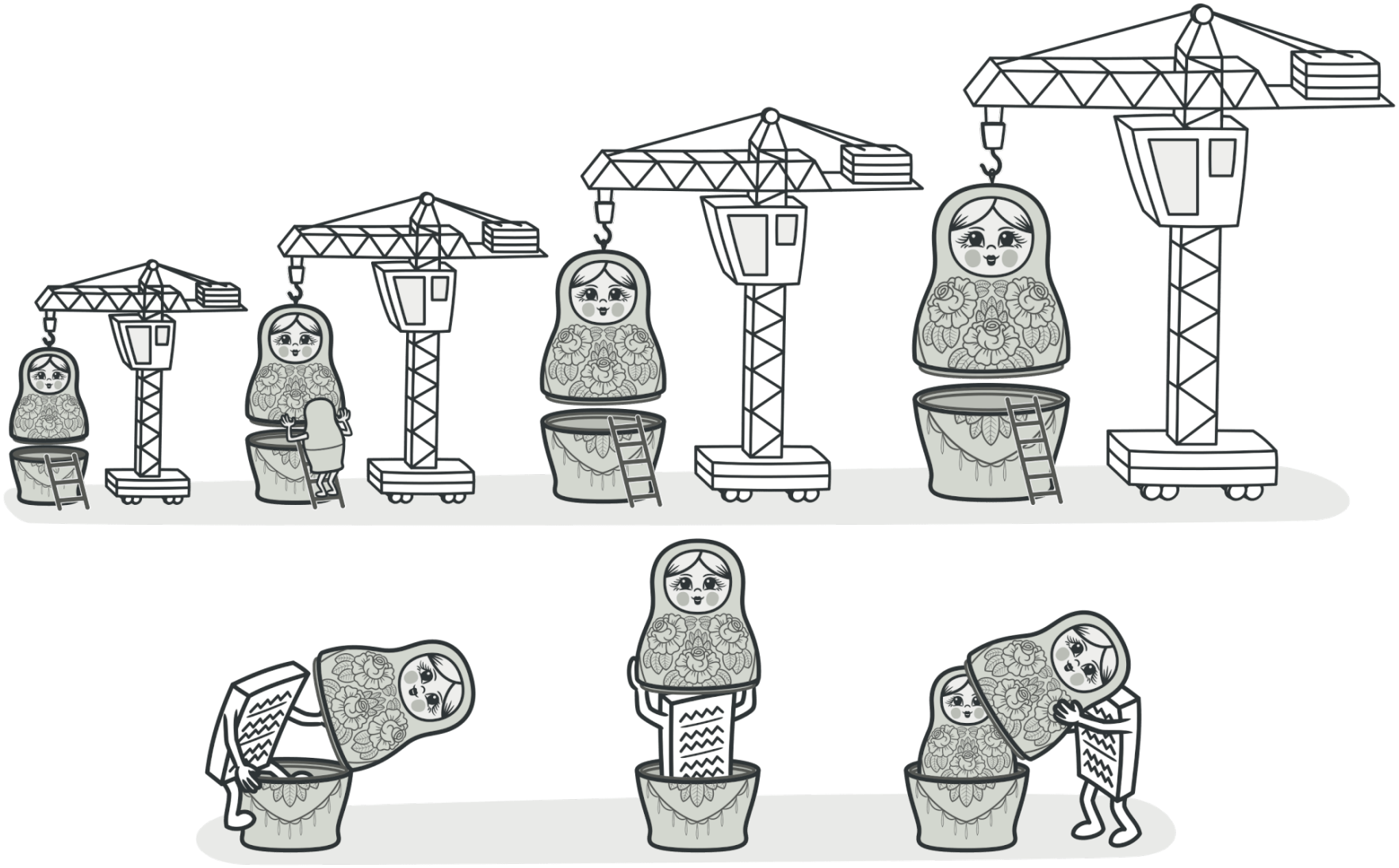
Декоратор



Кейс: Кофейня

Кейс: Кофейня

Кейс: Кофейня

Как добавить к напиткам различные виды дополнений?



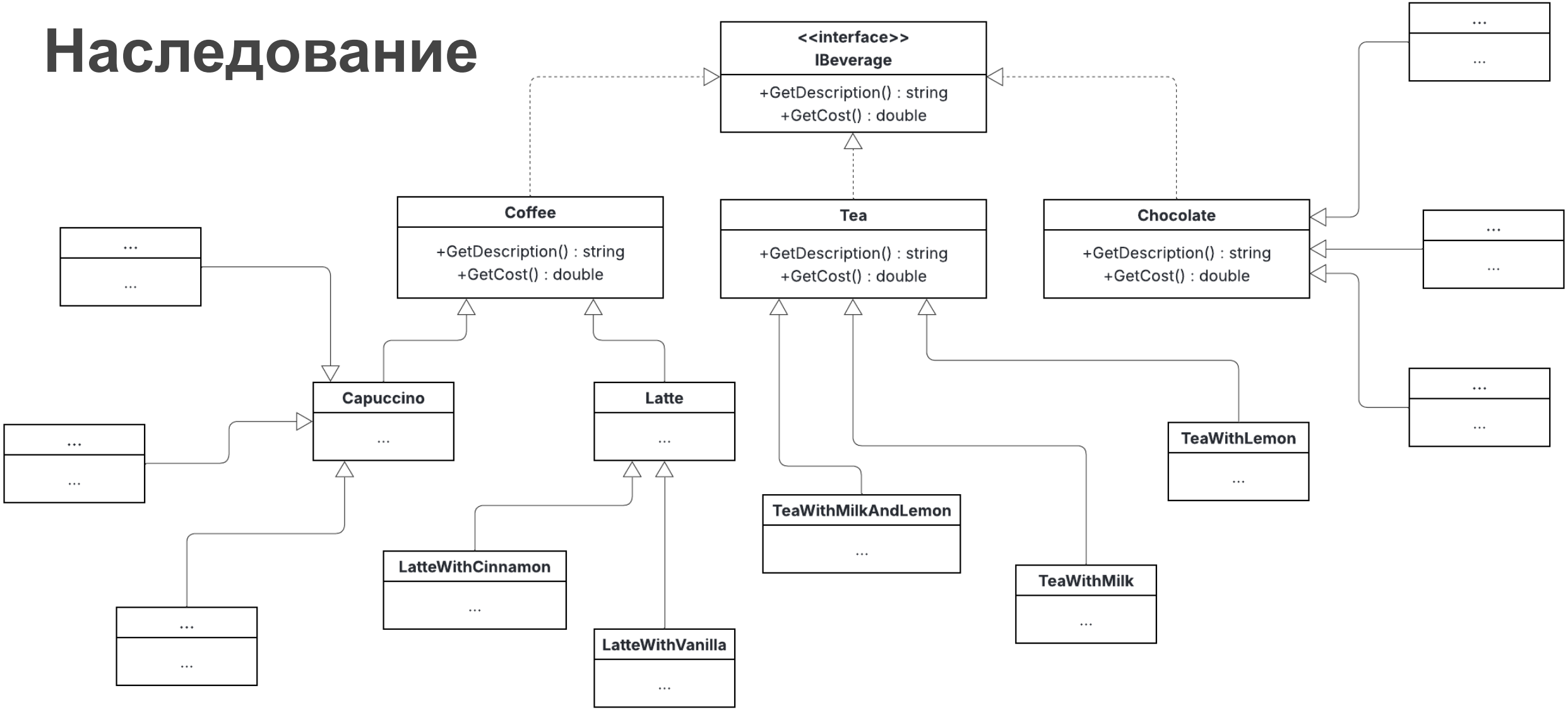







Решение (?)

Наследование



Решение (?)

Наследование: проблема 1

Пусть есть n вариантов дополнений
Сколько понадобится классов-наследников для конкретного напитка?

$$2^n$$

Для 10 дополнений: $2^{10} = 1024$!!!

Решение (?)

Наследование: проблема 2

Пусть нам нужно изменить одно дополнение (например, стоимость сиропа)

SOLID

↓

Сколько классов придется изменить? $\Longrightarrow$ Нарушение OCP

$$2^{n-1} = 2^{10-1} = 2^9 = 512$$

Решение (!)

~~**Наследование**~~

1) Жёсткие иерархии
2) Поведение фиксируется во время компиляции
3) Трудно изменять

Композиция

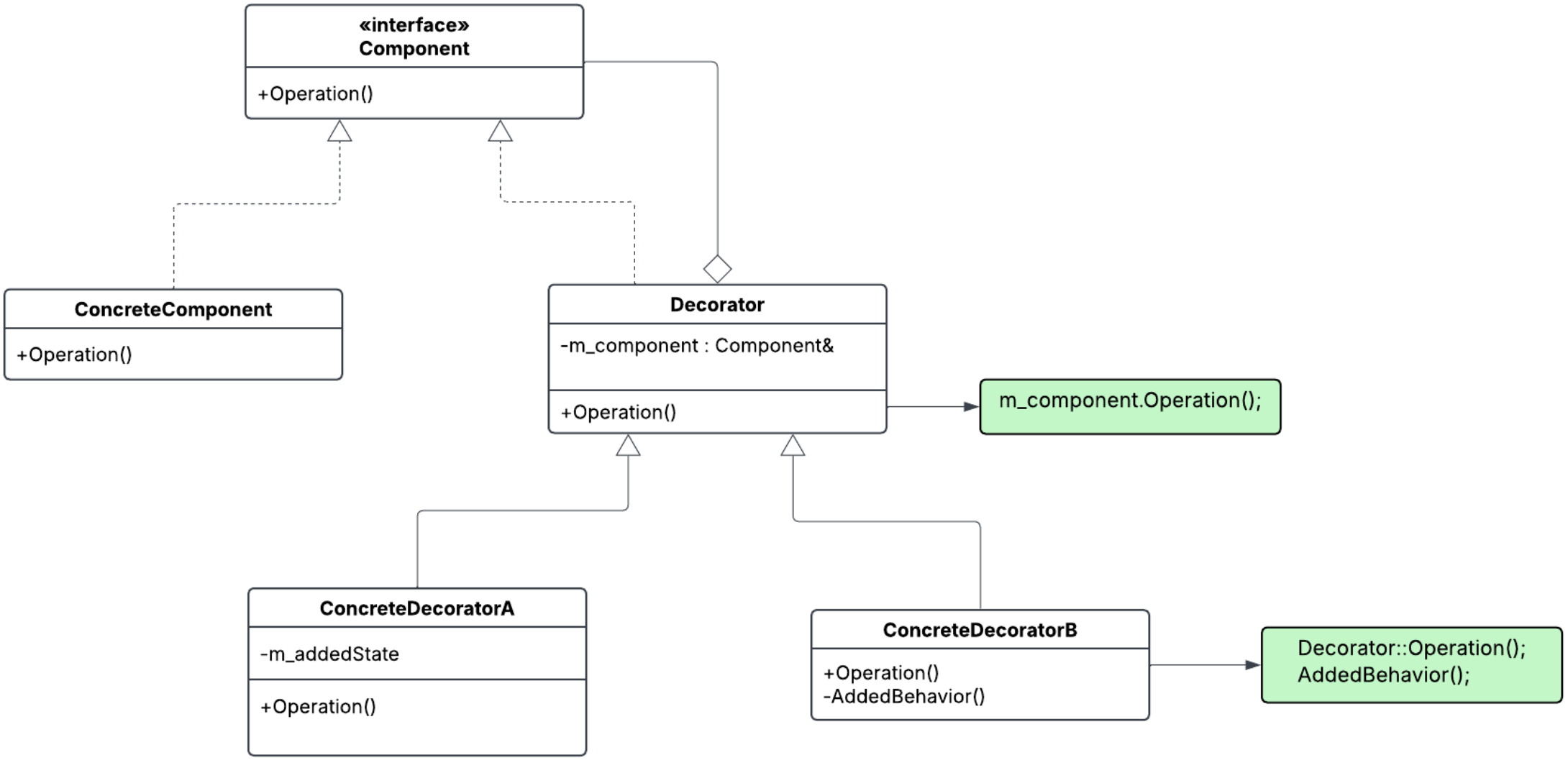
1) Объекты собираются из компонентов
2) Поведение можно менять во время исполнения программы
3) Гибкость и повторное использование кода

Паттерн «Декоратор»

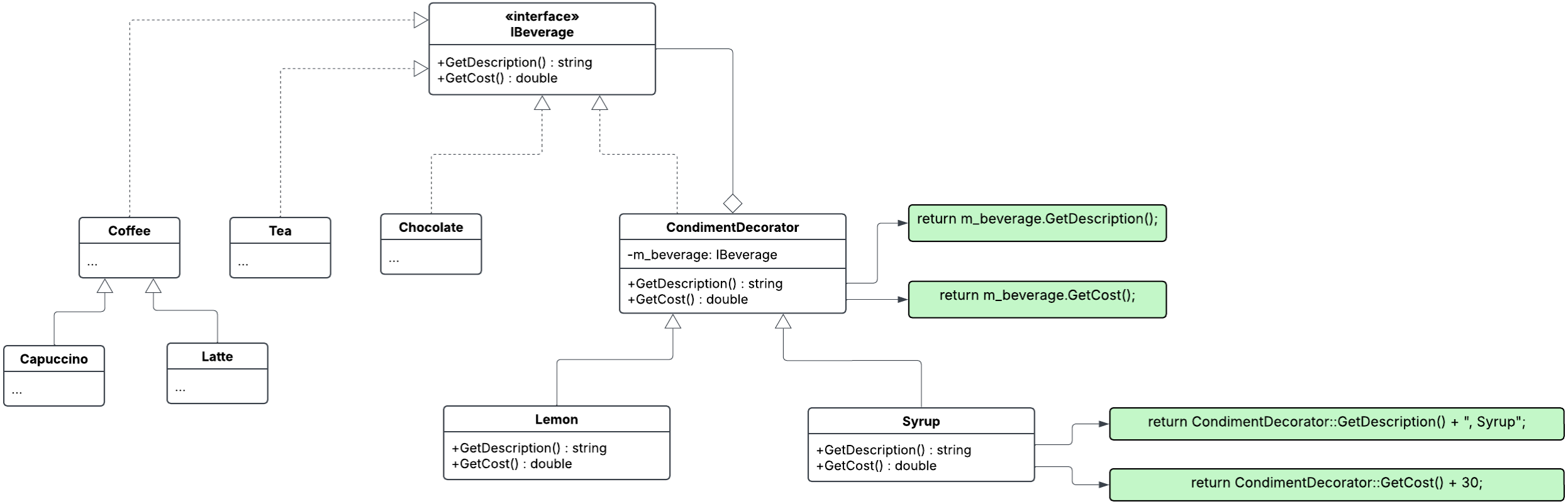
Декоратор - структурный паттерн проектирования, который позволяет динамически добавлять объектам новую функциональность

- Повторяет интерфейс декорируемого объекта
- Декорированный объект можно использовать как исходный
- Можно «оборачивать» в один или несколько декораторов

Декоратор: структура

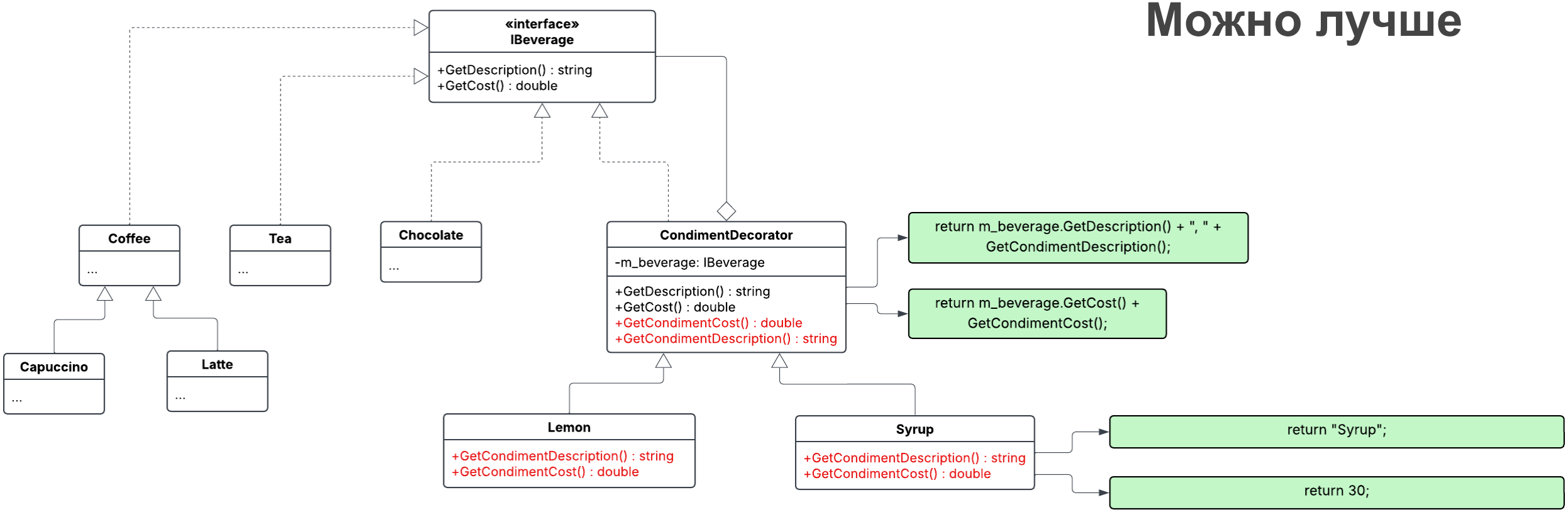


Декоратор применительно к Кофейне



Декоратор применительно к Кофейне

Можно лучше



Пример кода

Виртуальный класс «Уведомление»

```
class Notifier {
public:
    virtual ~Notifier() = default;
    virtual void Send(const std::string& message) = 0;
};
```

Базовый класс-заглушка

```
class EmptyNotifier : public Notifier {
public:
    void Send(const std::string& message) override {}
};
```

Декоратор

```
class NotifierDecorator : public Notifier {
protected:
    std::unique_ptr<Notifier> notifier;
public:
    NotifierDecorator(std::unique_ptr<Notifier> n) : notifier(std::move(n)) {}
};
```

Пример кода

Конкретный декоратор, наследуемый от базового класса Декоратор

```
class EmailDecorator : public NotifierDecorator {
public:
    EmailDecorator(std::unique_ptr<Notifier> n) : NotifierDecorator(std::move(n)) {}

    void Send(const std::string& message) override {
        notifier->Send(message);
        std::cout << "Sending email: " << message << std::endl;
    }
};
```

Пример кода

```
auto sms = std::make_unique<SMSDecorator>(std::make_unique<EmptyNotifier>());

auto email_and_telegram = std::make_unique<TelegramDecorator>(
    std::make_unique<EmailDecorator>(
        std::make_unique<EmptyNotifier>()
    )
);

auto all = std::make_unique<TelegramDecorator>(
    std::make_unique<SMSDecorator>(
        std::make_unique<EmailDecorator>(
            std::make_unique<EmptyNotifier>()
        )
    )
);

sms->Send("sms");
email_and_telegram->Send("email and telegram");
all->Send("all");
```

Использование