Getting to Know the Structural Design Patterns

Esteban Herrera
JAVA ARCHITECT

@eh3rrera www.eherrera.net

Structural Patterns

Decorator

Bridge

Facade

Proxy

Composite

Adapter

Flyweight

Class Structural Patterns

Adapter

Object Structural Patterns

Decorator

Bridge

Facade

Proxy

Composite

Flyweight

Object Structural Patterns

Decorator

Bridge

Facade

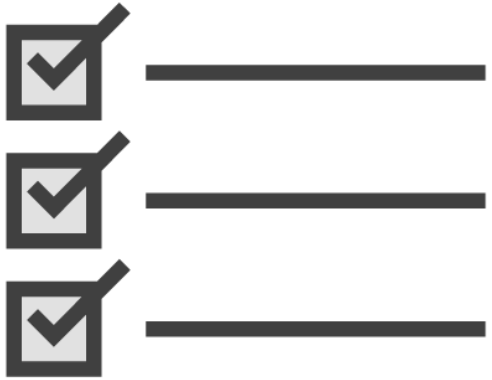
Proxy

Composite

Adapter

Flyweight

Structural Patterns You Should Know



- Facade
- Decorator
- Adapter
- Proxy

The Facade Pattern

The Facade Pattern

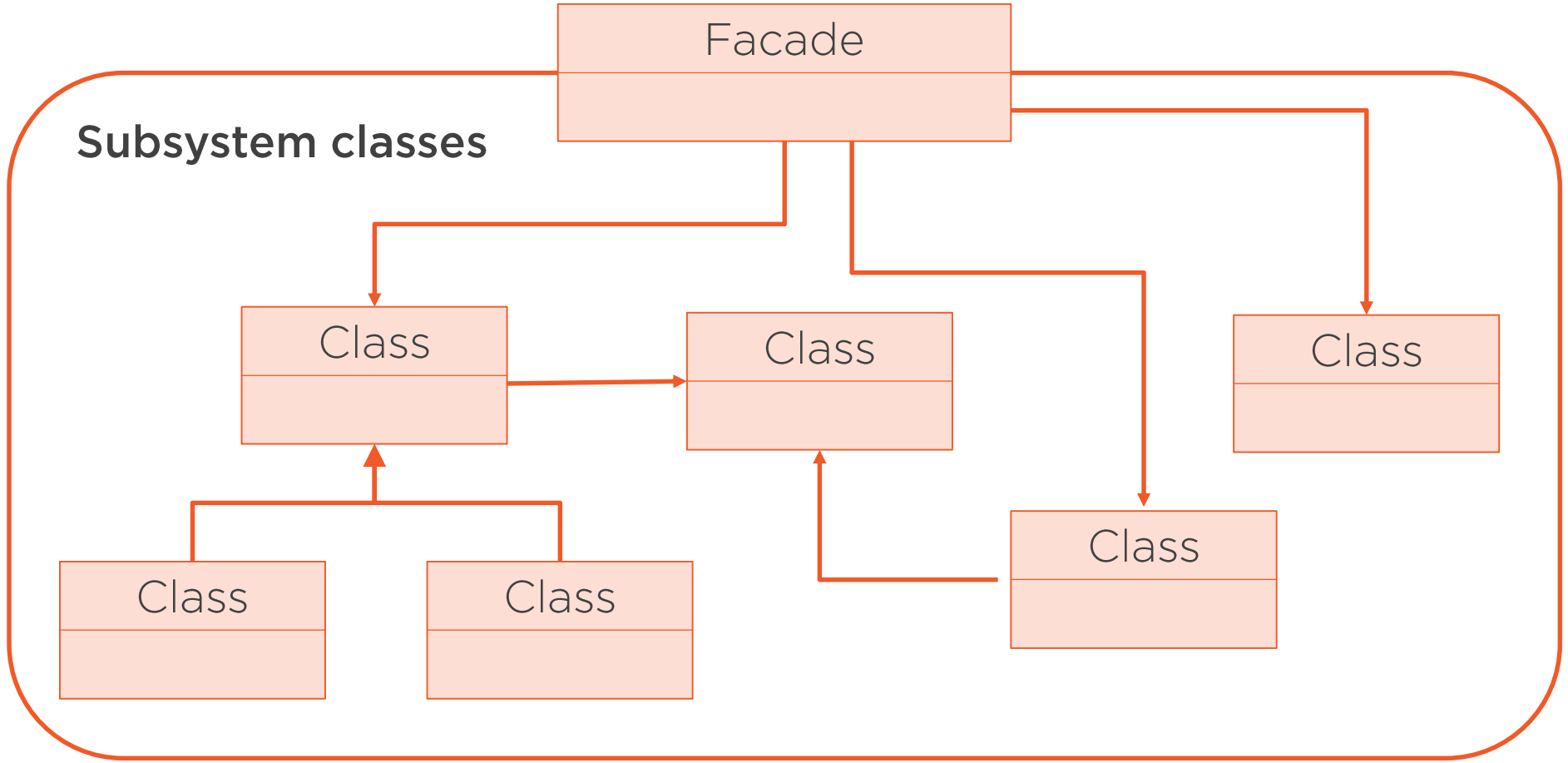
Provides a simplified interface to use subsystem classes

- It doesn't encapsulate subsystem classes

It may provide additional functionality

It decouples a client from the subsystem

The Facade Pattern



The Facade Pattern Example

```
order.registerOrder(order);
customer.sendNotification(order, Order.RECEIVED);
warehouse.sendNotification(order, Order.RECEIVED);
sales.generateOrderReport(order);
```

The Facade Pattern Example

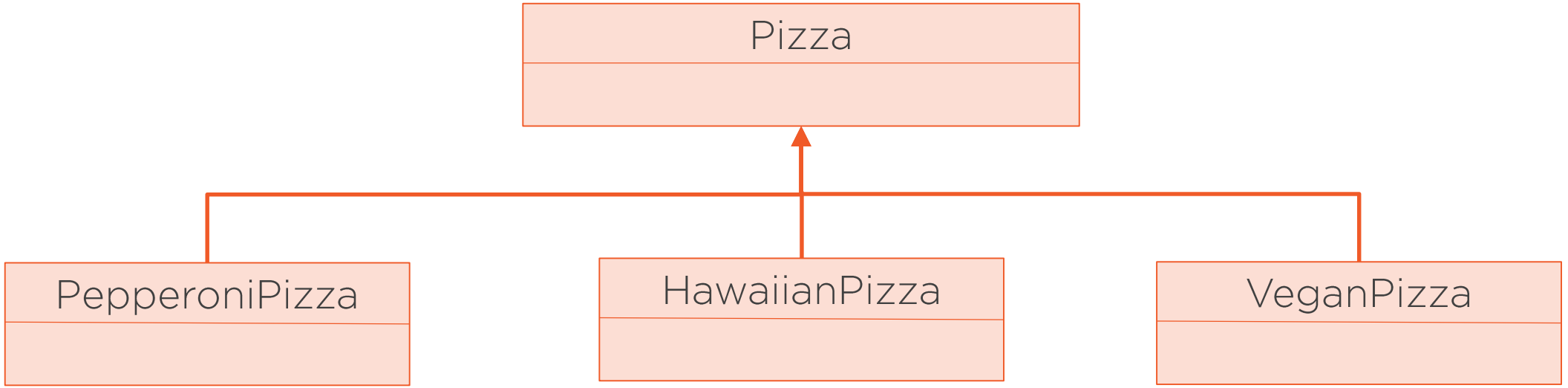
```
public class OrderFulfillmentFacade {
    public void receiveOrder(Order order) {
        order.registerOrder(order);
        customer.sendNotification(order, Order.RECEIVED);
        warehouse.sendNotification(order, Order.RECEIVED);
        sales.generateOrderReport(order);
    }
    // ...
}
```

The Facade Pattern Example

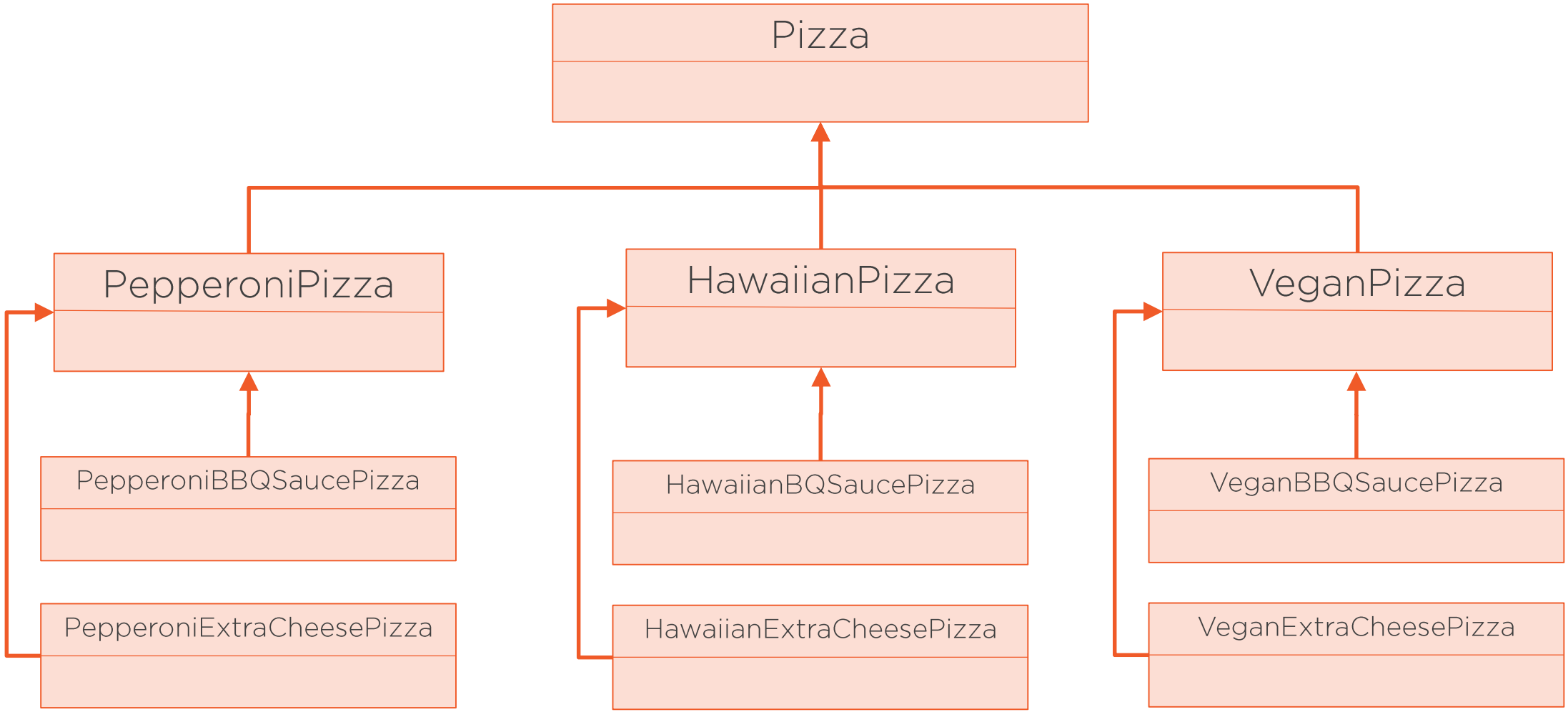
```
OrderFulfillmentFacade facade =
    new OrderFulfillmentFacade(
        order, customer, warehouse, sales
);
// ...
facade.receiveOrder(order);
```

The Decorator Pattern

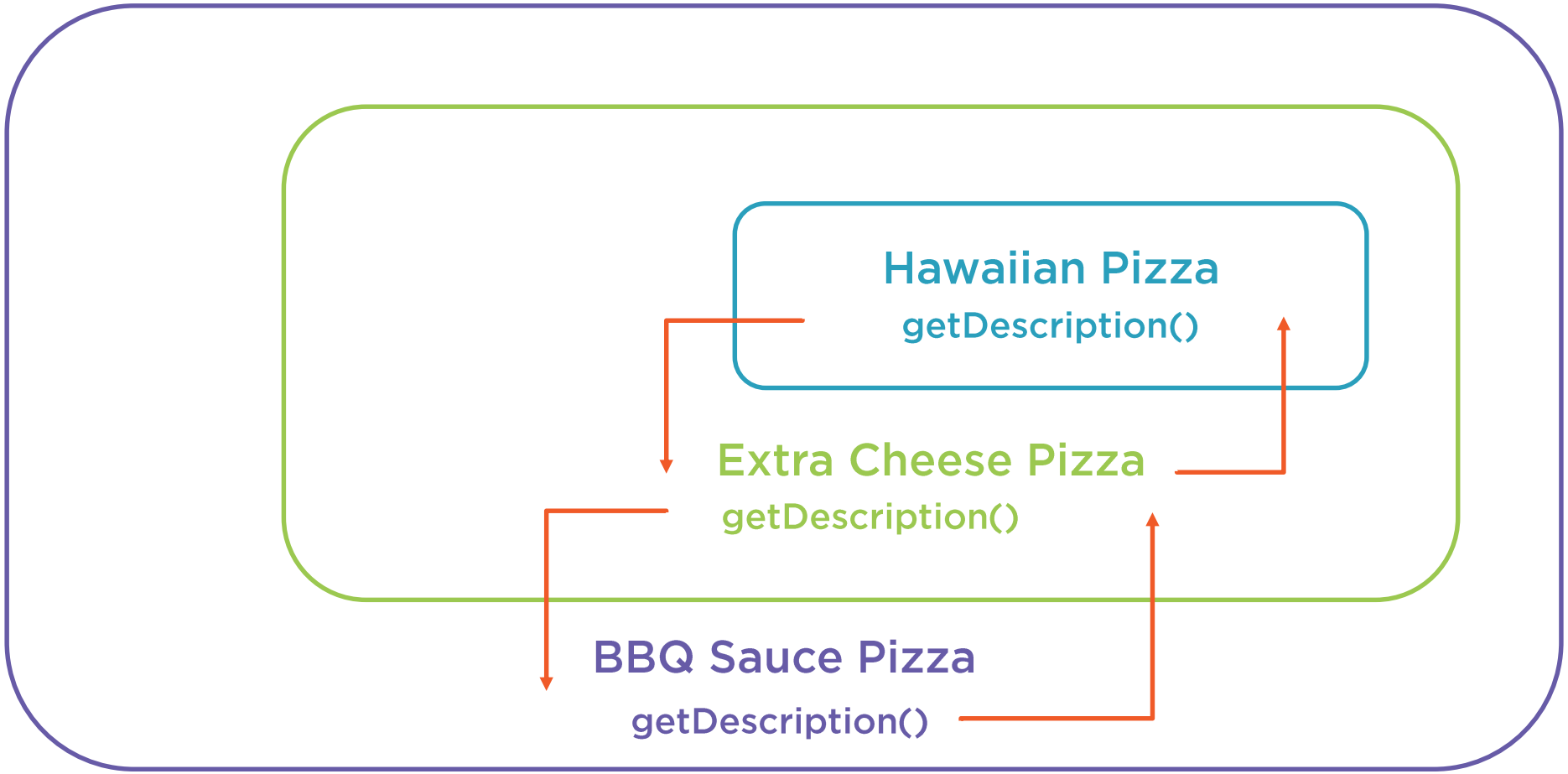
A Pizza Class Hierarchy



A Pizza Class Hierarchy

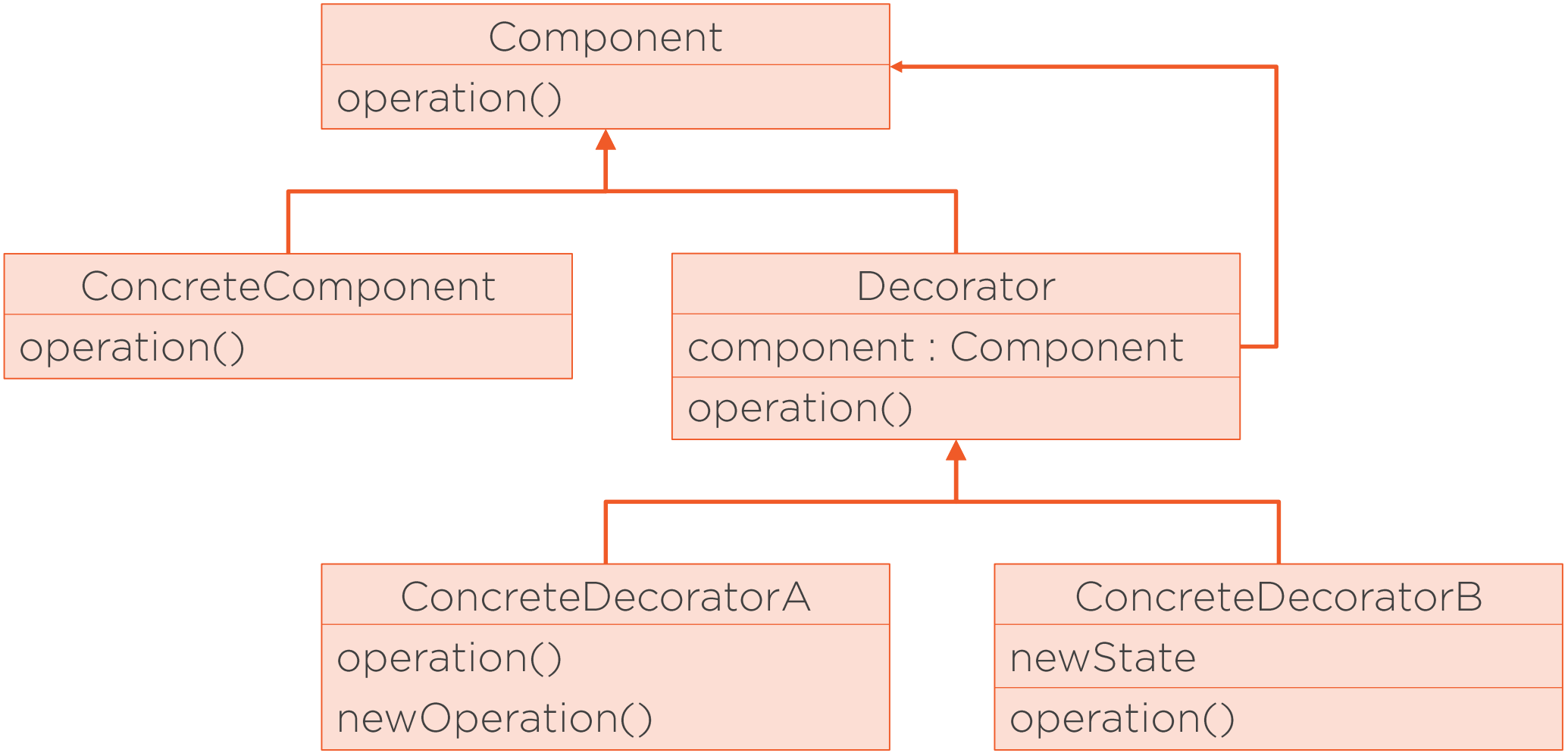


Decorators



Hawaiian Pizza, extra cheese, BBQ sauce

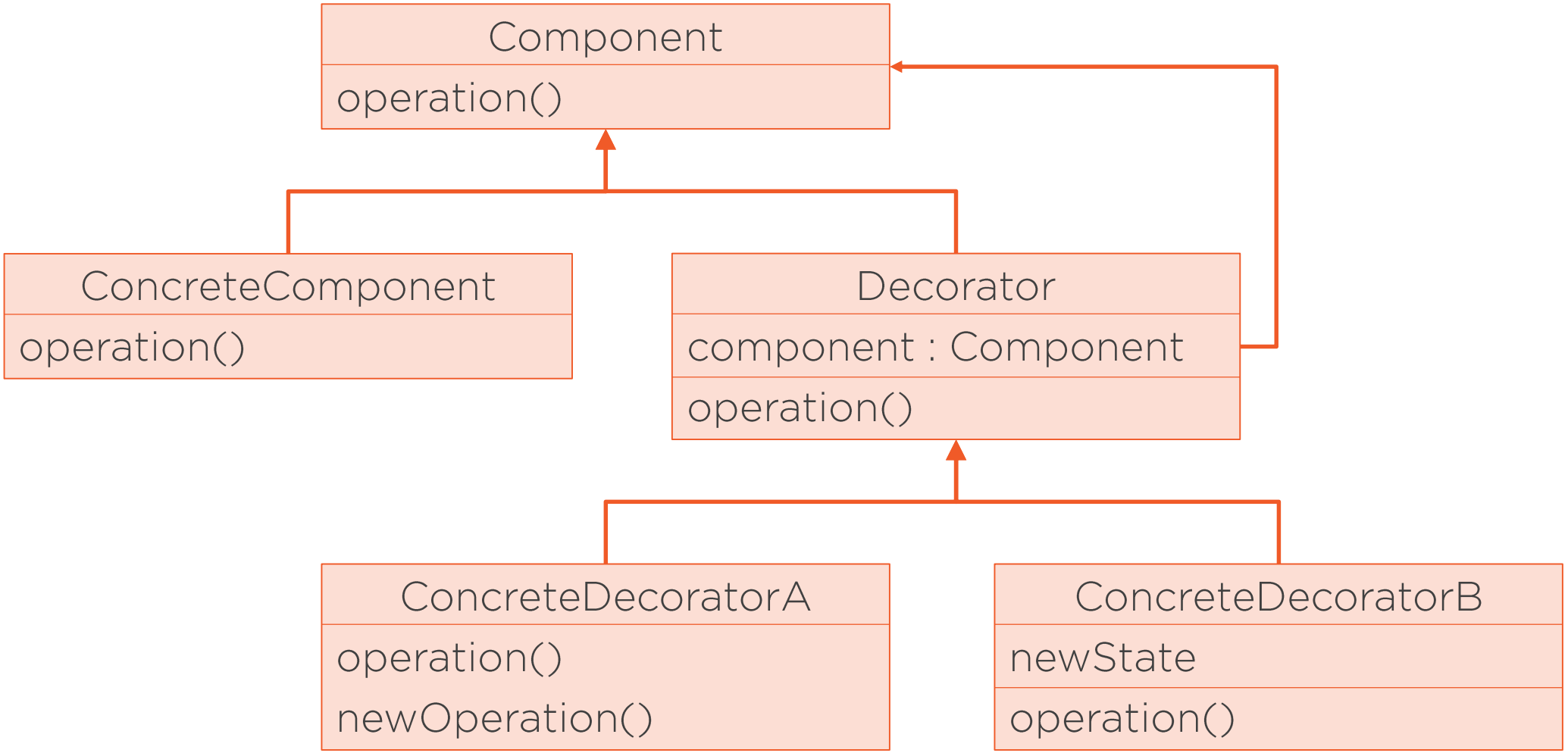
The Decorator Pattern



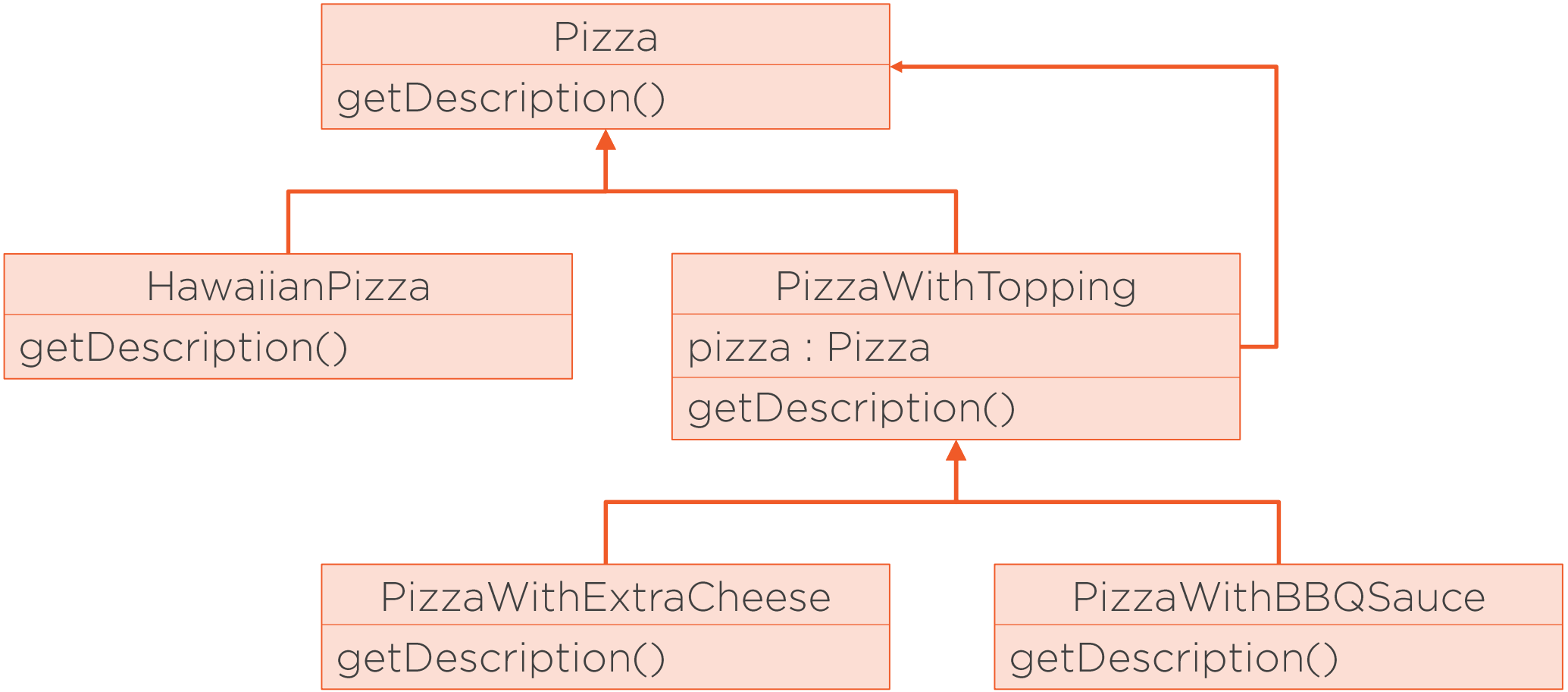
The Decorator pattern doesn't use inheritance to get behavior.

But for type matching.

The Decorator Pattern



Decorator Pattern Example



HawaiianPizza Class

```
public class HawaiianPizza extends Pizza {
    public String getDescription() {
        return "Hawaiian Pizza";
    }
    // ...
}
```

PizzaWithTopping Class

```
public abstract class PizzaWithTopping extends Pizza {
    protected Pizza pizza;

    public abstract String getDescription();
}
```

PizzaWithExtraCheese Class

```
public class PizzaWithExtraCheese extends PizzaWithTopping {
    public PizzaWithExtraCheese(Pizza pizza) {
        this.pizza = pizza;
    }

    @Override
    public String getDescription() {
        return pizza.getDescription() + ", extra cheese";
    }
}
```

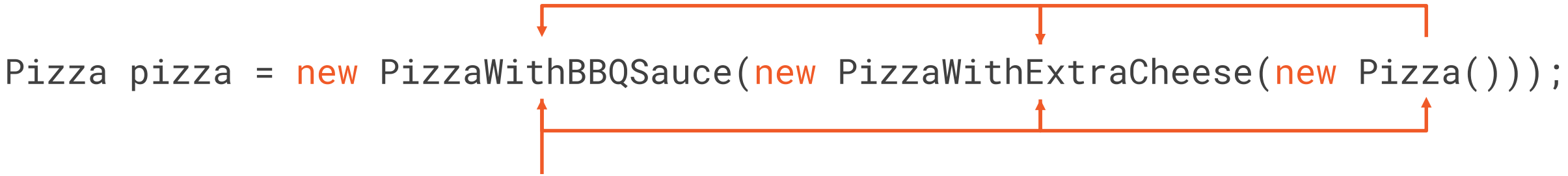
PizzaWithBBQSauce Class

```
public class PizzaWithBBQSauce extends PizzaWithTopping {
    public PizzaWithBBQSauce (Pizza pizza) {
        this.pizza = pizza;
    }

    @Override
    public String getDescription() {
        return pizza.getDescription() + ", BBQ sauce";
    }
}
```

Using the Decorator Pattern

```
Pizza pizza = new Pizza();

pizza = new PizzaWithExtraCheese(pizza);
pizza = new PizzaWithBBQSauce(pizza);
```

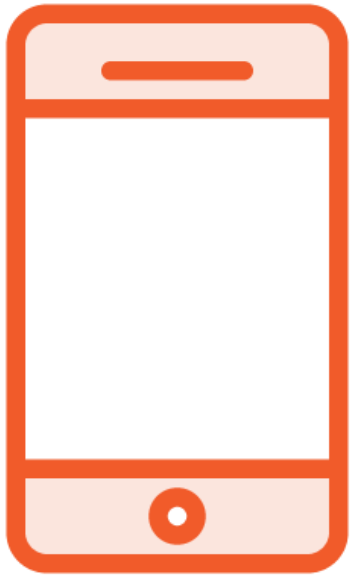
Using the Decorator Pattern



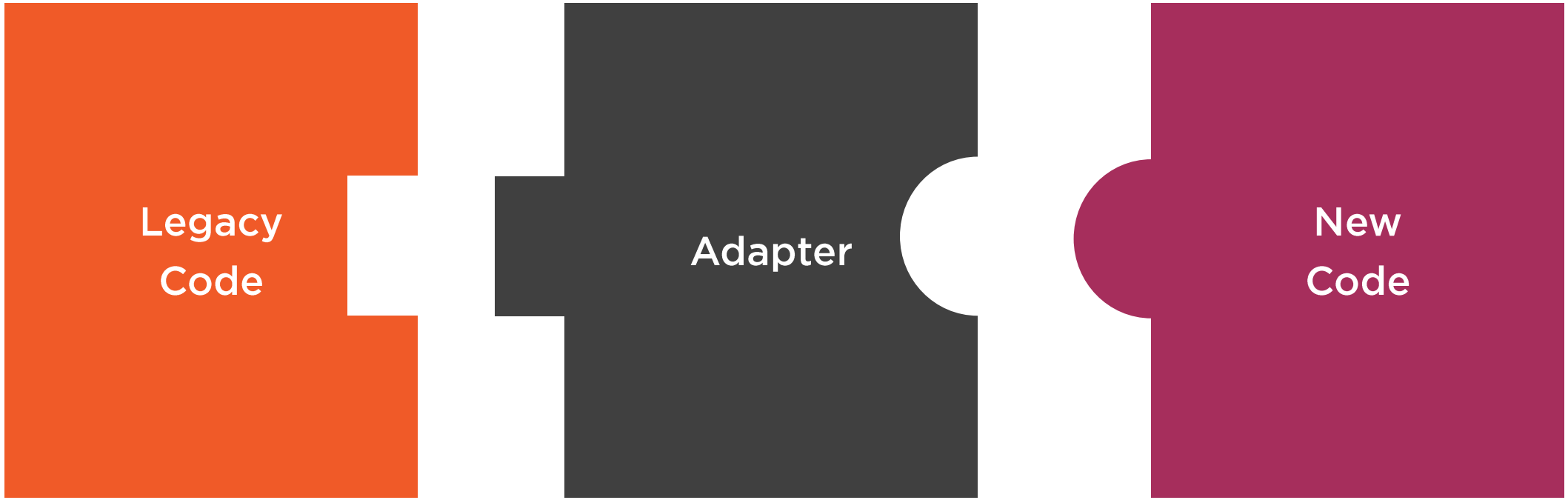
```
Pizza pizza = new PizzaWithBBQSauce(new PizzaWithExtraCheese(new Pizza()));

System.out.println(pizza.getDescription());


// Hawaiian Pizza, extra cheese, BBQ sauce
```

The Adapter Pattern

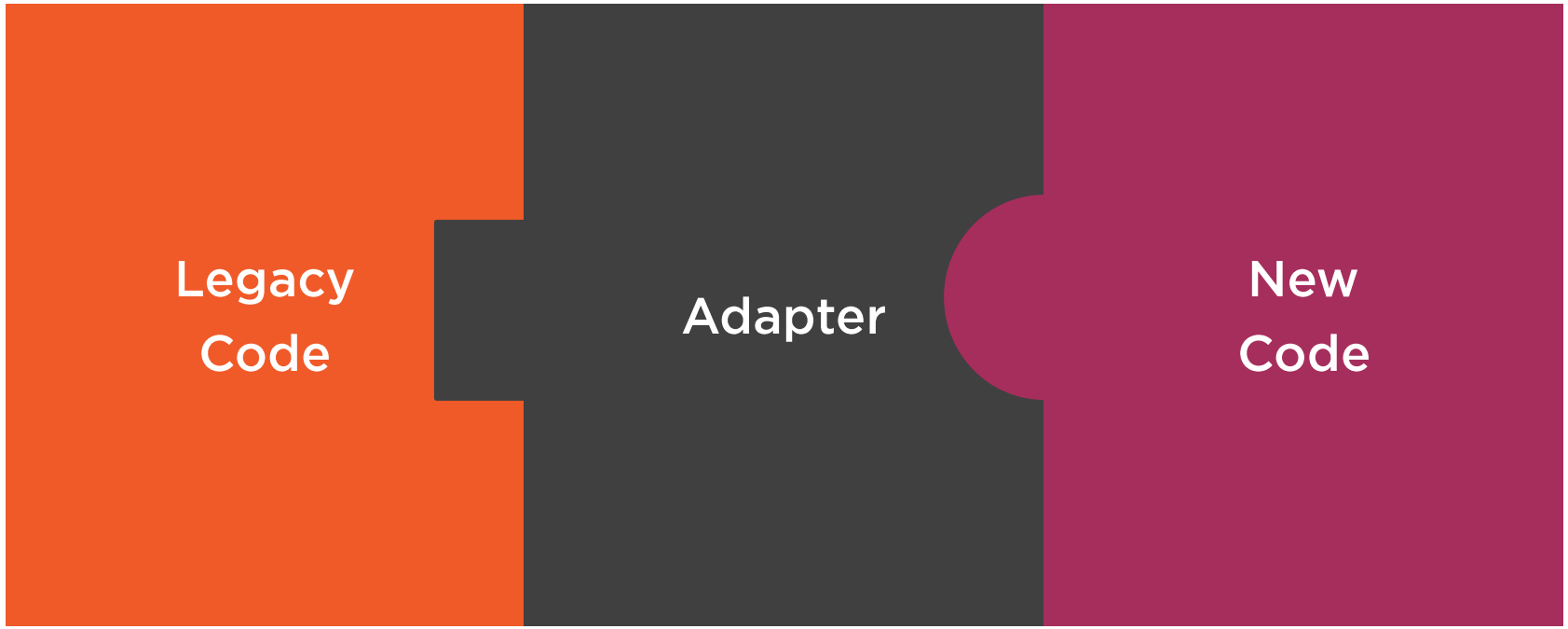
The Problem



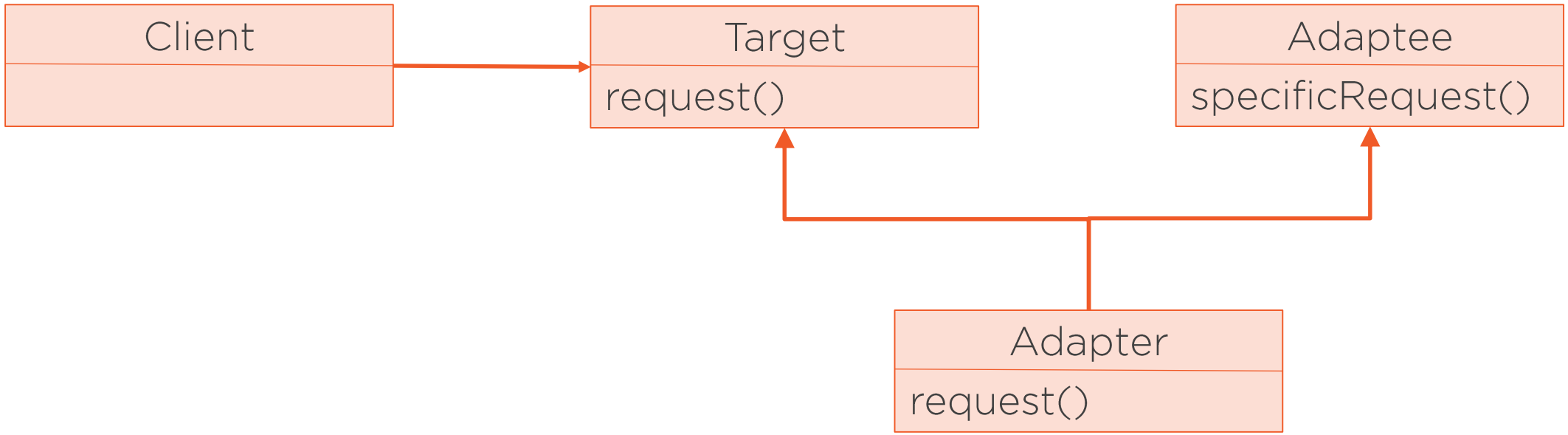
Adapting Code



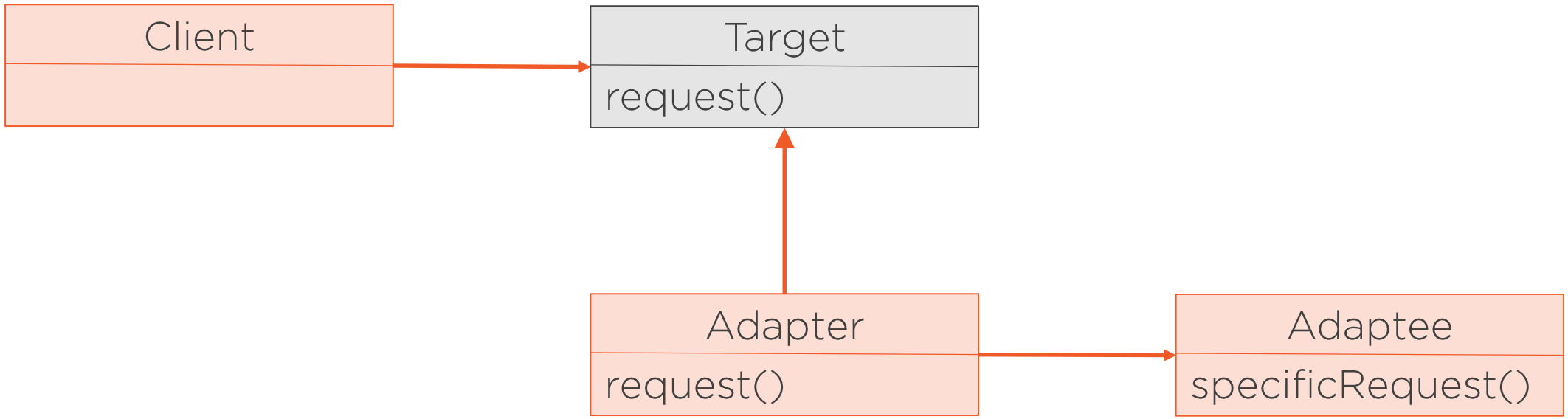
Adapting Code



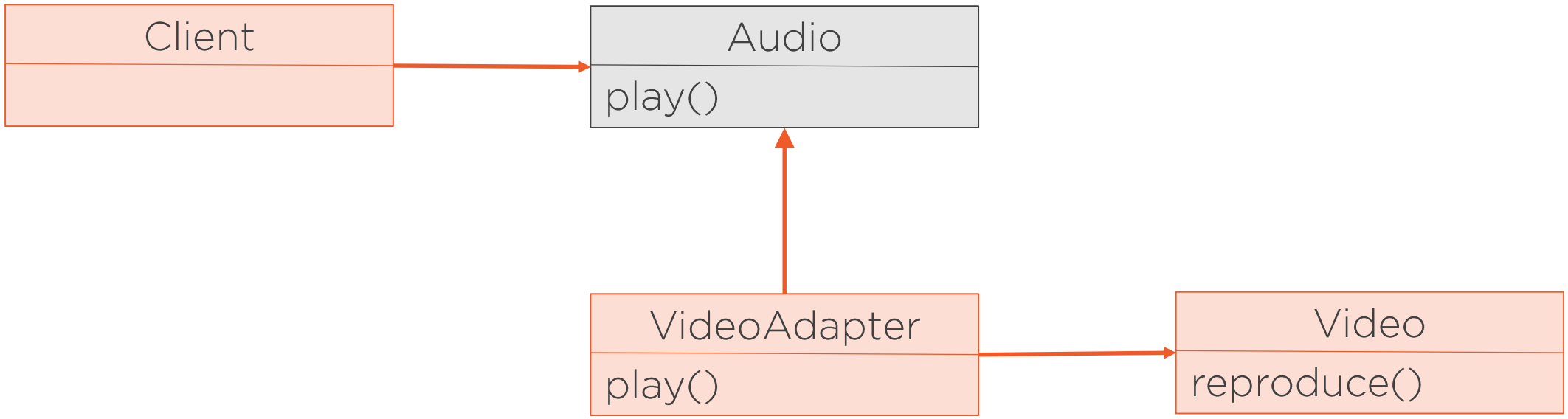
Class Adapter Pattern



Object Adapter Pattern



Object Adapter Pattern Example



VideoAdapter Class

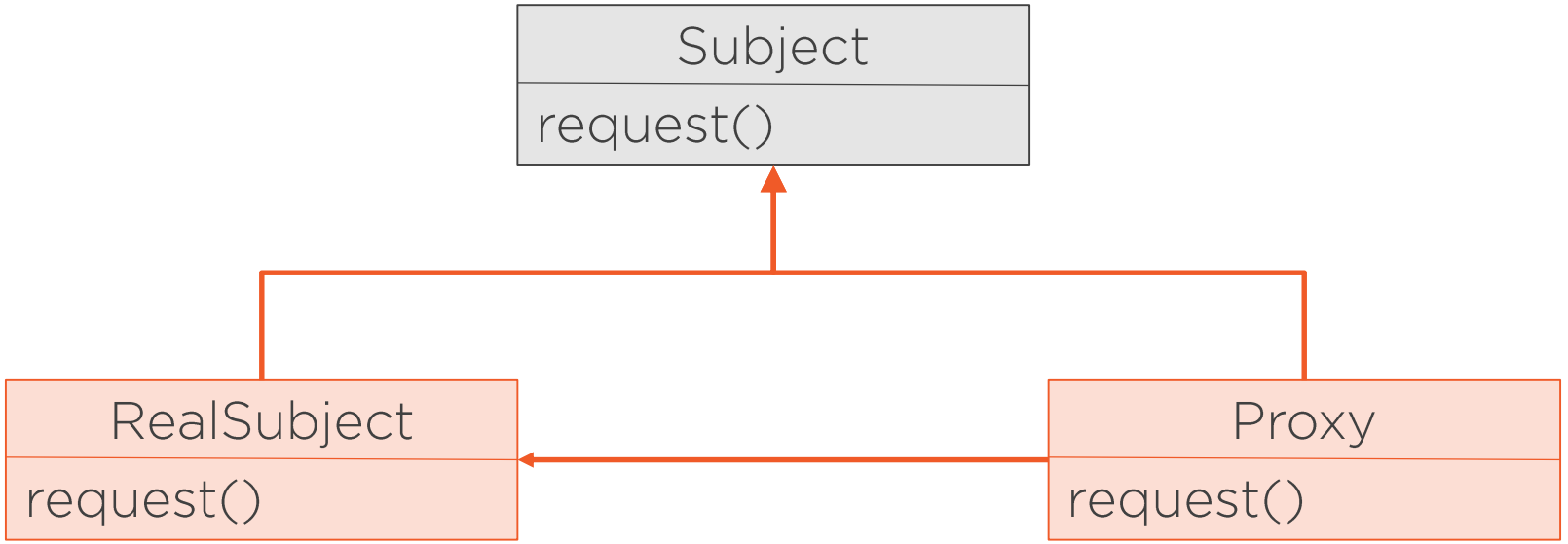
```
public class VideoAdapter implements Audio {
    private Video video;

    public VideoAdapter(Video video) {
        this.video = video;
    }

    public void play() {
        Audio audio = extractAudio(video);
        audio.play();
    }

    private Audio extractAudio(Video video) {
        // ...
    }
}
```

The Proxy Pattern

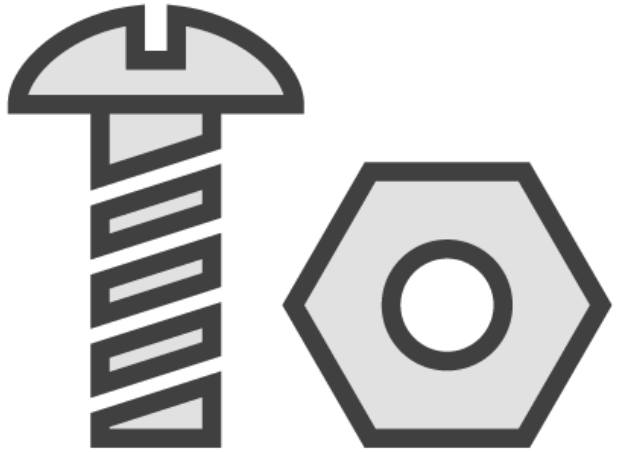
Using an Object

Using a Proxy to an Object

The Proxy Pattern

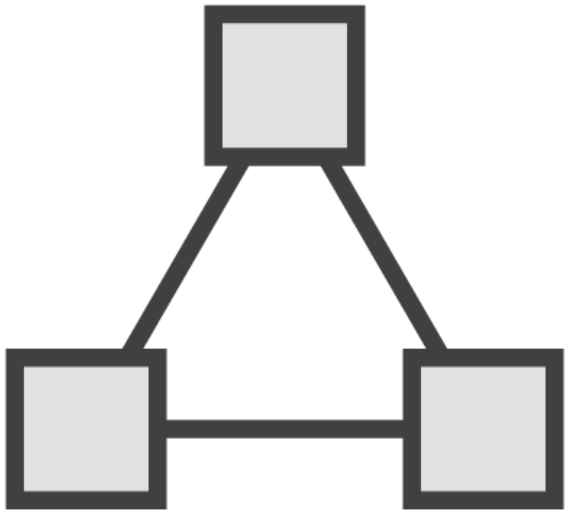


Common Uses of a Proxy



- Remote Calls
- Security
- Cache
- Virtual

How Is Proxy Related to Other Patterns?



Facade

Decorator

Adapter

Things to Remember

Facade

- Simplifies an interface to a subsystem, decoupling a client from it.

Decorator

- Attach additional responsibilities to an object dynamically.

Adapter

- Convert the interface of a class into another interface clients expect.

Proxy

- Provides a surrogate for another object to control access to it.

Things to Remember

Proxy and Decorator

Decorator and Adapter

Proxy, Decorator, and Adapter

Adapter and Facade