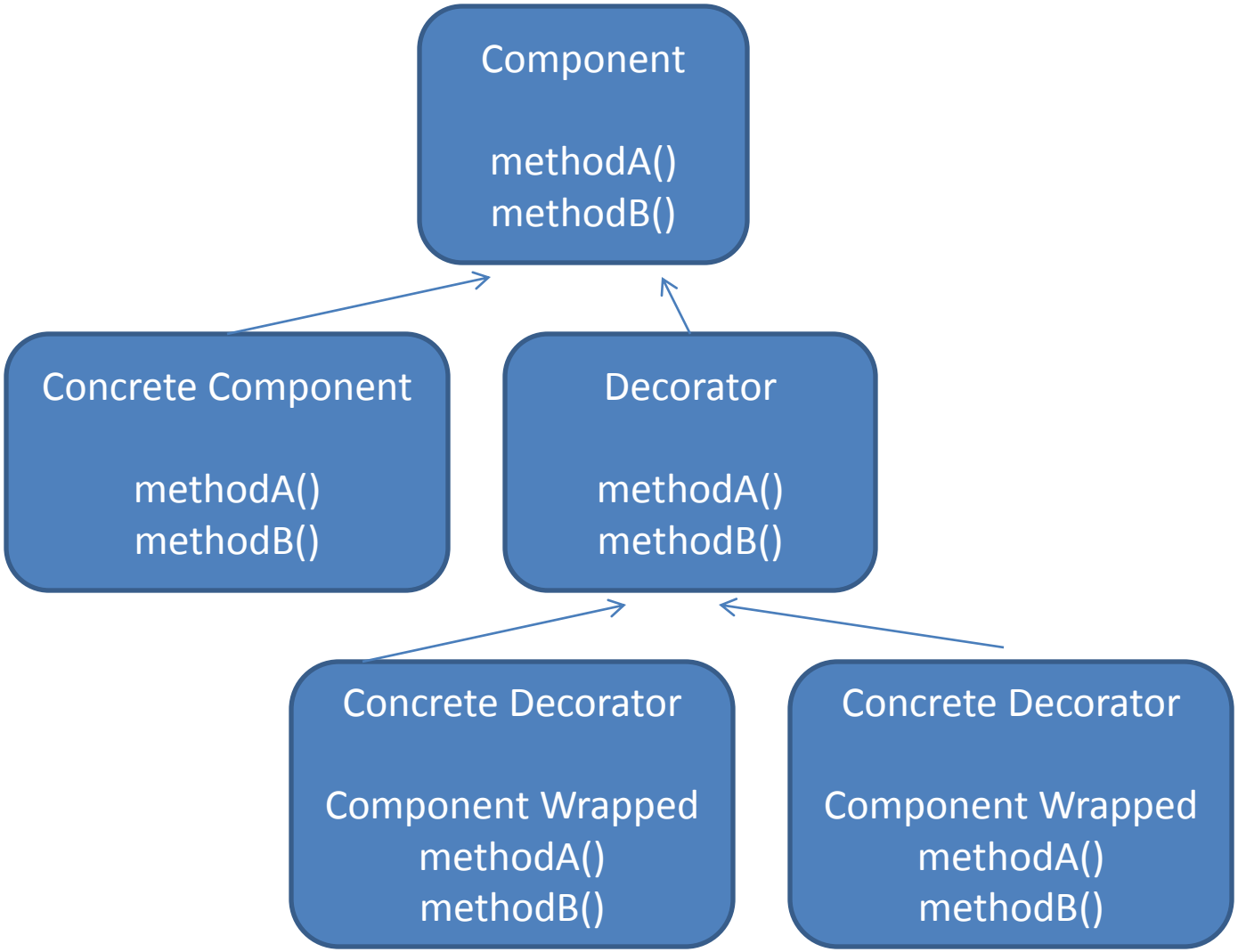
The Decorator Pattern

Decorator

Definition

- The decorator pattern attaches additional functionality or responsibility to an object dynamically.
- It provides a flexible alternative to sub classing for extending functionality. Since in case we are sub classing we have to alter the code but with decorator. We can add the functionality with out modifying the original code.

Class Diagram



Key Points

Decorators have the same supertype as the object they are decorating.

Decorators can be applied at run time to any number of levels.

The Decorator has the same supertype as the object it is decorating we either pass around a decorated object to the decorator or the object itself.

The decorator adds its own behavior either before or after delegating to the object it decorated to do the rest of the job.

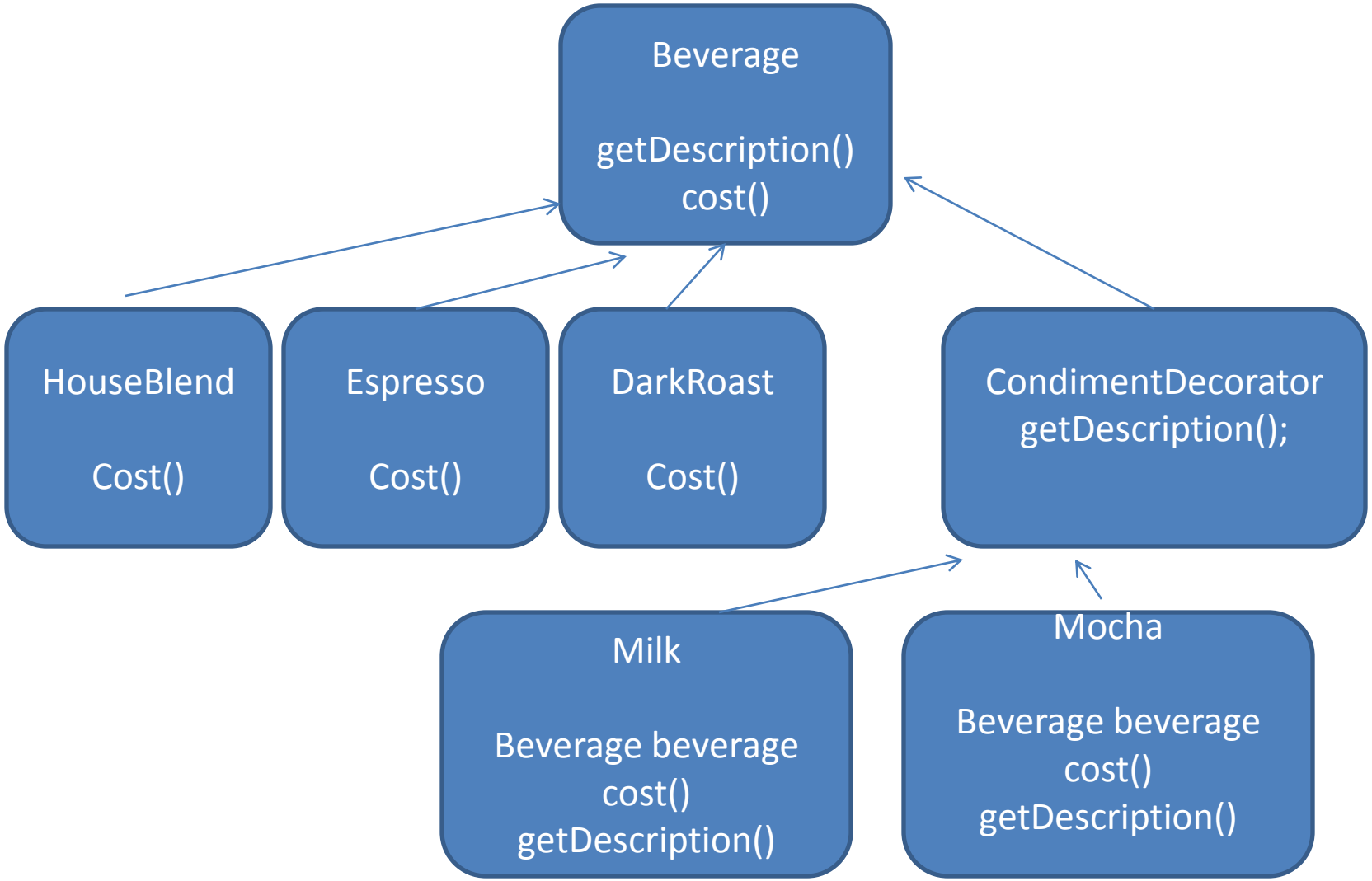
Design Problem

Lets consider a coffee shop which serves Beverages to the people coming to the shop.

People can ask for a normal espresso coffee, DarkRoast coffee, Espresso Coffee with Mocha and Milk and Soft Cream or be it any combination.

We have to build a system which is “scalable”, “maintainable”, “easy to add new toppings”, “easy to alter the cost of coffee or toppings” etc etc..

Class Diagram For Coffee Shop



Abstract Beverage Class/ Or Interface

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

Abstract Class CondimentDecorator

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

We have added the abstract signature since we want every decorator to tell that it is added to the coffee.

Expresso Beverage

```
public class Espresso extends Beverage {
    public Espresso() {
        description = “Espresso”;
    }
    public double cost()
        return 0.99;
}
```

Condiments Class (Mocha)

```
public class Mocha extend CondimentDecorator{
    Beverage bevarage;
    public Mocha(Beverage b)
        {this.beverage = b;}
    public String getDescription()
        return beverage.getDescription() + “Mocha”;
    public double cost()
        return beverage.cost()+0.20;
}
```

Main Class Creating an Espresso, and a Double Mocha Whip Dark Roast.

```
public class CoffeeShop{
    public static void main()
        {
        Beverage beverage = new Espresso();
        Beverage bvg1 = new DarkRoast();
        bvg1 = new Mocha(bvg1);
        bvg1= new Mocha(bvg1);
        bvg1 = new Whip(bvg1);
        System.out.println("Final Cost"+ bvg1.cost());
        }
```