# Using Structural Design Patterns

**Annapurna Agrawal**
AUTHOR

@annapurna_23 linkedin.com/in/annapurna-agrawal

# Design Patterns

**Creational Pattern**

Provides object creation mechanism

**Structural Pattern**

Explains how to assemble objects and classes into larger, flexible structures

**Behavioral Pattern**

Deals with algorithms and assignment of responsibilities between objects

# Design Patterns

**Structural Pattern**

**Explains how to assemble objects and classes into larger structures, while keeping these structures flexible and efficient**

# Structural Design Patterns

Adapter

Bridge

Composite

Decorator

Facade

Flyweight

Proxy

# Structural Design Patterns

## Adapter

## Decorator

# Structural Design Patterns

## Adapter

Allows objects with incompatible interfaces to collaborate

## Decorator

# Real World Scenario

American

Asian

# Real World Scenario


# Problem: Quick Scenario

**Notes Application**

**Share Option**

```
waShare()
.
.
// Application code
```

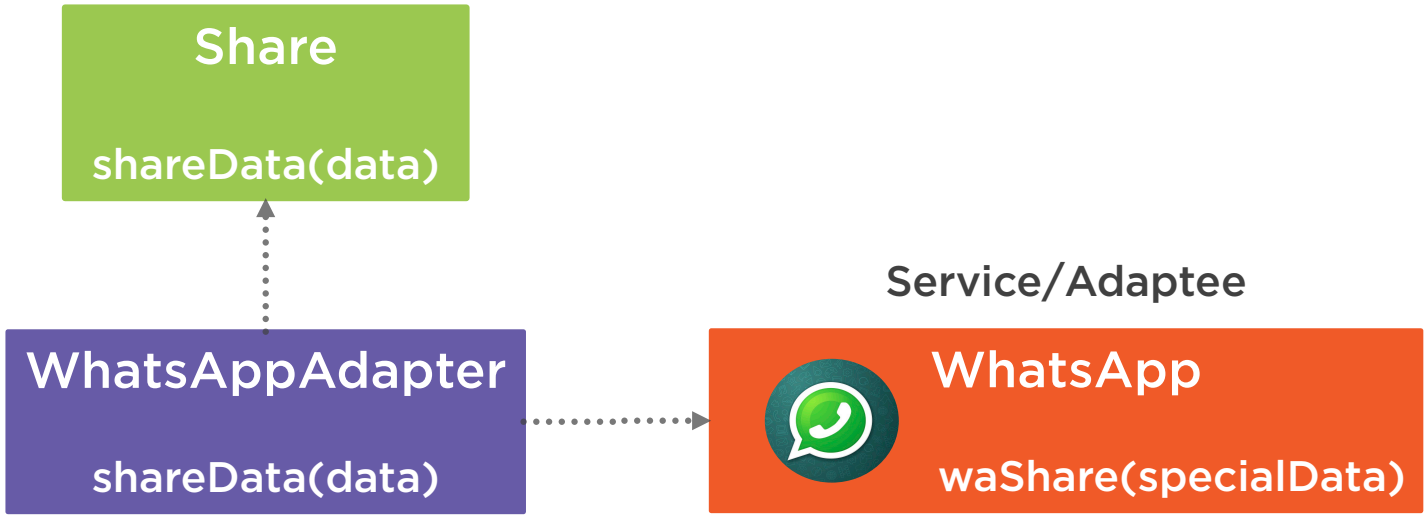
# Problem: Quick Scenario

**Notes Application**

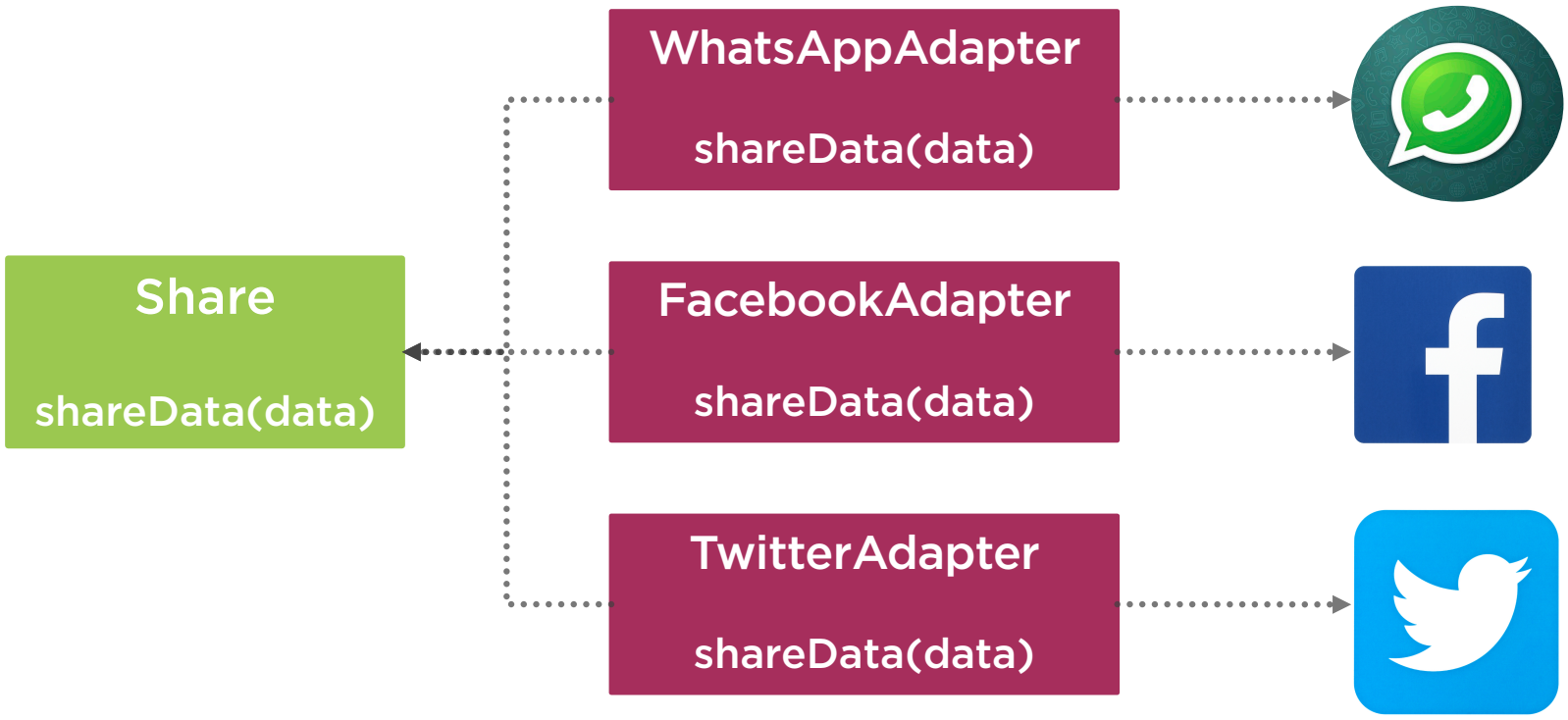
**Share Option**

```
wAppShare()
.
.
// Application code
```

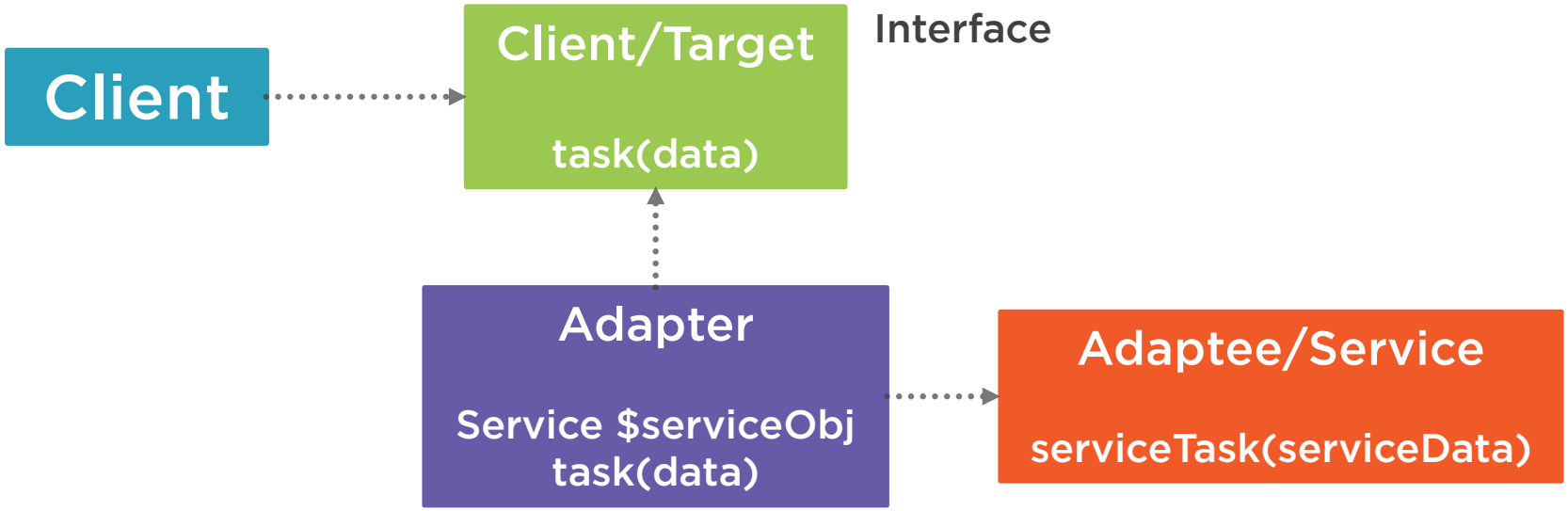
# Solution: Adapter Pattern



# Solution: Adapter Pattern



# Adapter Pattern



Demo

## Implementing Adapter Pattern

# Adapter

Adapter class serves as the bridge between some existing service code and our app code

Adapter pattern makes the existing or new incompatible APIs work, without changing the existing code

# Structural Design Patterns

## Adapter

Allows objects with incompatible interfaces to collaborate

## Decorator

# Structural Design Patterns

## Adapter

Allows objects with incompatible interfaces to collaborate

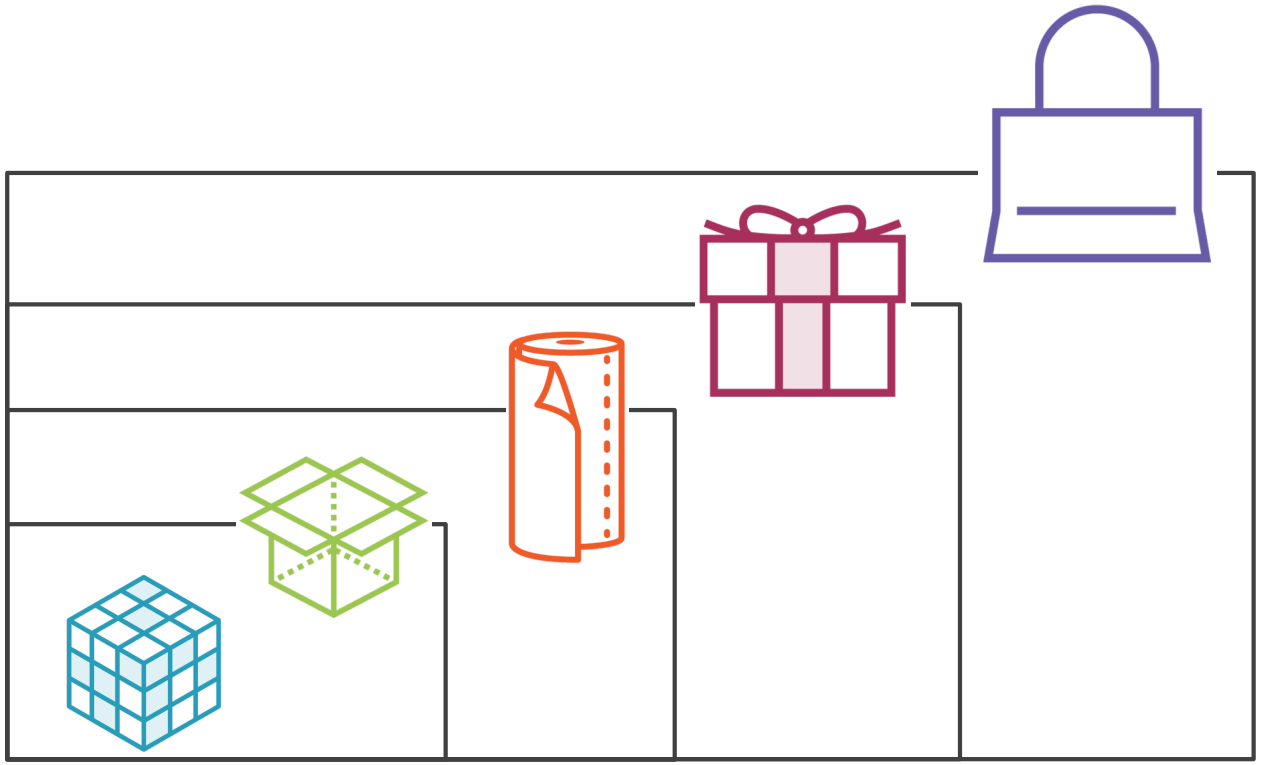
## Decorator

Add new behaviours to existing objects by placing these objects inside special wrapper objects that contain the behaviours

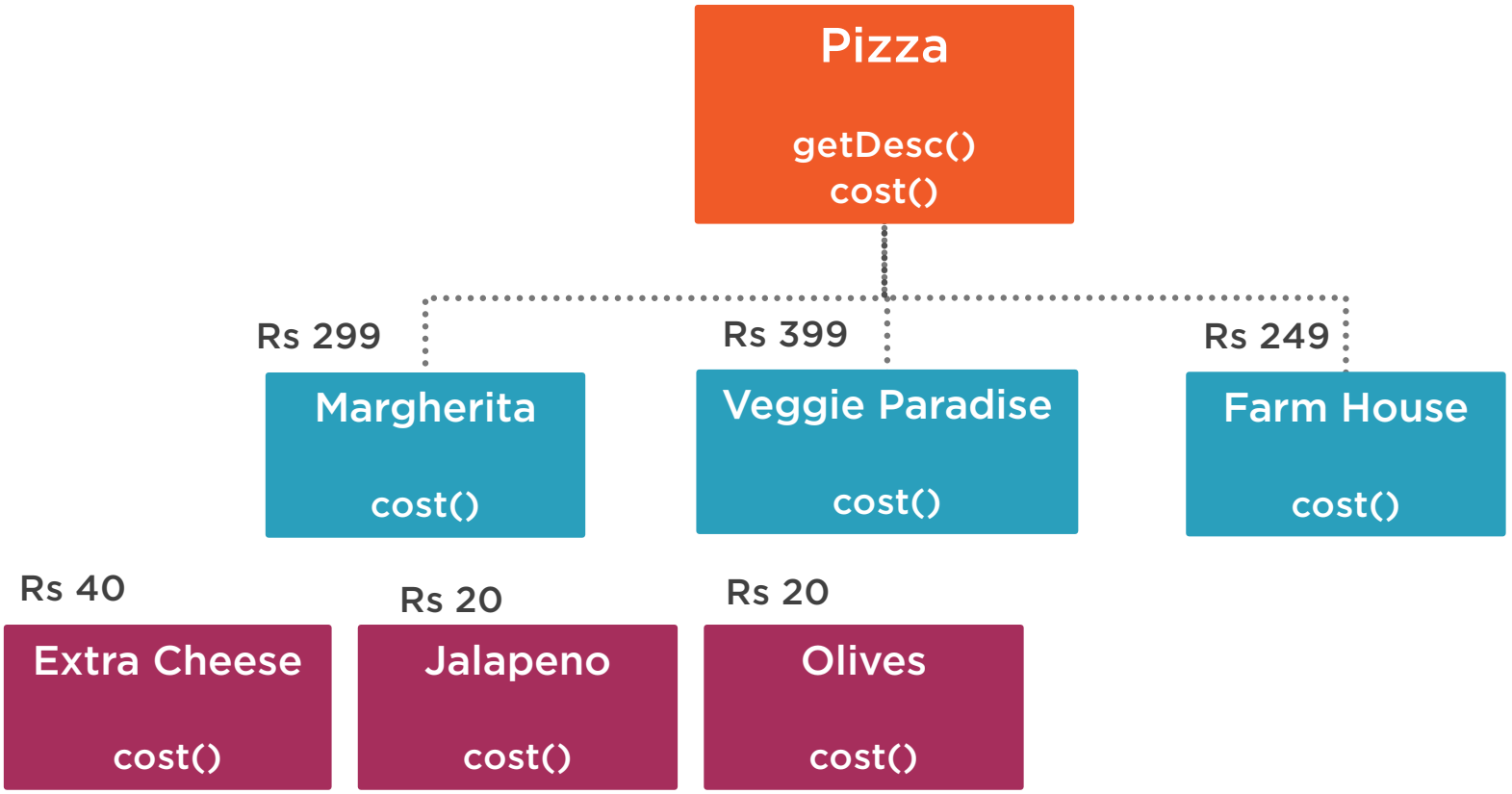
# Real World Scenario

Gift Box Paper Ribbon Gift Bag

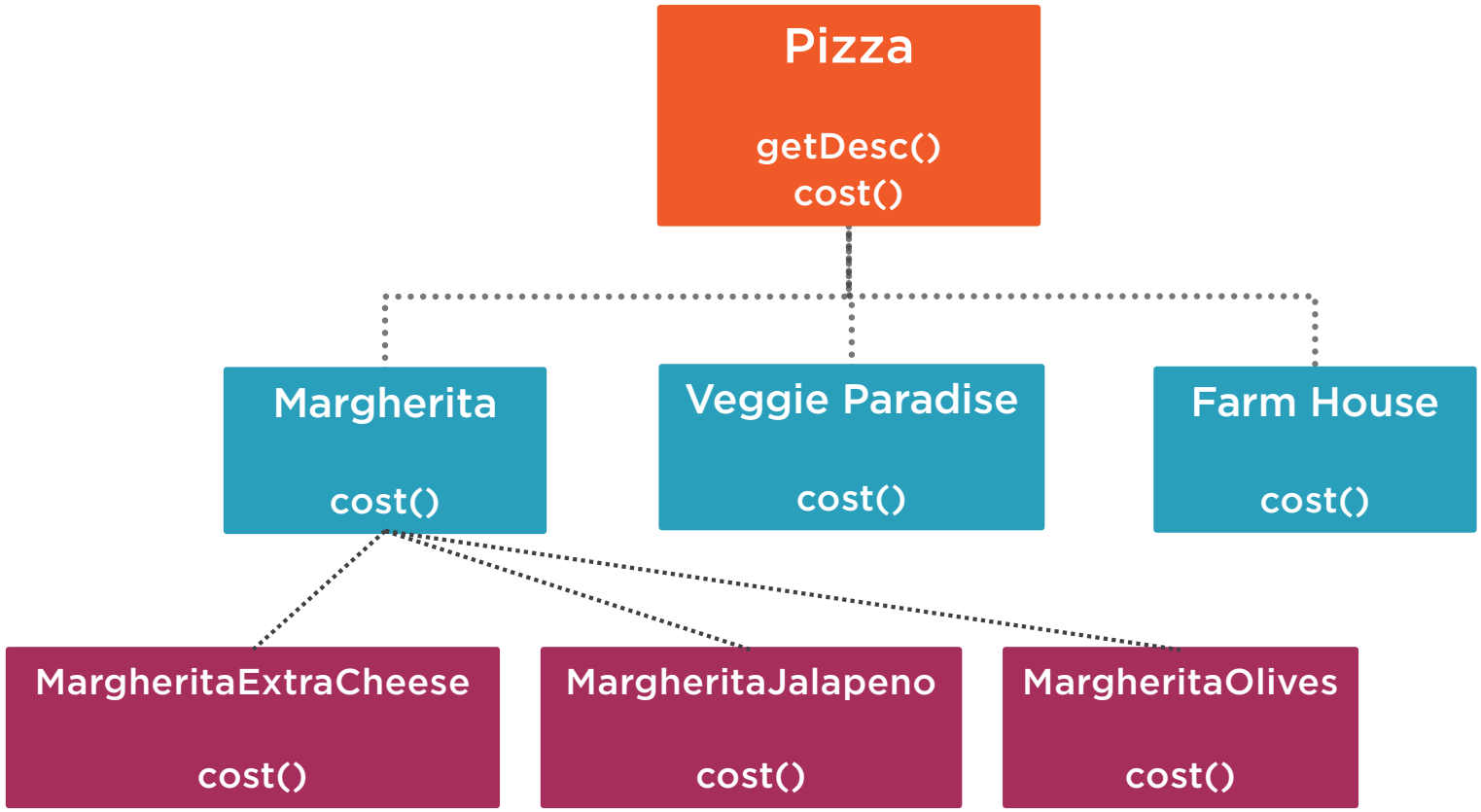
# Real World Scenario



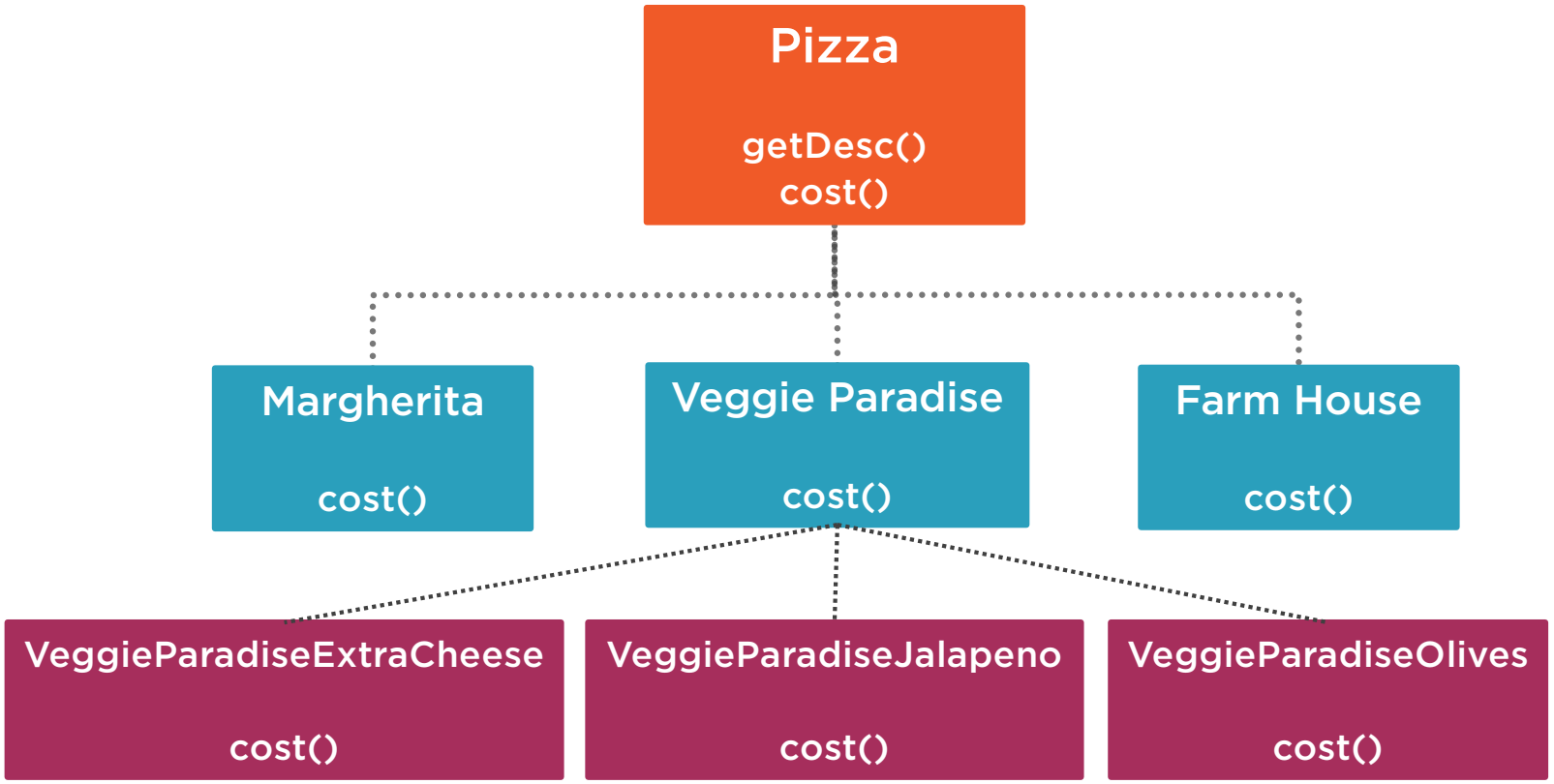
# Problem: Quick Scenario



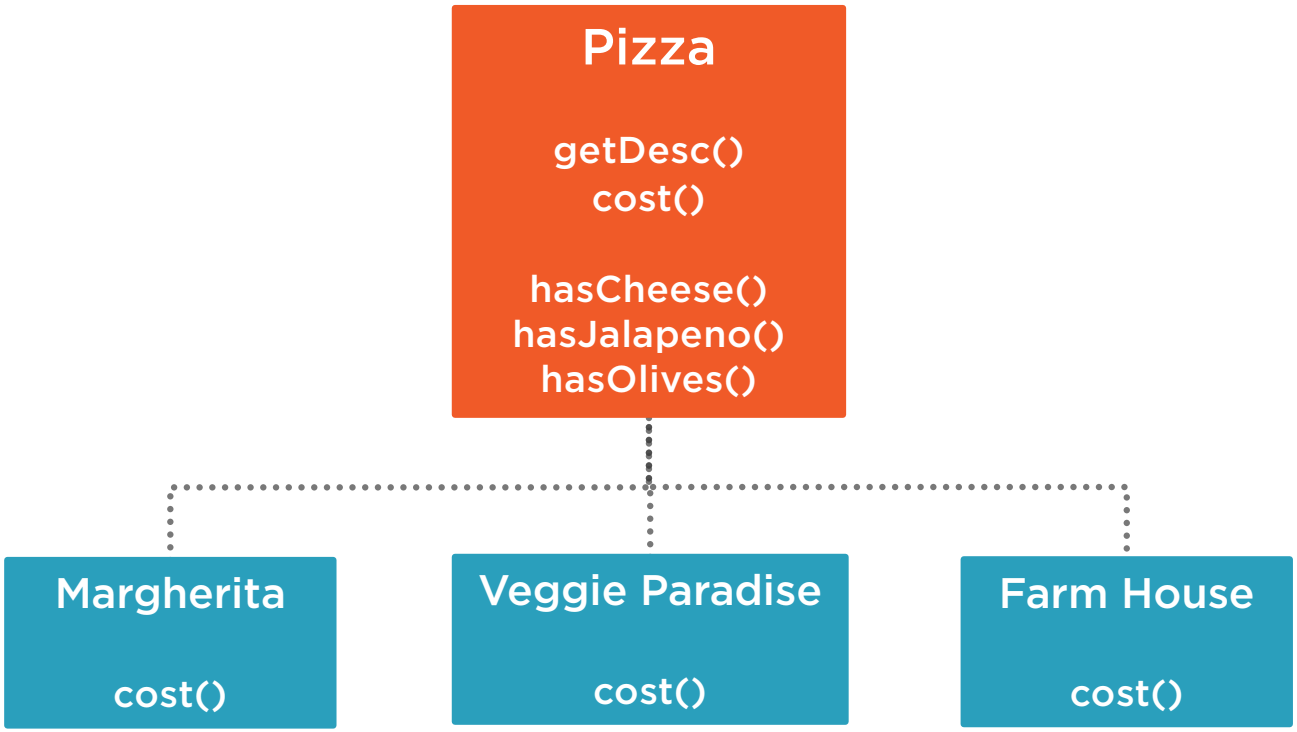
# Solution 1: Quick Scenario



# Solution 1: Quick Scenario



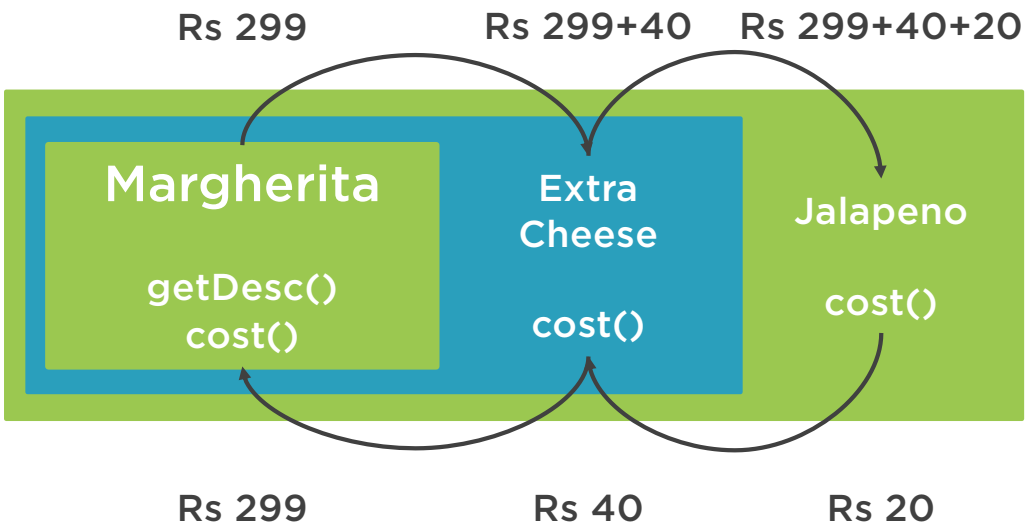
# Solution 2: Quick Scenario



# Solution 2: Quick Scenario

```
hasCheese() {
    $cost += 40;
}
hasJalapeno() {
    $cost += 20;
}
hasOlives() {
    $cost += 20;
}
```

# Solution: Decorator Pattern



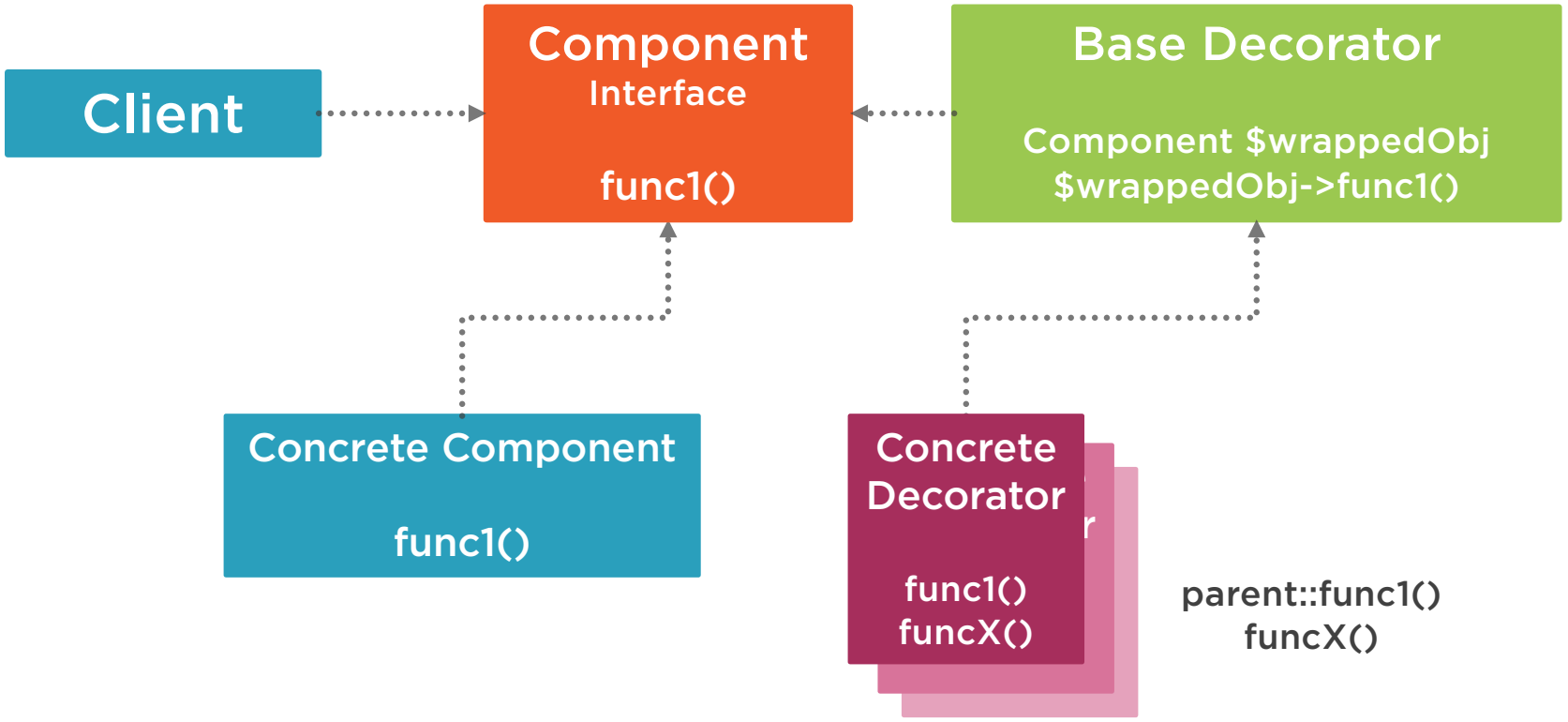
# Solution: Decorator Pattern

# Structural Design Patterns

## Decorator

**Add new behaviours to existing objects, dynamically, by placing these objects inside special wrapper objects that contain the behaviours**

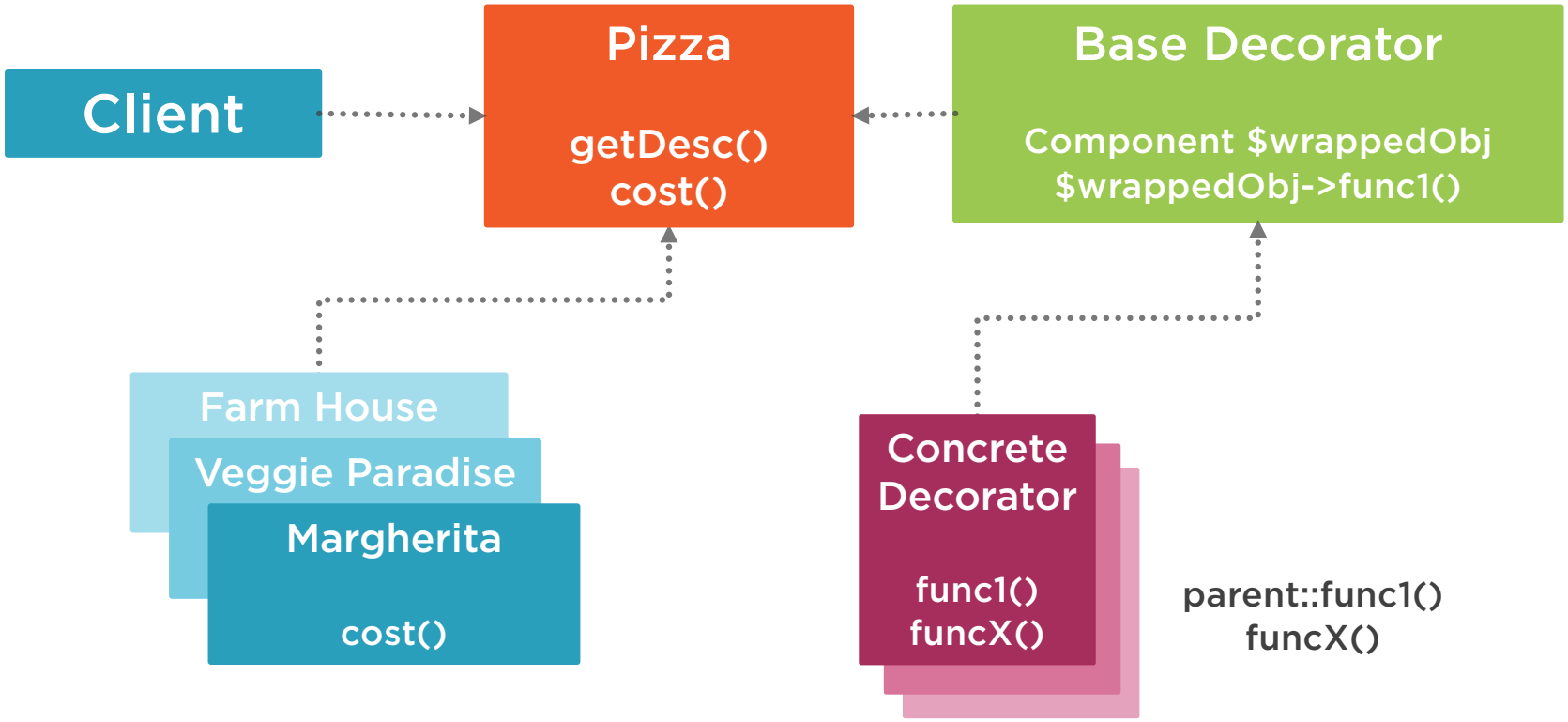
# Decorator Pattern



Demo

## Implementing Decorator Pattern

# Decorator Pattern



# Decorator

To assign behaviours to objects dynamically, without modifying the code that uses the object

Used when is it complex or not possible to extend the object's behaviour using inheritance

# Summary

Structural pattern ease the design by identifying simple way to realize relationships between entities

Adapter pattern is used to match interfaces of different classes

Decorator pattern is used to add responsibilities to objects dynamically

# Structural Pattern in PHP

# Up Next:
# Implementing Behavioural Design Patterns