

Structural Pattern: Facade



Kevin Dockx

Architect

@KevinDockx <https://www.kevindockx.com>



Coming Up



Describing the facade pattern

Implementation:

- Discount calculator service

Structure of the facade pattern



Coming Up



Use cases for this pattern

Pattern consequences

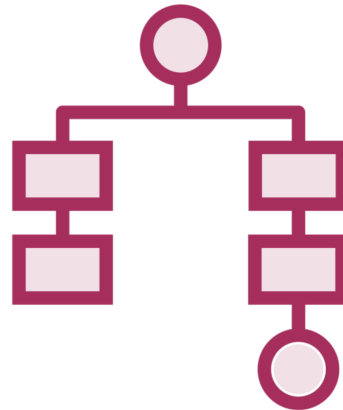
Related patterns



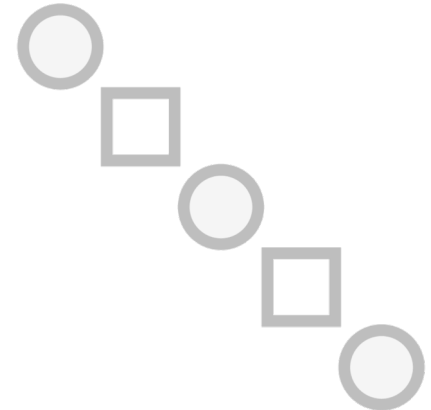
Describing the Facade Pattern



Creational



Structural



Behavioral



Facade

The intent of this pattern is to provide a unified interface to a set of interfaces in a subsystem. It defines a higher-level interface that makes the subsystem easier to use.



```
OrderService orderService = new ();
CustomerDiscountBaseService customerDiscountBaseService = new ();
DayOfTheWeekFactorService dayOfTheWeekFactorService = new ();

if (!orderService.HasEnoughOrders(customerId)) {
    discount = 0;
} else {
    discount = customerDiscountBaseService.CalculateDiscountBase(customerId) *
               dayOfTheWeekFactorService.CalculateDayOfTheWeekFactor(); }
}
```

Describing the Facade Pattern

```
OrderService orderService = new ();
CustomerDiscountBaseService customerDiscountBaseService = new ();
DayOfTheWeekFactorService dayOfTheWeekFactorService = new ();

if (!orderService.HasEnoughOrders(customerId)) {
    discount = 0;
} else {
    discount = customerDiscountBaseService.CalculateDiscountBase(customerId) *
               dayOfTheWeekFactorService.CalculateDayOfTheWeekFactor(); }
}
```

Describing the Facade Pattern

```
OrderService orderService = new ();
CustomerDiscountBaseService customerDiscountBaseService = new ();
DayOfTheWeekFactorService dayOfTheWeekFactorService = new ();

if (!orderService.HasEnoughOrders(customerId)) {
    discount = 0;
} else {
    discount = customerDiscountBaseService.CalculateDiscountBase(customerId) *
               dayOfTheWeekFactorService.CalculateDayOfTheWeekFactor(); }
}
```

Describing the Facade Pattern


```
OrderService orderService = new ();
CustomerDiscountBaseService customerDiscountBaseService = new ();
DayOfTheWeekFactorService dayOfTheWeekFactorService = new ();

if (!orderService.HasEnoughOrders(customerId)) {
    discount = 0;
} else {
    discount = customerDiscountBaseService.CalculateDiscountBase(customerId) *
               dayOfTheWeekFactorService.CalculateDayOfTheWeekFactor(); }
}
```

Describing the Facade Pattern

A facade hides away the complexity of this calculation and encourages reuse

Describing the Facade Pattern

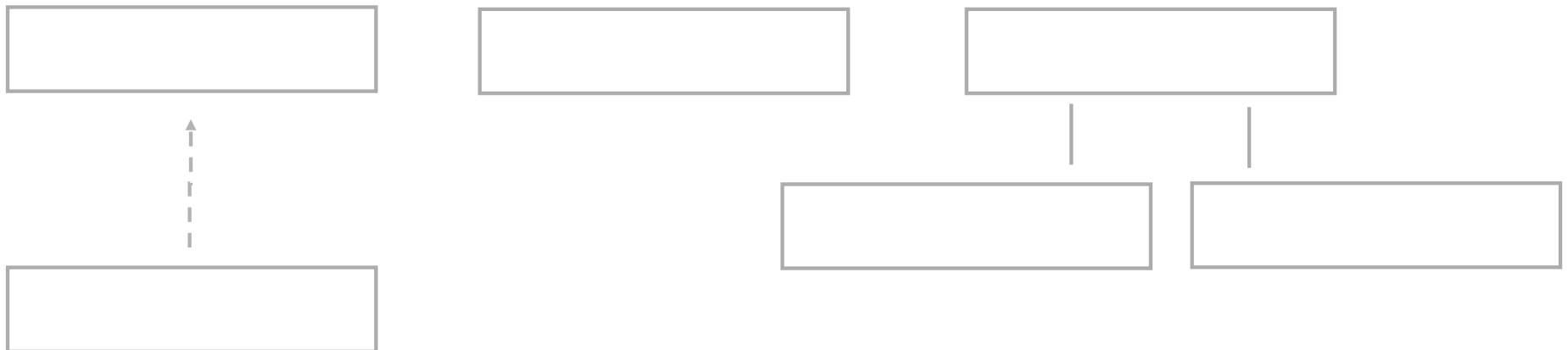
DiscountFacade

int CalculateDiscount()

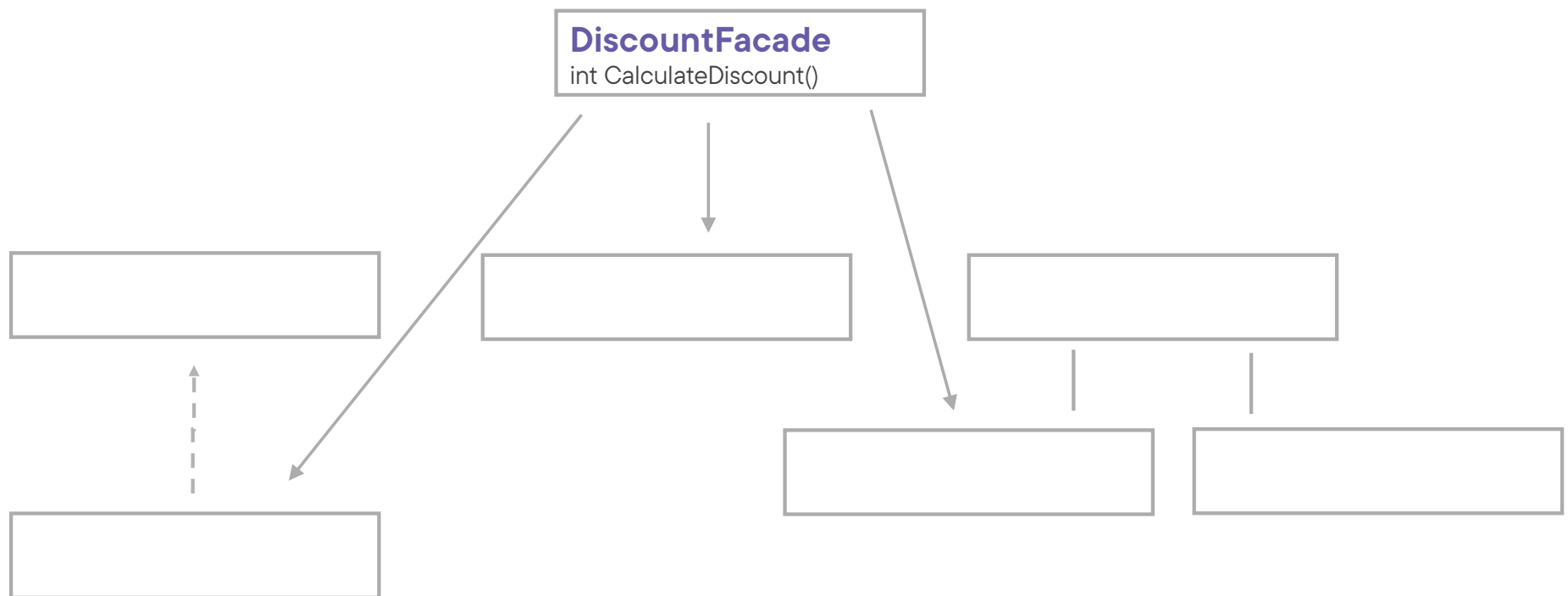


Describing the Facade Pattern

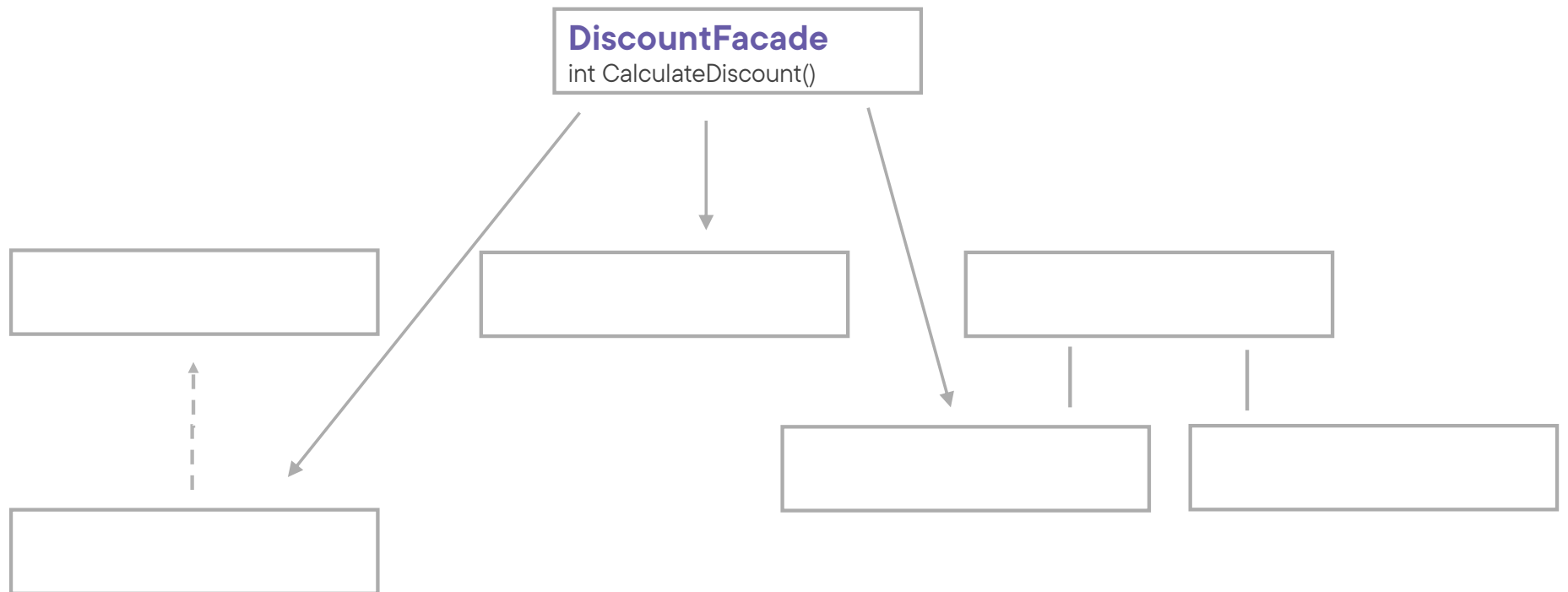
DiscountFacade
int CalculateDiscount()



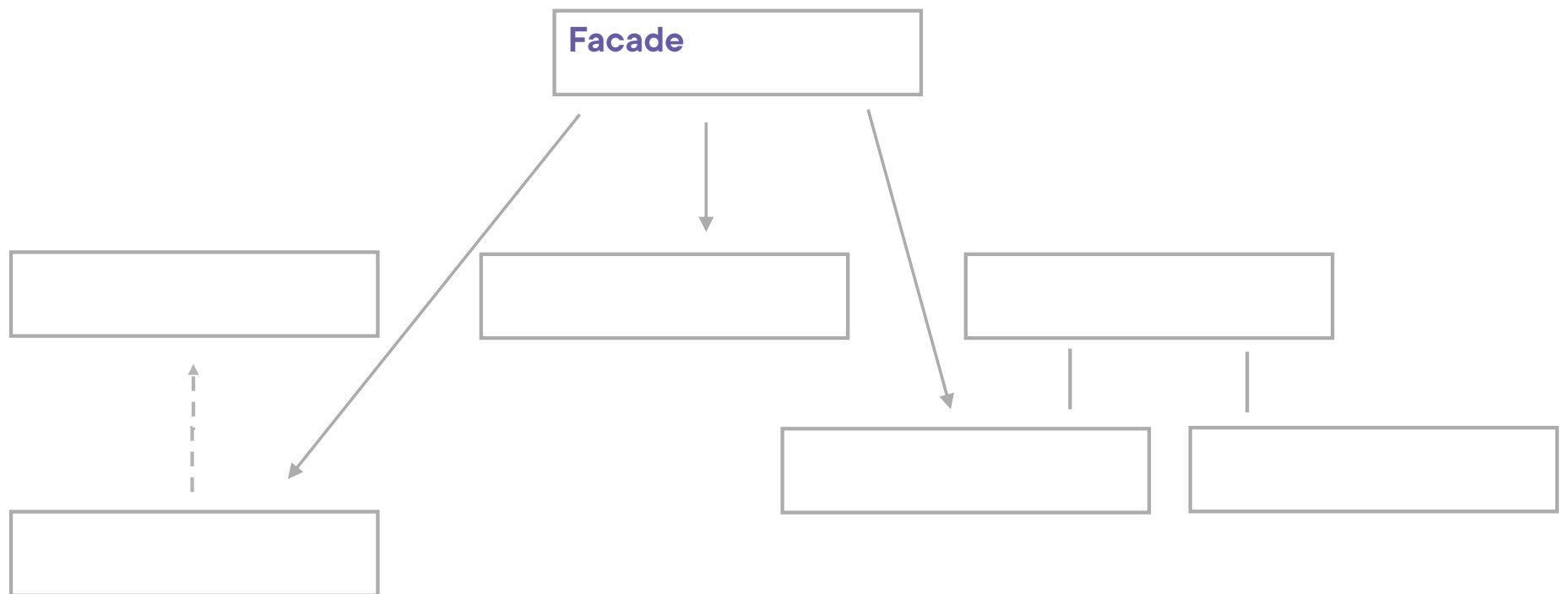
Describing the Facade Pattern



Structure of the Facade Pattern



Structure of the Facade Pattern

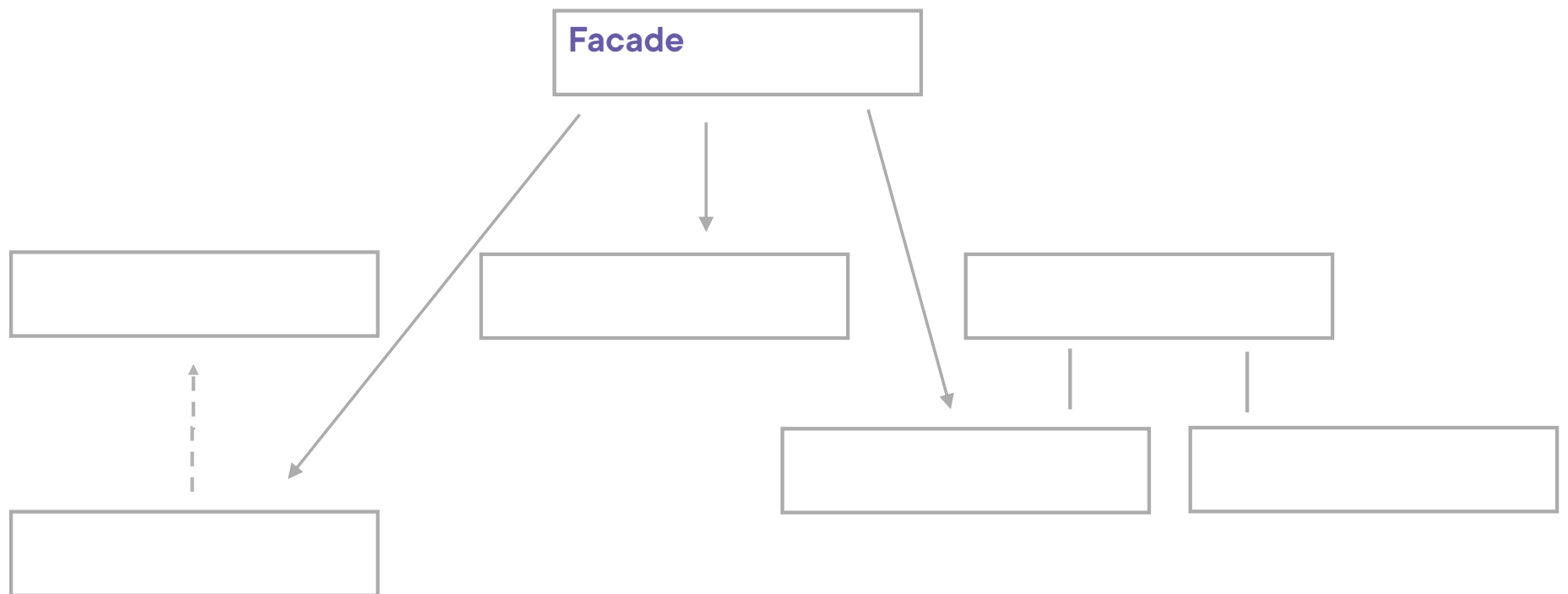




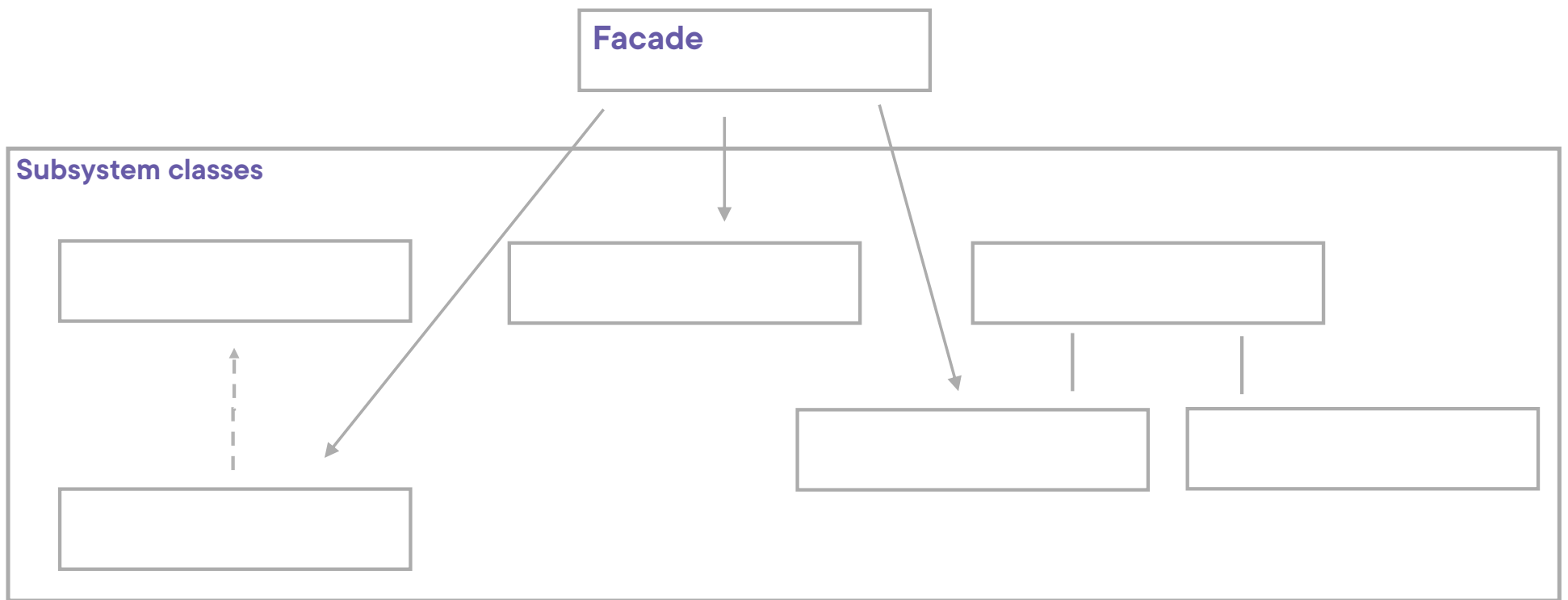
Facade knows which subsystem classes are responsible for a request, and delegates client requests to appropriate subsystem objects



Structure of the Facade Pattern



Structure of the Facade Pattern

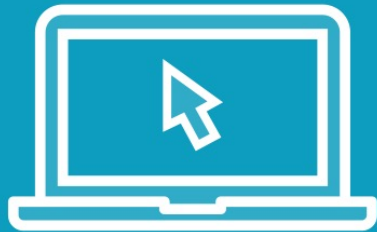




Each **subsystem class** implements subsystem functionality. They don't know about the **Facade**, but they do handle work assigned by it.



Demo



Implementing the facade pattern



Use Cases for the Composition Pattern



When you want to provide a simple interface into a complex subsystem



When there are many dependencies between a client and the implementation classes of the abstraction



Pattern Consequences



The number of objects clients have to deal with are reduced



It promotes weak coupling between the subsystem and its clients, enabling subsystem components to vary without affecting the client: **open/closed principle**



Clients are not forbidden to use subsystem classes

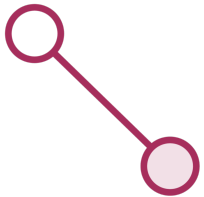


Related Patterns



Abstract factory

Can provide an interface for creating subsystem objects



Mediator

Also abstracts functionality of existing classes, but its purpose is abstracting communication between objects, while facade is about promoting easy of use.



Adapter

Adapter makes existing interfaces useable by wrapping one object, while with facade you're defining a new interface for an entire subsystem.



Summary



Intent of the facade pattern:

- Make it easier for a client to use subsystems by providing one or more interfaces into those subsystems

Higher-level layer to promote ease of use



Up Next:
Structural Pattern: Proxy

