

Шаблоны проектирования

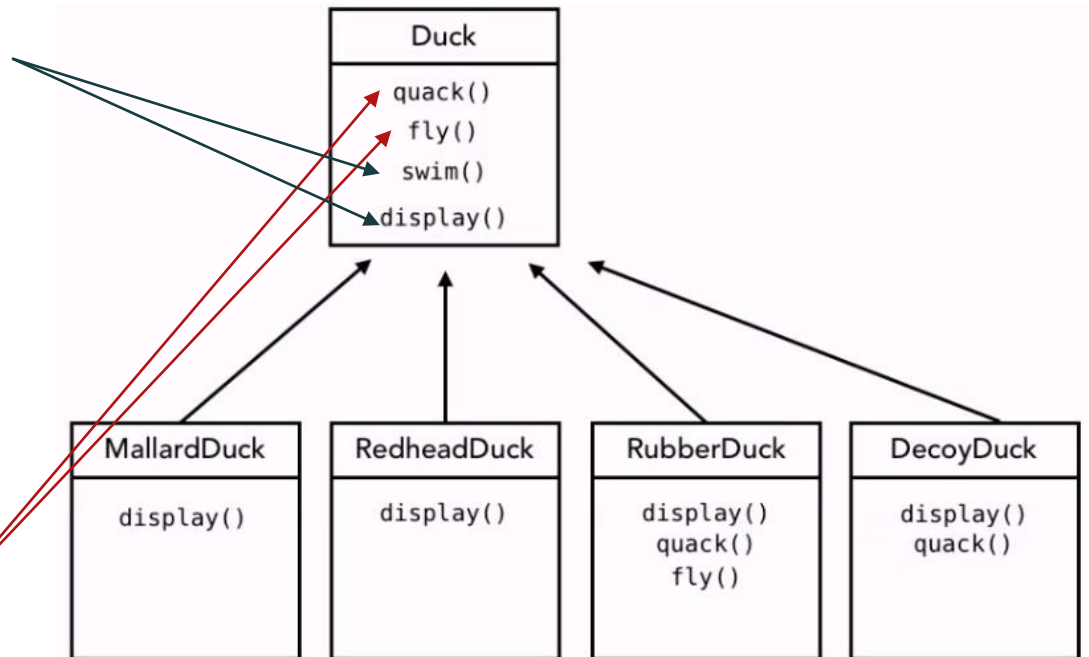
ГЛУЩЕНКО АЛЁНА

Что рассмотрим:

- ▶ Паттерны:
 - ▶ Стратегия
 - ▶ Наблюдатель
 - ▶ Декоратор
- ▶ Реализация:
 - ▶ java

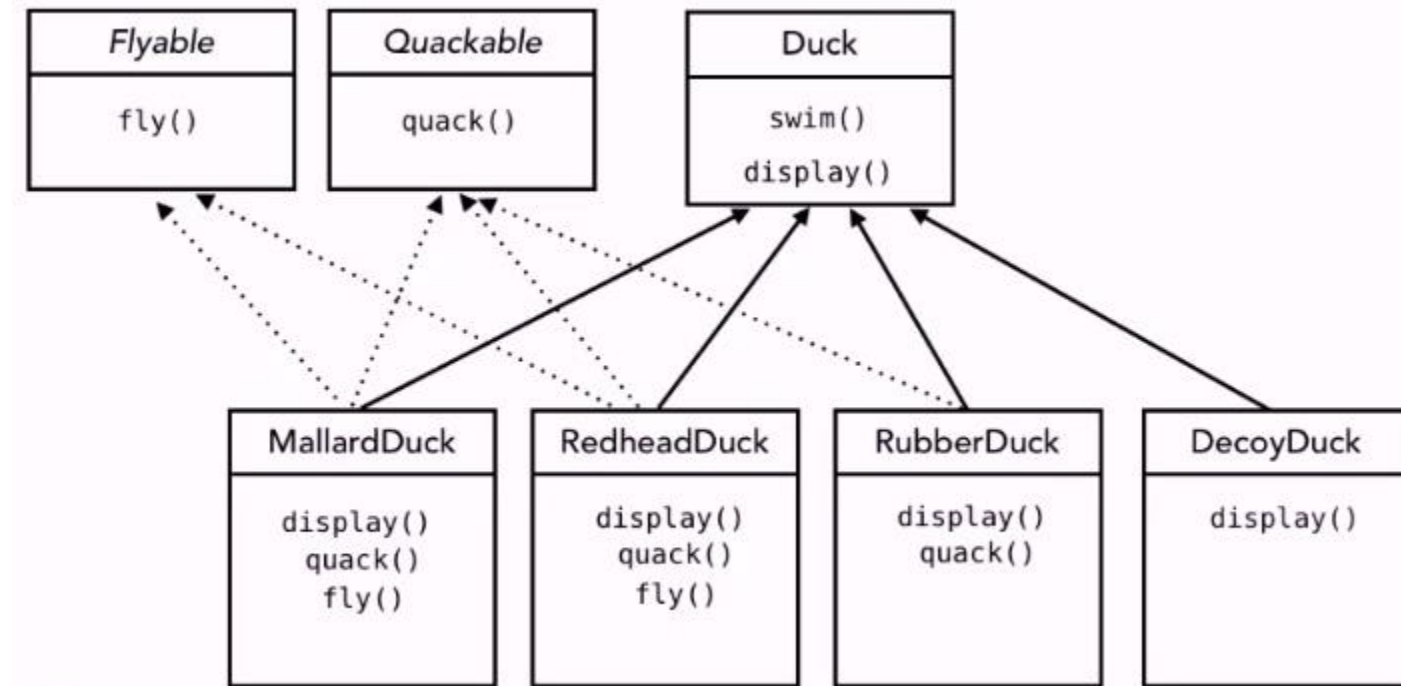
Стратегия. Проблемы наследования.

- ▶ Проблема используя наследование:
 - ▶ Жесткие структуры которые сложно поддаются изменениям
 - ▶ дублируется код, даже ненужный
 - ▶ Суперкласс не дает представления о всех классах
 - ▶ Изменение в классе может привести к проблемам в части уже созданных наследуемых классов



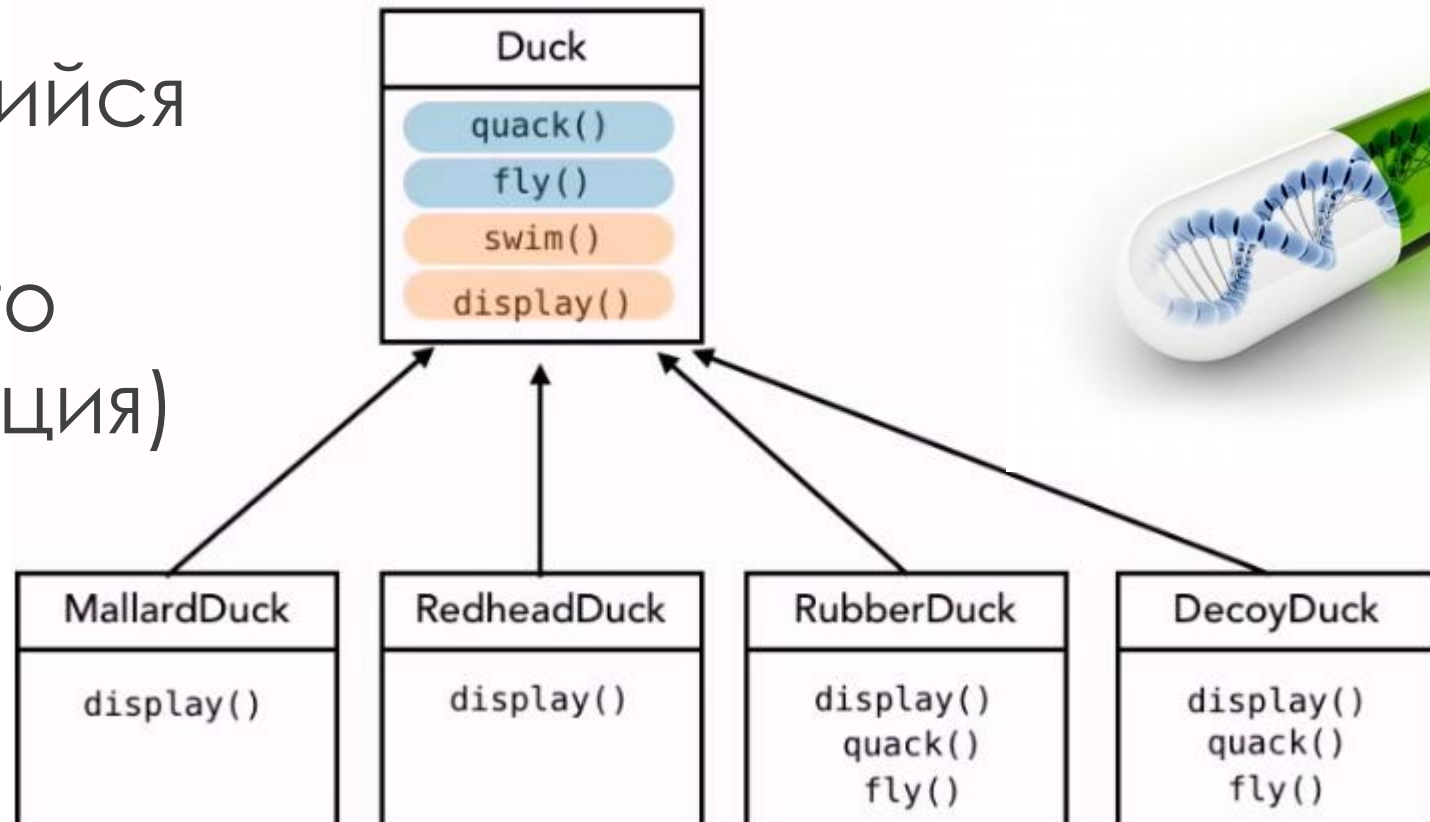
Стратегия и Интерфейсы.

- ▶ **Позволяют разделять общие черты для разных классов**
- ▶ **Но не обязательно наследовать их все для разных наследников**
- ▶ Но это тоже мешает использовать код повторно если у нас 10ки классов и все их надо как-то наследовать и определять..
- ▶ А если бы пришлось изменить интерфейсы? Вручную. Переписать. В каждом реализуемом классе!



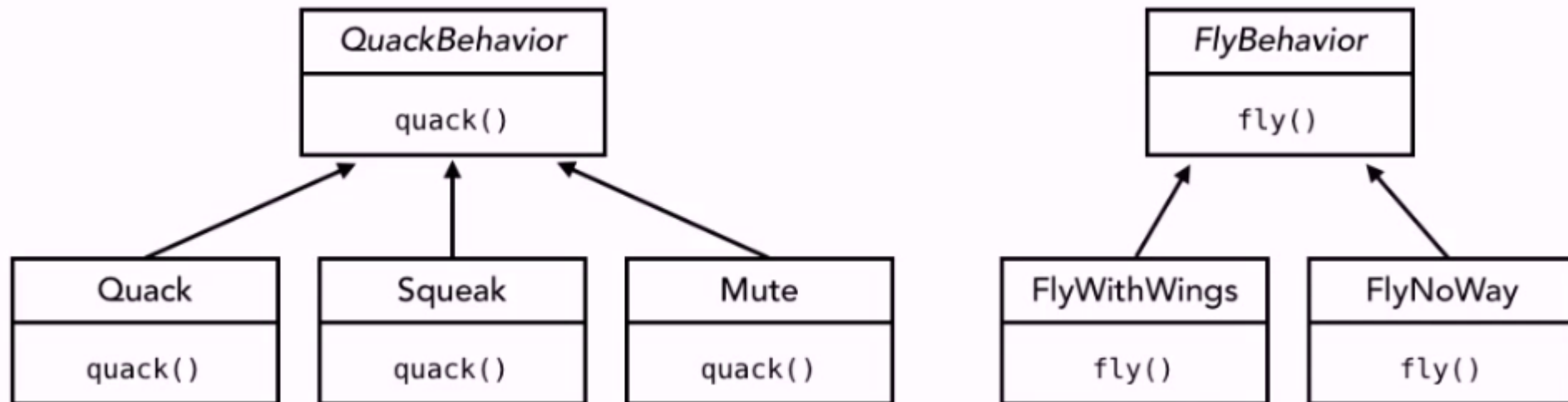
Стратегия. Правильный подход.

- ▶ Отделить изменяющийся код от постоянного (инкапсуляция)



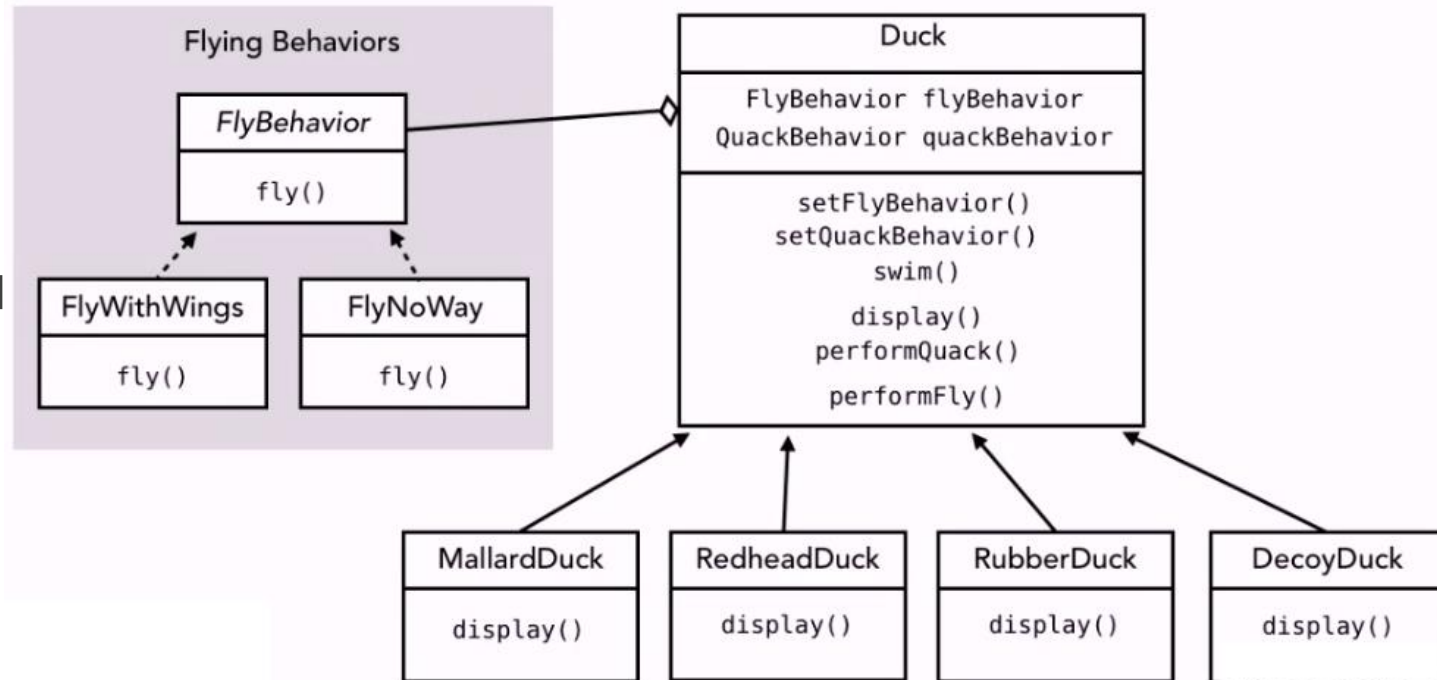
Стратегия. Правильный подход.

- Програмируем в интерфейс, а не в реализацию: то есть выносим изменяющиеся части(формы поведения) в соответствующие интерфейсы, а не реализуем их непосредственно в классе. Таким образом вариации данного поведения могут быть представлены подклассами(конкретными формами) такого интерфейса



Стратегия.

- ▶ Что это дает:
 - ▶ Изменение объекта на этапе выполнения
 - ▶ Мы не привязаны к реализации. Суперкласс ДАЕТ полную информацию о своих подклассах



Реализация

```
public abstract class Duck {
    FlyBehavior flyBehavior;
    QuackBehavior quackBehavior;

    public Duck() {
    }

    public void setFlyBehavior(FlyBehavior fb) {
        flyBehavior = fb;
    }

    public void setQuackBehavior(QuackBehavior qb) {
        quackBehavior = qb;
    }

    abstract void display();

    public void performFly() {
        flyBehavior.fly();
    }
}
```

```
public class MallardDuck extends Duck {

    public MallardDuck() {

        quackBehavior = new Quack();
        flyBehavior = new FlyWithWings();

    }

    public void display() {
        System.out.println("I'm a real Mallard duck");
    }
}
```

```
public class RubberDuck extends Duck {

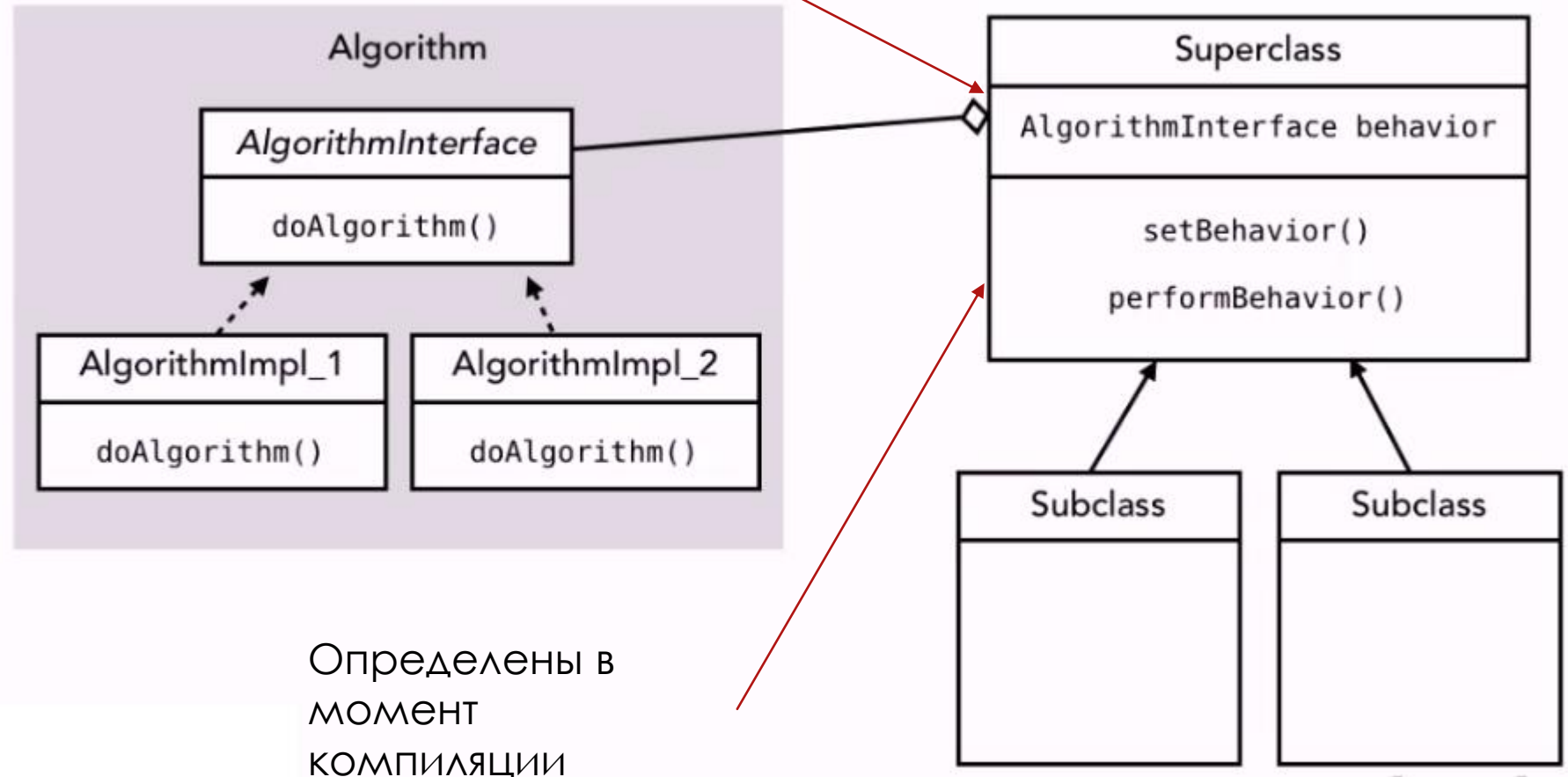
    public RubberDuck() {
        flyBehavior = new FlyNoWay();
        quackBehavior = new Squeak();
    }

    public void display() {
        System.out.println("I'm a rubber duckie");
    }
}
```

Стратегия. Общая диаграмма классов

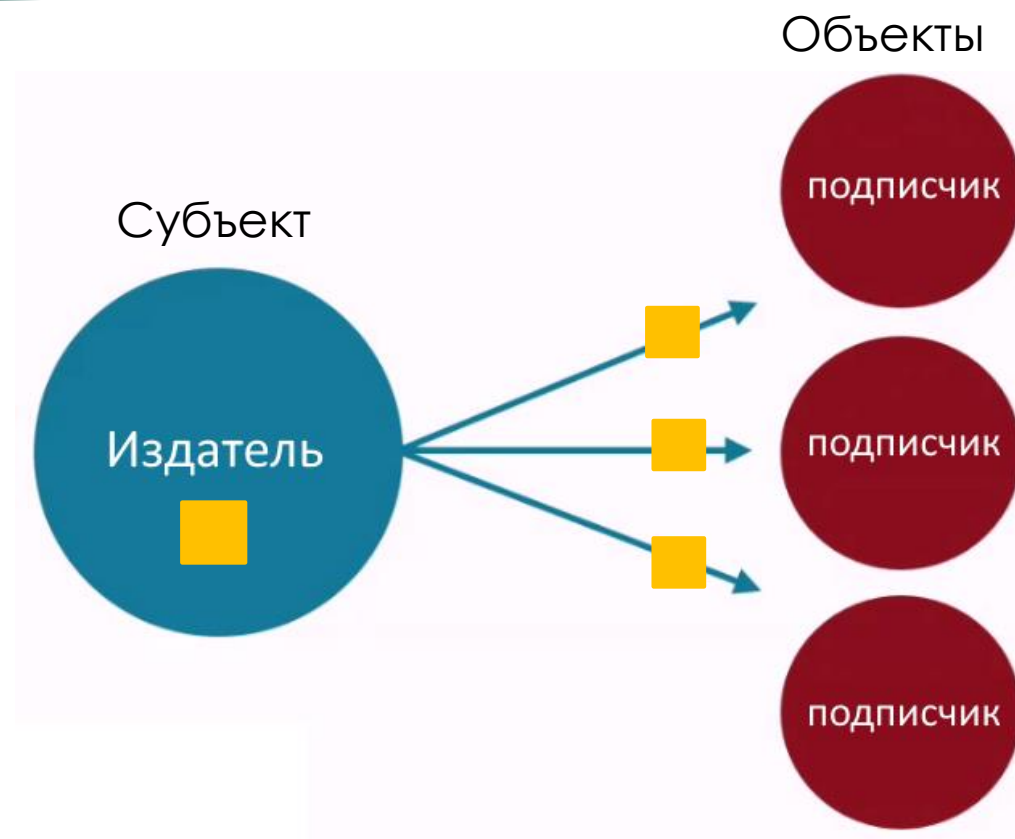
Изменяемы со временем

- ▶ Иерархия наследования через «является» - неизменяемые общие моменты
- ▶ Созданные объекты и формы поведения в стратегии связаны через «содержат»



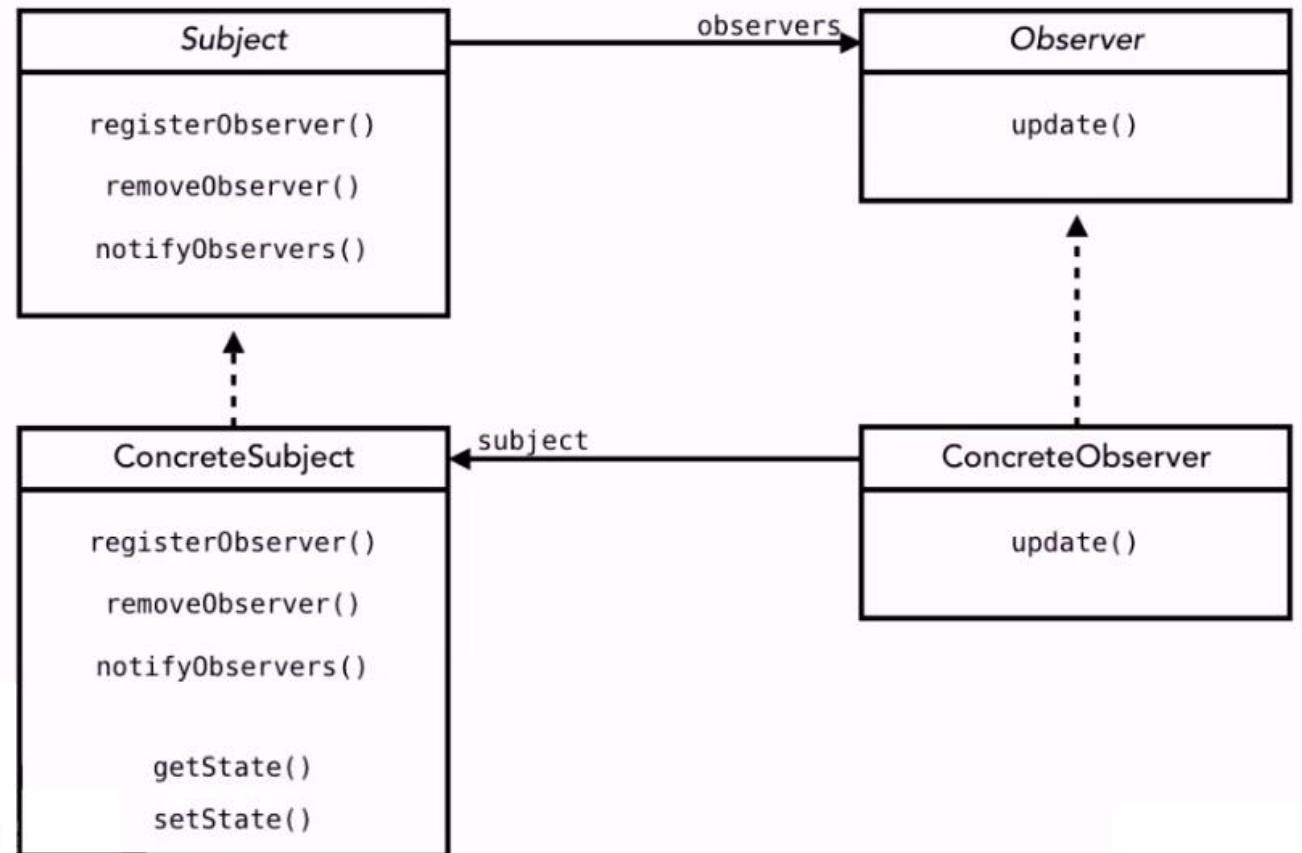
Наблюдатель.

- ▶ Например: подписка на журнал
 - ▶ Пока вы подписаны и газета публикуется – вы получаете газету
 - ▶ Подписчиком может быть кто-угодно
 - ▶ Подписчики могут хранить те данные, которые им интересны и при их изменении они уведомляются
- ▶ Зависимость один ко многим
- ▶ Издатель – единственный владелец данных

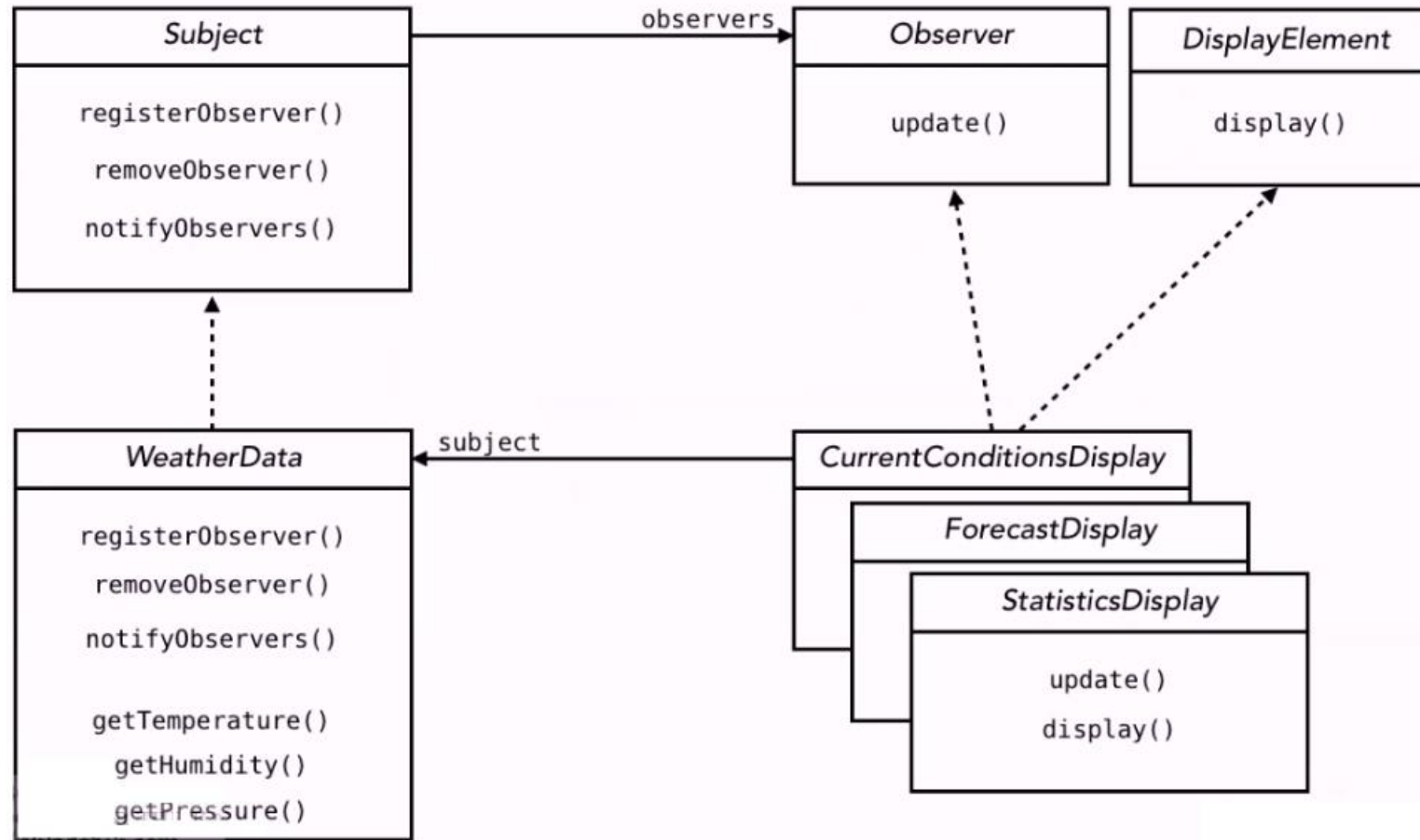


Наблюдатель. Схема.

- ▶ notifyObservers – вызывает метод update у наблюдателей; вызывается при изменении данных; реализован в ConcreteSubject
- ▶ Subject вызывает метод update
- ▶ В java реализован данный паттерн, но мы все равно напишем свой.



Наблюдатель. Задача о метеостанции.



Наблюдатель. Реализация.

```
public interface Subject {  
    public void registerObserver(Observer o);  
    public void removeObserver(Observer o);  
    public void notifyObservers();  
}
```

```
public interface Observer {  
    public void update(float temp, float humidity, float pressure);  
}
```

```
public class WeatherData implements Subject {  
    private ArrayList<Observer> observers;  
    private float temperature;  
    private float humidity;  
    private float pressure;  
  
    public WeatherData() {  
        observers = new ArrayList<Observer>();  
    }  
  
    public void registerObserver(Observer o) {  
        observers.add(o);  
    }  
  
    public void removeObserver(Observer o) {  
        int i = observers.indexOf(o);  
        if (i >= 0) {  
            observers.remove(i);  
        }  
    }  
  
    public void notifyObservers() {  
        for (int i = 0; i < observers.size(); i++) {  
            Observer observer = (Observer)observers.get(i);  
            observer.update(temperature, humidity, pressure);  
        }  
    }  
  
    public void notifyObservers() {  
        for (int i = 0; i < observers.size(); i++) {  
            Observer observer = (Observer)observers.get(i);  
            observer.update(temperature, humidity, pressure);  
        }  
    }  
  
    public void measurementsChanged() {  
        notifyObservers();  
    }  
  
    public void setMeasurements(float temperature, float humidity, float pressure) {  
        this.temperature = temperature;  
        this.humidity = humidity;  
        this.pressure = pressure;  
        measurementsChanged();  
    }  
}
```

```
public class CurrentConditionsDisplay implements Observer, DisplayElement {  
    private float temperature;  
    private float humidity;  
    private Subject weatherData;  
  
    public CurrentConditionsDisplay(Subject weatherData) {  
        this.weatherData = weatherData;  
        weatherData.registerObserver(this);  
    }  
  
    public void update(float temperature, float humidity, float pressure) {  
        this.temperature = temperature;  
        this.humidity = humidity;  
        display();  
    }  
  
    public void display() {  
        System.out.println("Current conditions: " + temperature  
            + "F degrees and " + humidity + "% humidity");  
    }  
}
```

Декоратор.

- ▶ Пример: делаем кофе
- ▶ Добавки изменяют стоимость
- ▶ Есть разные сорта кофе
- ▶ Но реализовывать все возможные сочетания ...

DRINKS (HOT & ICED)	12oz	16oz
Brewed Coffee	\$0.69	\$0.89
Cafe au lait	\$1.79	\$2.19
Caffe Latte	\$1.79	\$2.19
Caffe Mocha	\$2.05	\$2.39
Caramel Macchiato	\$1.79	\$2.39
Chai Latte	\$1.99	\$2.39
Hot Chocolate	\$1.59	\$1.85
Espresso		\$0.89
Flavor Shot		\$0.30
Reusable Mug Discount		\$0.15
Tea		\$0.39

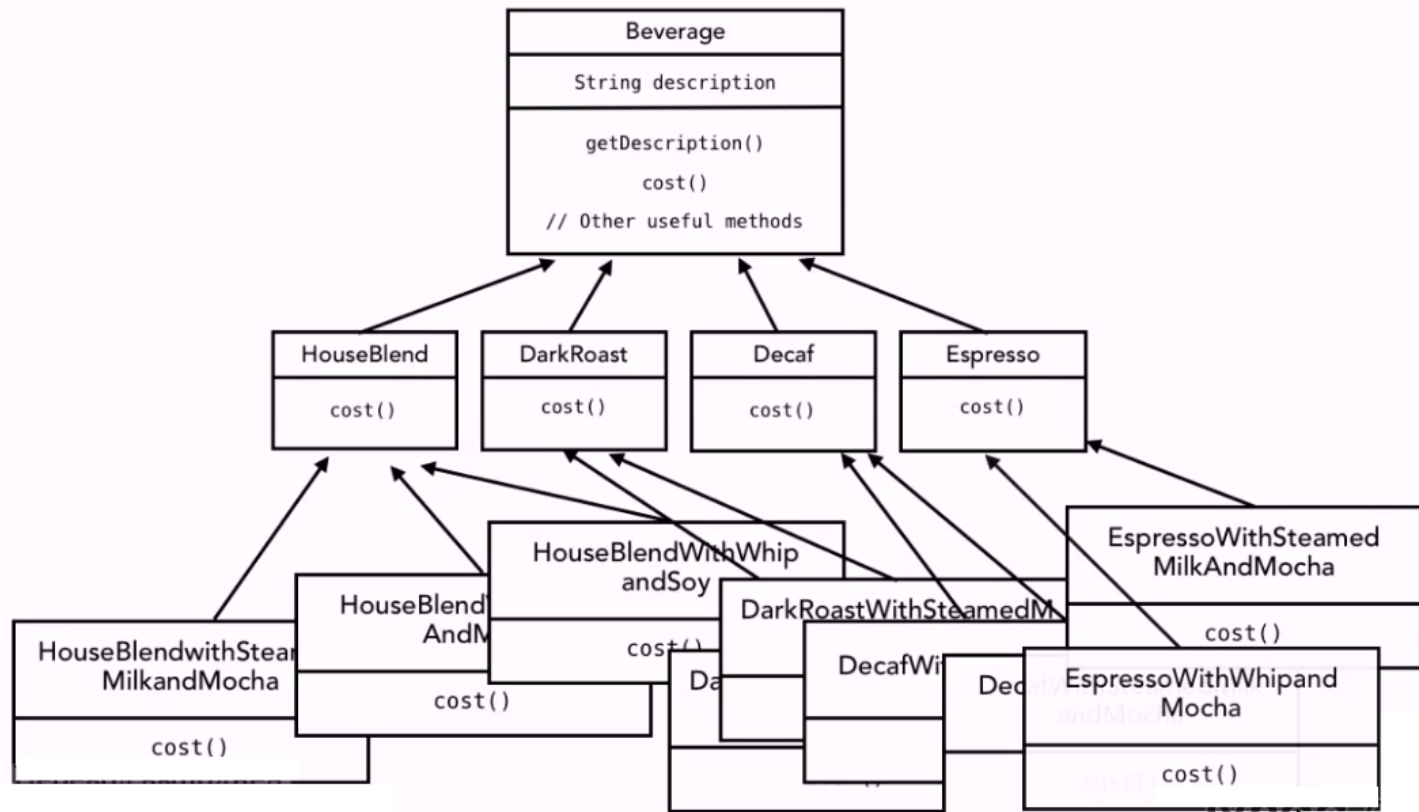


at Gordon Avenue Market



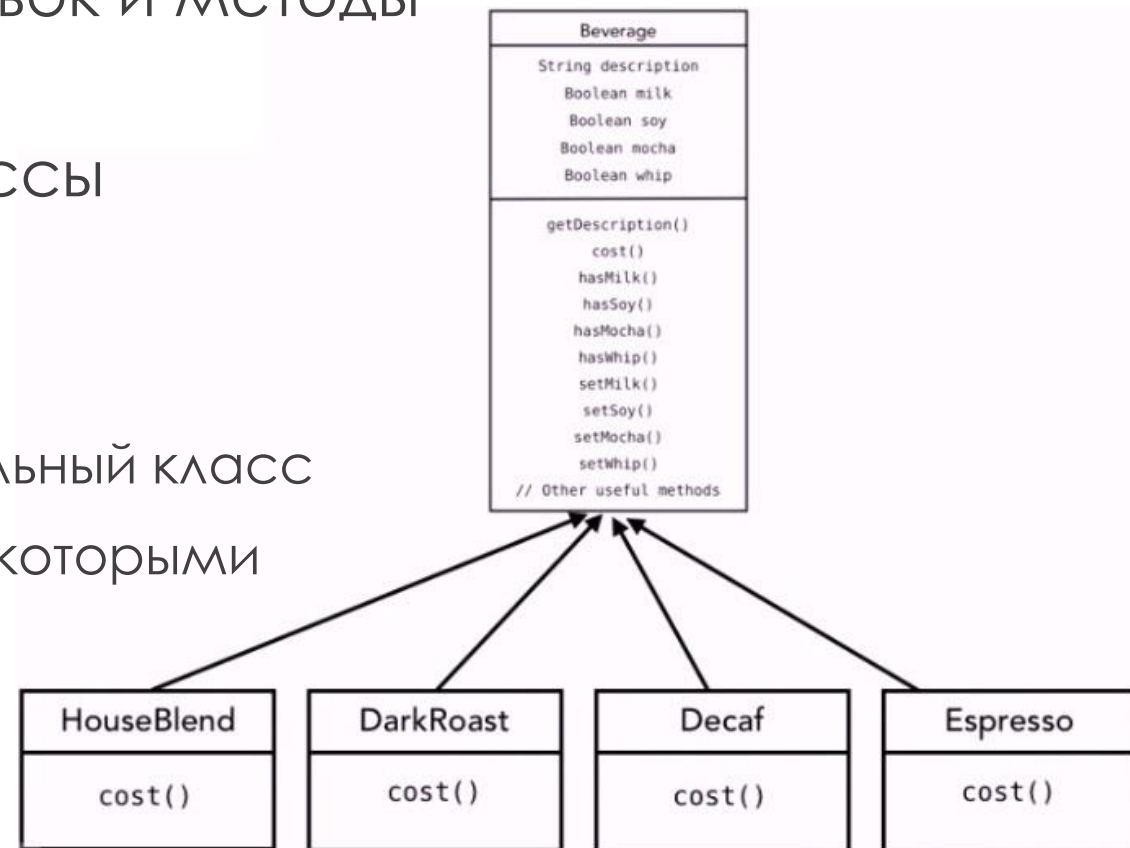
Декоратор.

- ▶ Очень не кошерно
- ▶ А вообще декоратор используется в java, к примеру, для работы с файлами



Декоратор. Введение дополнительных полей в суперкласс.

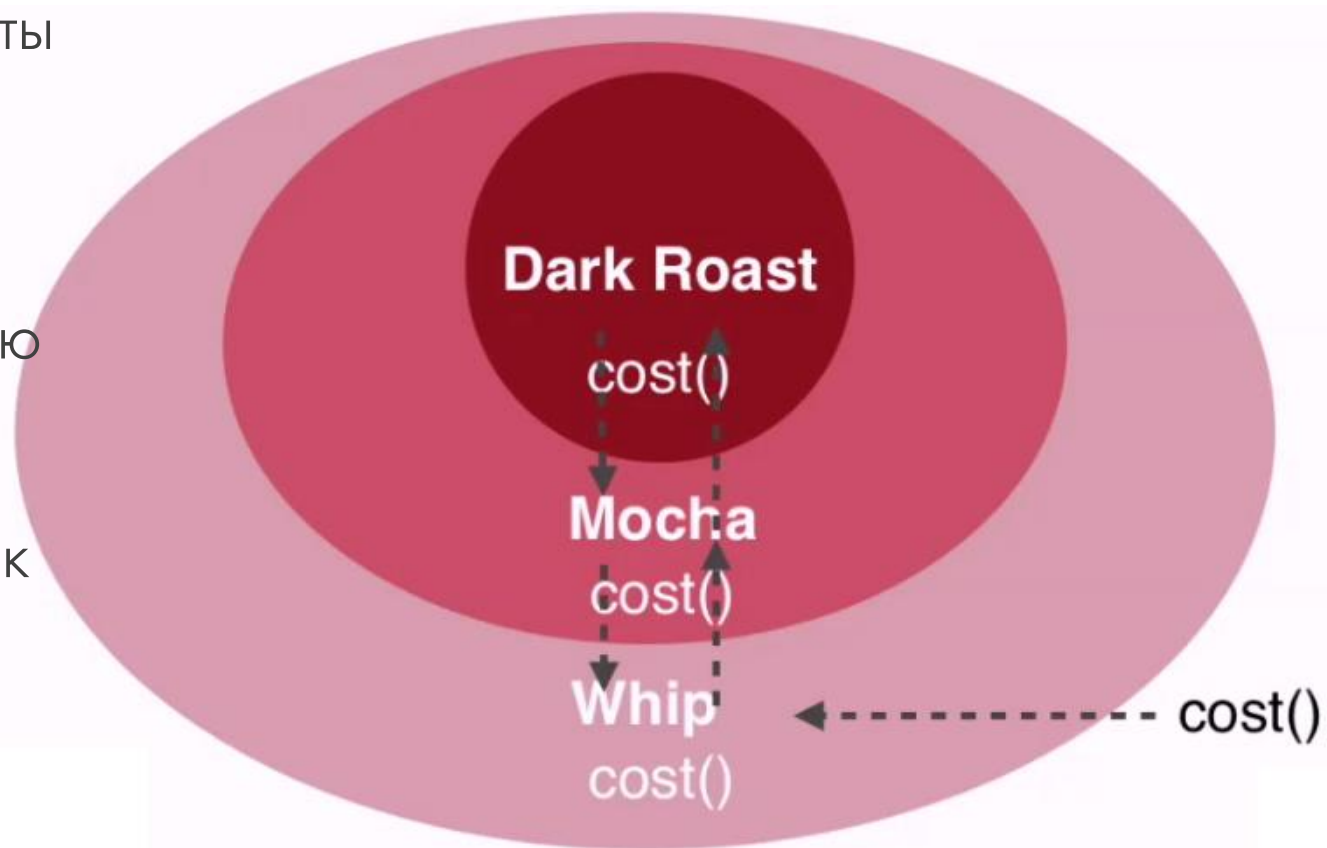
- ▶ Ввести поля в суперклассе для добавок и методы set
- ▶ Для видов кофе используем подклассы
- ▶ Минусы:
 - ▶ Довольно хлопотно считать цену
 - ▶ Изменение цены влияет на каждый отдельный класс
 - ▶ Некоторые добавки нельзя сочетать с некоторыми видами кофе
 - ▶ изменение меню добавок через изменение суперкласса



Декоратор.

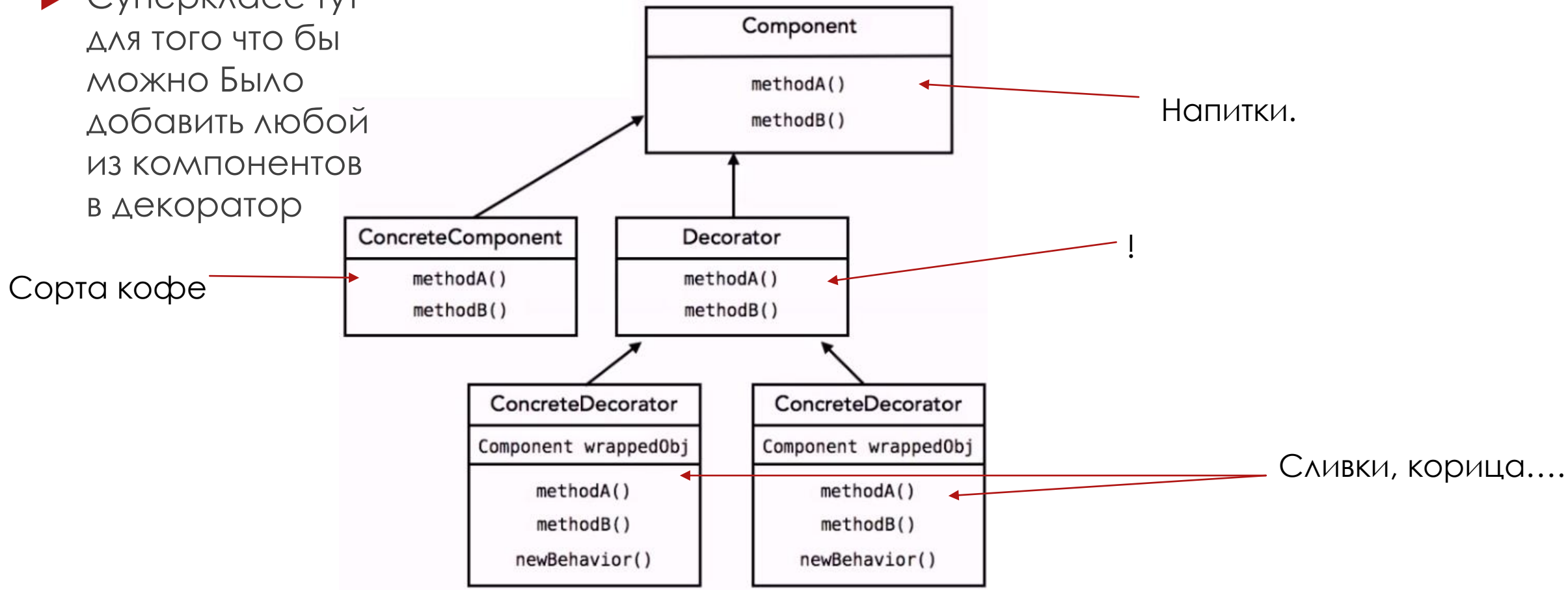
- ▶ Принцип: классы должны быть открыты для расширения, но закрыты для изменения
- ▶ Вместо классов будем создавать объекты и объединять их
- ▶ Каждый метод `cost` отвечает за свою часть цены, а общая стоимость вычисляется через делегирование
- ▶ Шаблон динамически подключает к объекту дополнительные обязанности

Черный мокко со сливками



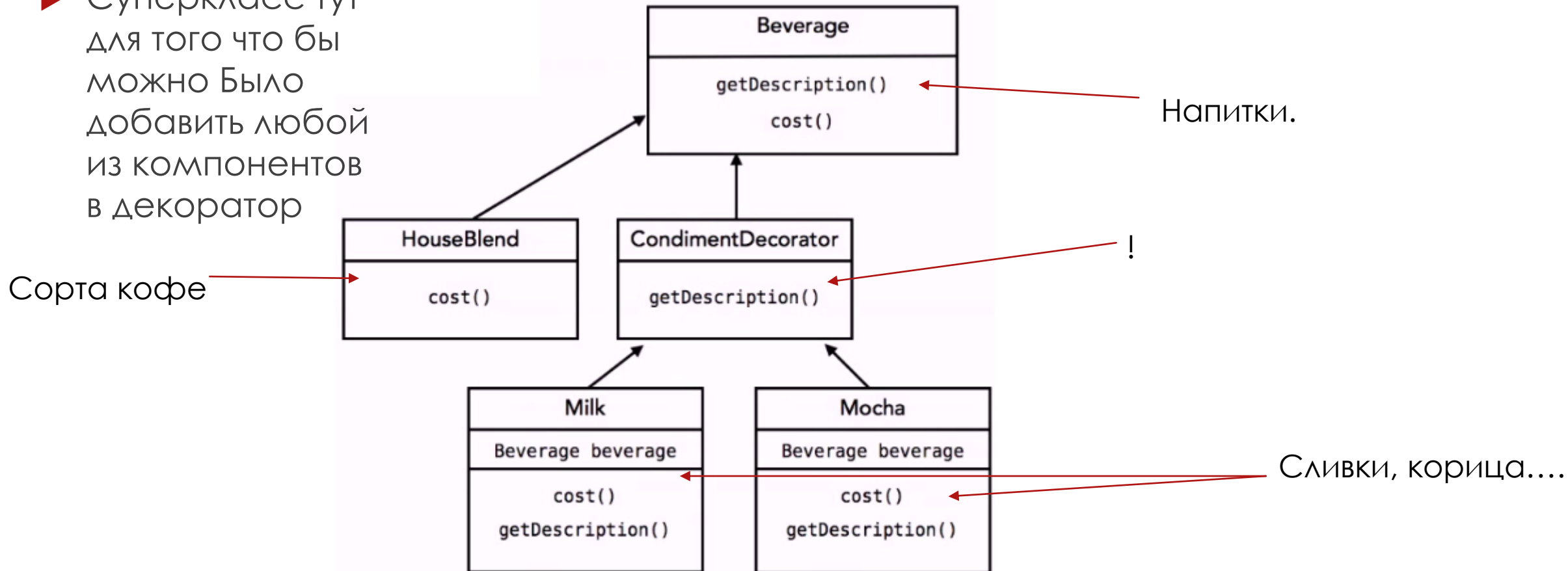
Декоратор. Схема.

- Суперкласс тут для того что бы можно было добавить любой из компонентов в декоратор



Декоратор. Схема.

- Суперкласс тут для того что бы можно было добавить любой из компонентов в декоратор



Декоратор. Реализация.

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

```
public class Decaf extends Beverage {  
    public Decaf() {  
        description = "Decaf Coffee";  
    }  
  
    public double cost() {  
        return 1.05;  
    }  
}
```

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

```
public class Milk extends CondimentDecorator {  
    Beverage beverage;  
  
    public Milk(Beverage beverage) {  
        this.beverage = beverage;  
    }  
  
    public String getDescription() {  
        return beverage.getDescription() + ", Milk";  
    }  
  
    public double cost() {  
        return .10 + beverage.cost();  
    }  
}
```

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

Dark Roast

cost()

Mocha

cost()

Whip

cost()