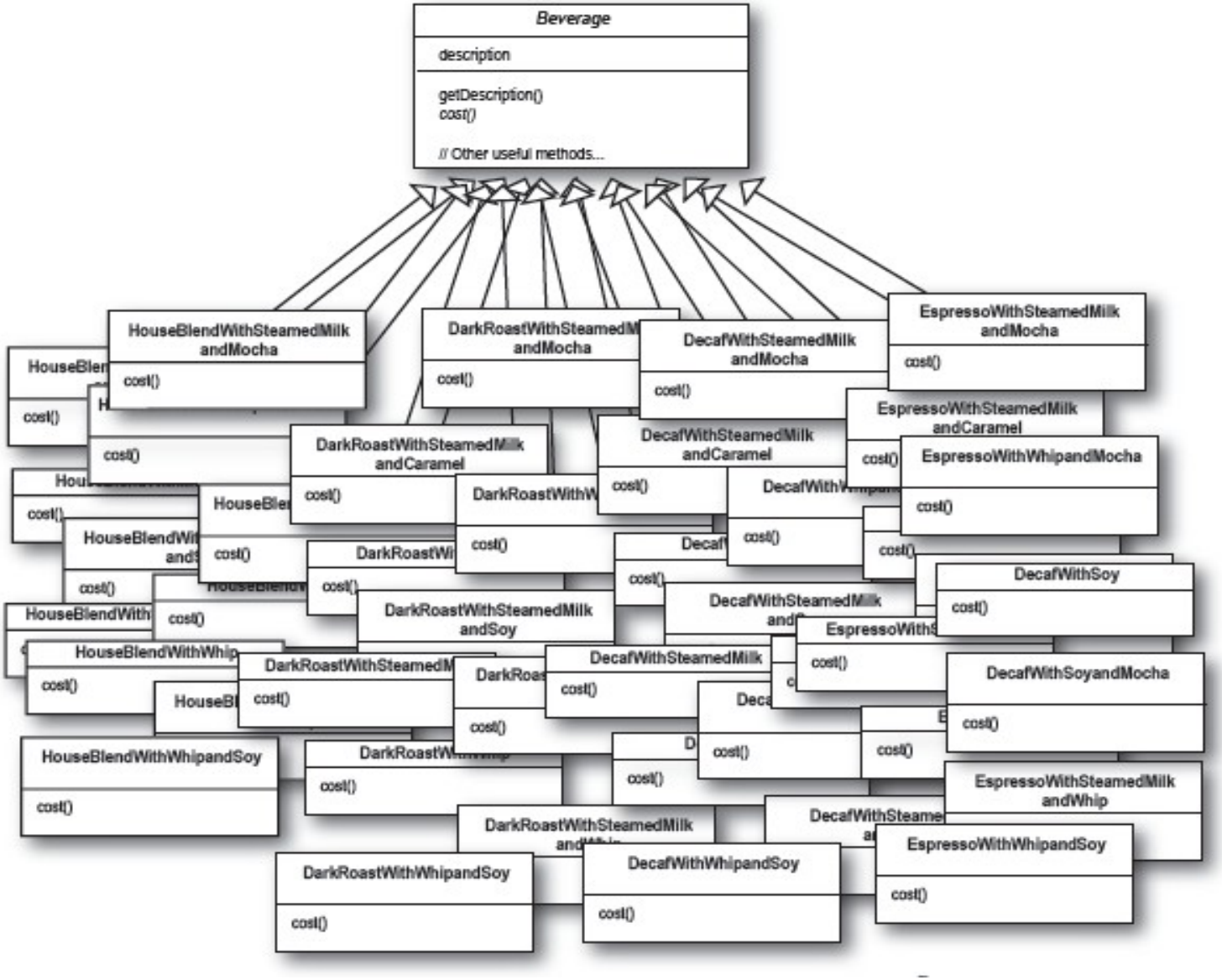
Decorator


I used to think real men subclassed everything. That was until I learned the power of extension at runtime, rather than at compile time. Now look at me!

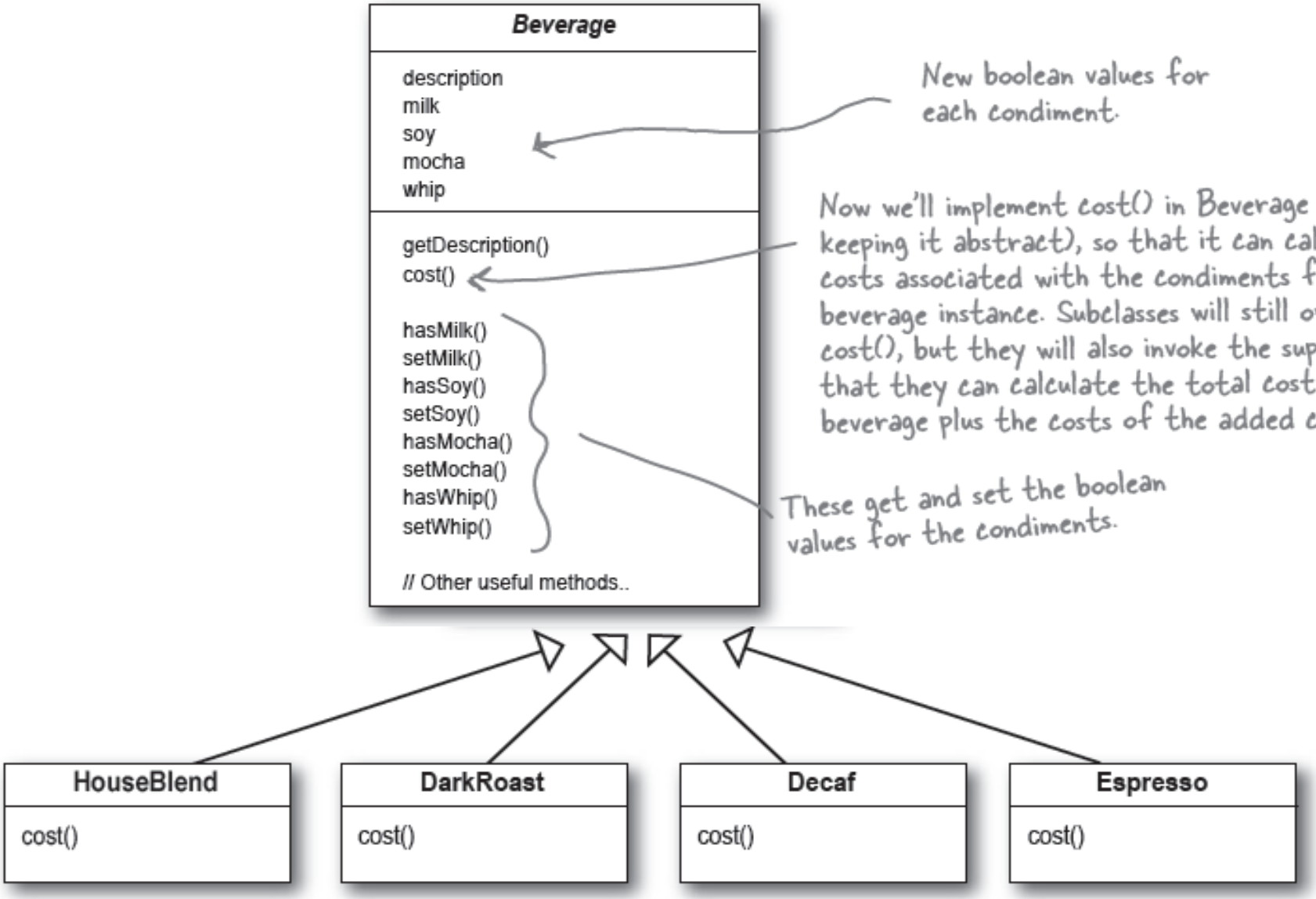
Príklad: Starbuzz Coffie



Koľko stojí káva s cukrom a mliekom?
Riešenie 1.



O niečo lepšie riešenie



Skúste si to!

Napíšte (stačí pseudokód) metódu cost() pre triedu Beverage a pre triedu DarkRoast.

Aké problémy môžu vzniknúť v súvislosti s týmto riešením?

Open-closed princíp

Triedy by mali byť zatvorené pokiaľ ide o zmeny, avšak otvorené, pokiaľ ide o rozšírenie

Koľko stojí káva s cukrom a mliekom?

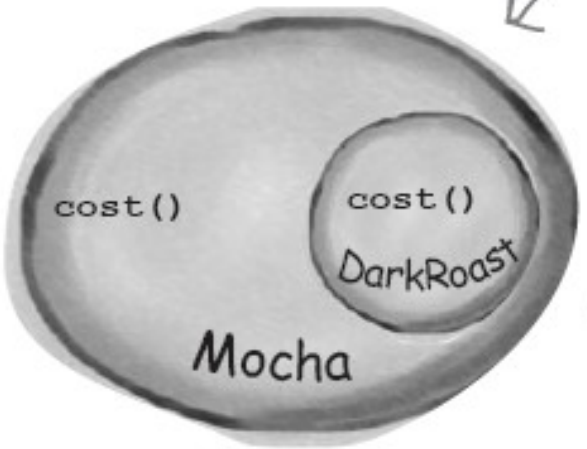
Resp. DarkRoast with Mocha and Whip?

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**

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.

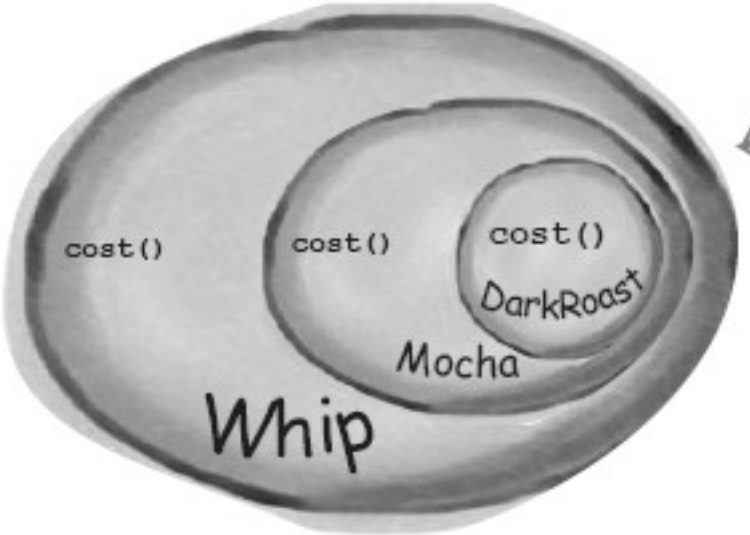
2 **The customer wants Mocha, so we create a Mocha object and wrap it around the DarkRoast.**



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).

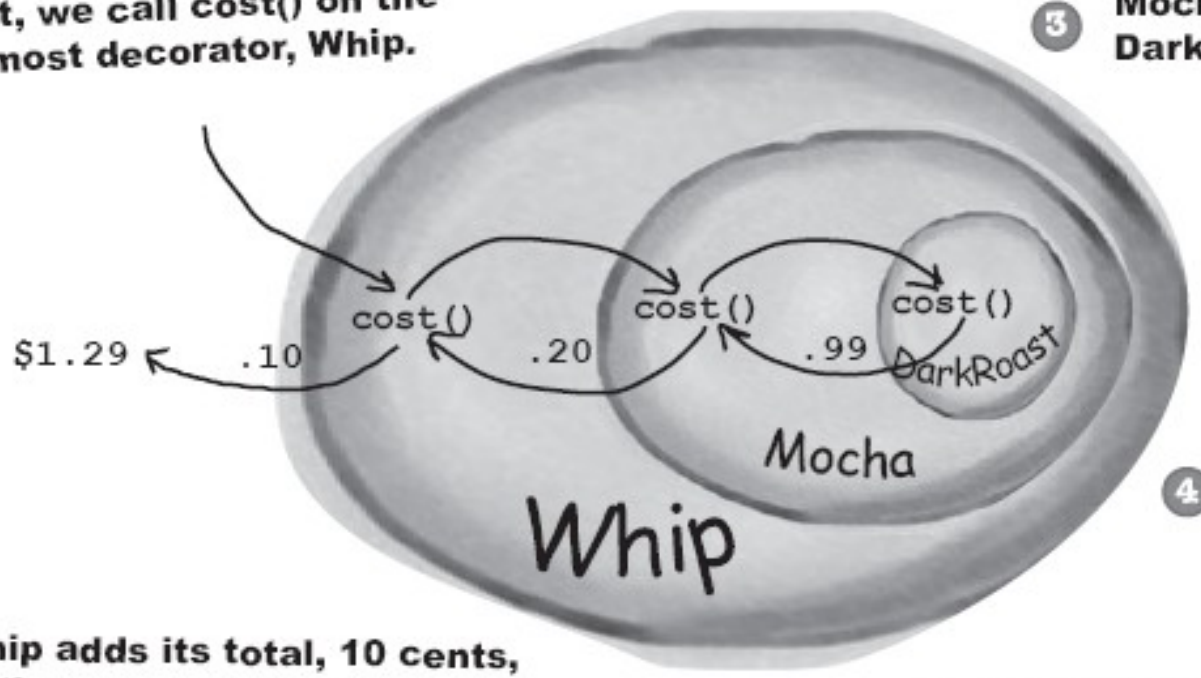
❸ The customer also wants Whip, so we create a Whip decorator and wrap Mocha with it.



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.**



Vlastnosti dekorátorov

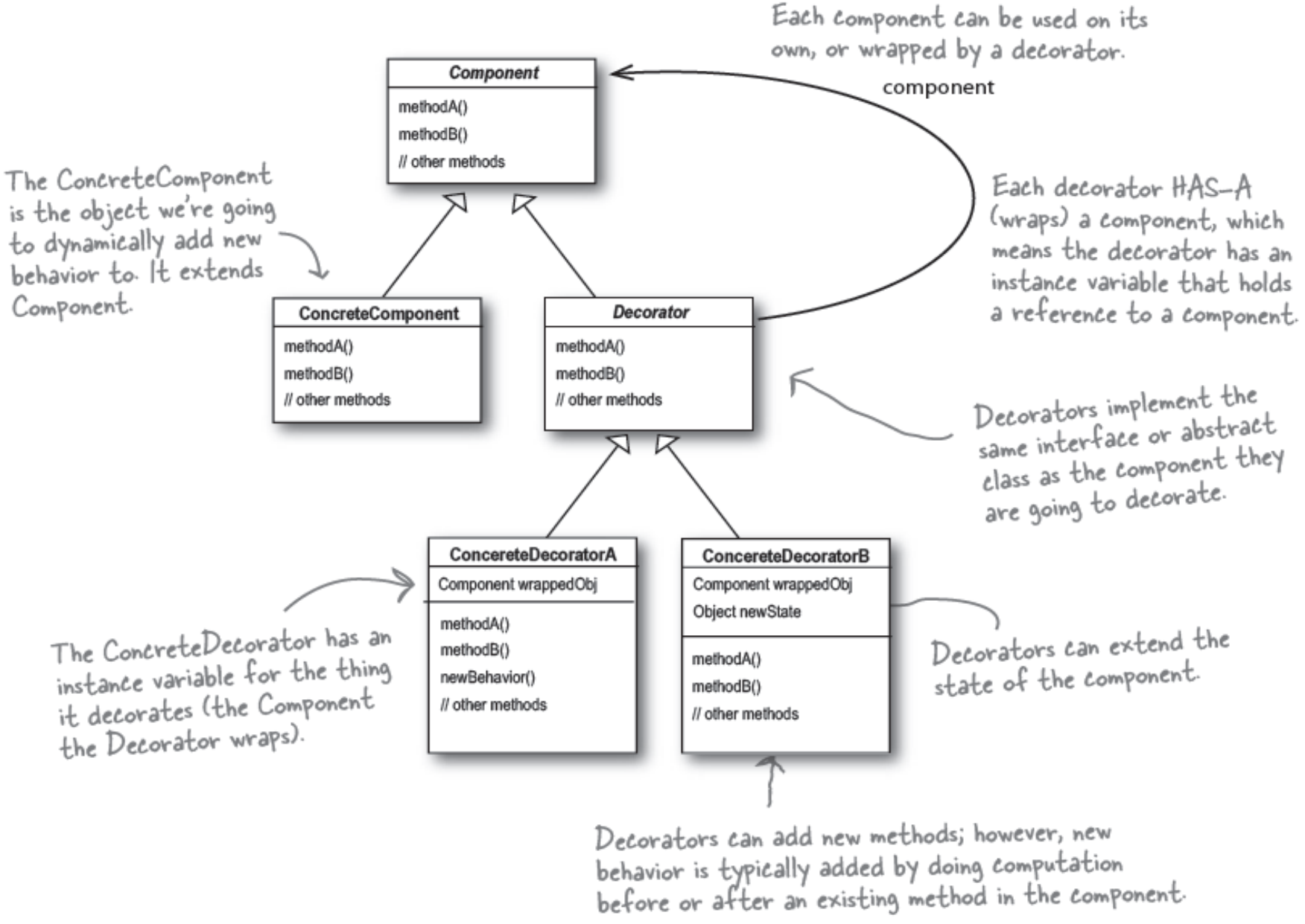
- Dekorátory majú spoločný nadtyp s triedami, ktoré dekorujú (vďaka tomu môžeme predávať dekorovaný objekt namiesto pôvodného)
- Jeden objekt môže byť dekorovaný jedným alebo viacerými dekorátormi
- Dekorátor môže pridávať svoje správanie predtým alebo potom než deleguje riadenie dekorovanému objektu
- Objekty môžu byť dekorované kedykoľvek počas behu programu

Vzor Dekorátor: definícia

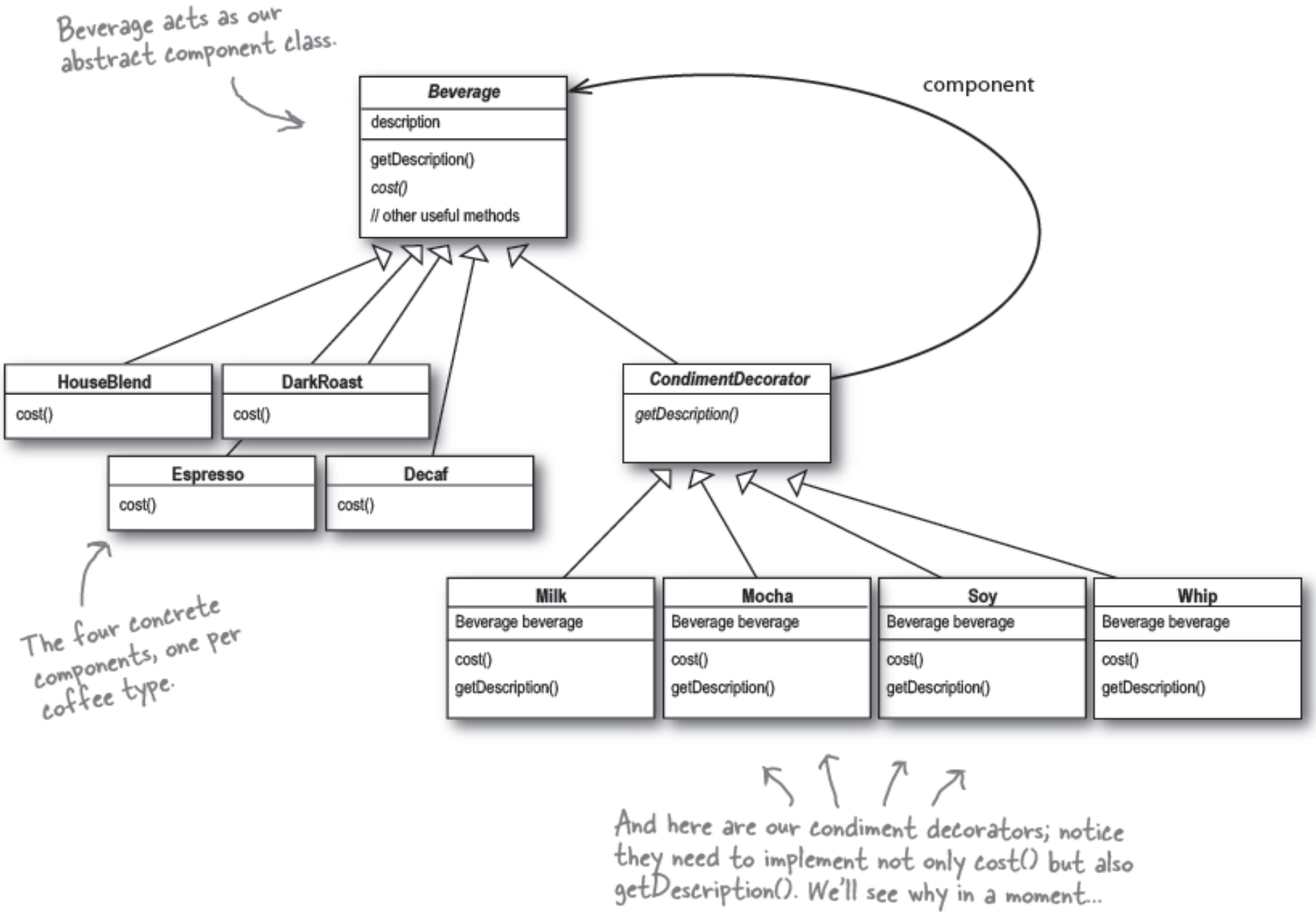
Vzor dekorátor dynamicky pridáva správanie a zodpovednosti iným objektom.

Dekorátory sú flexibilnou alternatívou dedičnosti

Vzor dekorátor: schéma



Káva s cukrom a mliekom



```
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.

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

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

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

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

```
public class Espresso extends Beverage {

    public Espresso() {
        description = "Espresso";
    }

    public double cost() {
        return 1.99;
    }
}
```

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.

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

```
public class Espresso extends Beverage {

    public Espresso() {
        description = "Espresso";
    }

    public double cost() {
        return 1.99;
    }
}
```

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.

Mocha is a decorator, so we extend CondimentDecorator.

Remember, CondimentDecorator extends Beverage.

```
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();
    }
}
```

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.

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.

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.

```
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());
```

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.