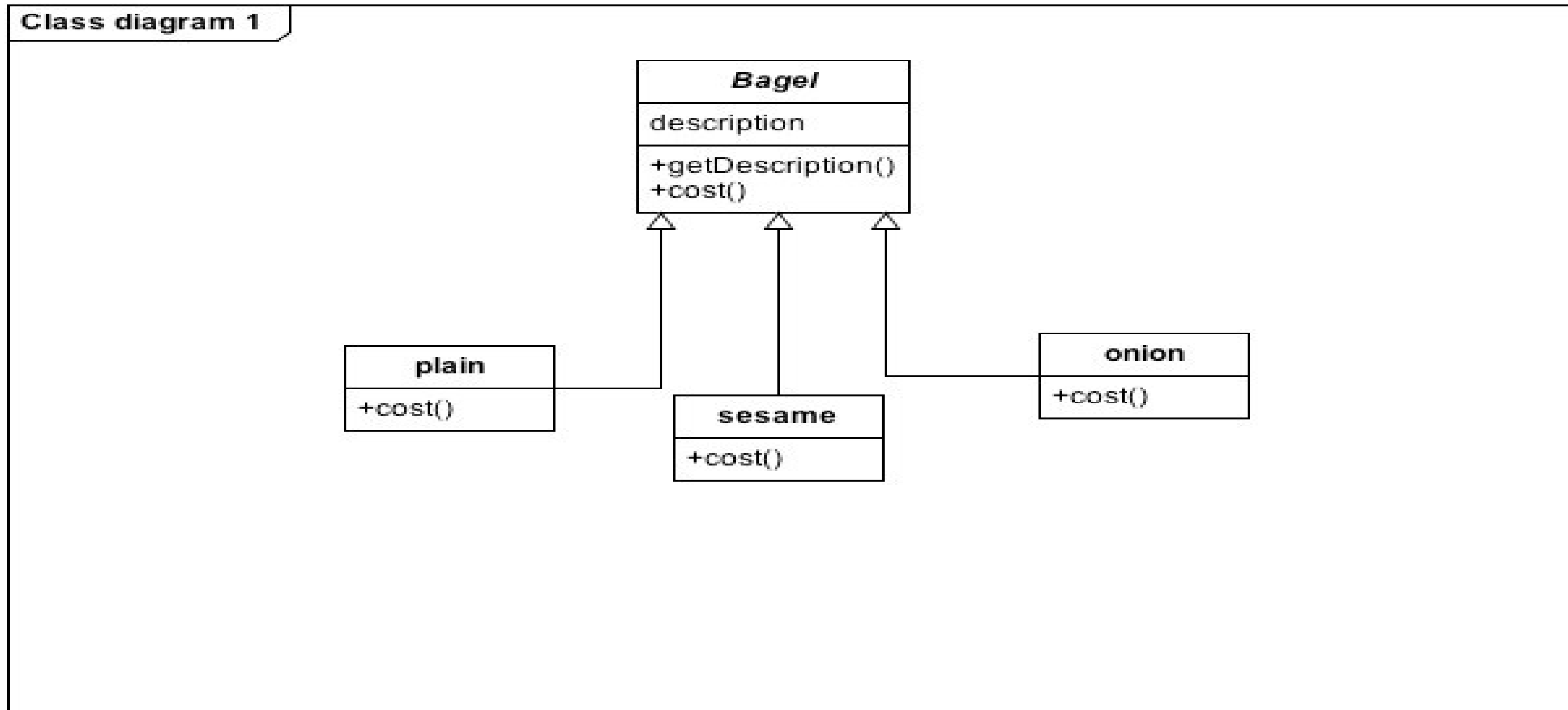
Decorator Pattern

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

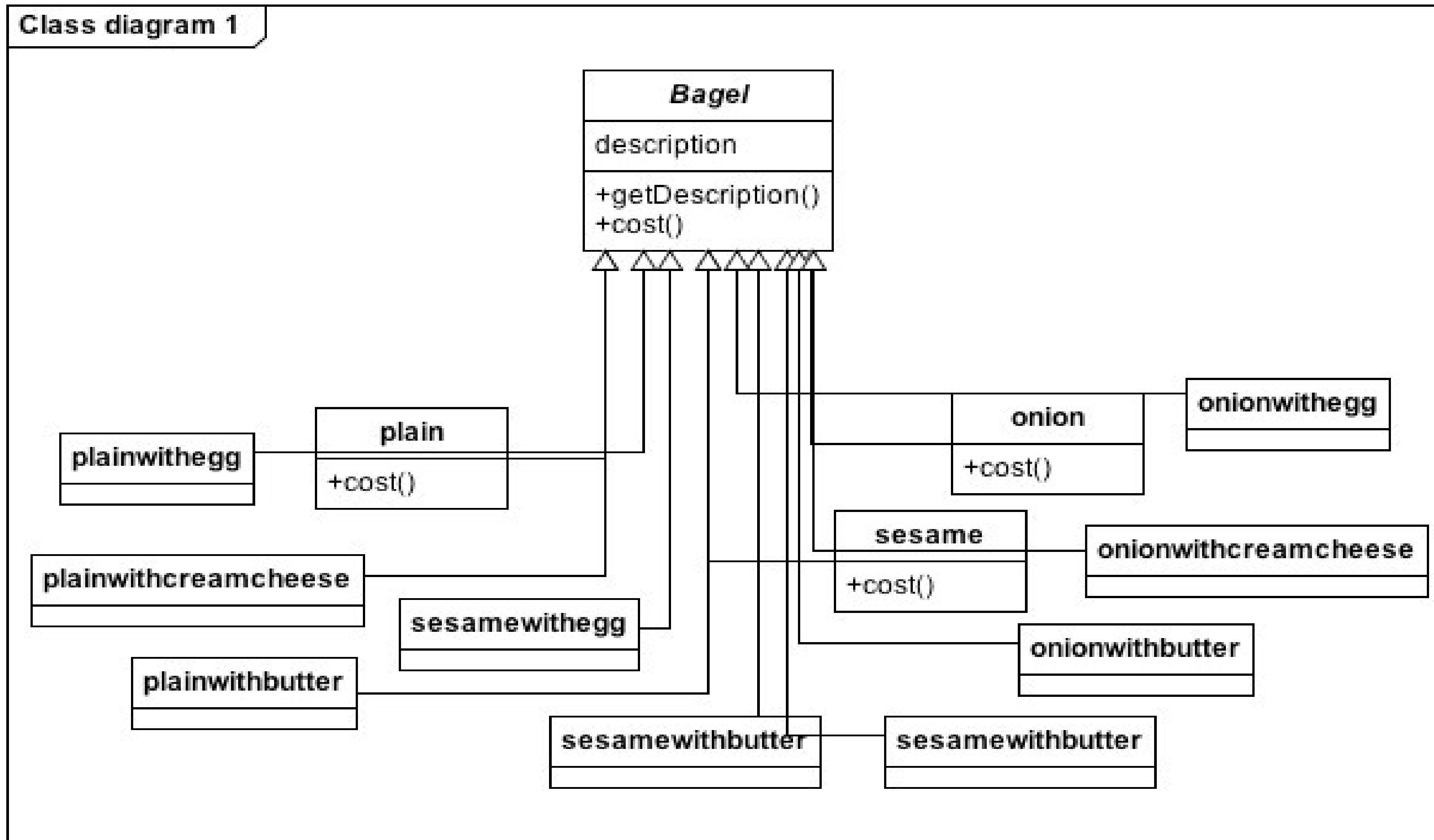
Problem

Overuse of inheritance often leads to an explosion of classes

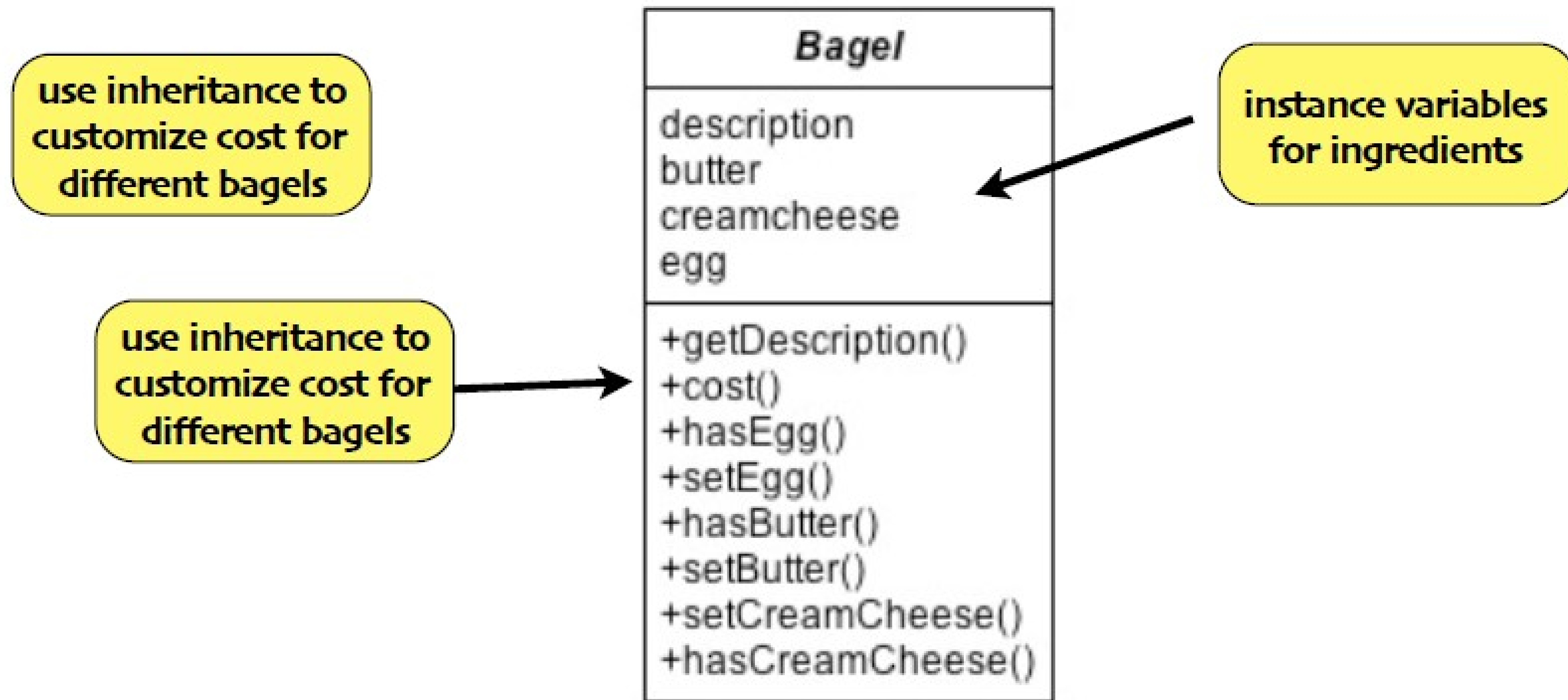
Example: Bagel Store



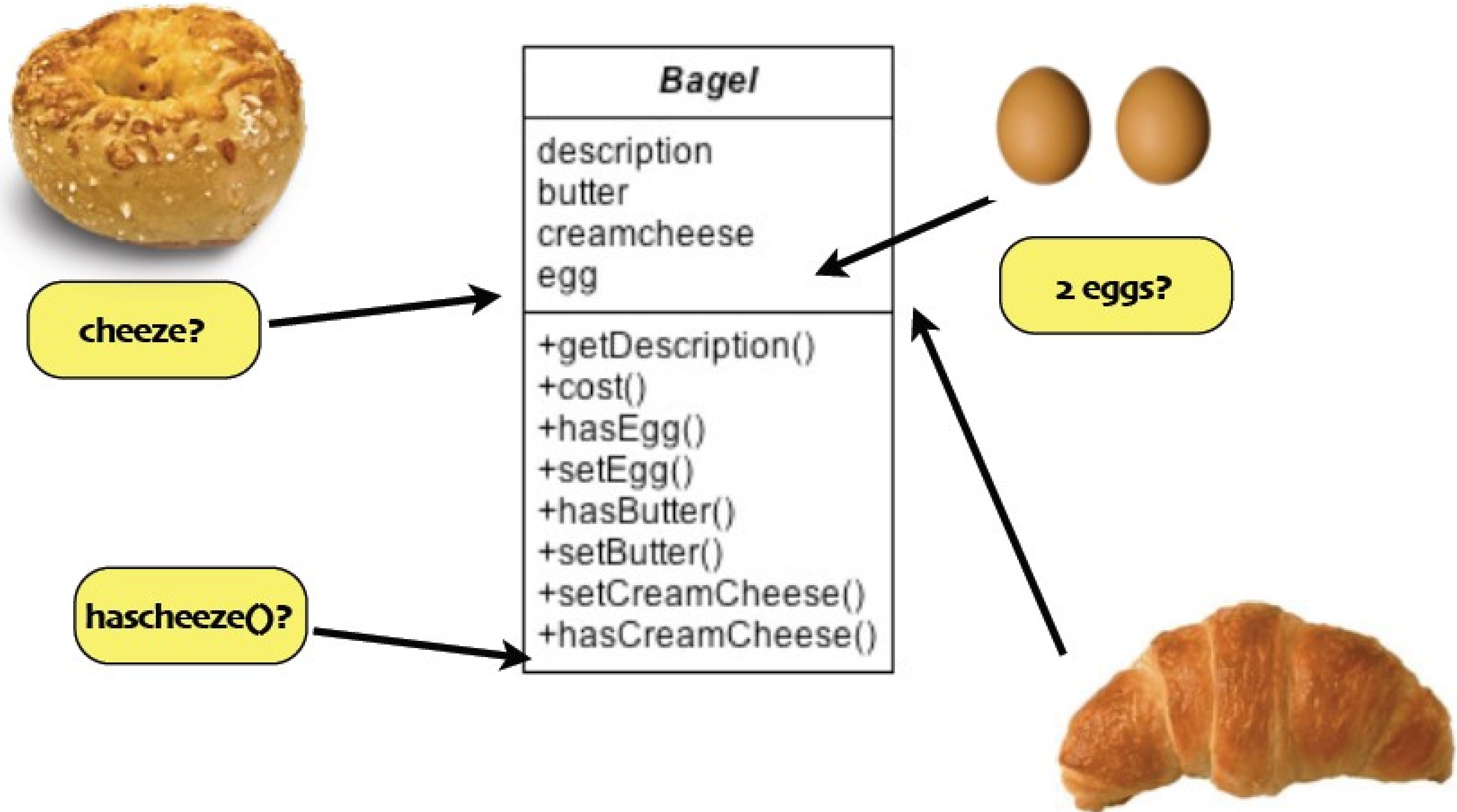
Class Explosion



Alternative Design



Disadvantages



Design Principle

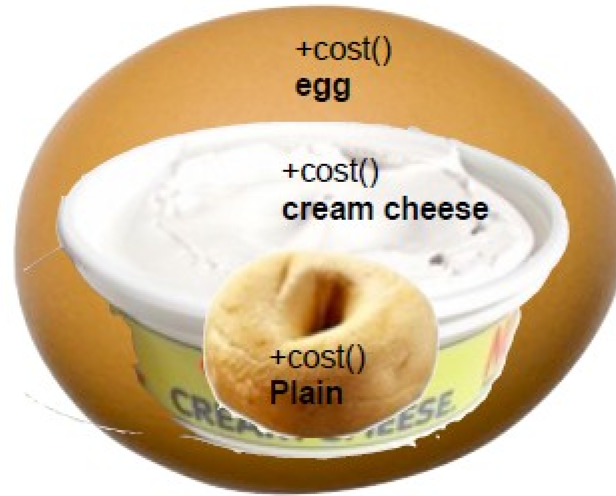
Classes should be open for extension, but closed for modification

What this means

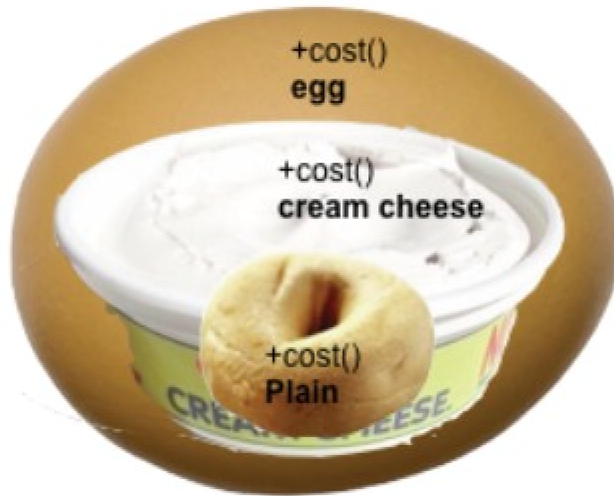
- ★**You can extend functionality through inheritance but that does not always achieve flexibility in our designs.**
- ★**We should be able to extend behavior without modifying existing code.**
- ★**You can use composition and delegation to add new behavior at runtime.**

>Favor composition over inheritance<

'Wrap' our bagels with decorators



Delegation to access behavior



1. egg.cost()
2. egg calls CC.cost()
3. CC calls plain.cost()
4. plain returns $plain
5. CC returns $plain + $cc
6. Egg returns $egg + ($plain+$cc)

Properties of Decorators

- ★**We have Objects and Decorators**
- ★**Decorators have the same supertype as objects they decorate.**
- ★**You can pass around a decorated object instead of the original.**
- ★**The Decorator adds it's own behavior before or after delegating to the object it decorates**
- ★**Objects can be decorated dynamically at runtime.**

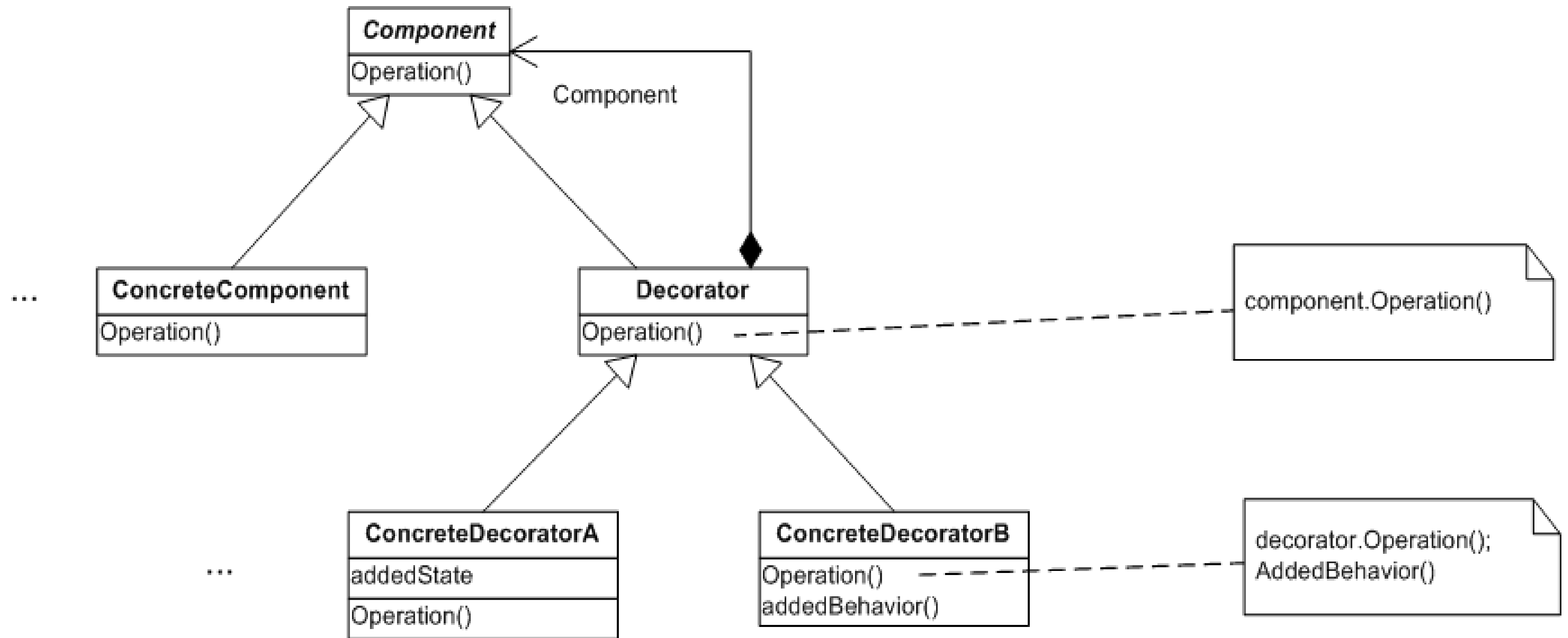
Concrete Decorators

Will contain a reference to the object they are decorating:

- Object could be original object that is being decorated
- Object could be another concrete decorator

They override method that needs 'decorated' and add to the existing behavior (delegate to existing behavior then add something to it)

Class Diagram: Component and Decorator are Abstract



Decorator Class

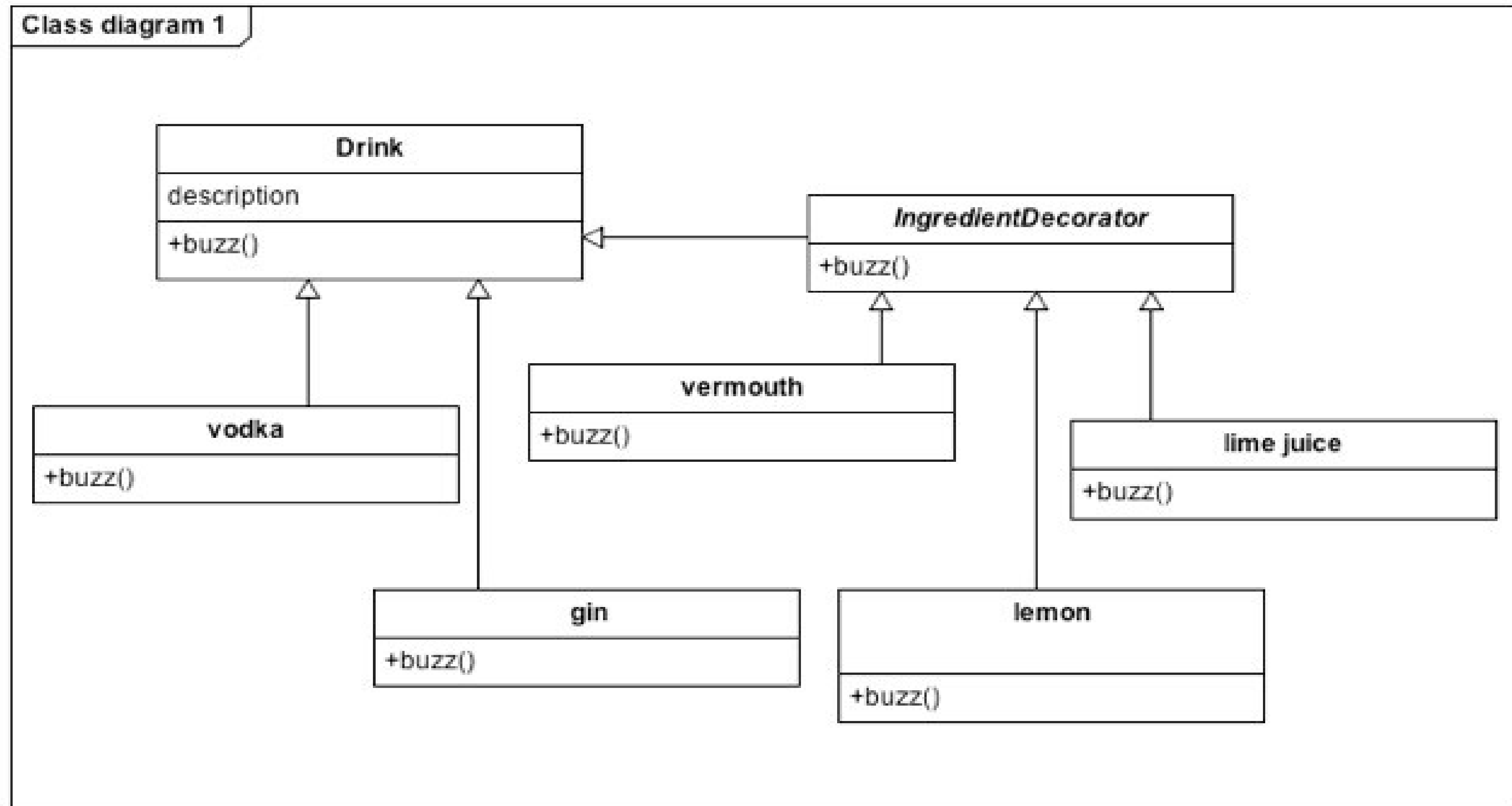
```
public class ConcreteDecoratorA extends Decorator {

    Component component;

    public ConcreteDecoratorA(Component component) {
        this.component=component;
    }

    public double methodA() {
        return 3 + component.methodA();
        //return component.methodA() + 3
        }
}
```

instance variable holding a pointer to a component

set this in constructor

add behavior before or after

Example: Make a Drink using Decorator



The main app.

```
public class MadMenDrinkMixer {
   public static void main(String[] args) {
      // Fix an old fashioned
      Drink drink = new Bourbon();
      drink = new Bitter(drink);
      drink = new Bitter(drink);
      drink = new Sugar(drink);
      drink = new Cherry(drink);
      System.out.println(drink.getDescription() +
" has: " +     drink.calories() + " calories\n");
   }
}
```

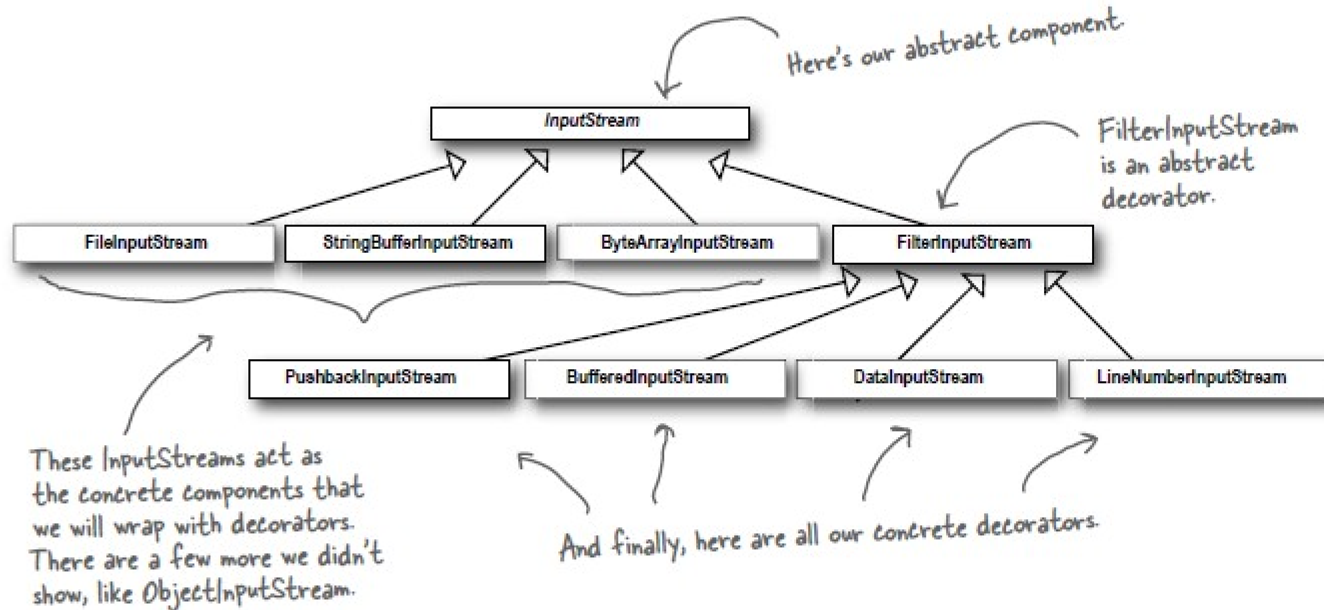
Drink class

```
    public abstract class Drink {
    String description = "unknown Drink";
public String getDescription() {
    return description;
}
public abstract double calories();
}

public class Bourbon extends Drink {
    public Bourbon() {
        description = "Bourbon";
    }
    public double calories() {
        return 7;
    }
}
```

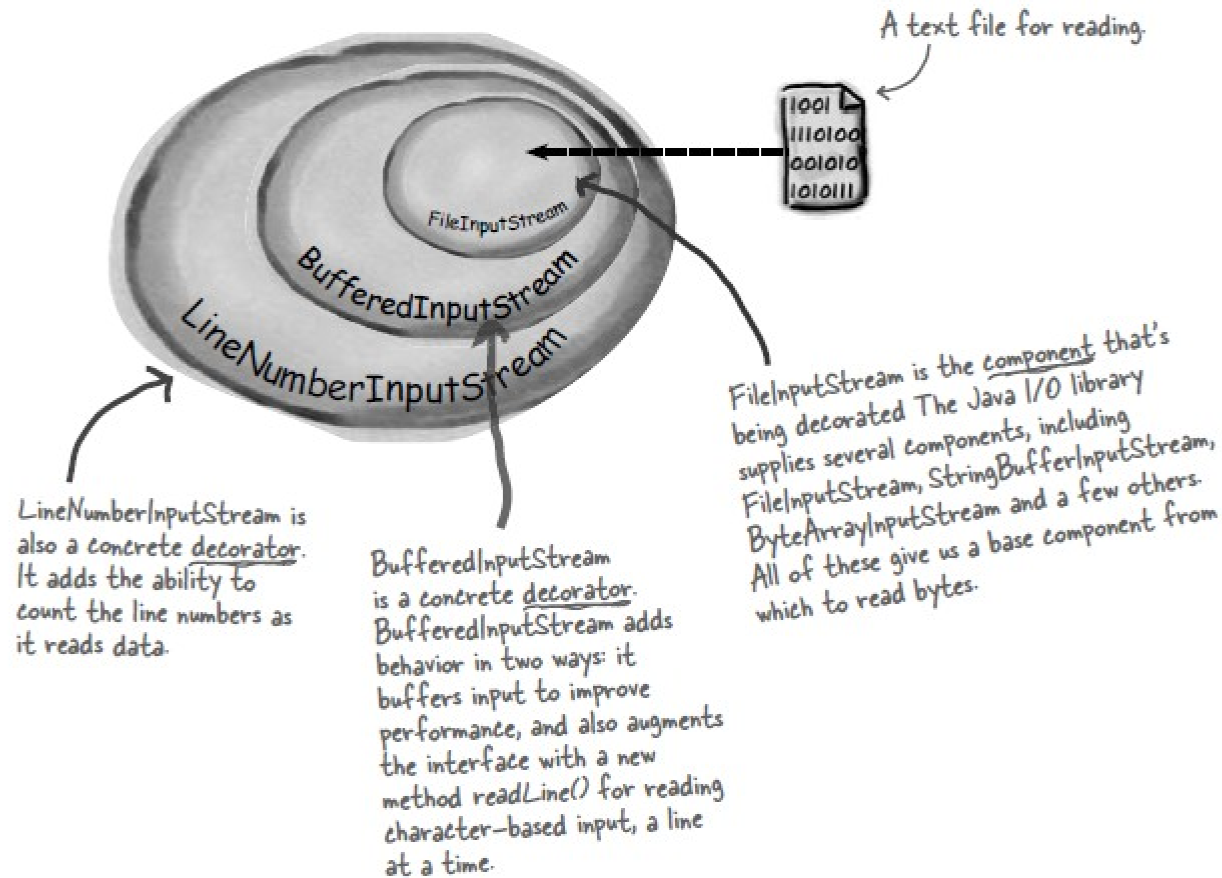
Ingredient Decorator

```
    public abstract class IngredientsDecorator extends Drink {
    public abstract String getDescription();
}

public class LimeJuice extends IngredientsDecorator {
    Drink drink;
    public LimeJuice(Drink drink) {
        this.drink=drink;
    }
    public String getDescription() {
        return drink.getDescription() + ", Lime Juice";
    }
    public double calories() {
        return 0 + drink.calories();
    }
}
```

Real World Decorators: Java I/O



Decorators are Wrapper Classes



BufferedInputStream and ***LineNumber***InputStream both extend ***Filter***InputStream, which acts as the abstract decorator class.