



The Decorator Pattern

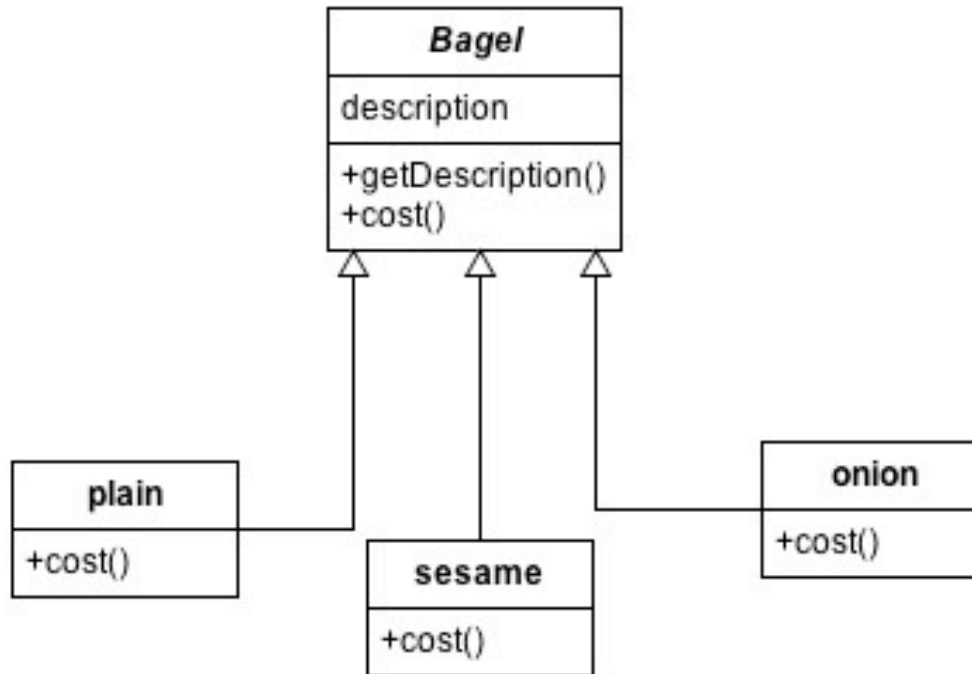
**Human Computer Interaction Research
University of Nevada, Reno**



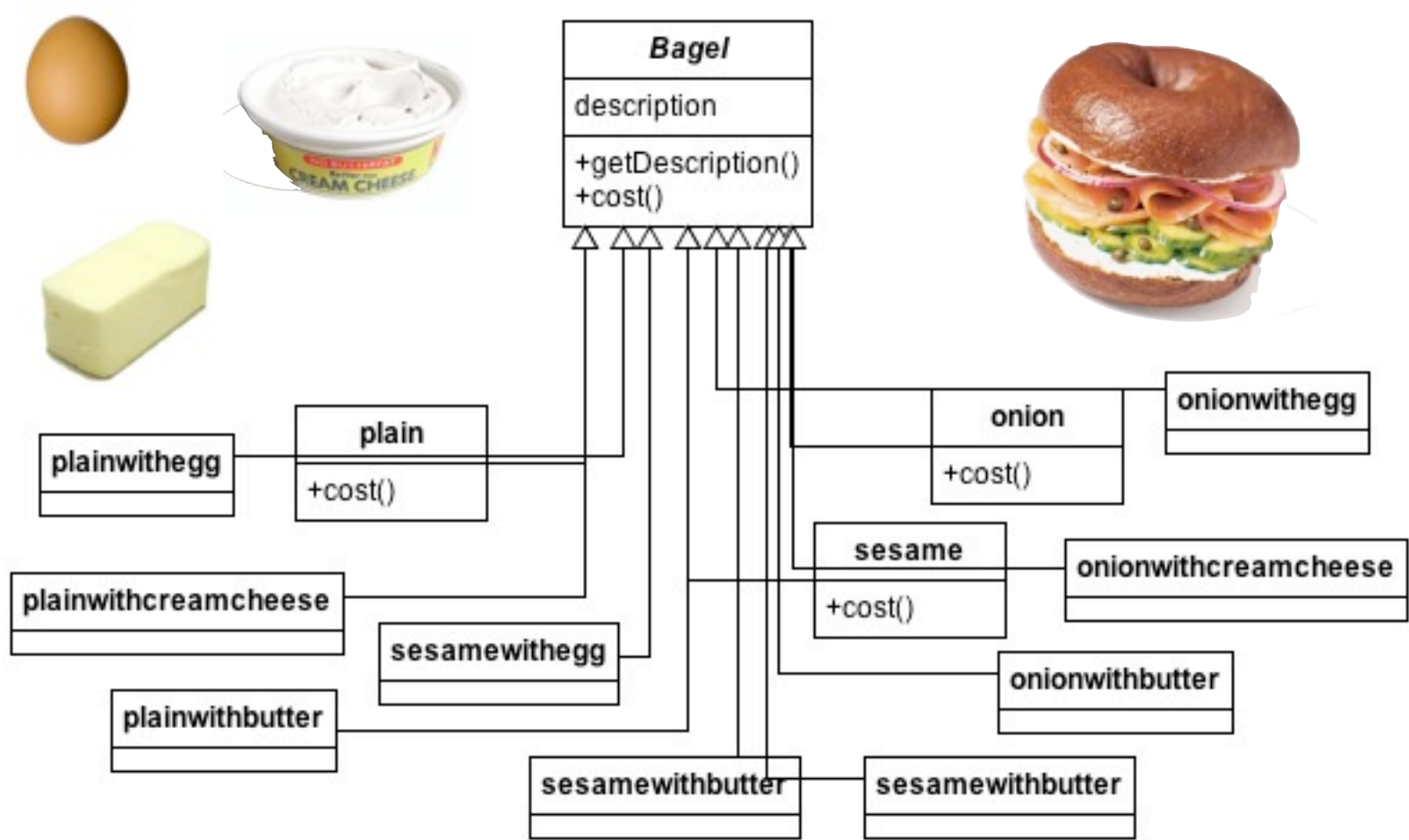
Problem

“Overuse of inheritance often leads to an explosion of classes”

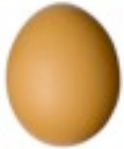
Example: Bagel Store



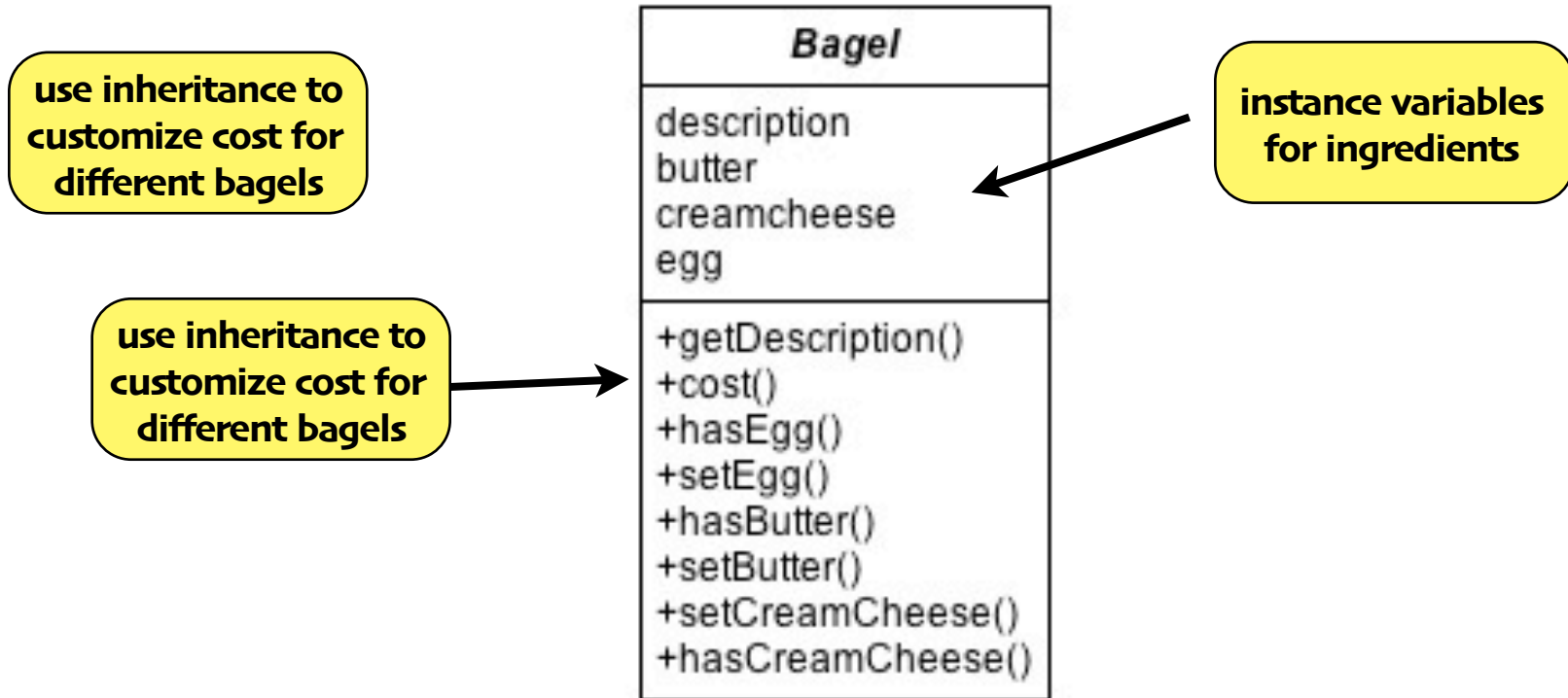
Class explosion



Class explosion



Alternative Design



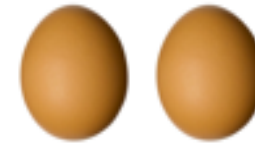
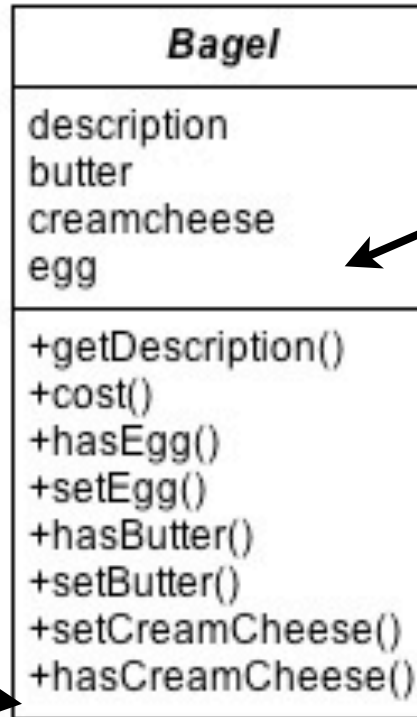
Disadvantages



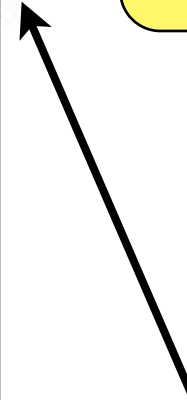
cheeze?



hascheezeO?



2 eggs?



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.

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”

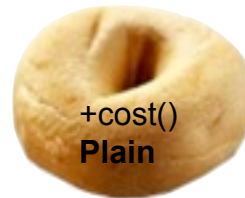
Decorator Pattern

The Decorator Pattern attaches additional responsibilities to an object dynamically.

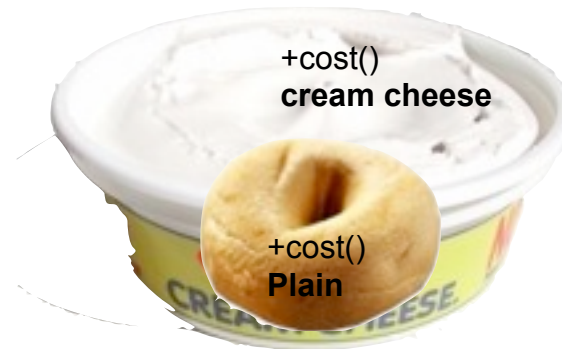
Decorators provide a flexible alternative to subclassing for extending functionality

“Wrap” bagels with decorators

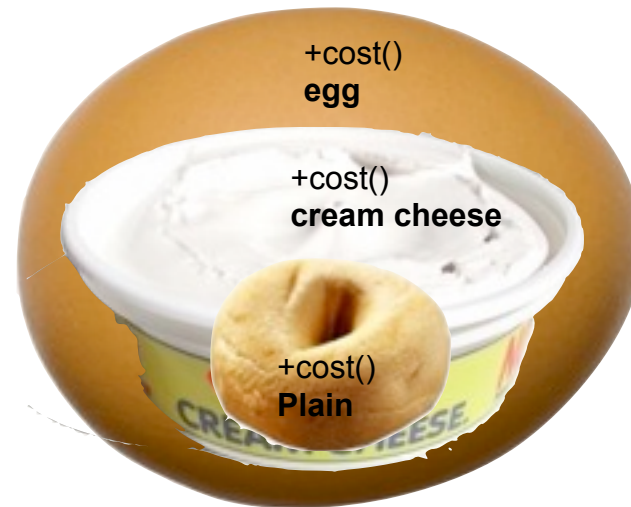
"Wrap" bagels with decorators



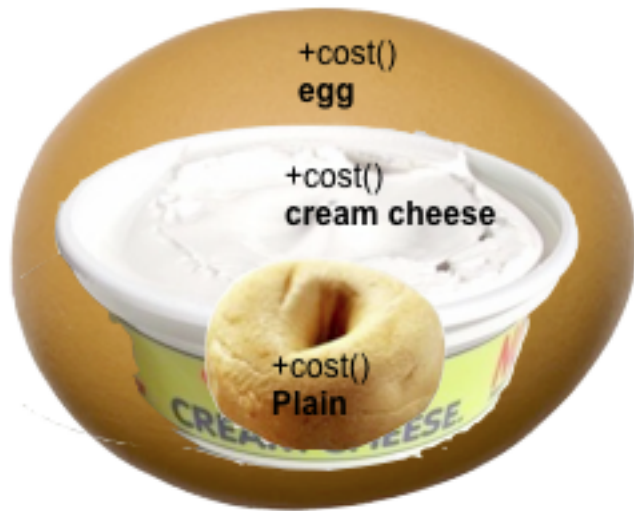
"Wrap" bagels with decorators



"Wrap" bagels with decorators



Delegation to access behavior

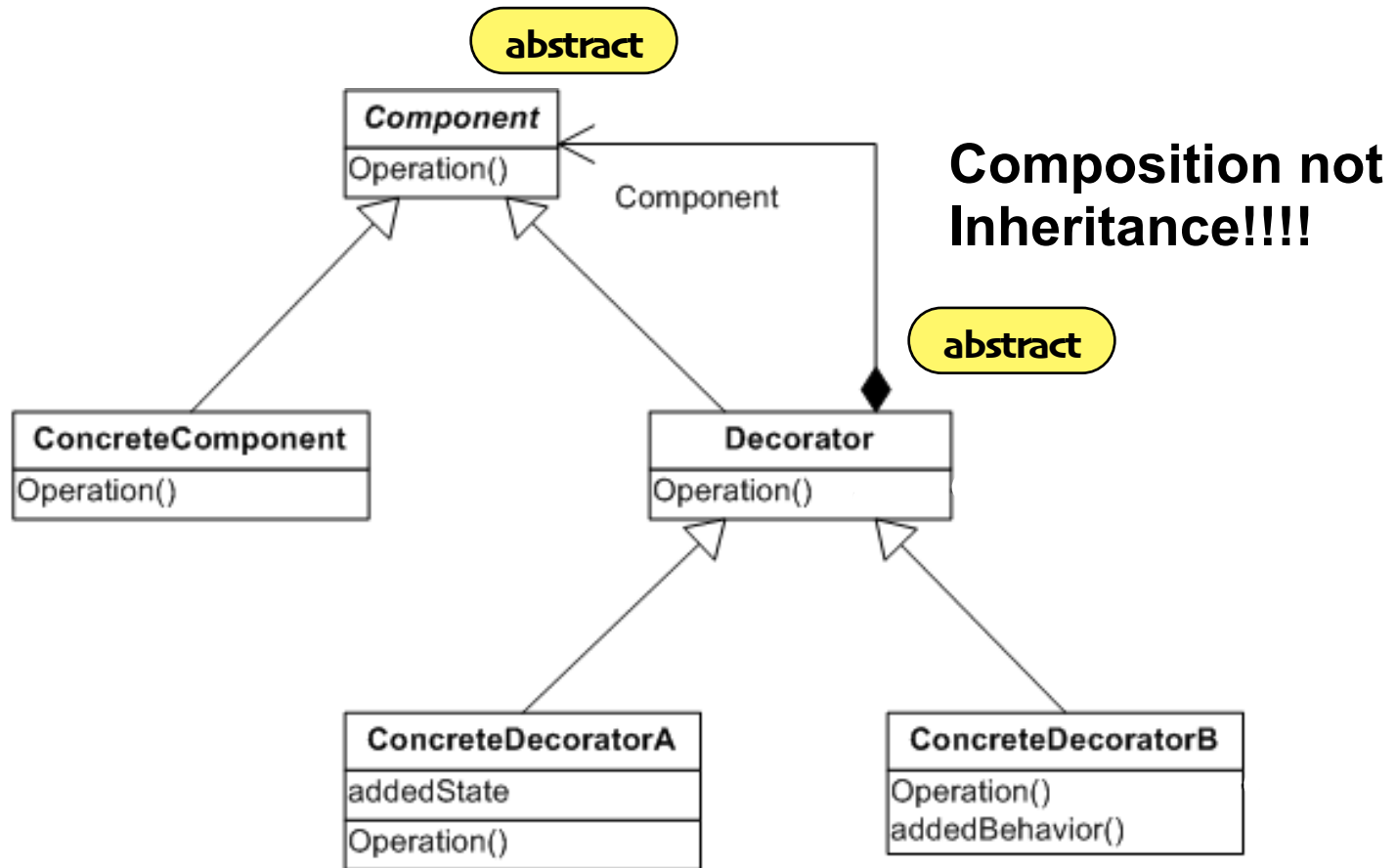


1. egg.cost()
2. egg calls CC.cost()
3. CC calls plain.cost()
4. plain returns \$plain
5. CC returns \$plain + \$cc
6. Egg returns \$egg + (\$plain + \$cc)

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 its own behavior **before** or **after** delegating to the object it decorates
- ★ Objects can be decorated dynamically at **runtime**.

Class diagram



Decorator Class

```
public class ConcreteDecoratorA extends Decorator {
```

```
    Component component;
```

instance variable holding a pointer to a component

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

set this in constructor

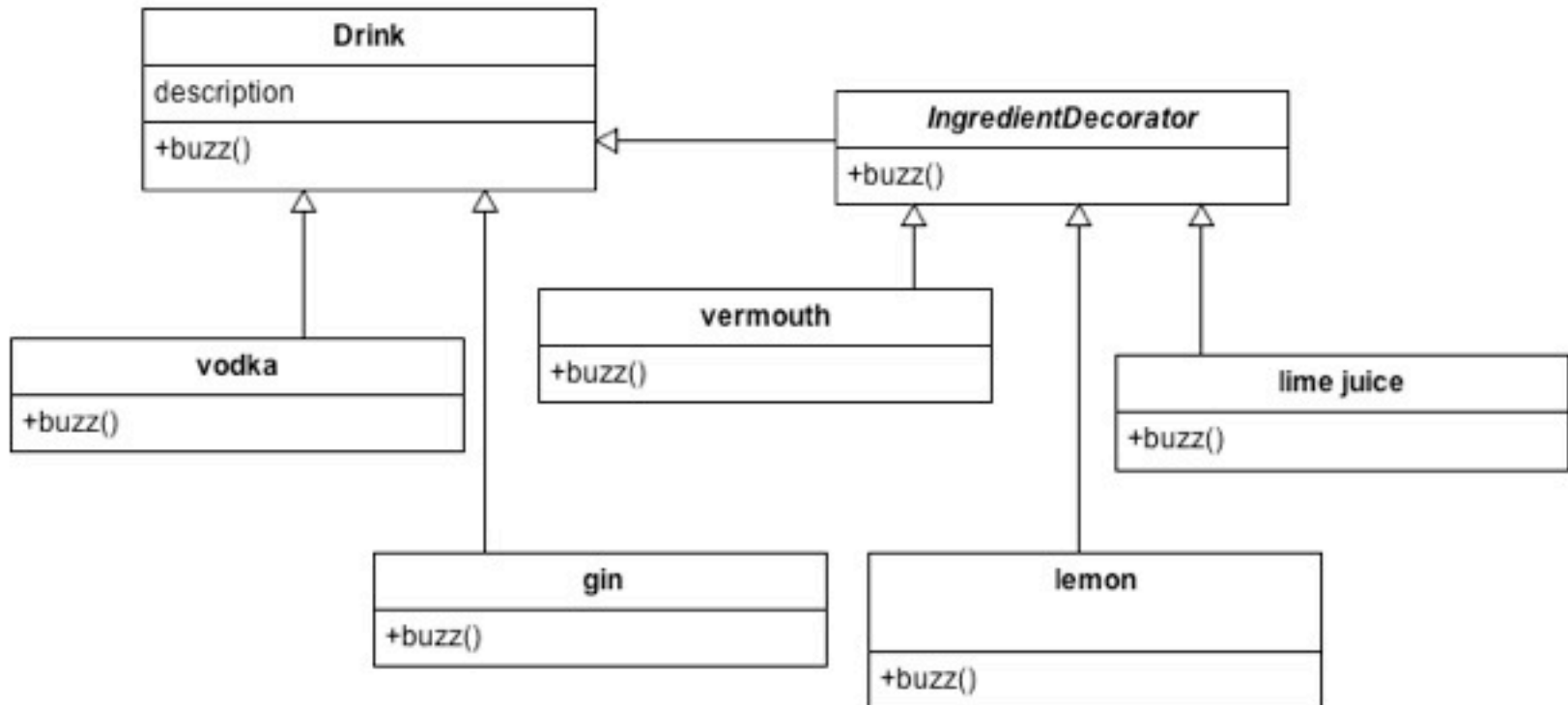
```
    public double methodA() {  
        return 3 + component.methodA();  
        //return component.methodA() + 3  
    }  
}
```

add behavior before or after

Exercise 1



Class Diagram



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();
    }
}
```

Exercise 2



**Human Computer Interaction Research
University of Nevada, Reno**