Design Patterns in C# Made Simple

Module 4 Constructing Flexible Behavior with the Strategy Pattern

Zoran Horvat
CEO at Coding Helmet

http://codinghelmet.com
zh@codinghelmet.com
zoranh75

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Full Screen

Search (Ctrl+Q)

TakeTwoOffer.cs* Book.cs Clip01Demo.cs

Demo | Demo.Clip01.TakeTwoOffer | DeductFromCheaper(decimal amount)

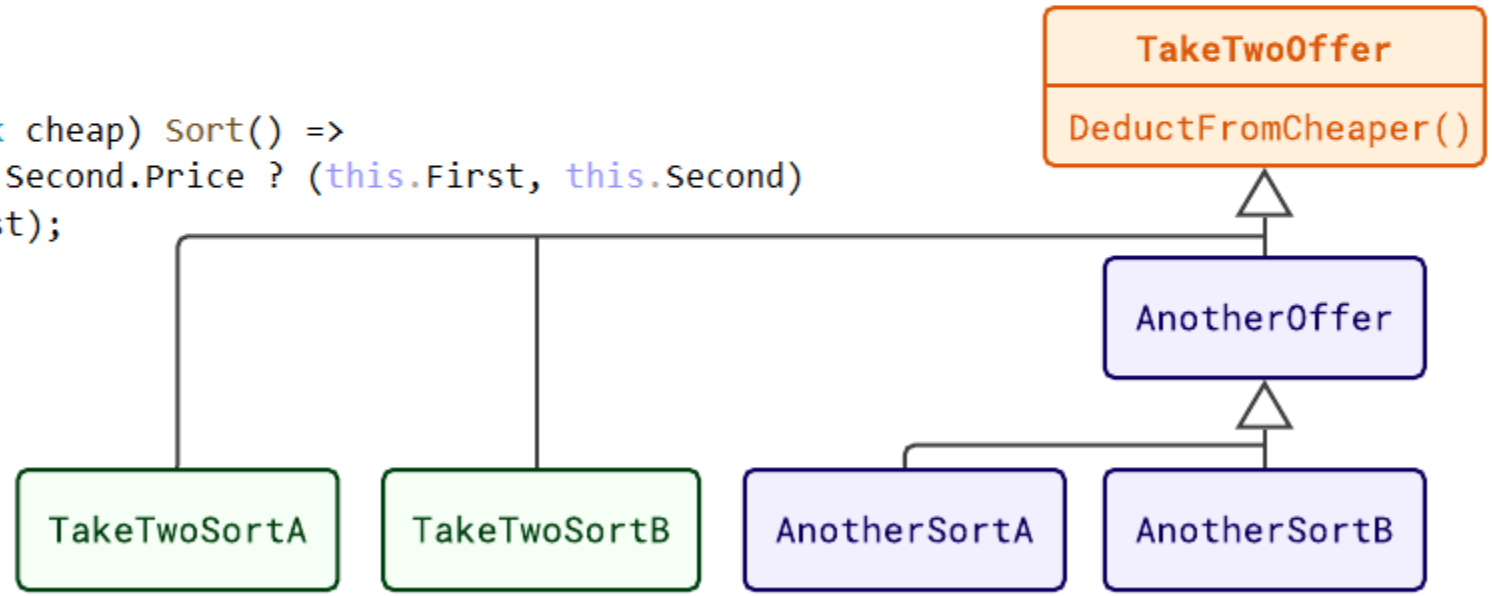
```
        1 reference
        public (Book first, Book second) Apply() => this.DeductFromCheaper(7);

        1 reference
        protected virtual (Book first, Book second) DeductFromCheaper(decimal amount)
        {
            var books = this.Sort();
            return (
                books.expensive,
                books.cheap.WithEffectivePrice(books.cheap.Price.SubtractAmount(amount)));
        }

        1 reference
        private (Book expensive, Book cheap) Sort() =>
            this.First.Price >= this.Second.Price ? (this.First, this.Second)
            : (this.Second, this.First);
    }
}
```

Axis of change #1

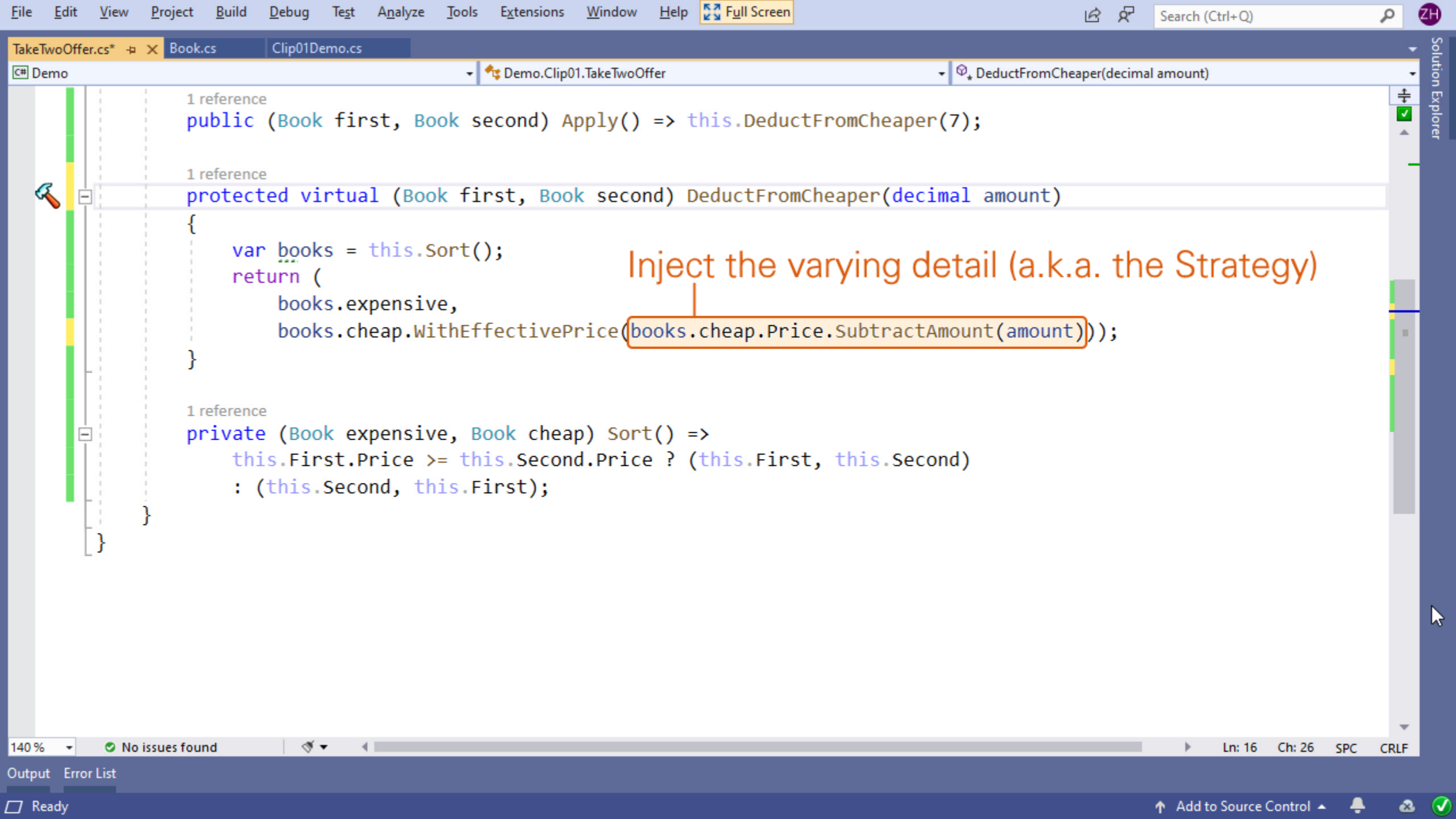
Axis of change #2



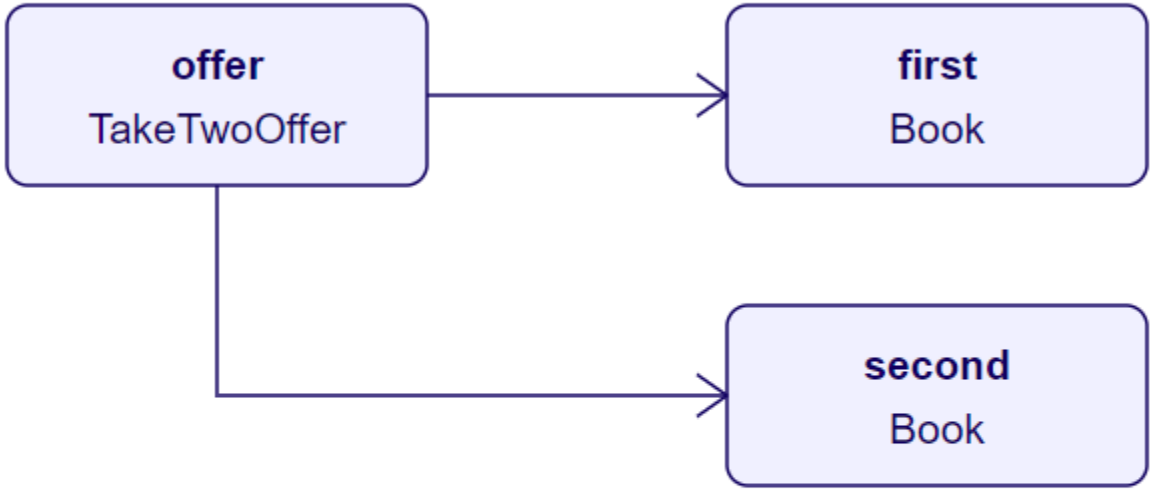
140 % No issues found Ln: 16 Ch: 26 SPC CRLF

Output Error List

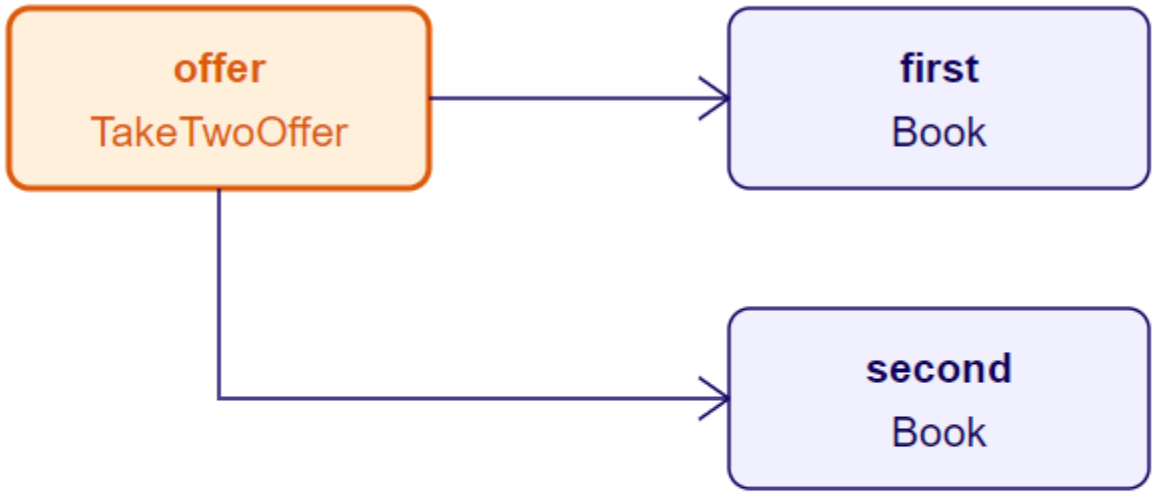
Ready Add to Source Control


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Full Screen
Search (Ctrl+Q)
TakeTwoOffer.cs* Book.cs Clip01Demo.cs
Demo
Demo.Clip01.TakeTwoOffer
DeductFromCheaper(decimal amount)
1 reference
public (Book first, Book second) Apply() => this.DeductFromCheaper(7);
1 reference
protected virtual (Book first, Book second) DeductFromCheaper(decimal amount)
{
var books = this.Sort();
return (
books.expensive,
books.cheap.WithEffectivePrice(books.cheap.Price.SubtractAmount(amount)));
}
1 reference
private (Book expensive, Book cheap) Sort() =>
this.First.Price >= this.Second.Price ? (this.First, this.Second)
: (this.Second, this.First);
}
}
Inject the varying detail (a.k.a. the Strategy)
140 %
No issues found
Ln: 16
Ch: 26
SPC
CRLF
Output
Error List
Ready
Add to Source Control
Solution Explorer

Motivation to Implement Strategy

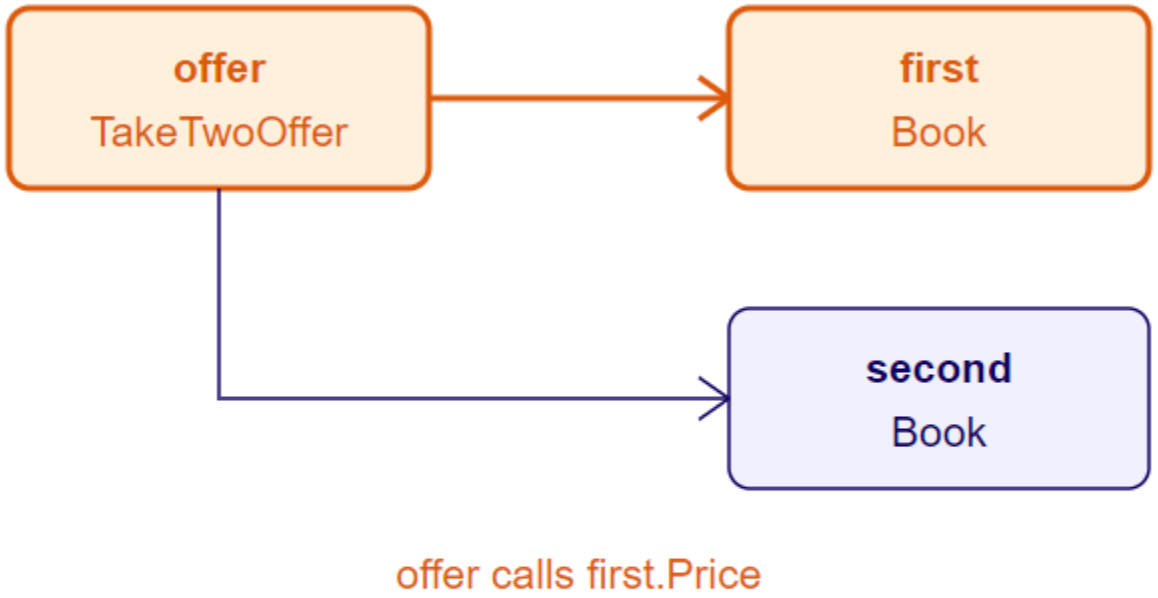


Motivation to Implement Strategy

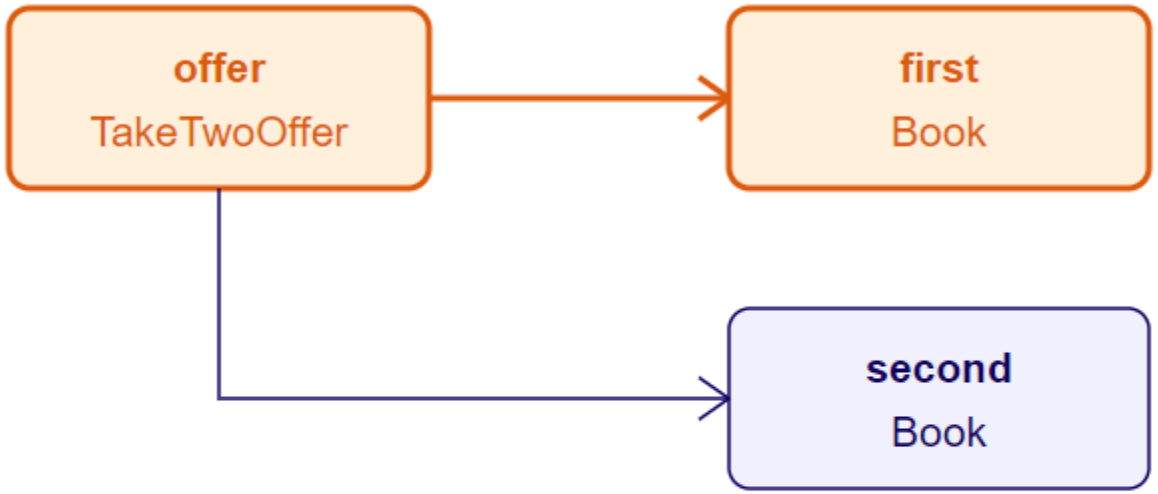


External code calls offer.Apply()

Motivation to Implement Strategy

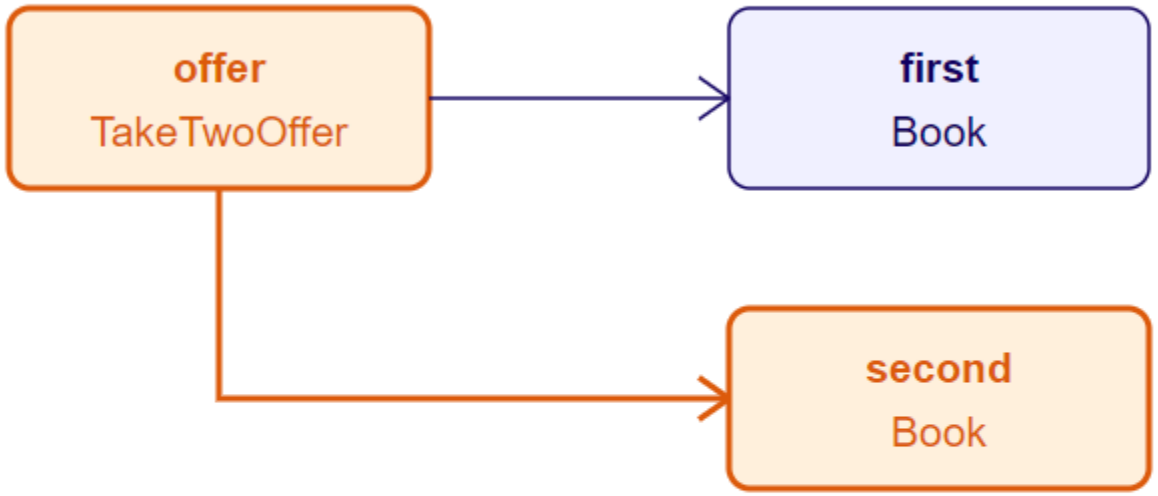


Motivation to Implement Strategy



first.Price returns 9.00 USD

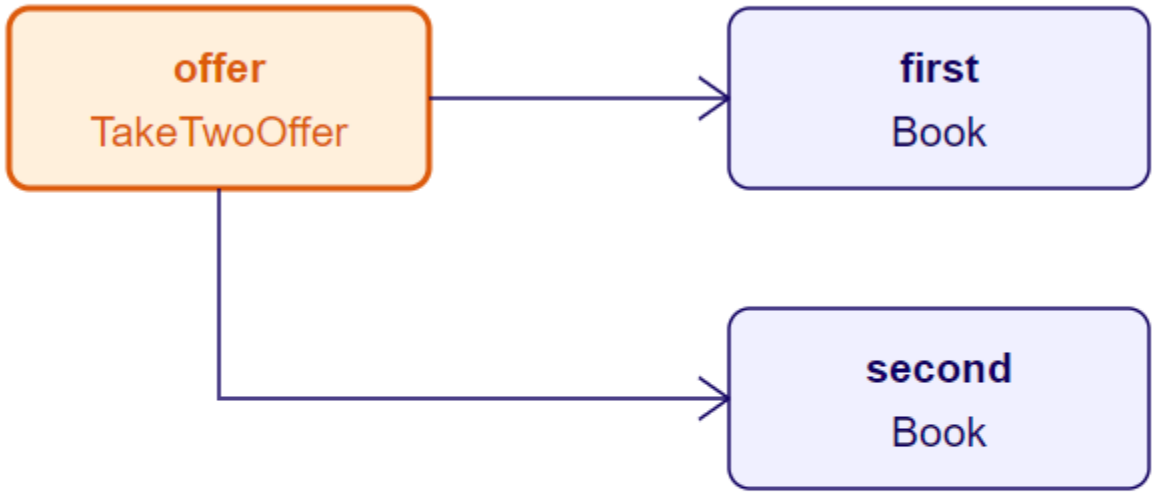
Motivation to Implement Strategy



offer calls second.Price

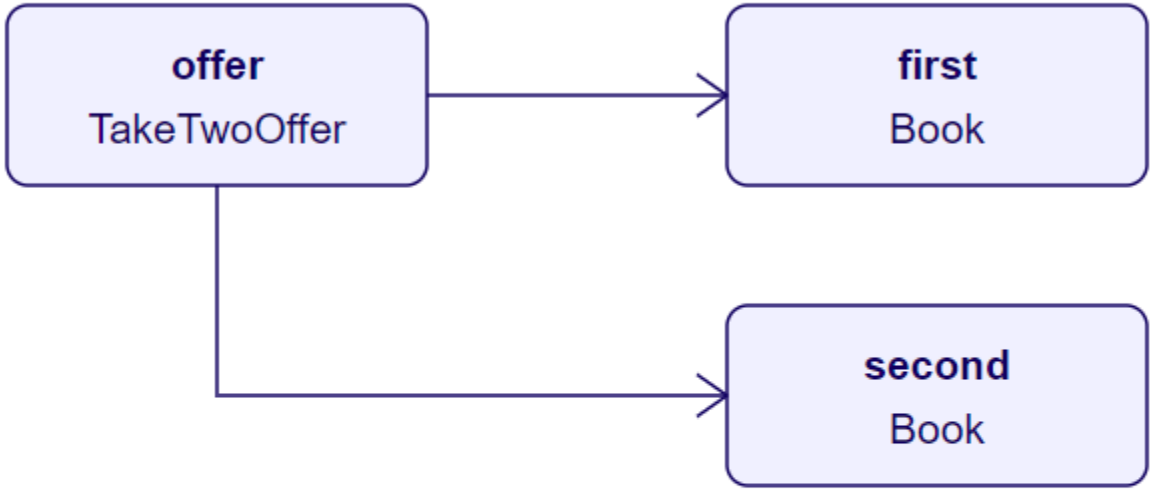
Motivation to Implement Strategy

Motivation to Implement Strategy

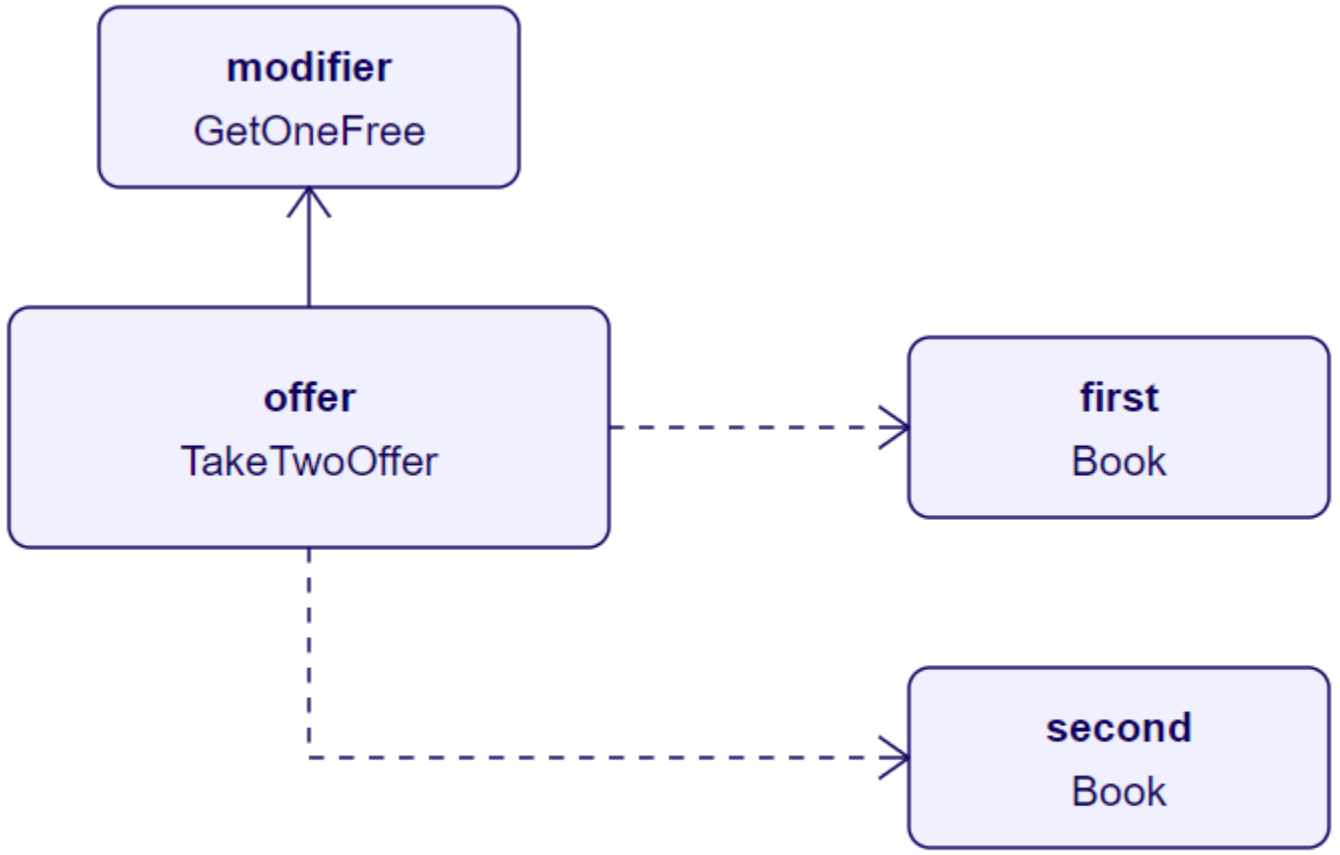


offer.Apply() returns (Design Patterns 35.00 USD, The Little Prince 2.00 USD (Was 9.00 USD))

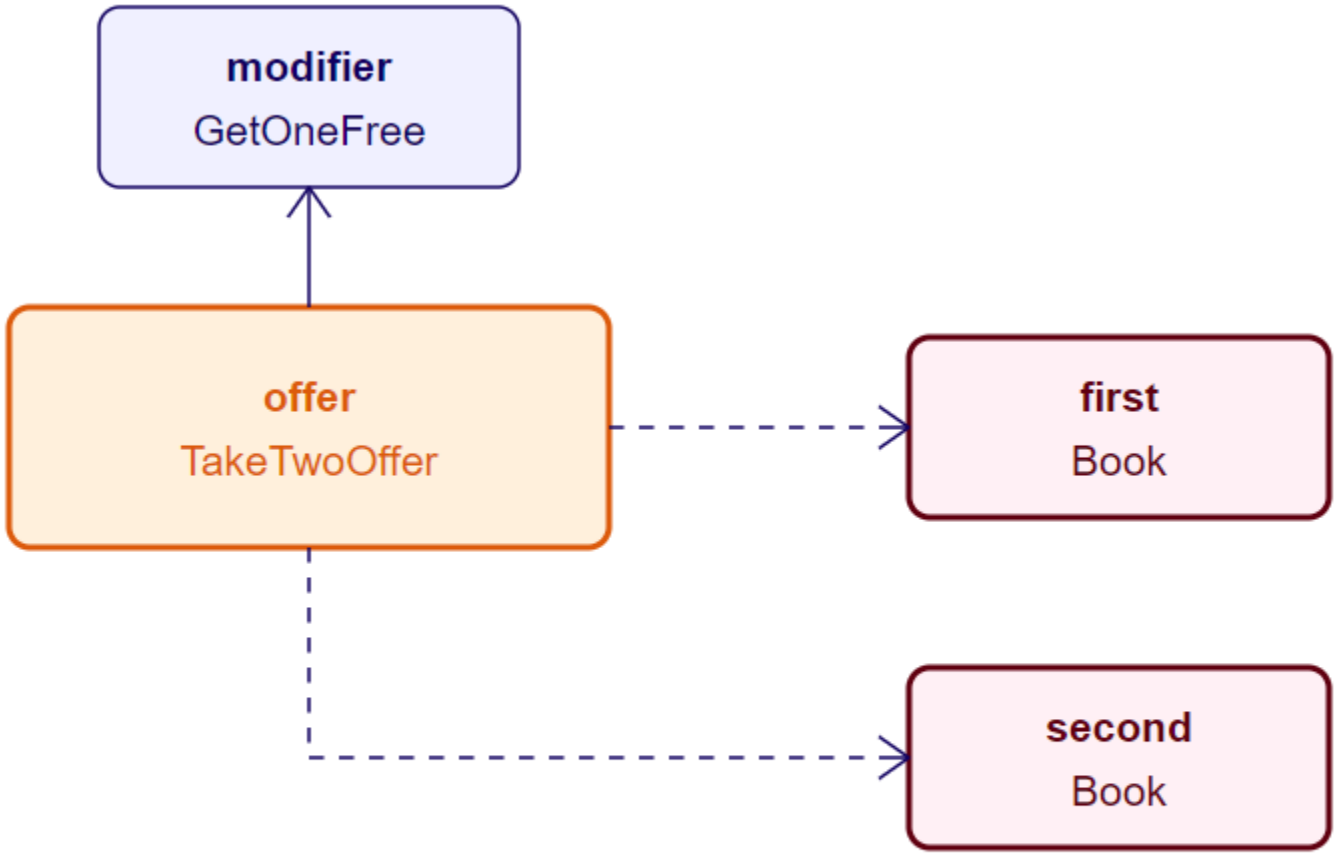
Motivation to Implement Strategy



Common Strategy Implementation

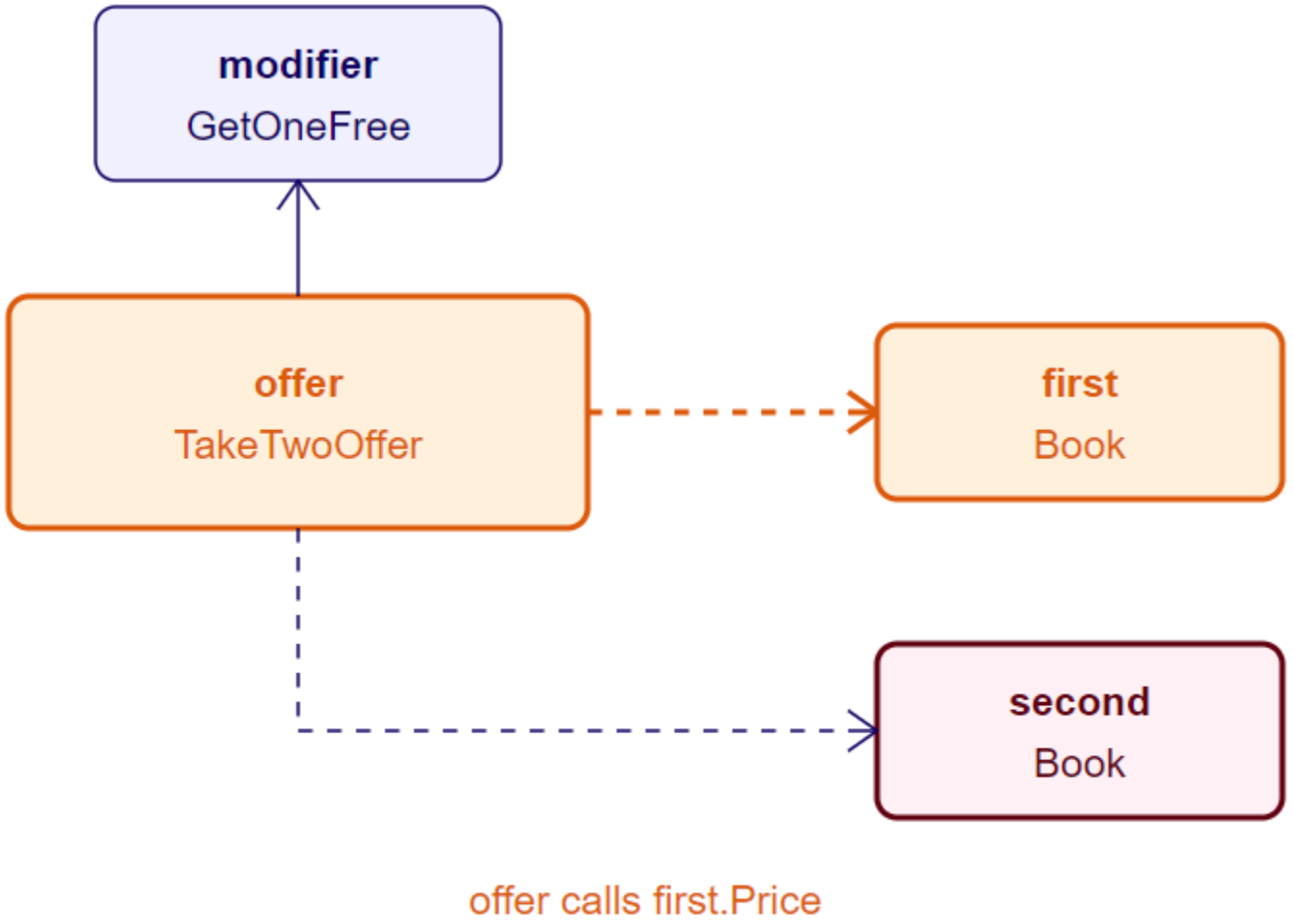


Common Strategy Implementation

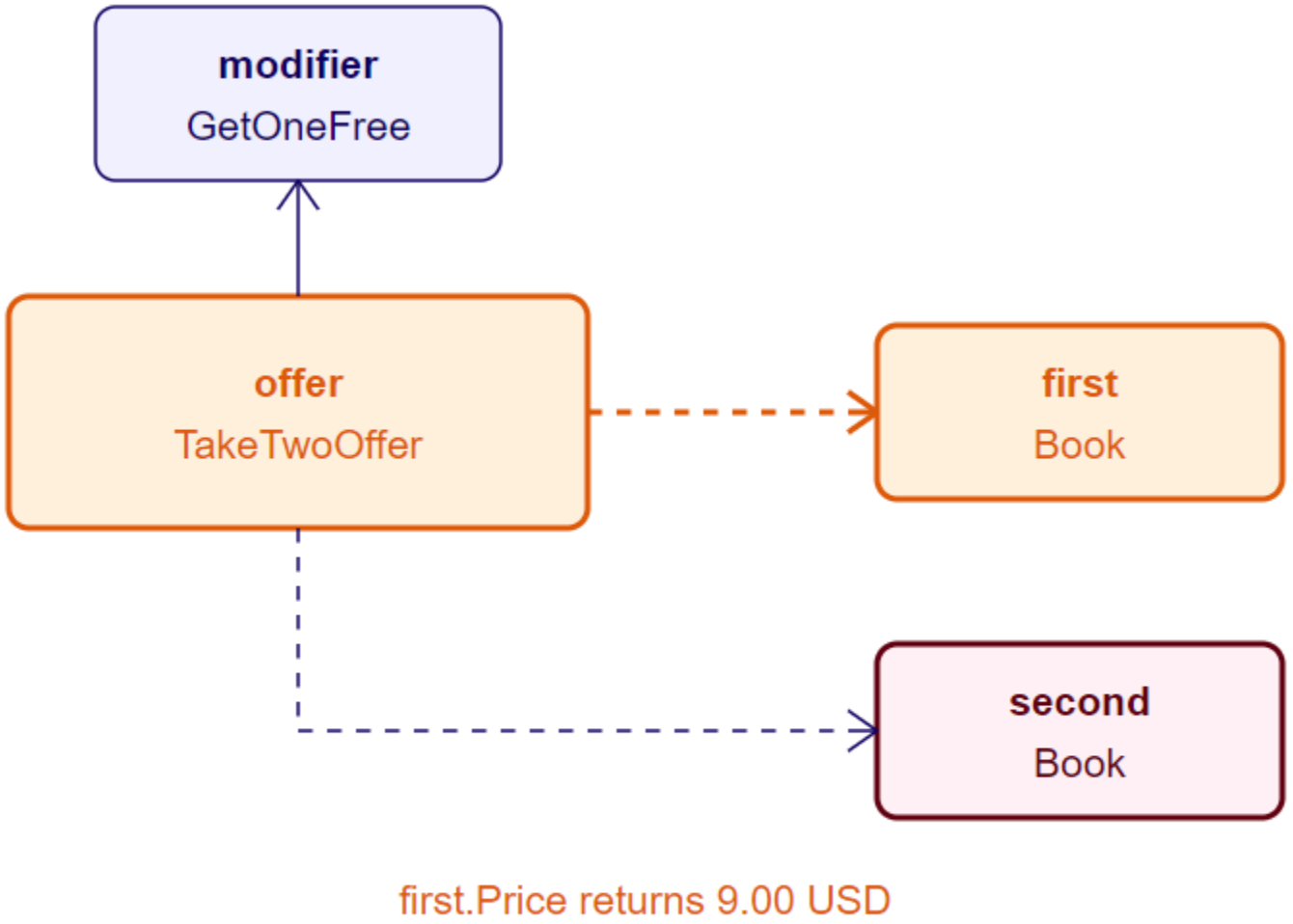


External code calls offer.ApplyTo(first, second)

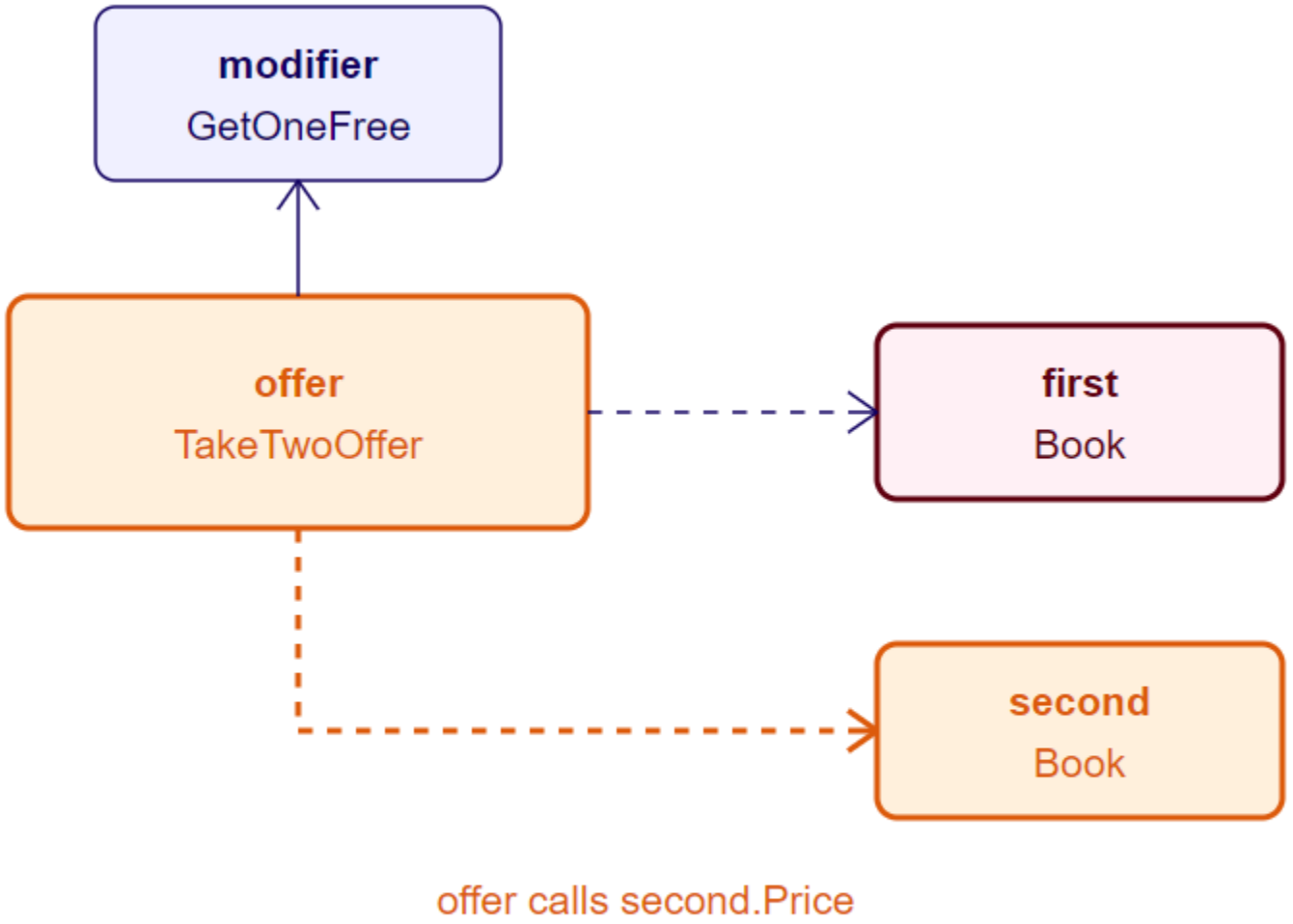
Common Strategy Implementation



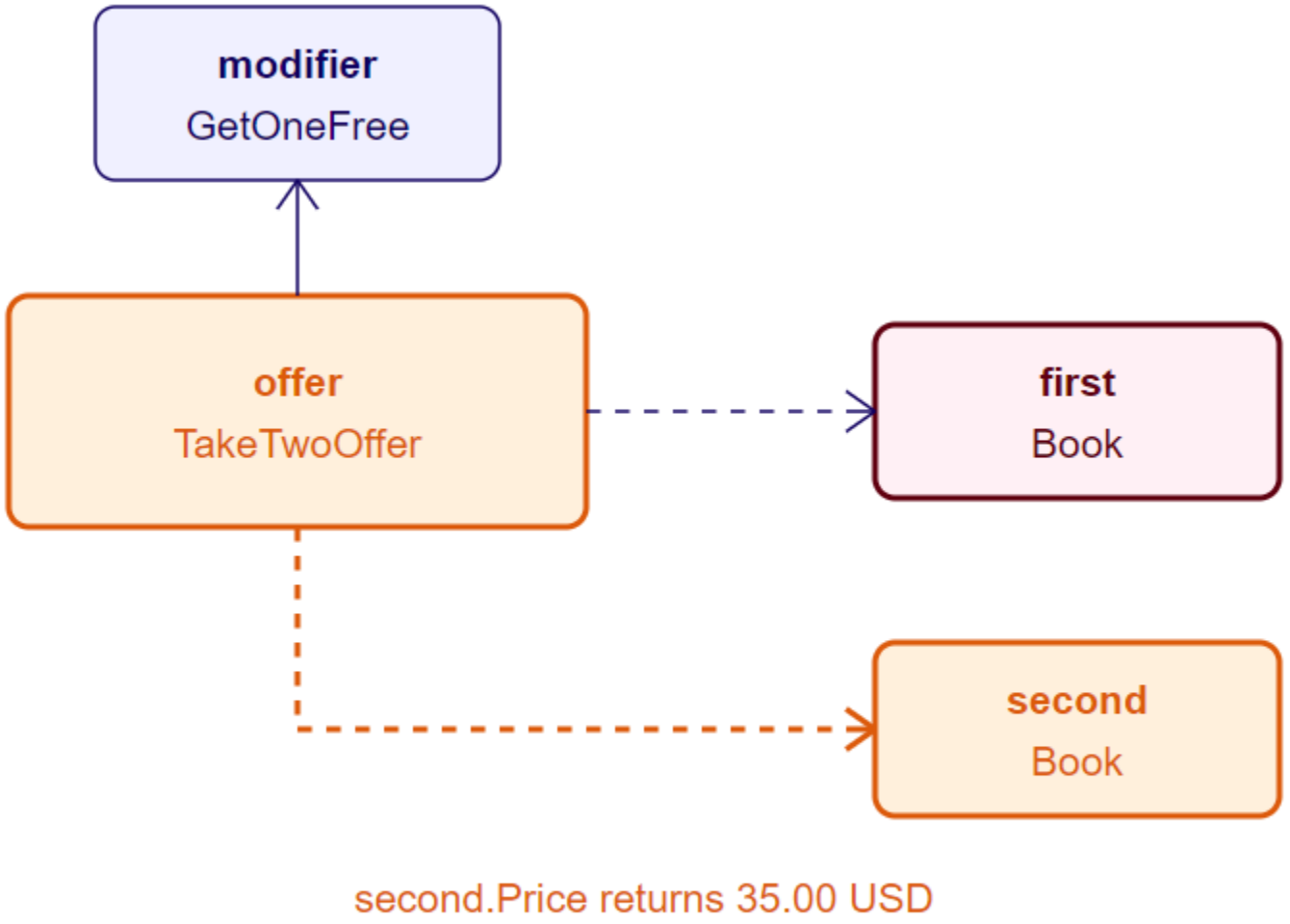
Common Strategy Implementation



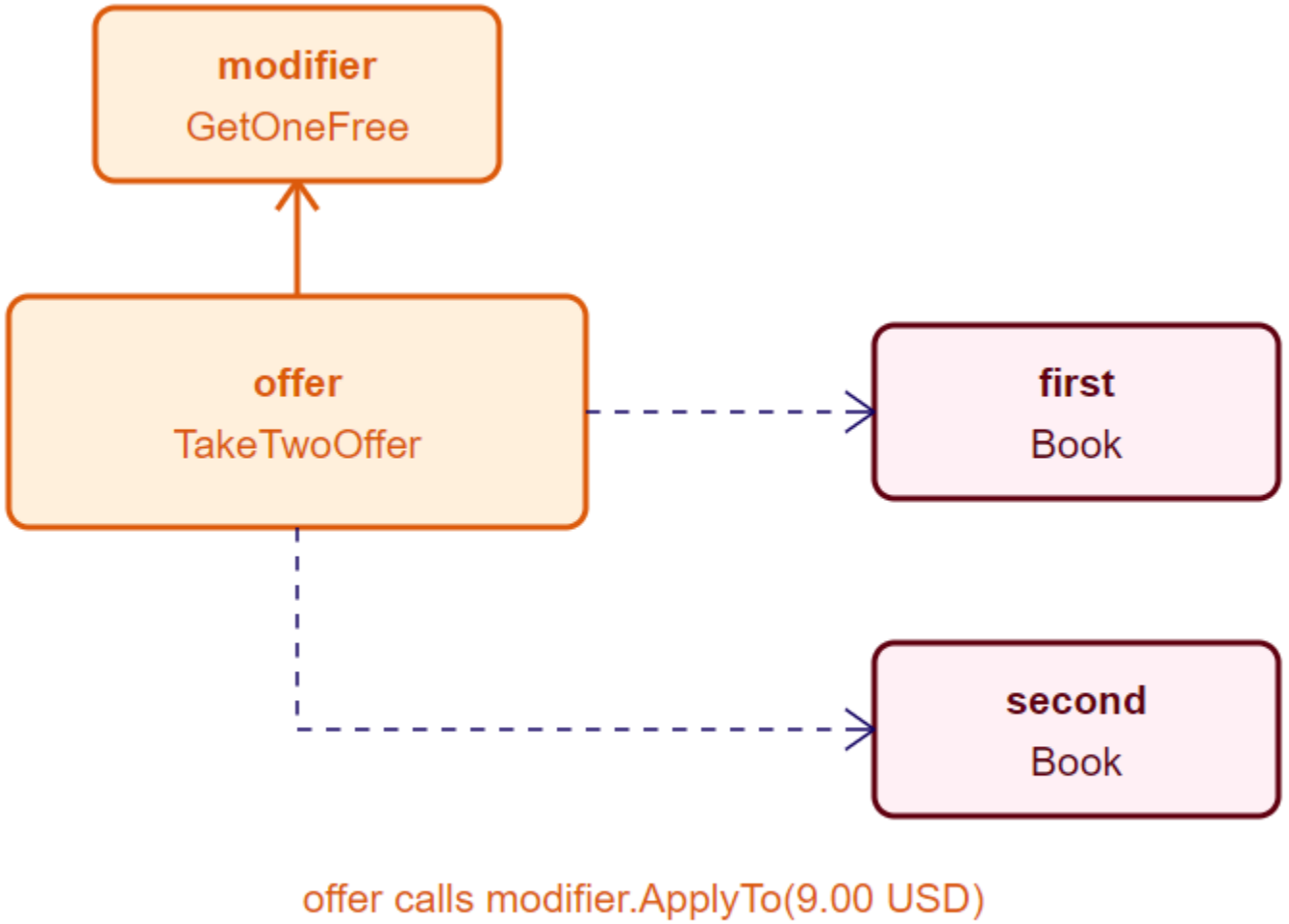
Common Strategy Implementation



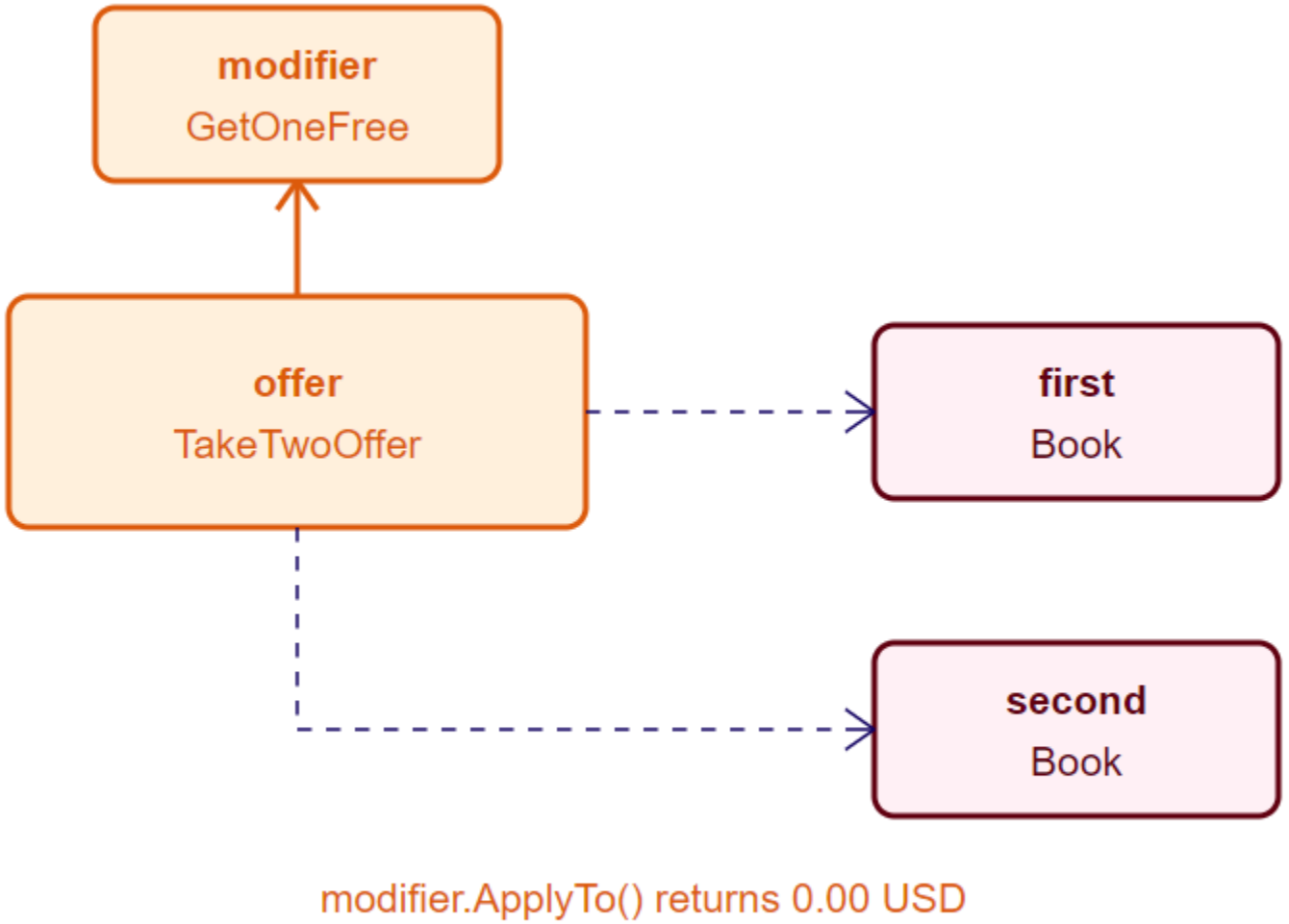
Common Strategy Implementation



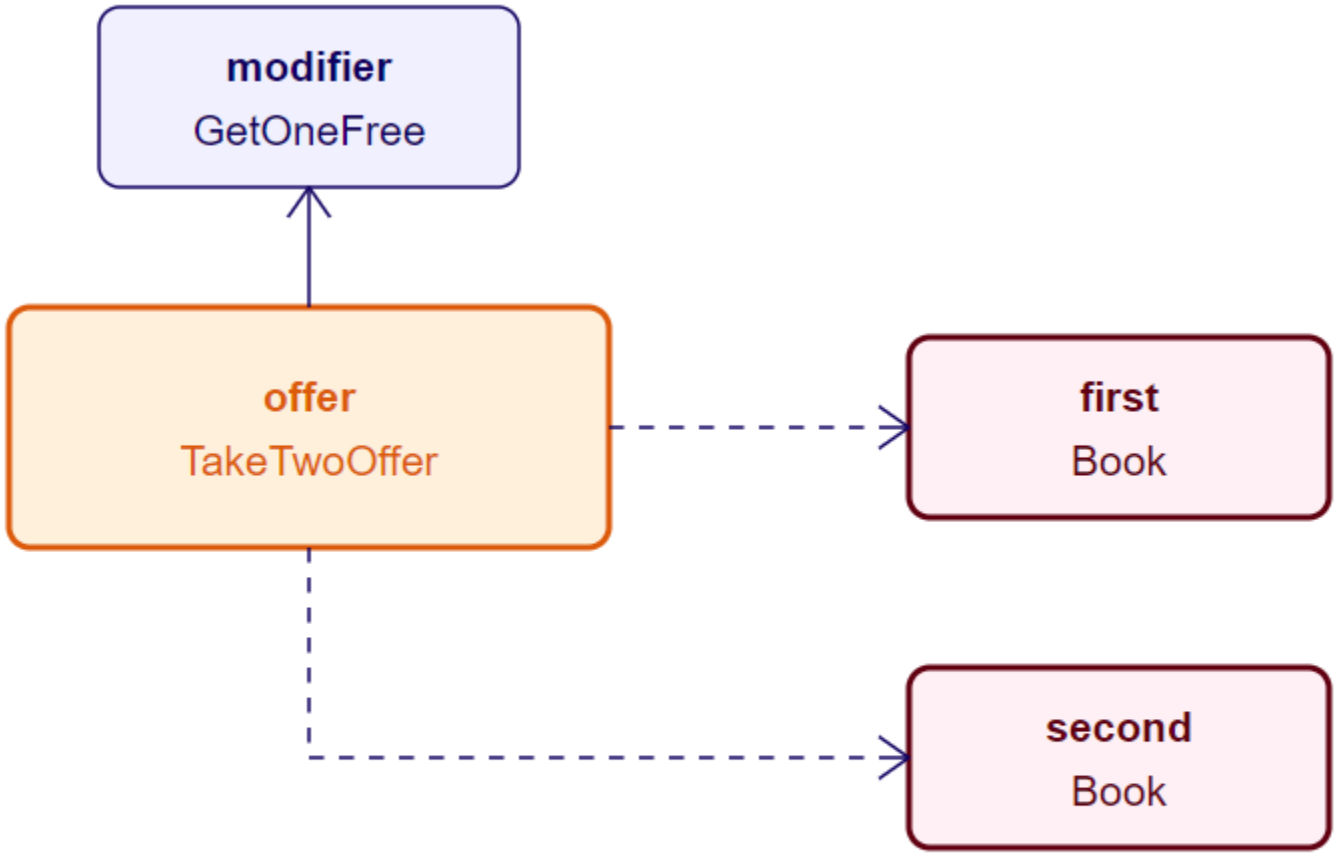
Common Strategy Implementation



Common Strategy Implementation

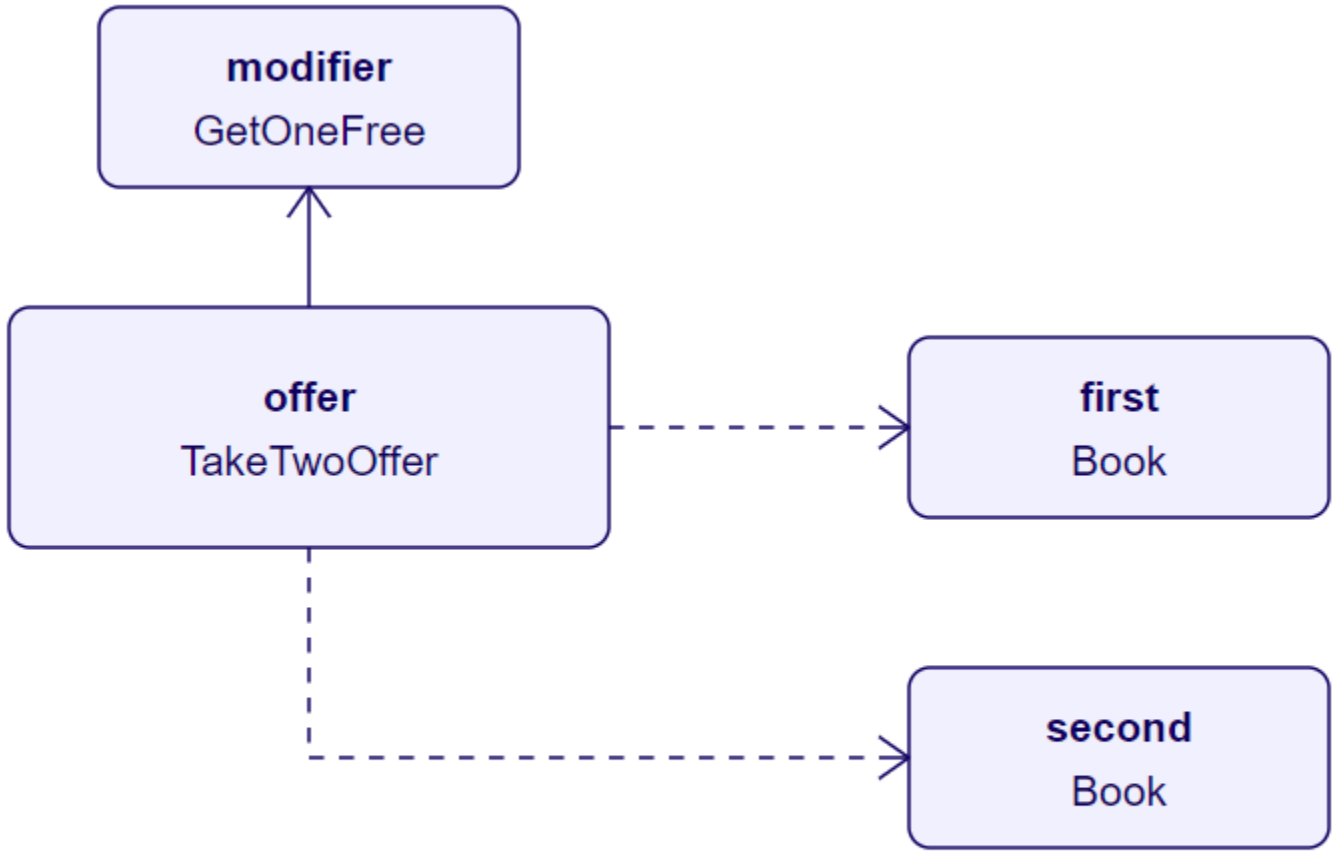


Common Strategy Implementation

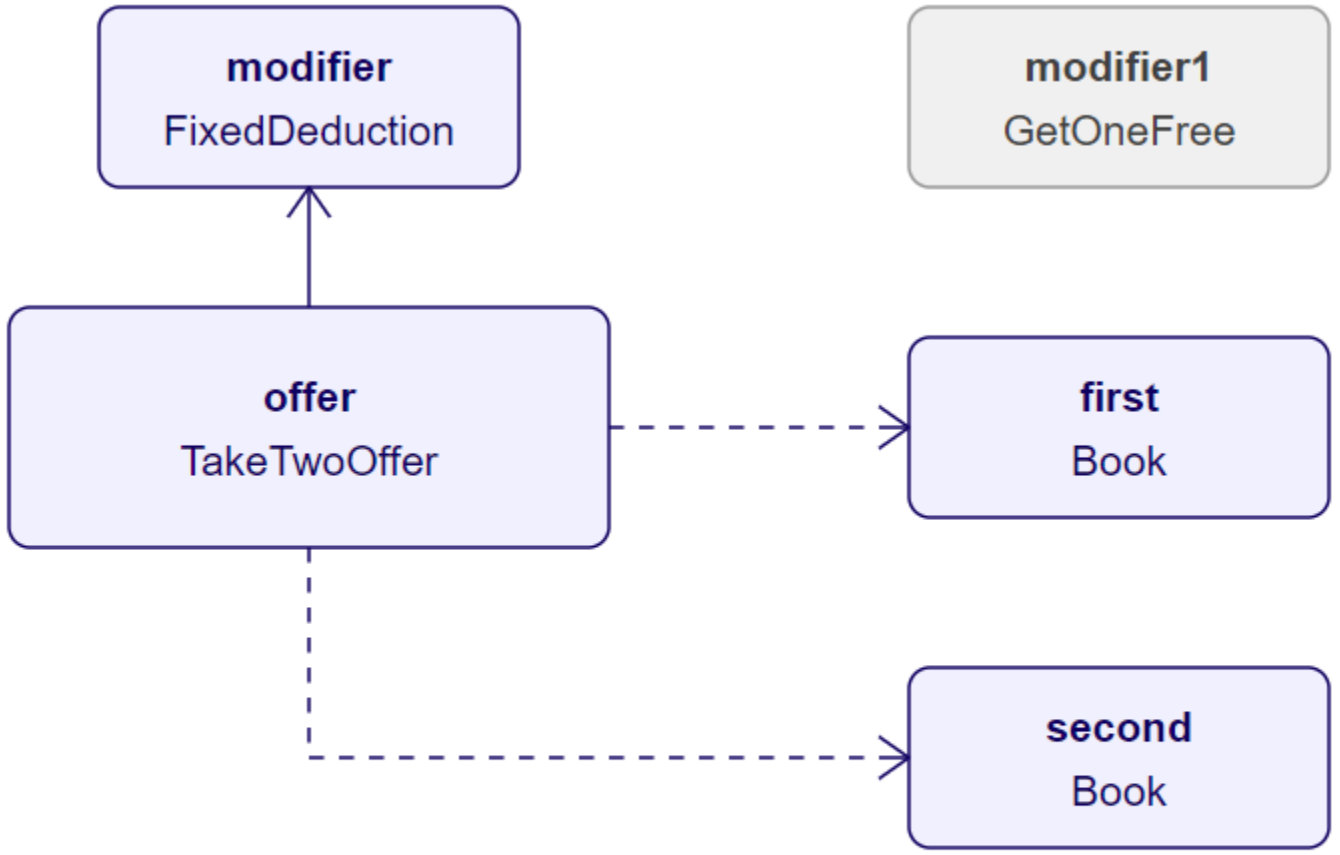


offer.ApplyTo() returns (Design Patterns 35.00 USD, The Little Prince 0.00 USD (Was 9.00 USD))

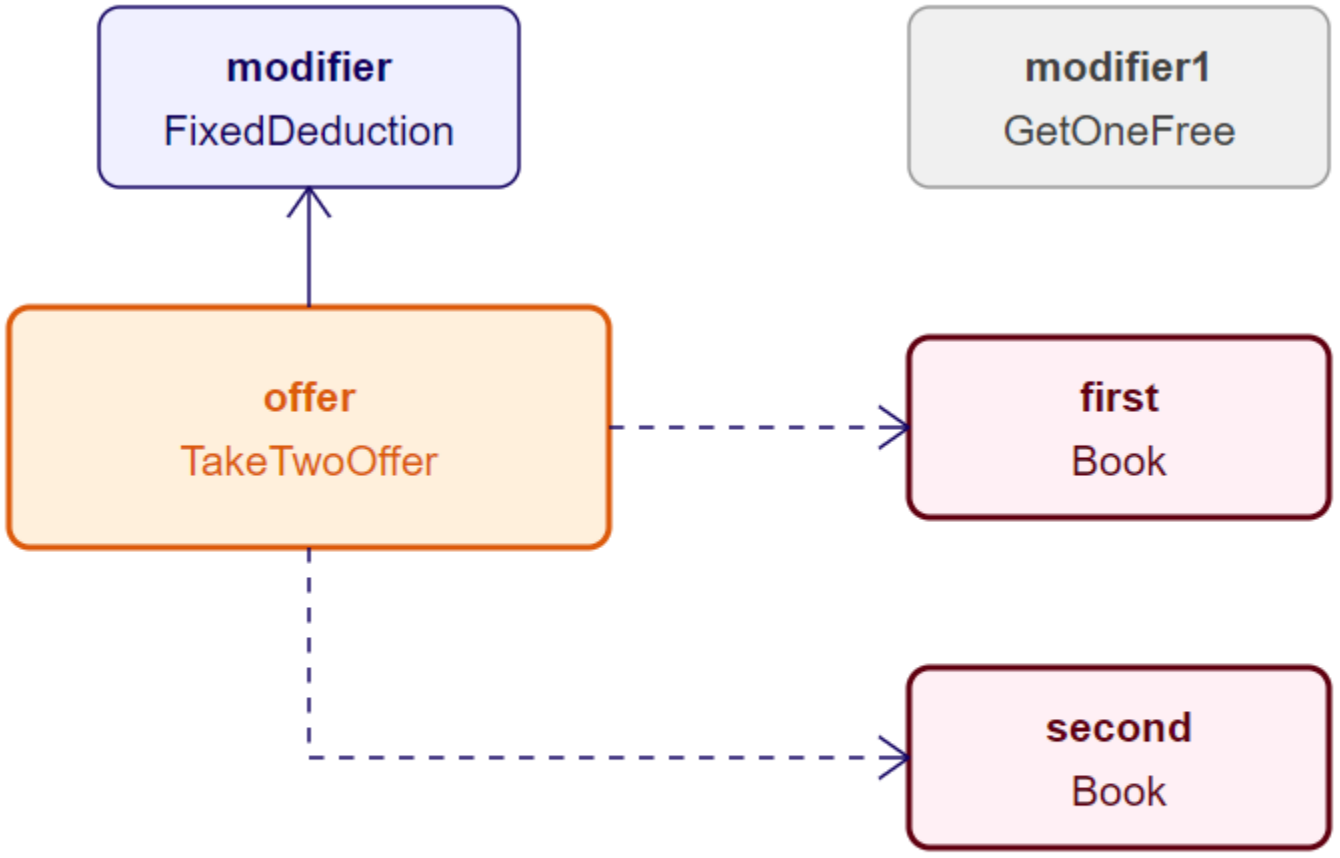
Common Strategy Implementation



Common Strategy Implementation

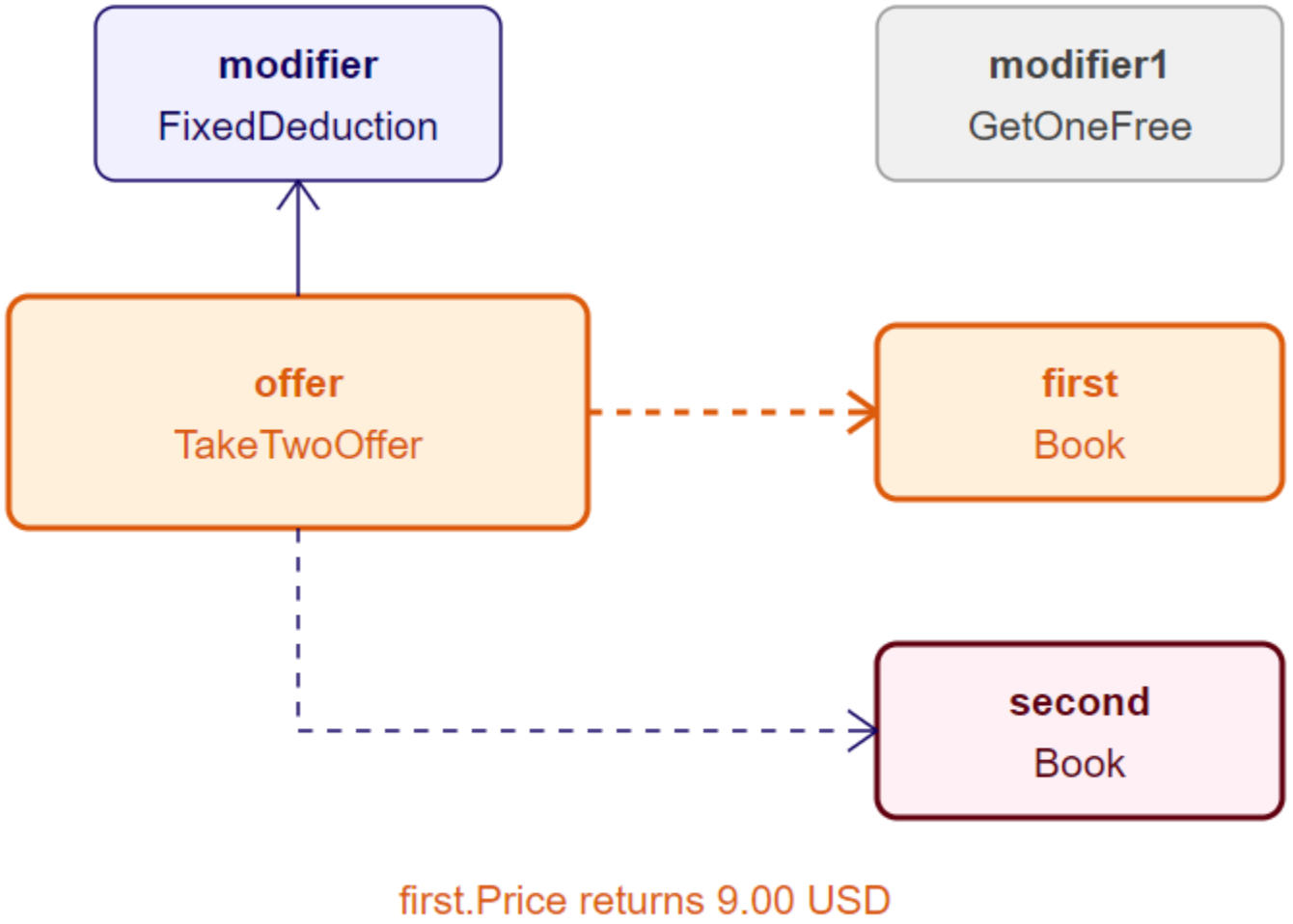


Common Strategy Implementation



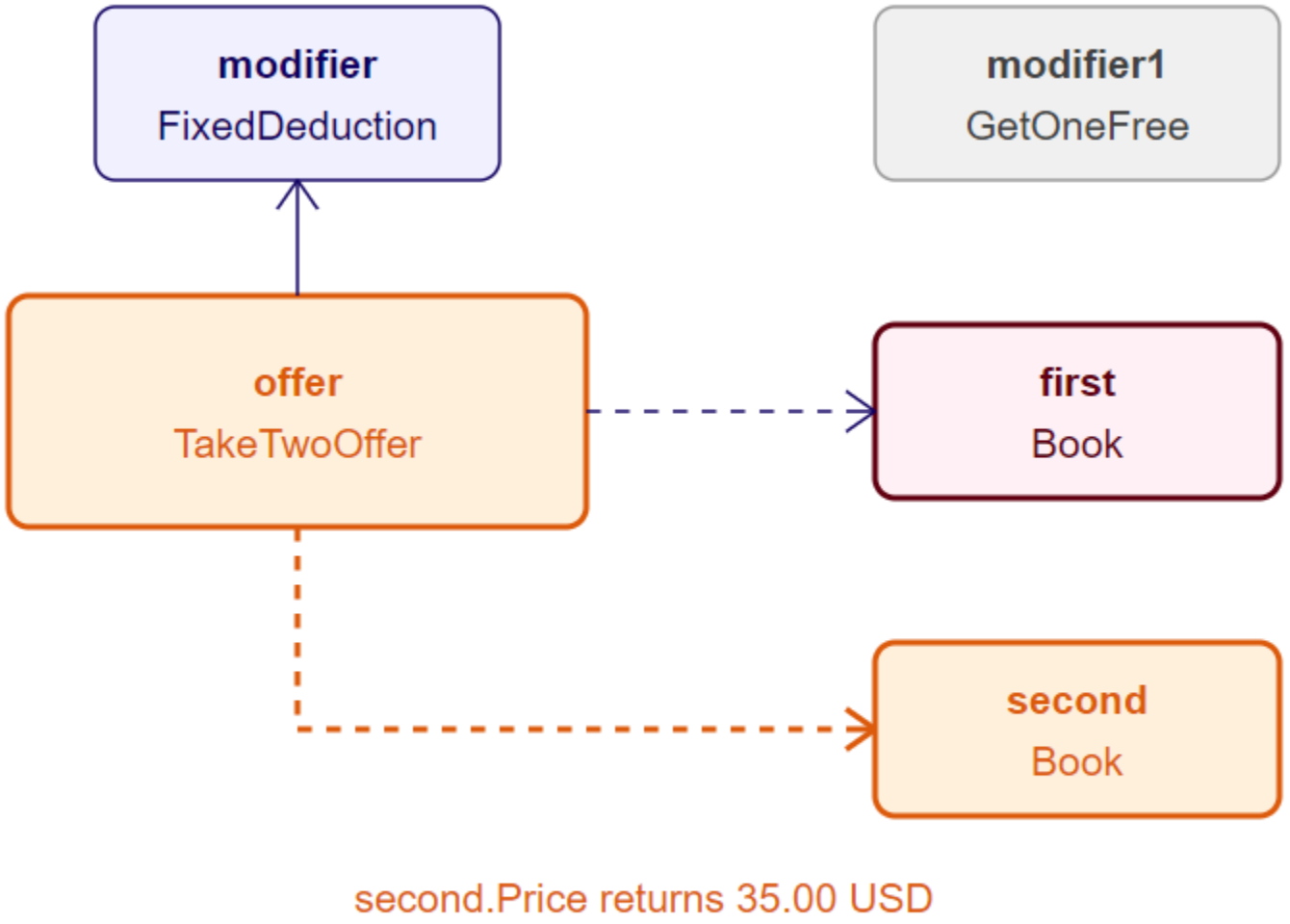
External code calls offer.ApplyTo(first, second)

Common Strategy Implementation

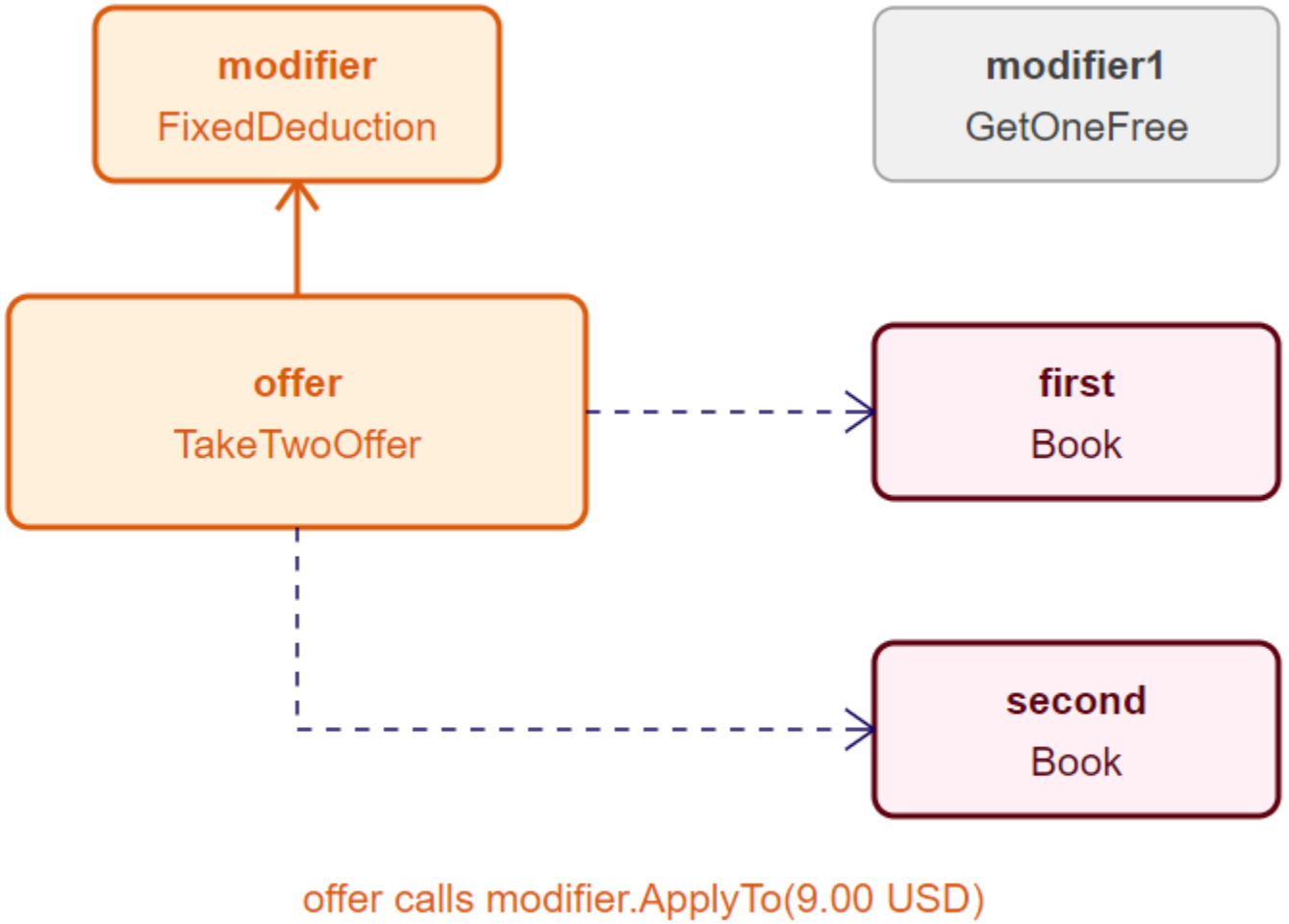


first.Price returns 9.00 USD

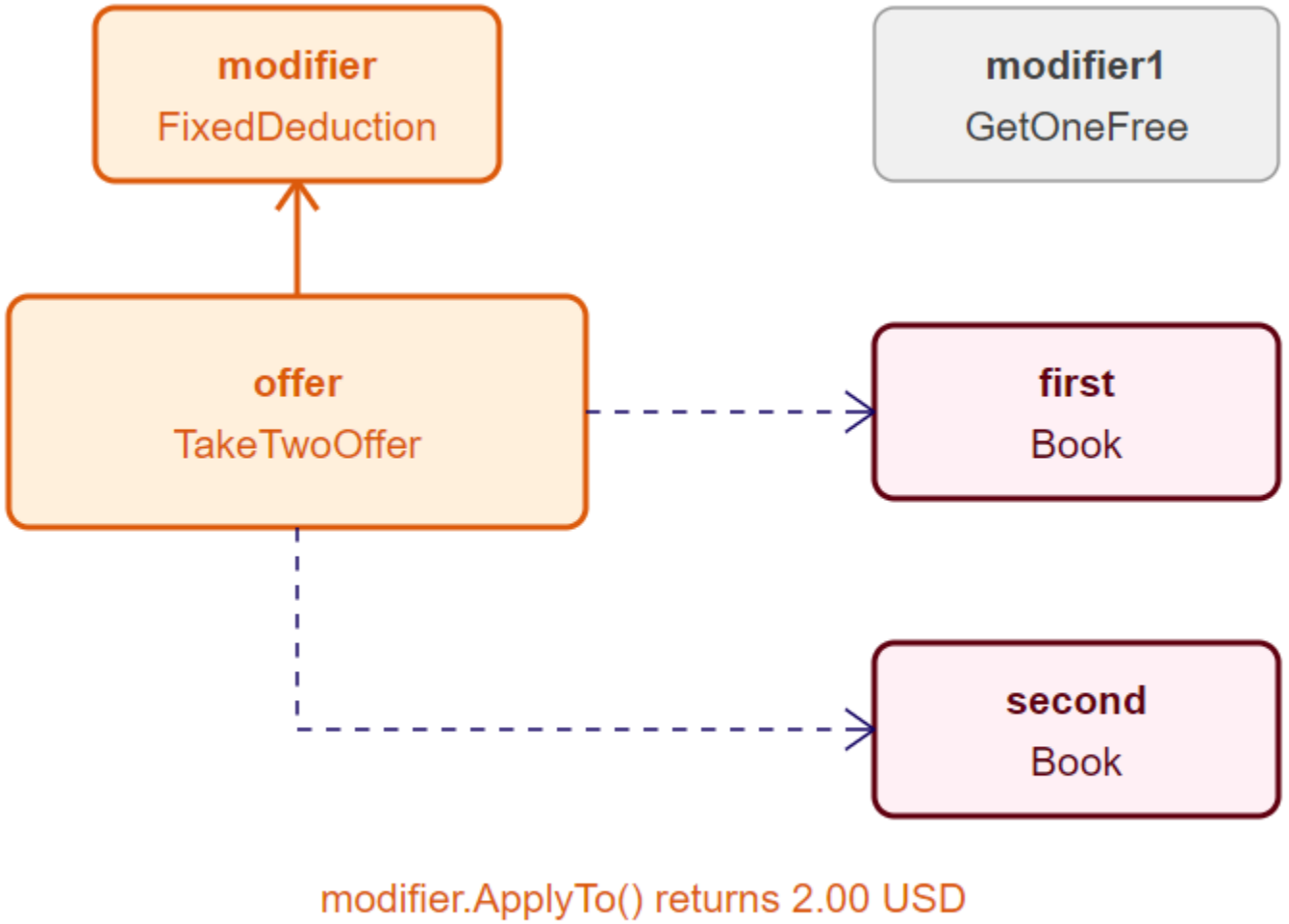
Common Strategy Implementation



Common Strategy Implementation

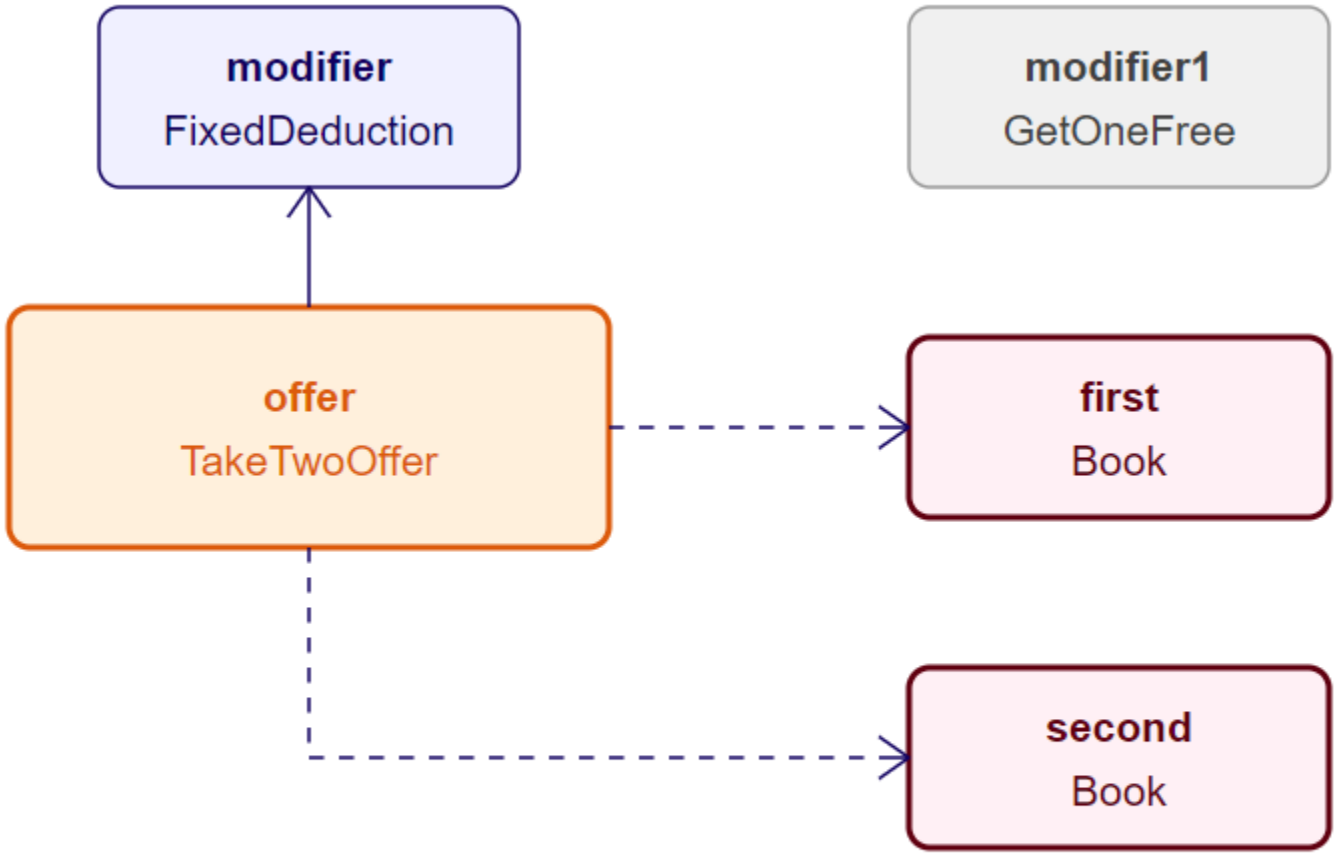


Common Strategy Implementation



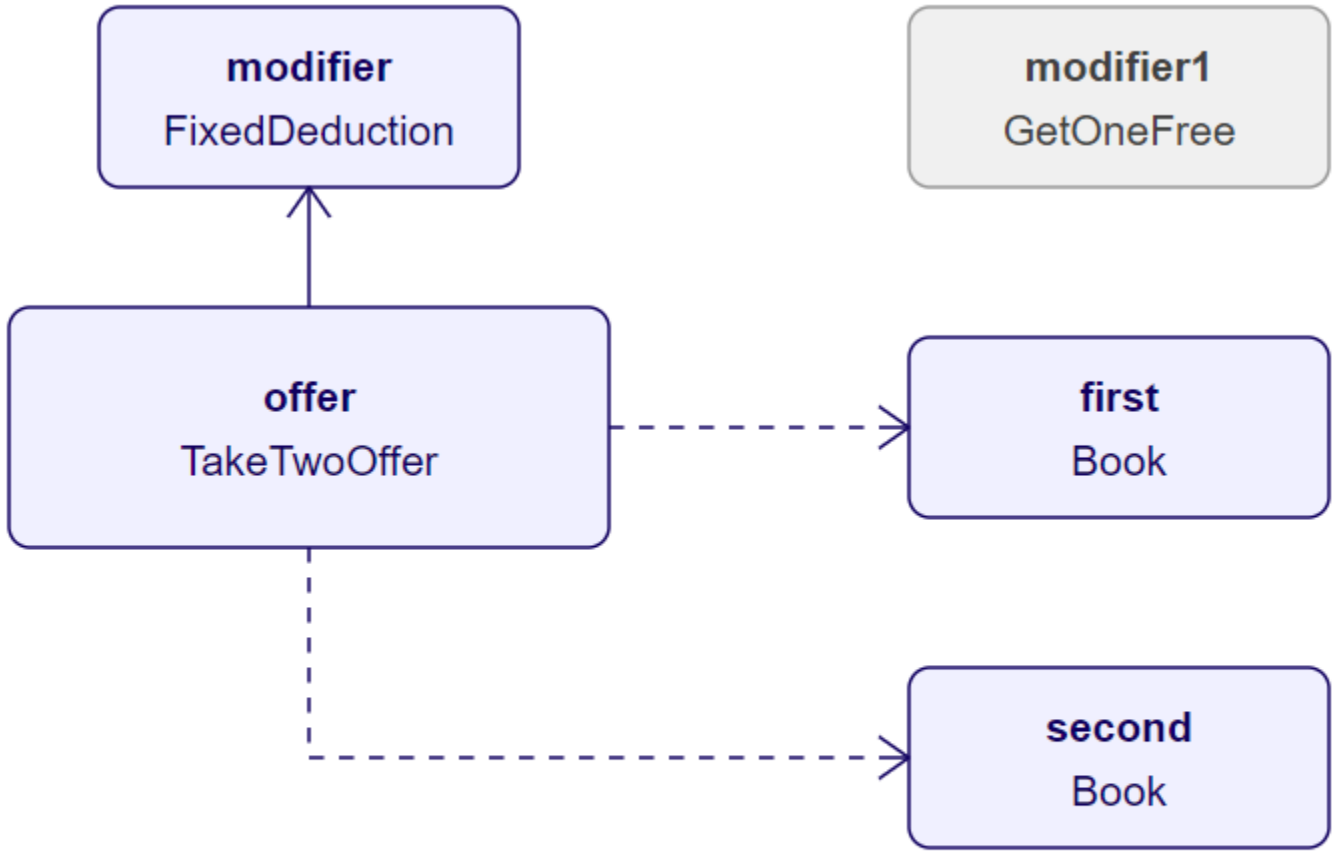
modifier.ApplyTo() returns 2.00 USD

Common Strategy Implementation



offer.ApplyTo() returns (Design Patterns 35.00 USD, The Little Prince 2.00 USD (Was 9.00 USD))

Common Strategy Implementation



C05

Understanding Axes of Change

Applying

Second for free

From second

With spillover

Calculating

Absolute amount

Relative to both

Relative to lower

Classes

Absolute from second

Understanding Axes of Change

Applying

Second for free

From second

With spillover

Calculating

Absolute amount

Relative to both

Relative to lower

Classes

Absolute from second

Relative to both from second

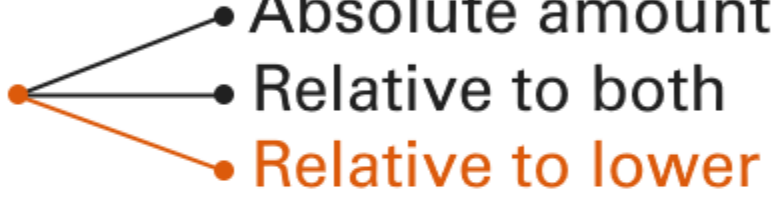
Understanding Axes of Change

Applying

Second for free

From second

With spillover



Calculating

Absolute amount

Relative to both

Relative to lower

Classes

Absolute from second

Relative to both from second

Relative to lower from second

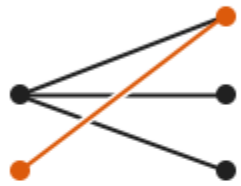
Understanding Axes of Change

Applying

Second for free

From second

With spillover



Calculating

Absolute amount

Relative to both

Relative to lower

Classes

Absolute from second

Relative to both from second

Relative to lower from second

Absolute with spillover

Understanding Axes of Change

Applying

Second for free

From second

With spillover

Calculating

Absolute amount

Relative to both

Relative to lower

Classes

Absolute from second

Relative to both from second

Relative to lower from second

Absolute with spillover

Relative to both with spillover

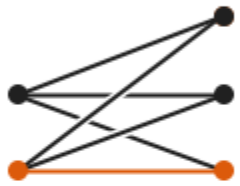
Understanding Axes of Change

Applying

Second for free

From second

With spillover



Calculating

Absolute amount

Relative to both

Relative to lower

Classes

Absolute from second

Relative to both from second

Relative to lower from second

Absolute with spillover

Relative to both with spillover

Relative to lower with spillover

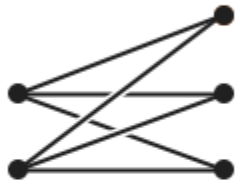
Understanding Axes of Change

Applying

Second for free

From second

With spillover



Calculating

Absolute amount

Relative to both

Relative to lower

Classes

Absolute from second

Relative to both from second

Relative to lower from second

Absolute with spillover

Relative to both with spillover

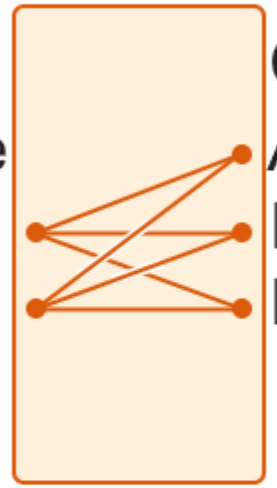
Relative to lower with spillover

Second for free

Understanding Axes of Change

Applying

Second for free
From second
With spillover



Calculating

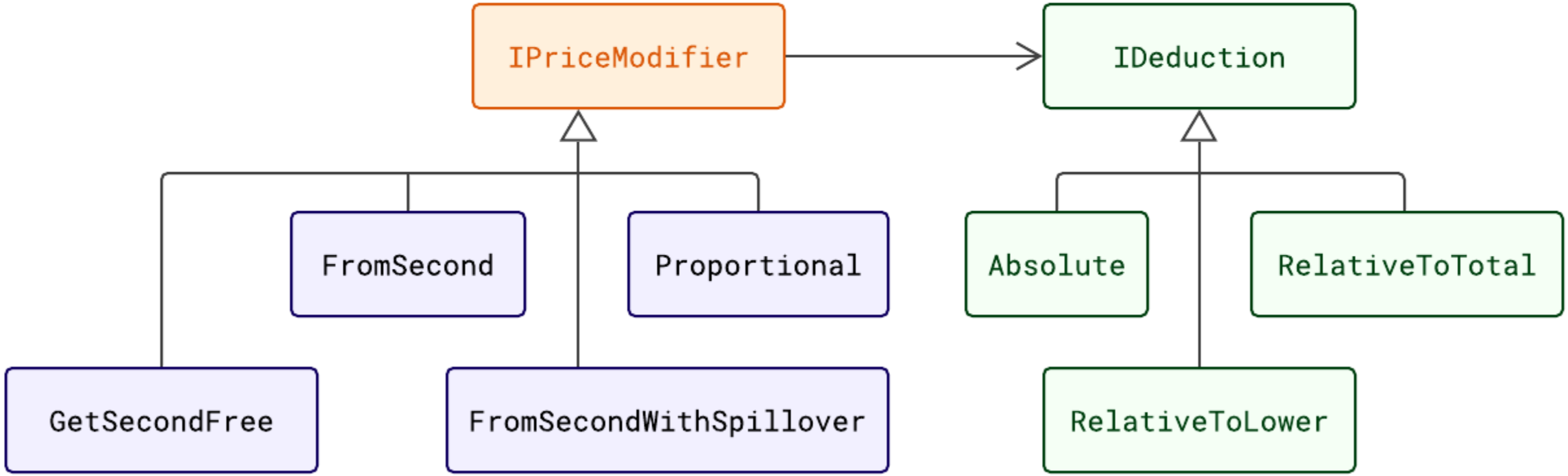
Absolute amount
Relative to both
Relative to lower

Combinatorial explosion of classes

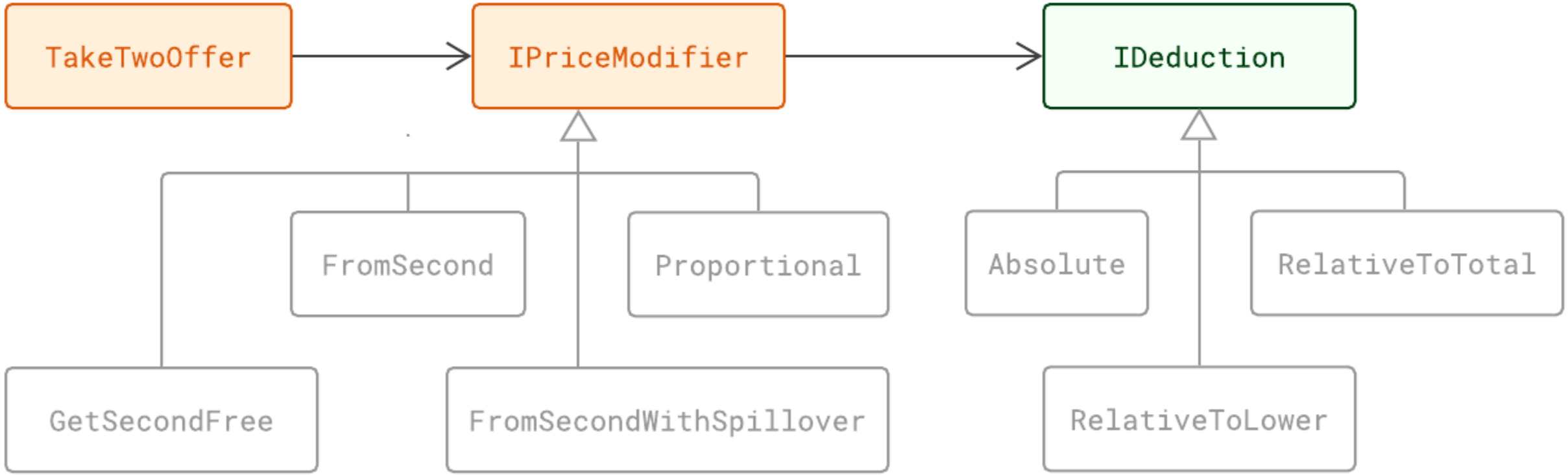
Classes

Absolute from second
Relative to both from second
Relative to lower from second
Absolute with spillover
Relative to both with spillover
Relative to lower with spillover
Second for free

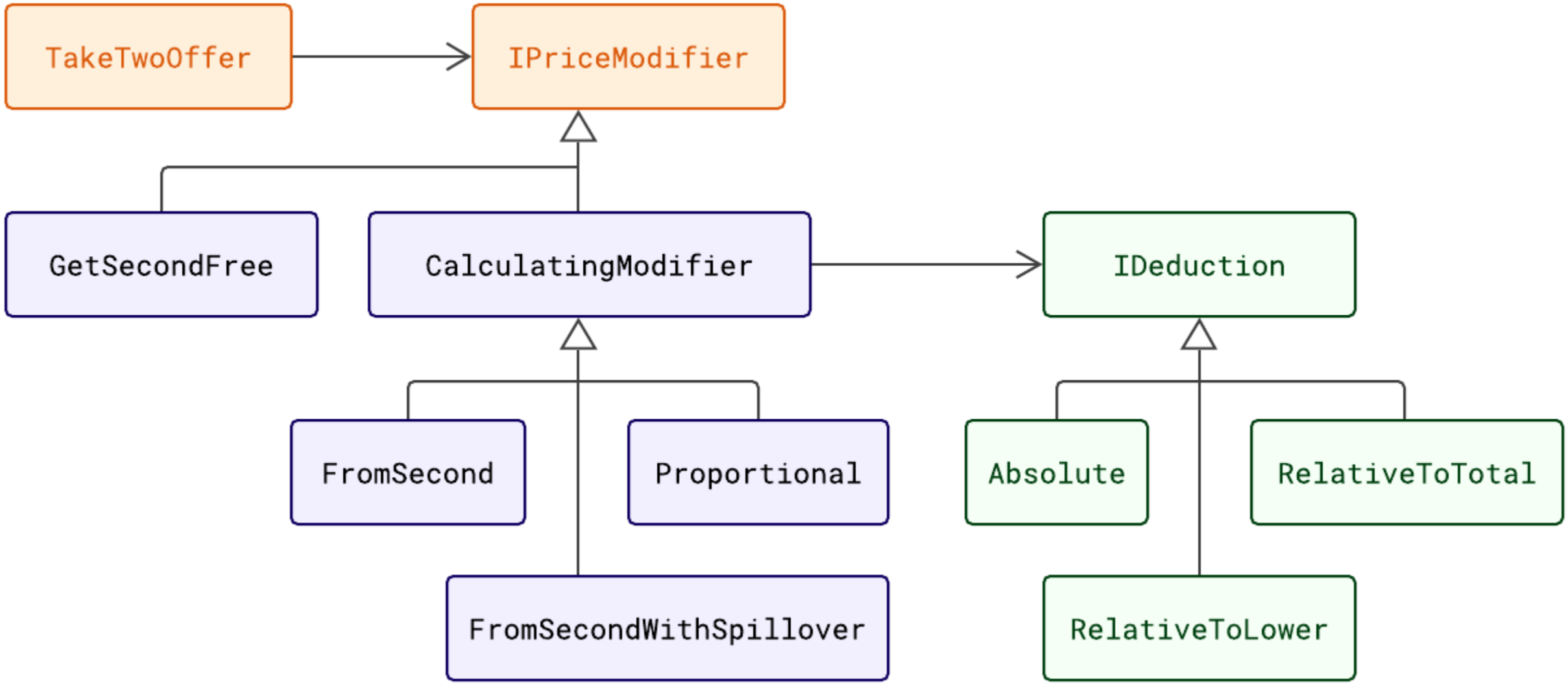
Avoiding Combinatorial Explosion



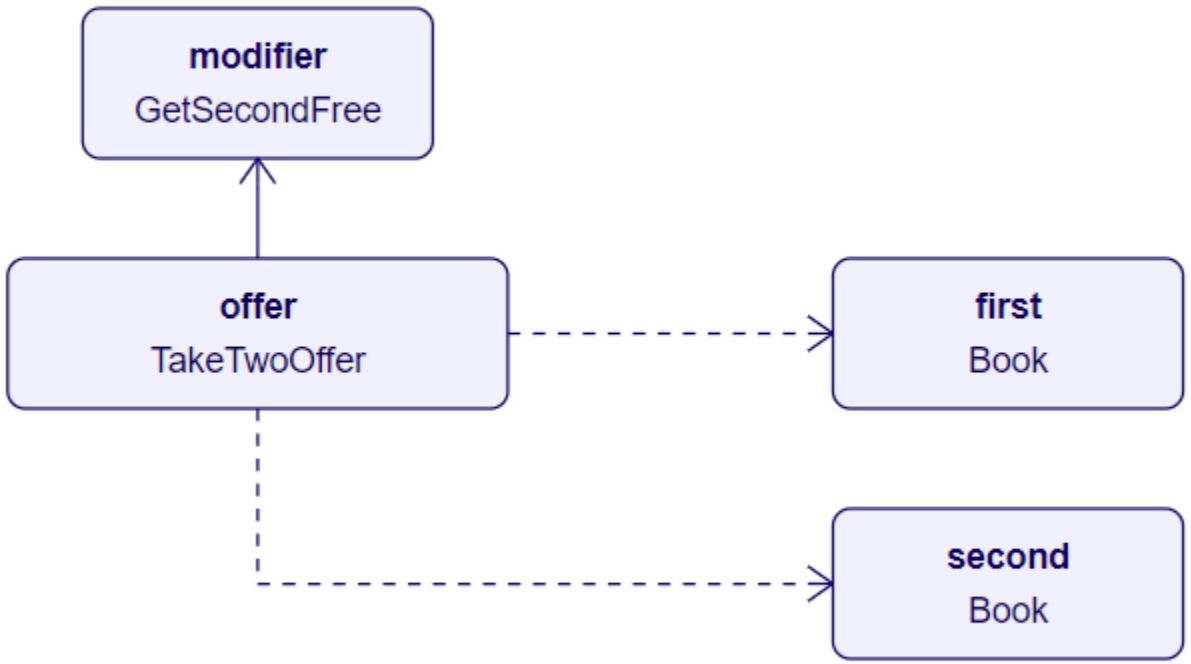
Avoiding Combinatorial Explosion



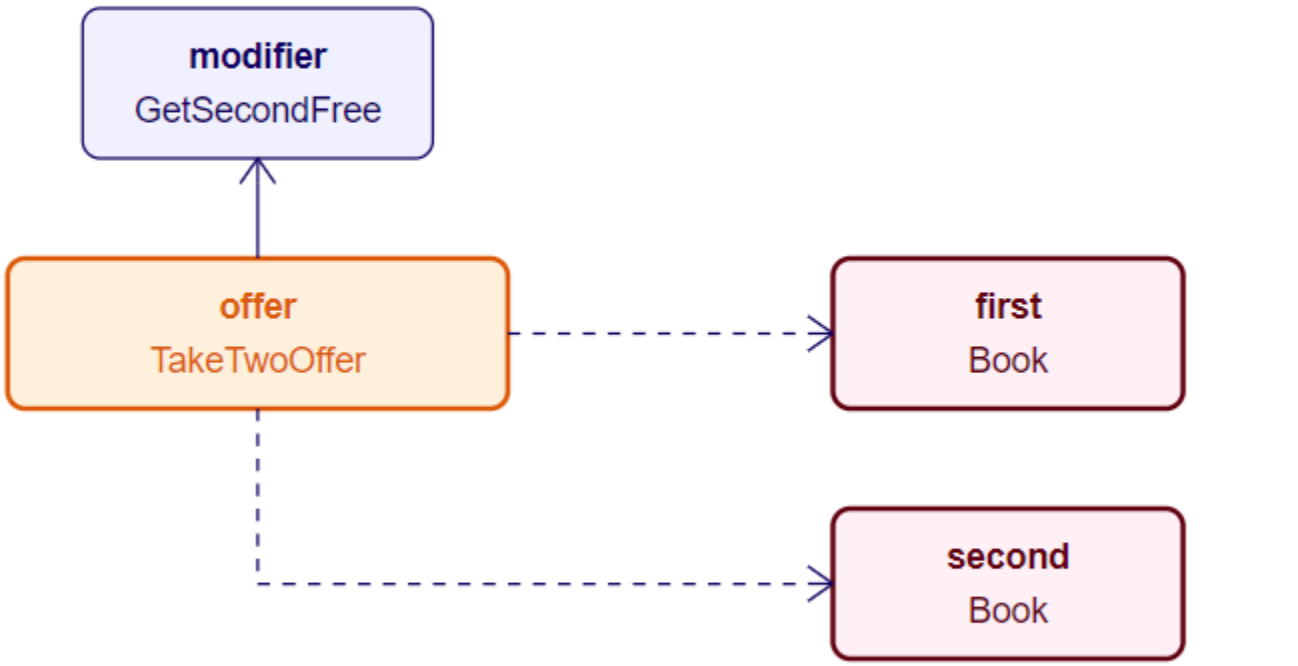
Avoiding Combinatorial Explosion



Nesting Strategies

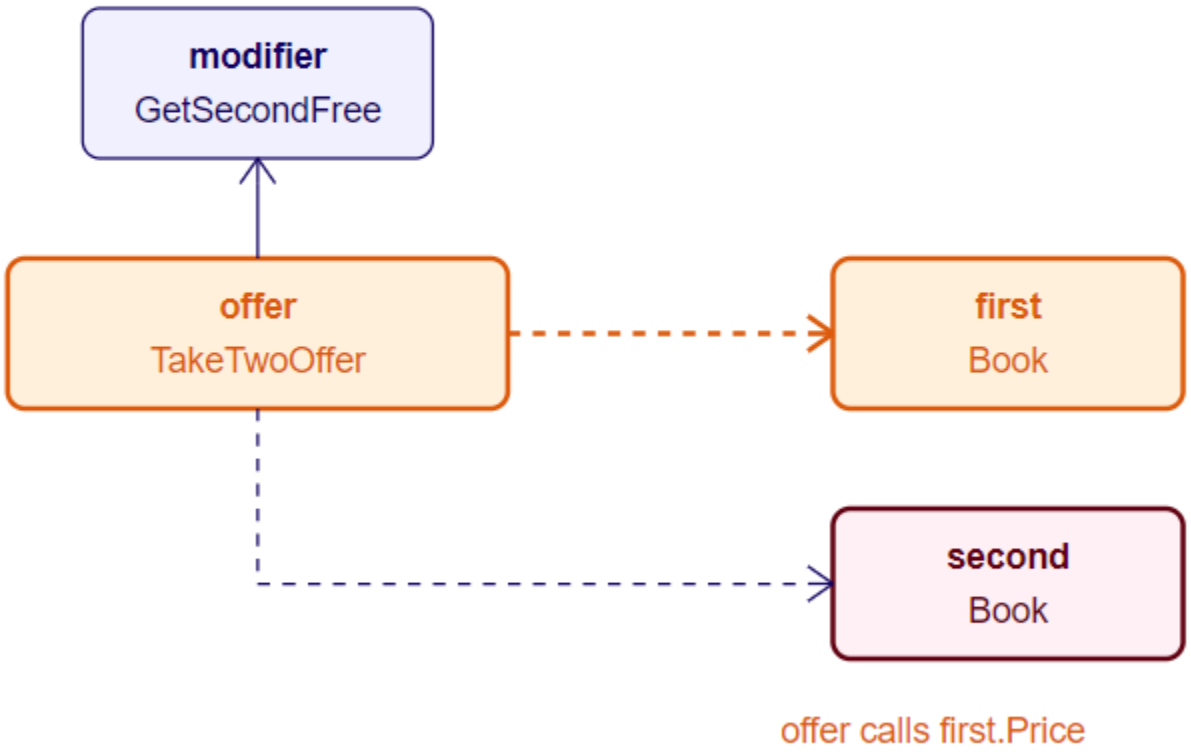


Nesting Strategies

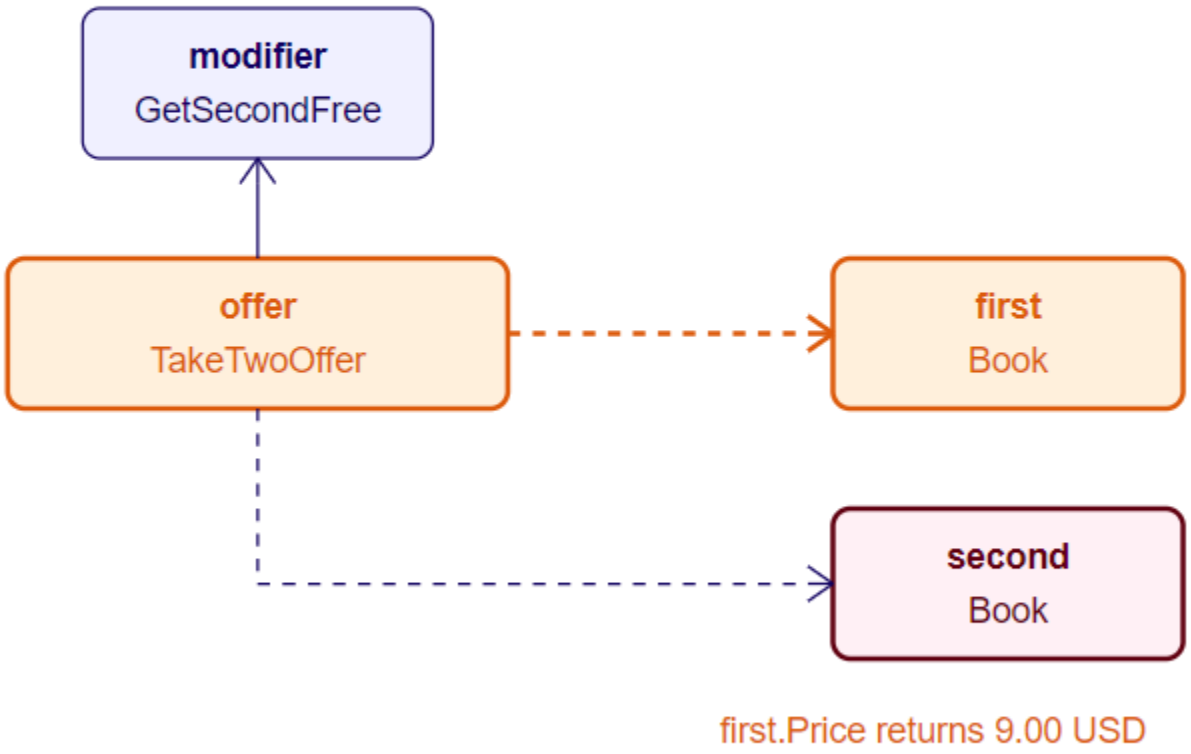


External code calls offer.ApplyTo(first, second)

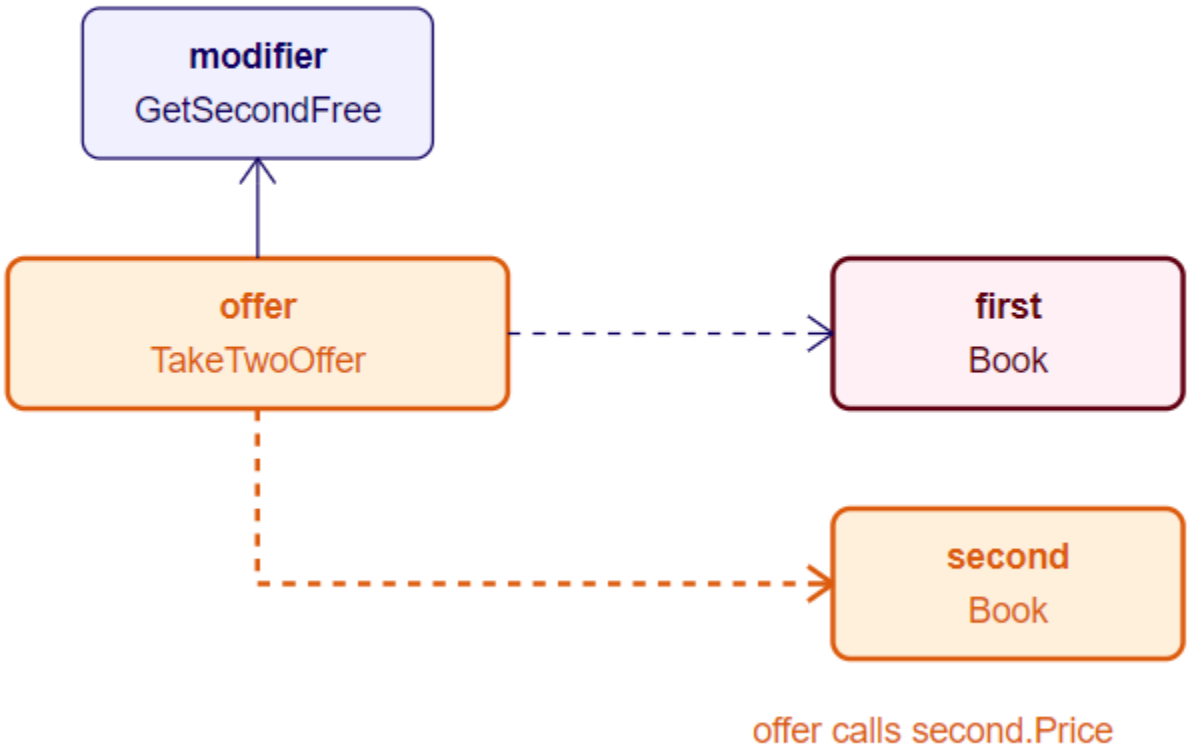
Nesting Strategies



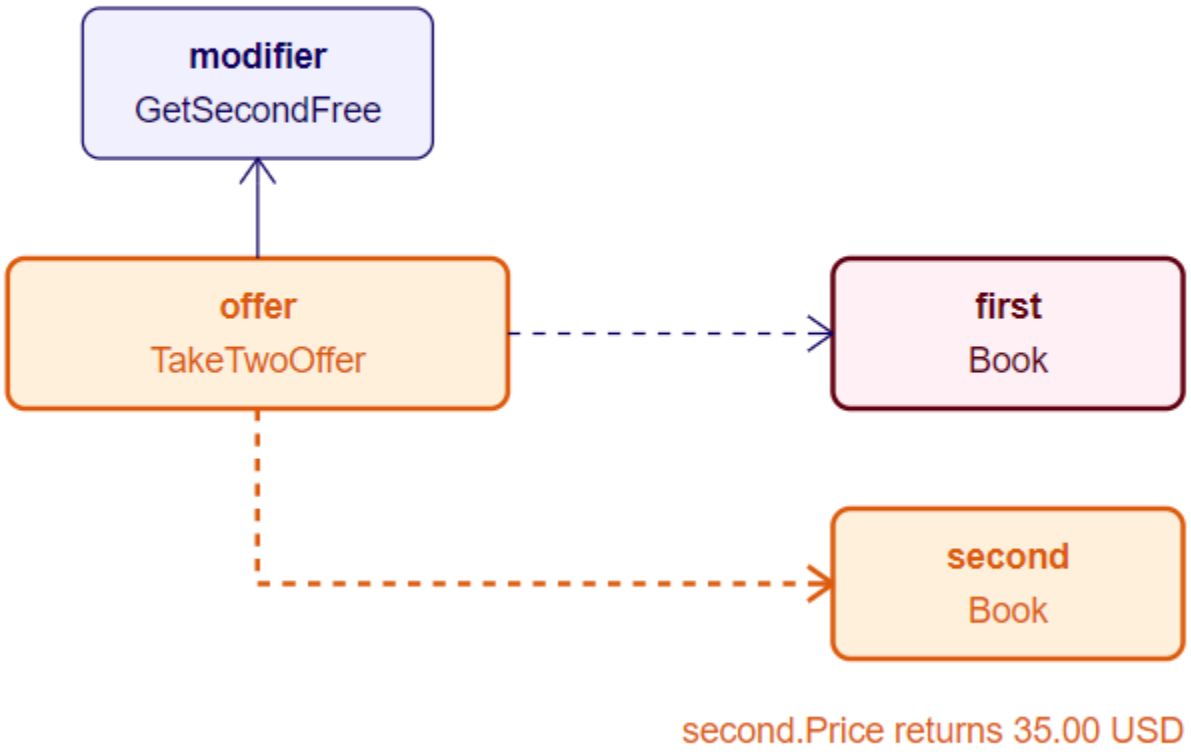
Nesting Strategies



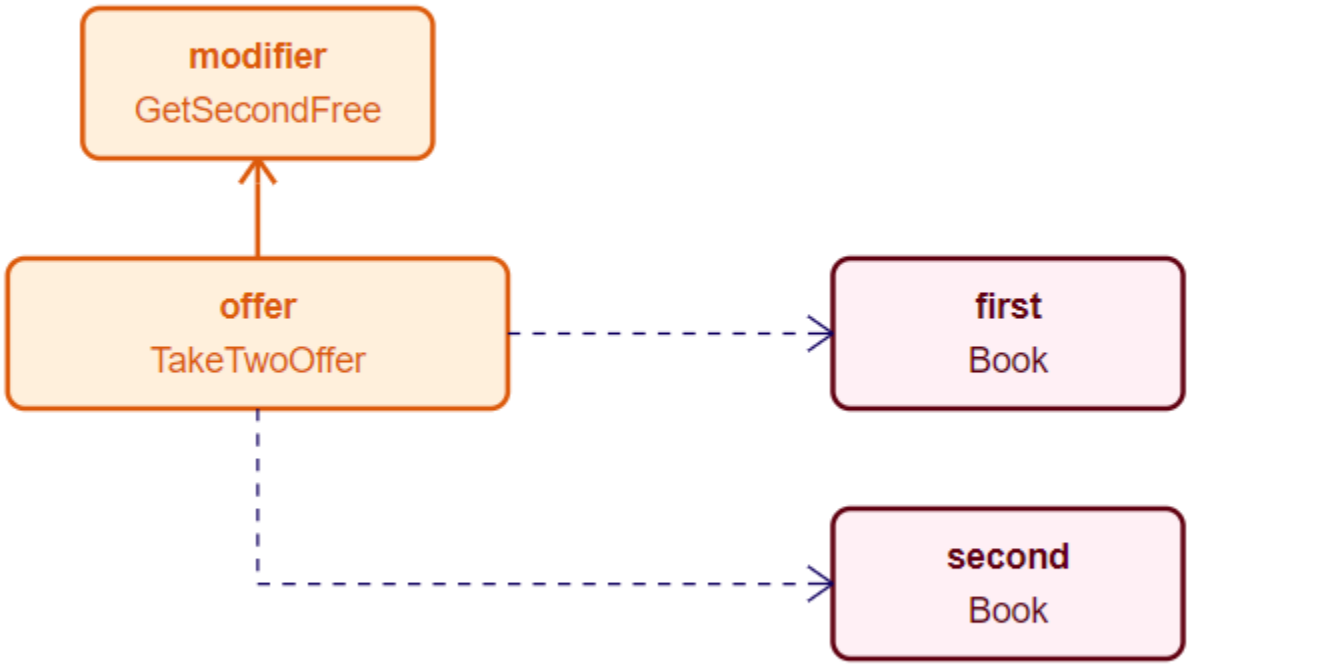
Nesting Strategies



Nesting Strategies

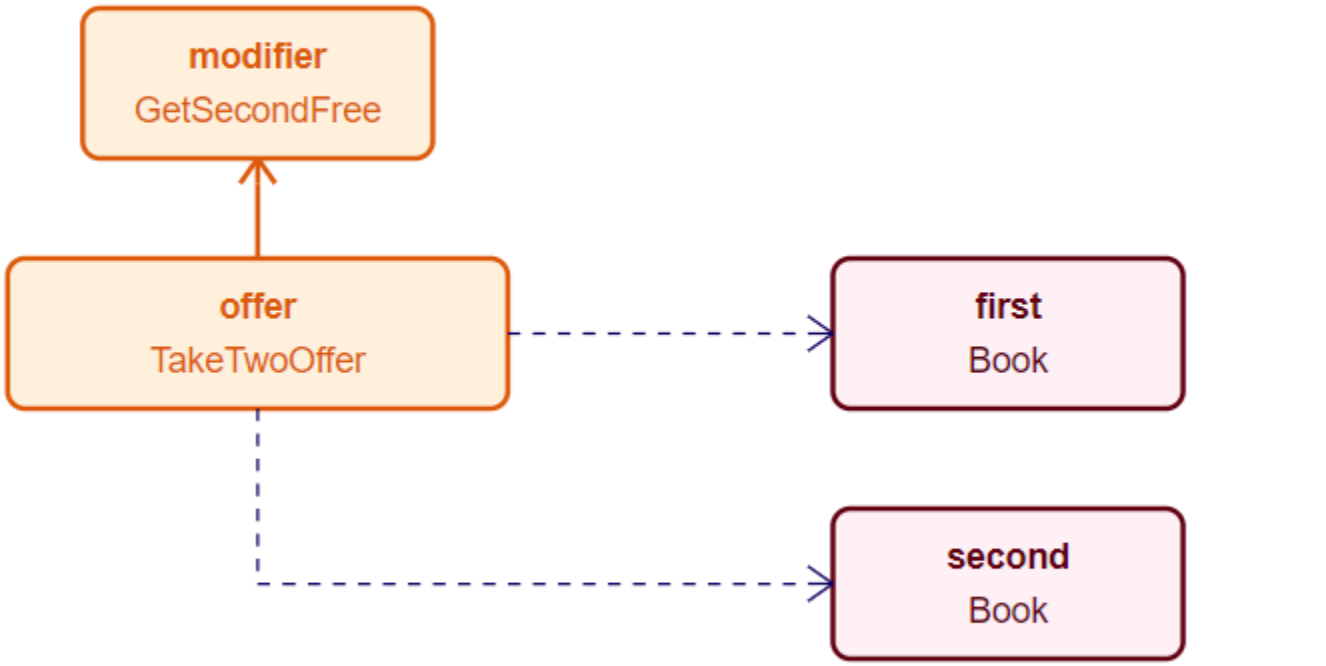


Nesting Strategies



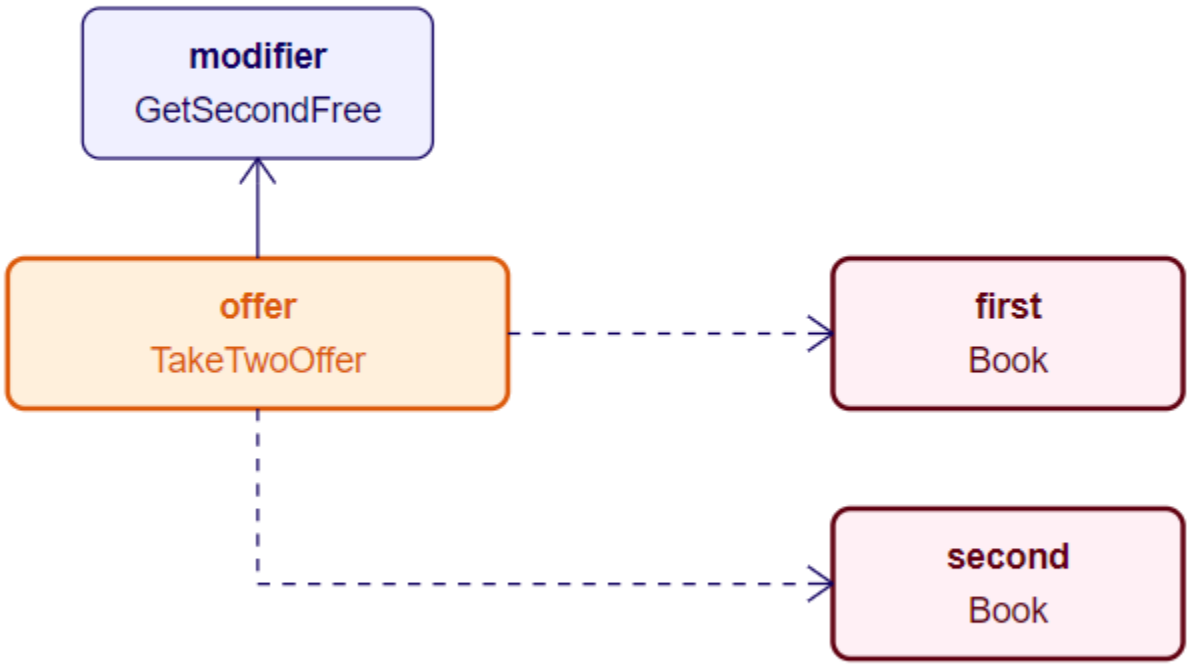
offer calls modifier.ApplyTo(35.00 USD, 9.00 USD)

Nesting Strategies



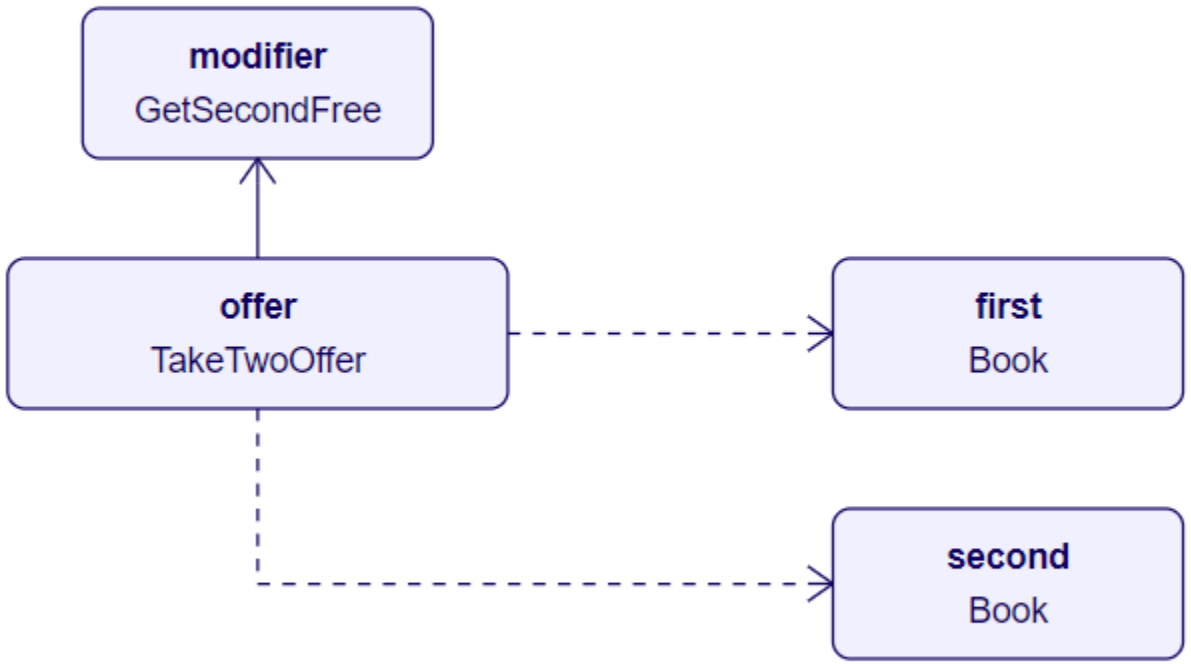
modifier.ApplyTo() returns (35.00 USD, 0.00 USD)

Nesting Strategies

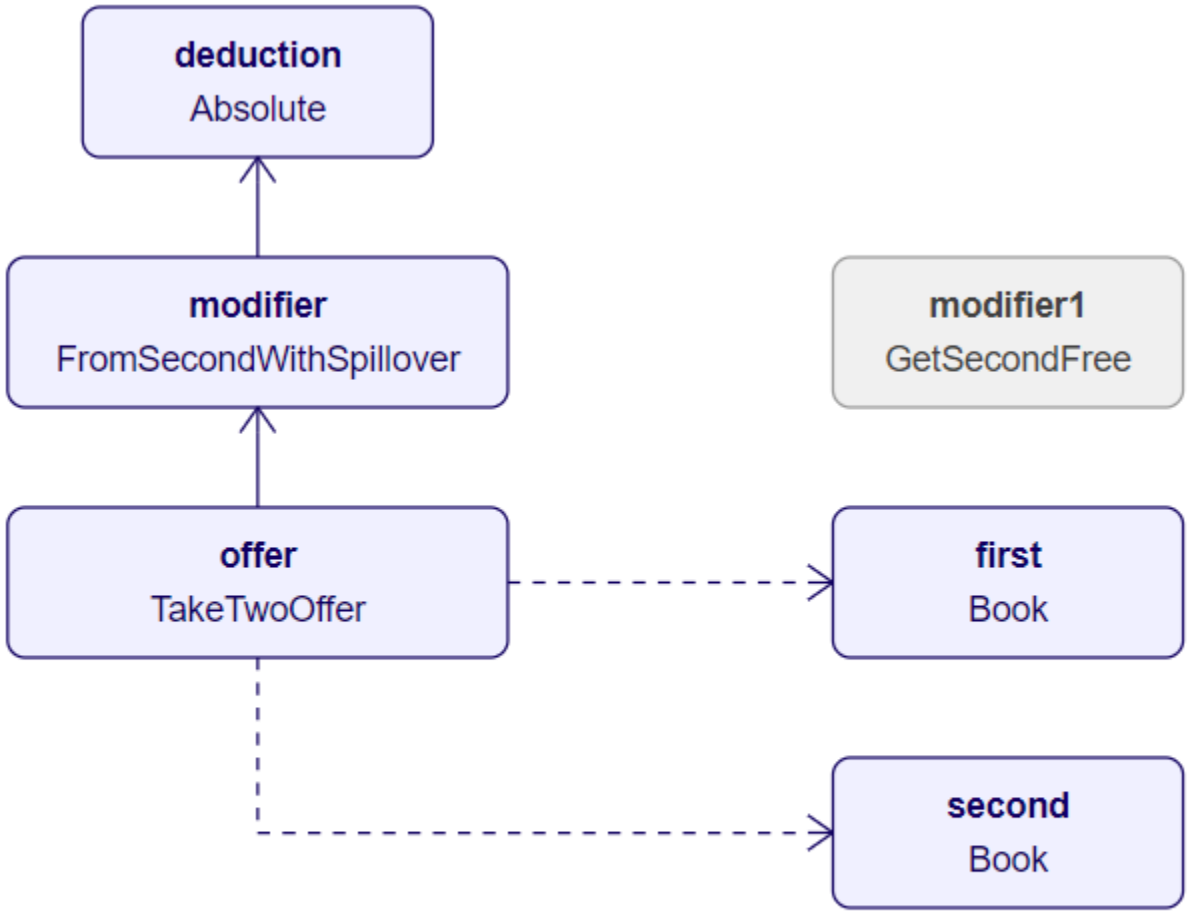


offer.ApplyTo() returns (Design Patterns 35.00 USD, The Little Prince 0.00 USD (Was 9.00 USD))

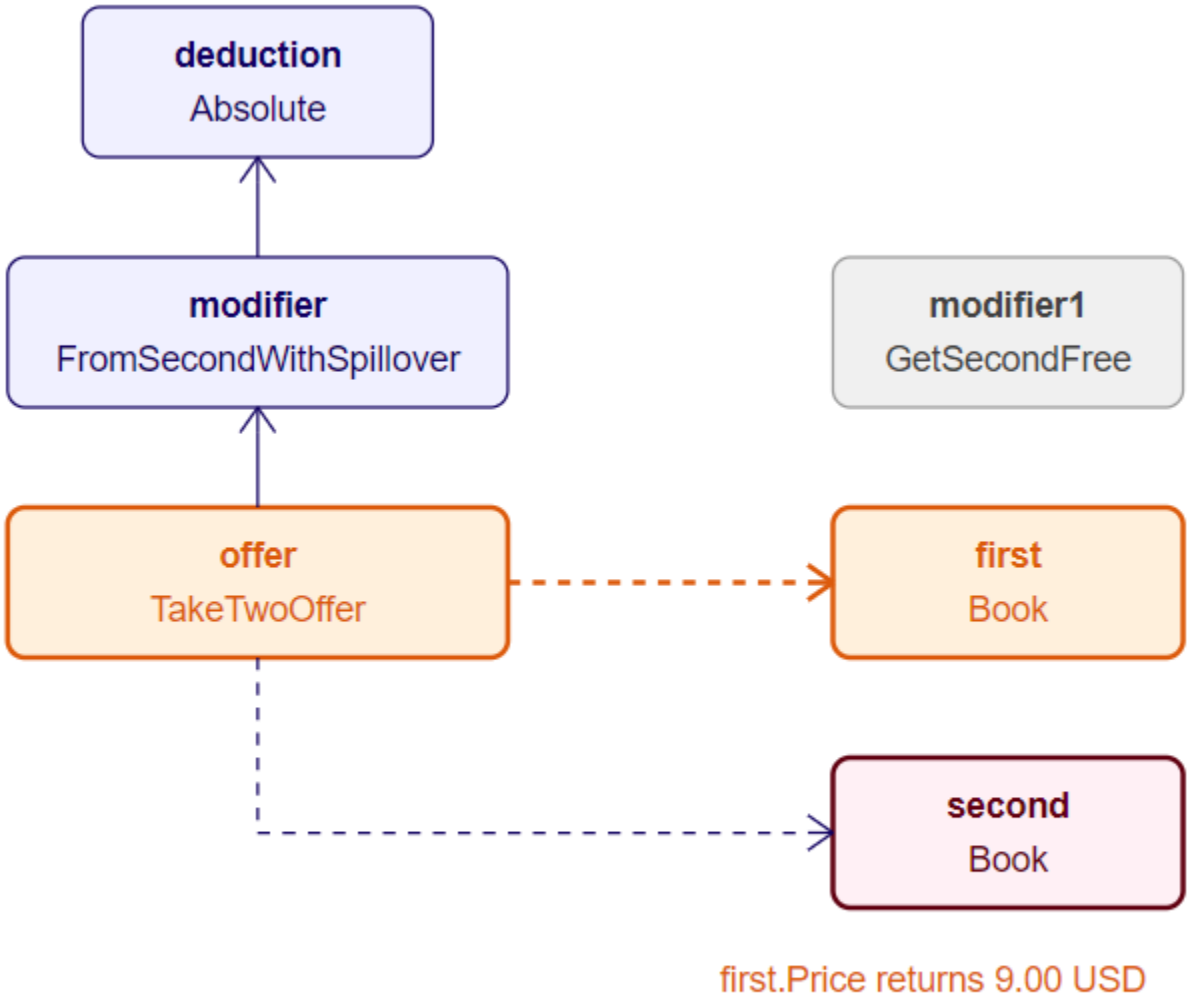
Nesting Strategies



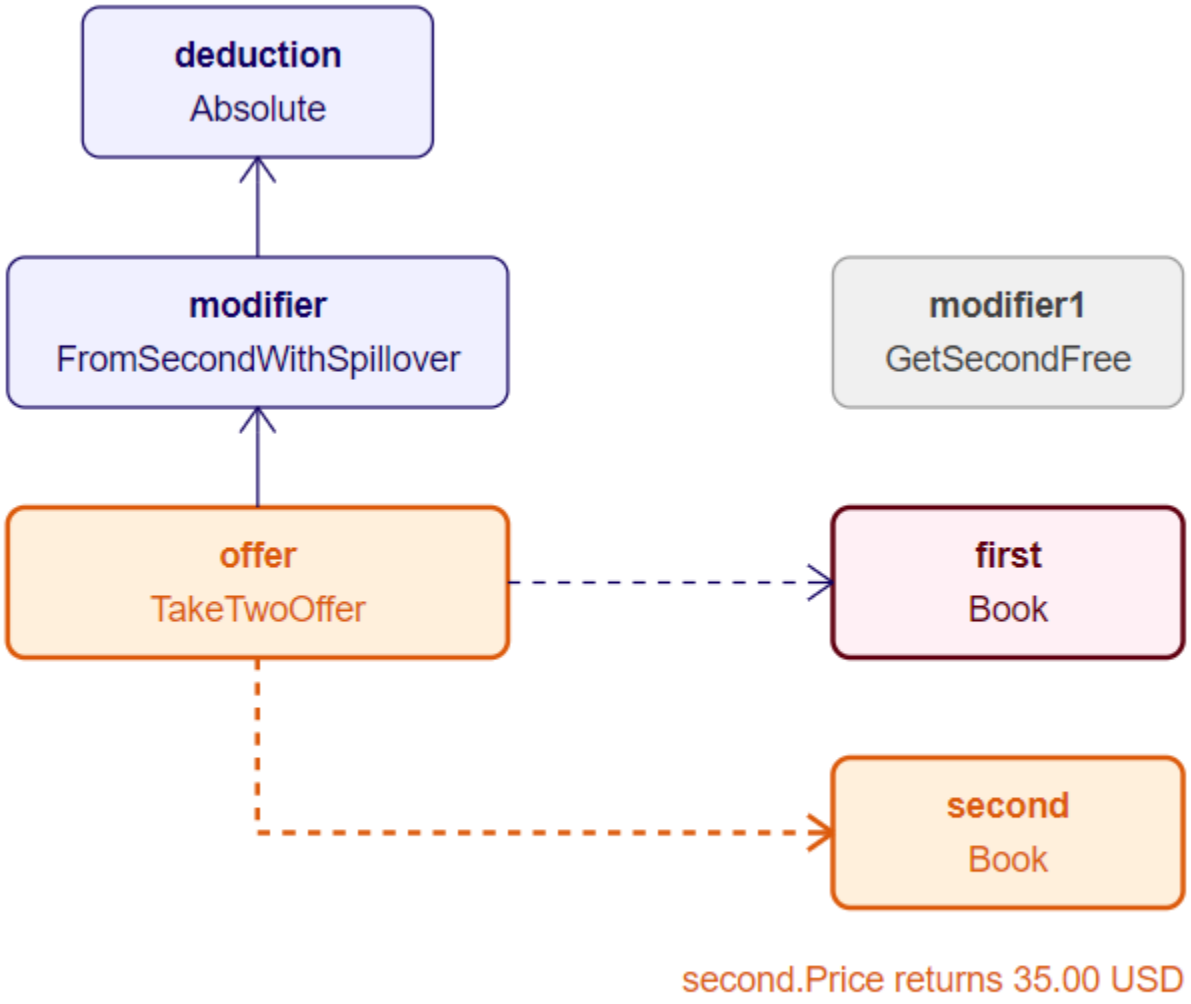
Nesting Strategies



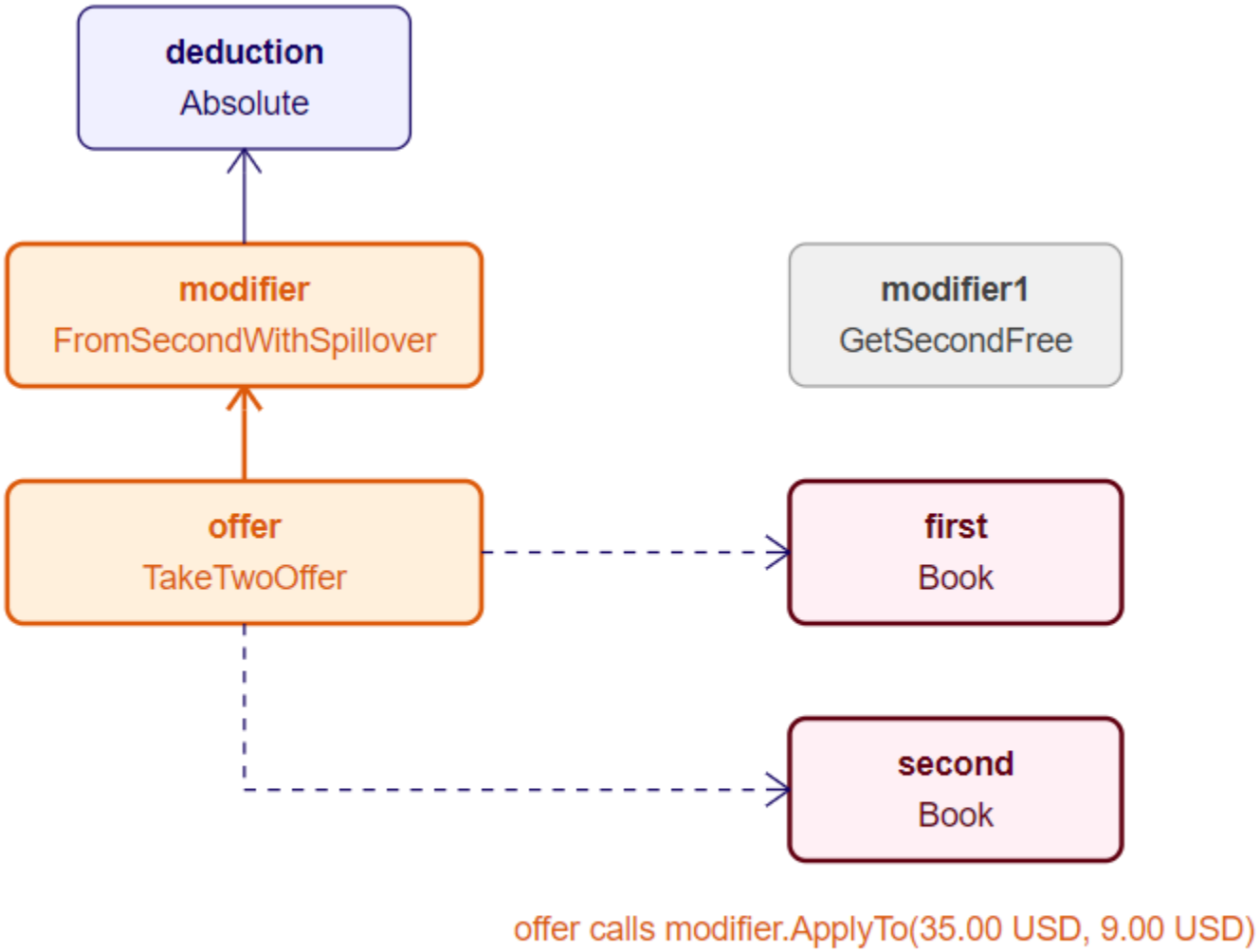
Nesting Strategies



Nesting Strategies

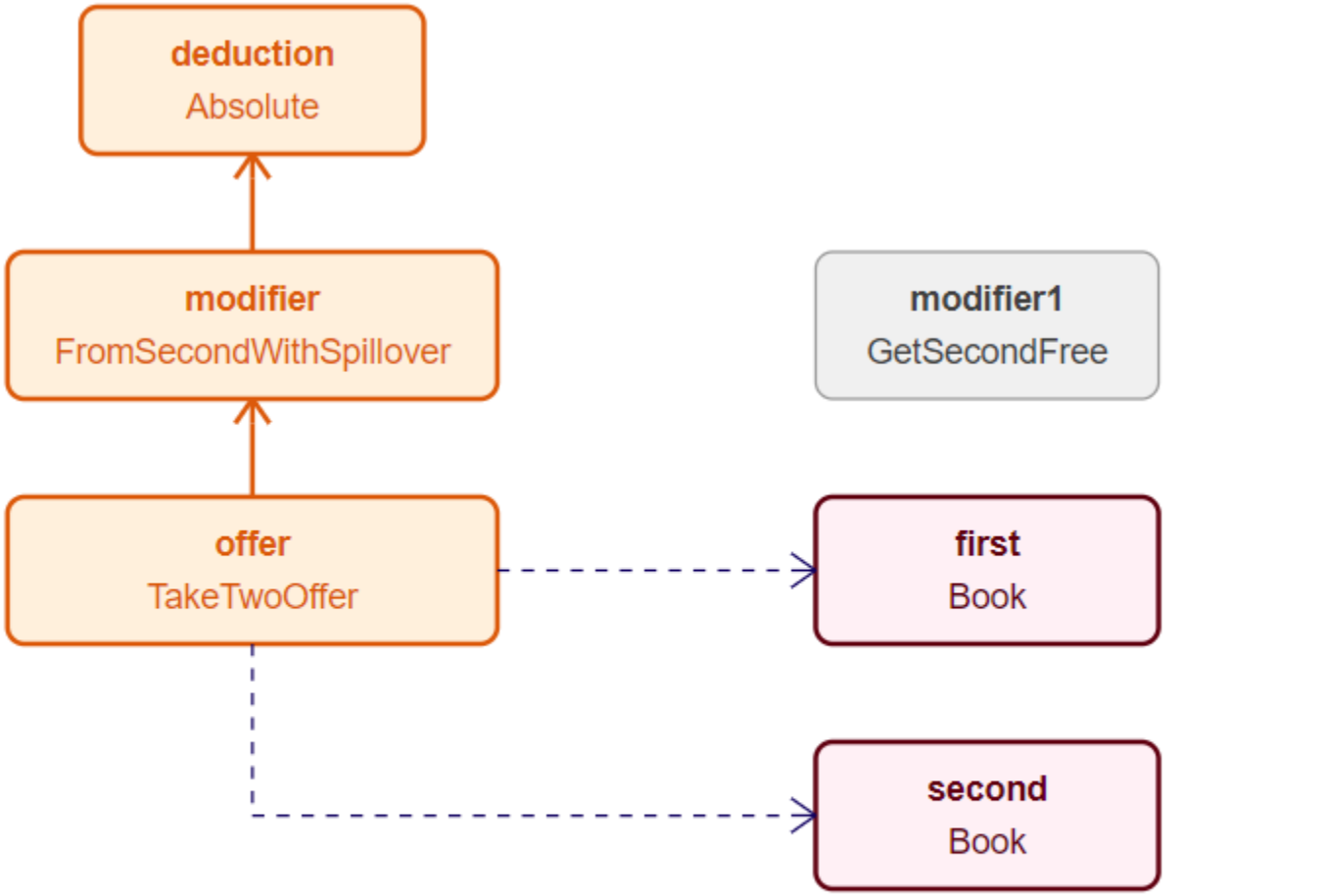


Nesting Strategies



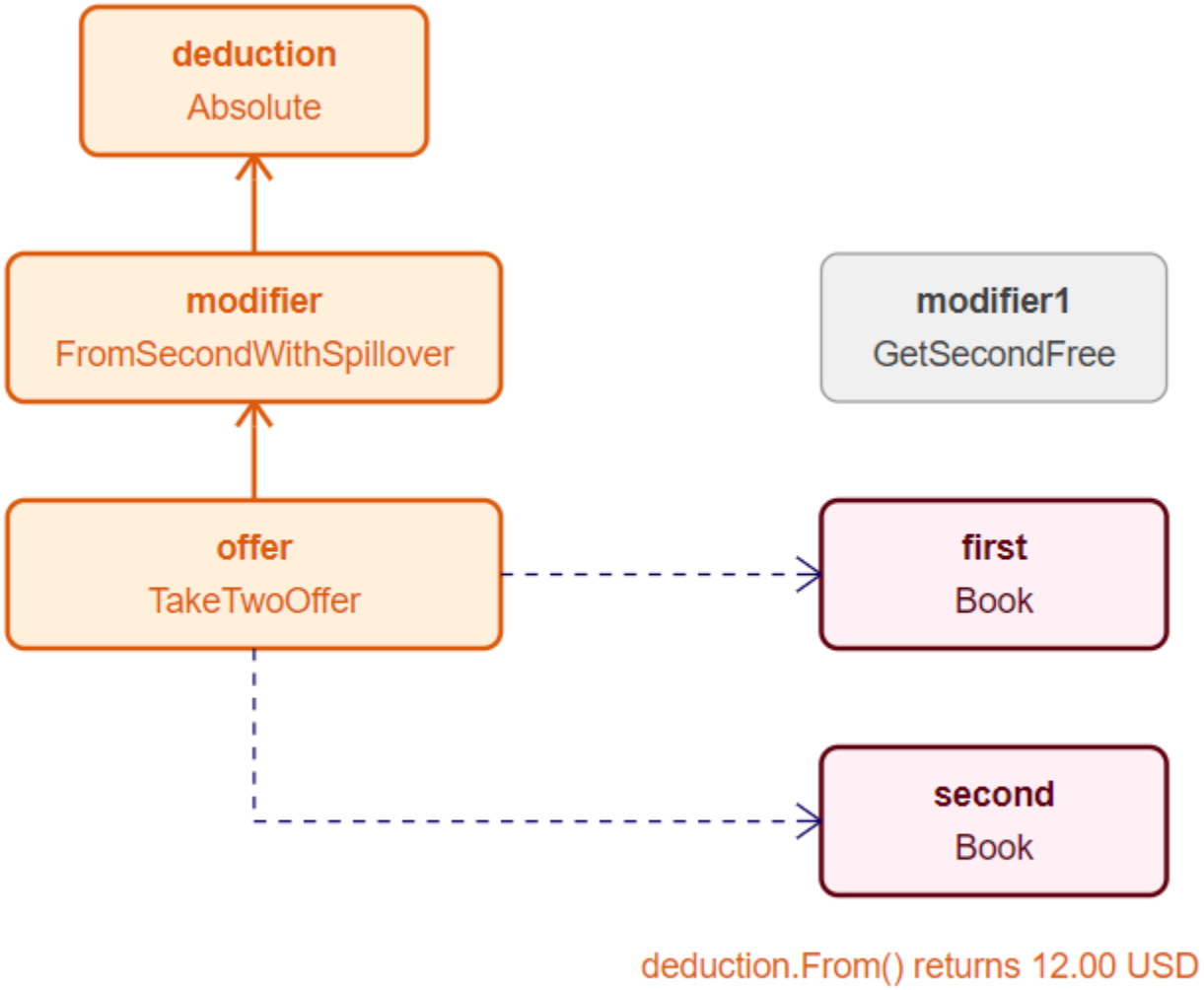
offer calls modifier.ApplyTo(35.00 USD, 9.00 USD)

Nesting Strategies



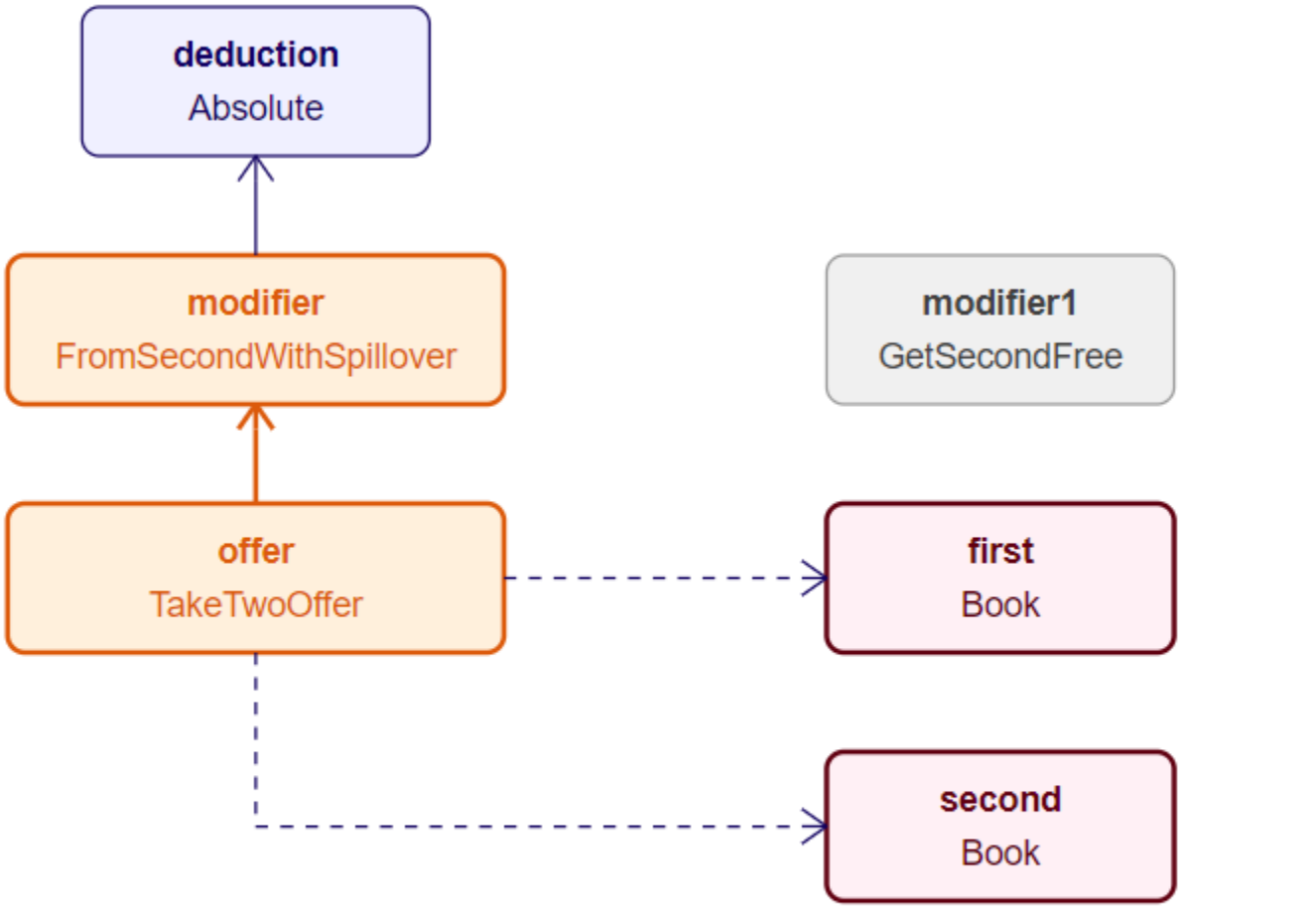
modifier calls deduction.From(35.00 USD, 9.00 USD)

Nesting Strategies



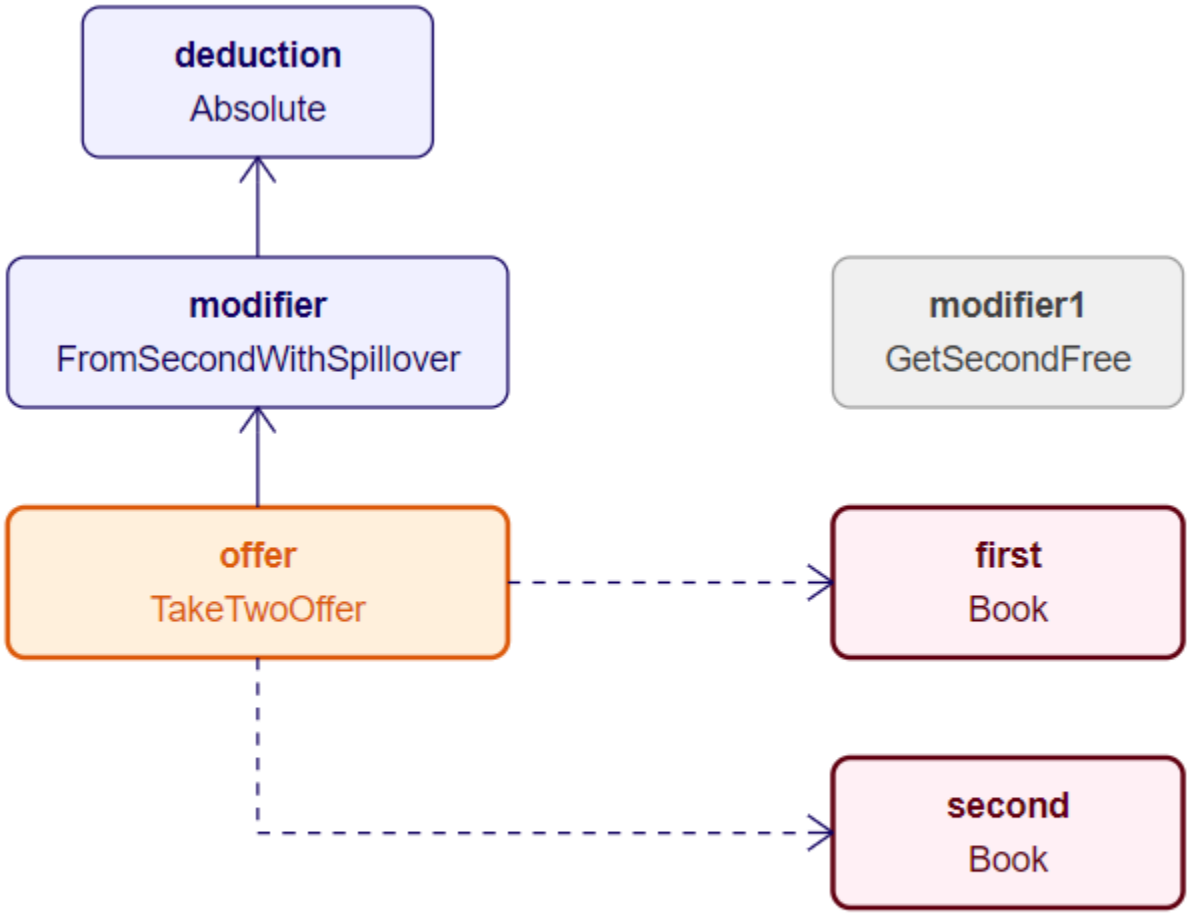
deduction.From() returns 12.00 USD

Nesting Strategies



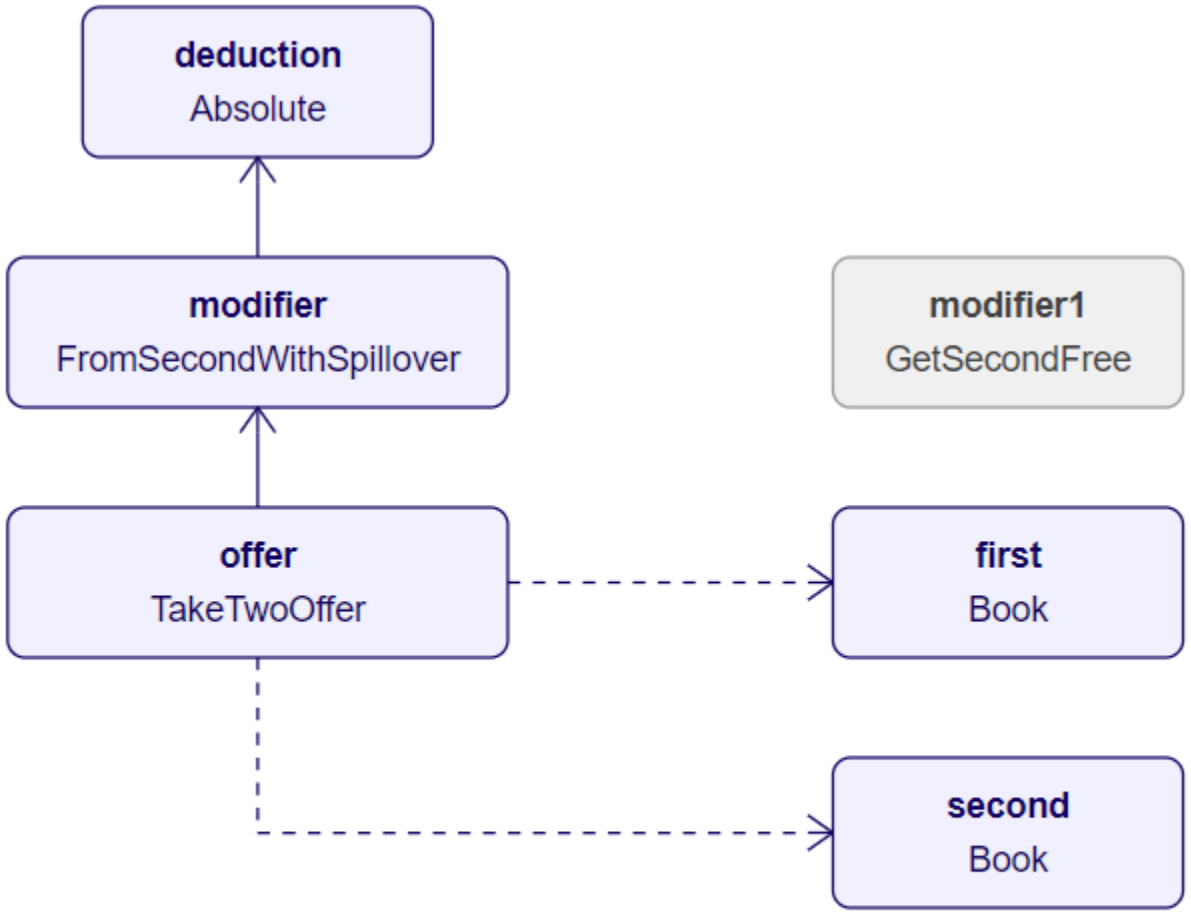
modifier.ApplyTo() returns (32.00 USD, 0.00 USD)

Nesting Strategies

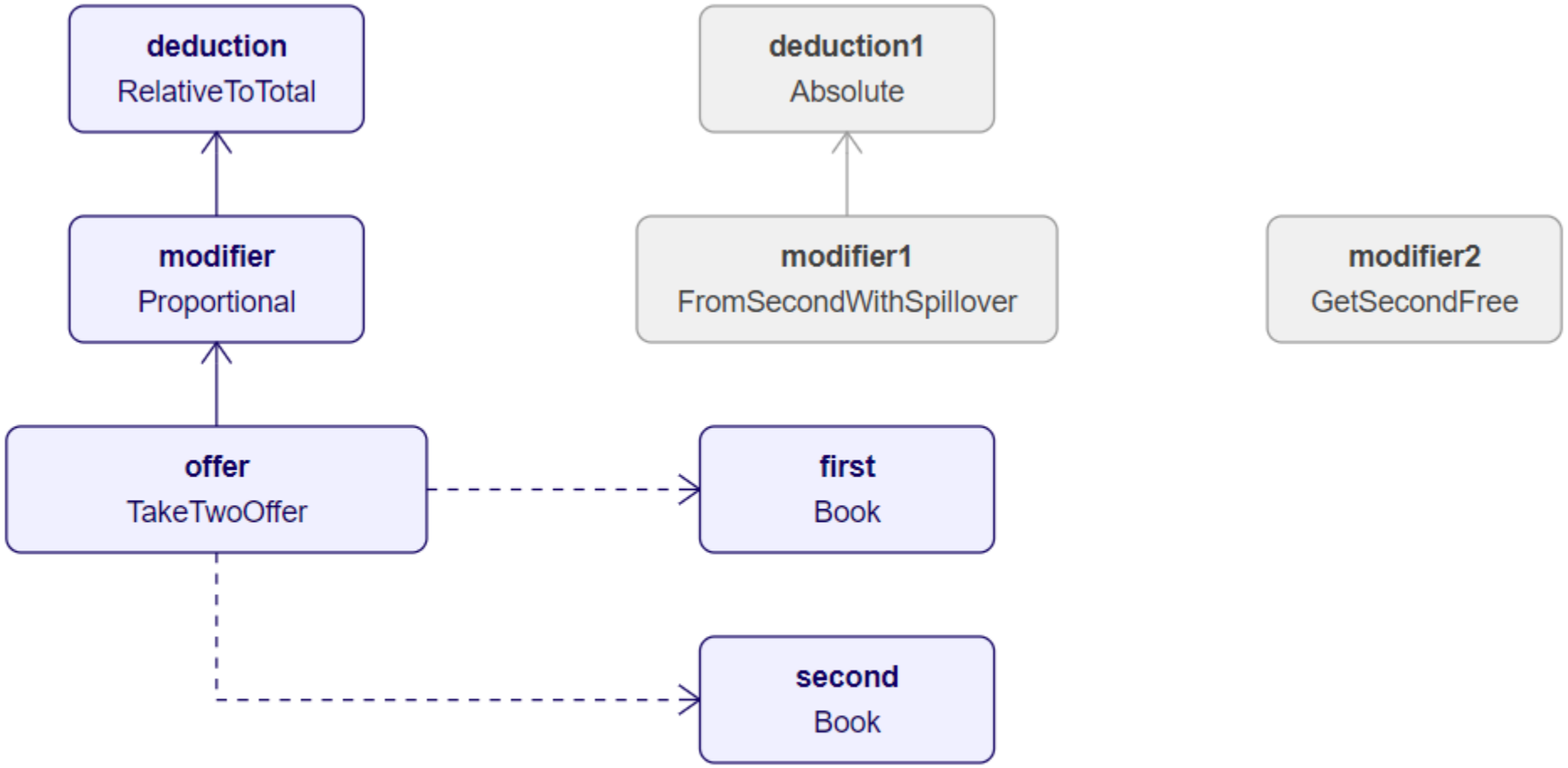


offer.ApplyTo() returns (Design Patterns 32.00 USD (Was 35.00 USD), The Little Prince 0.00 USD (Was 9.00 USD))

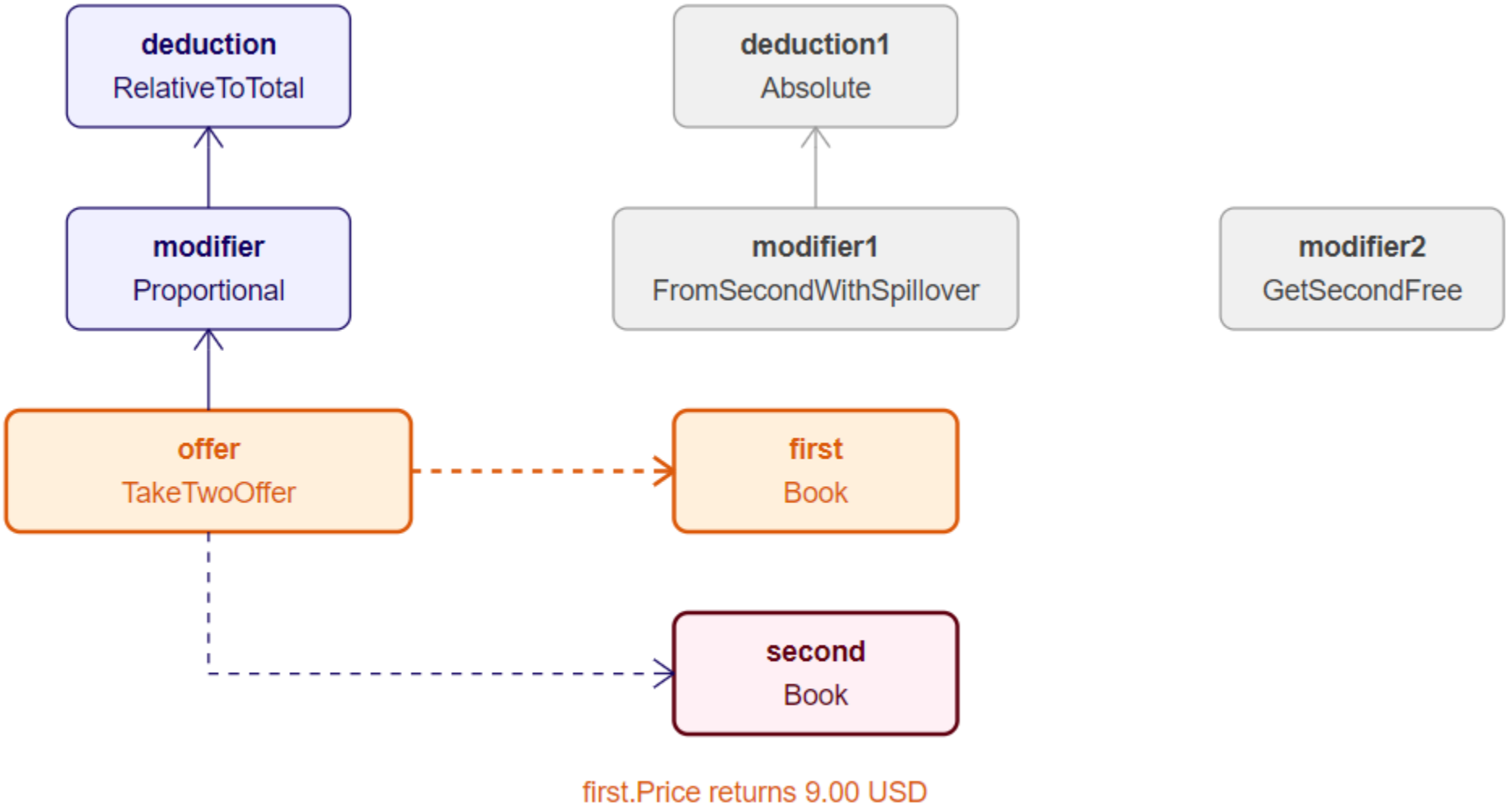
Nesting Strategies



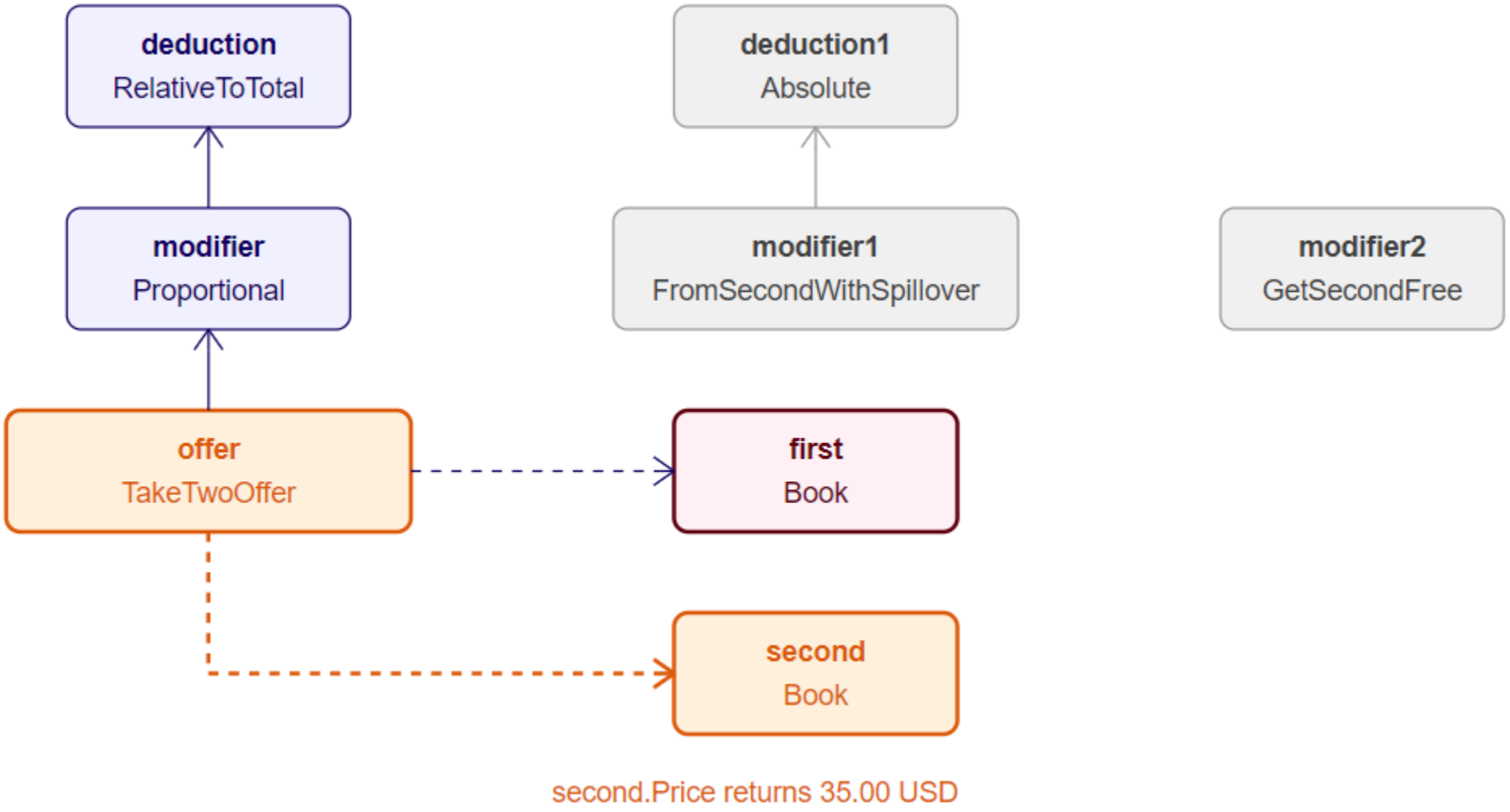
Nesting Strategies



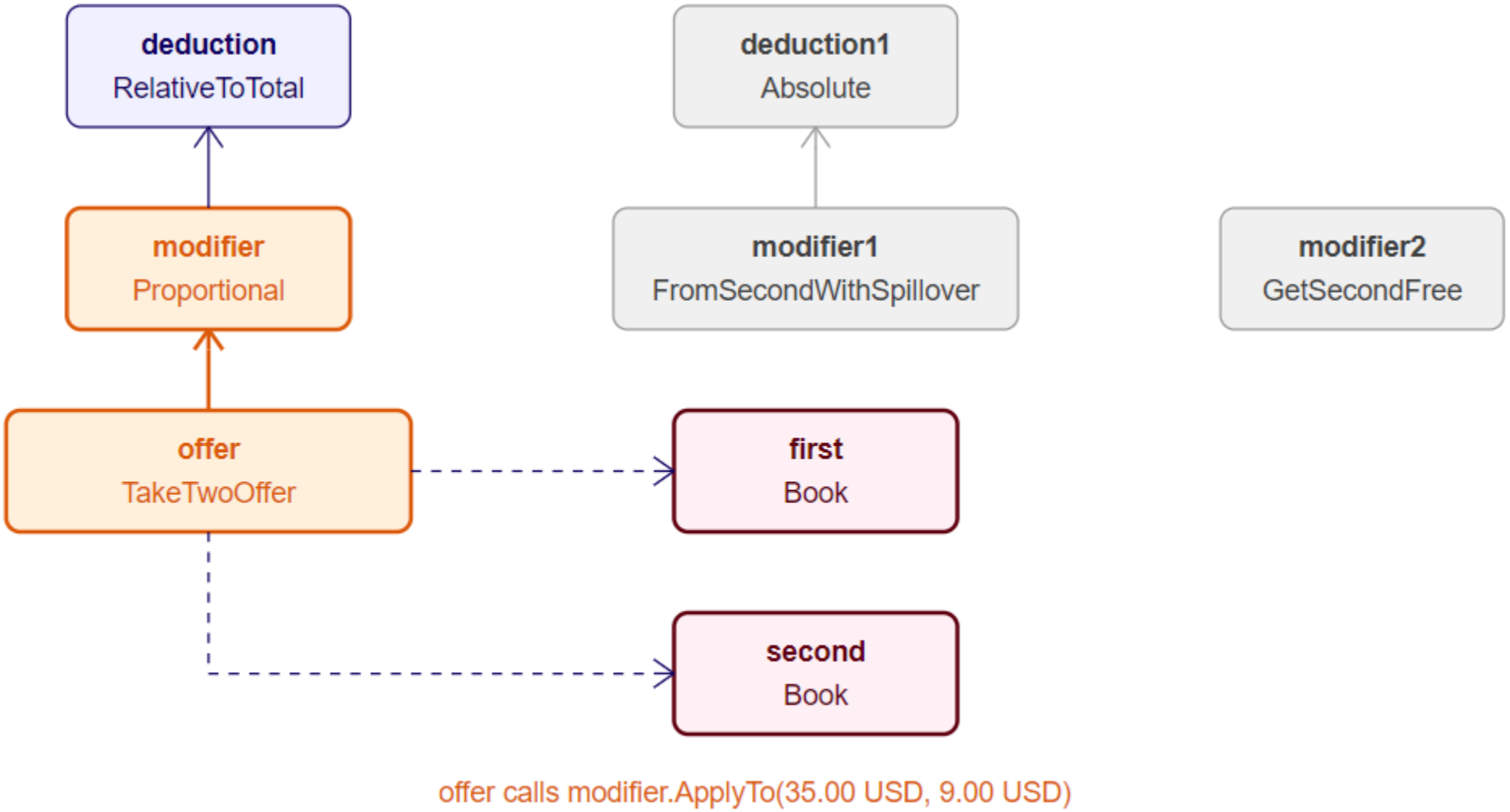
Nesting Strategies



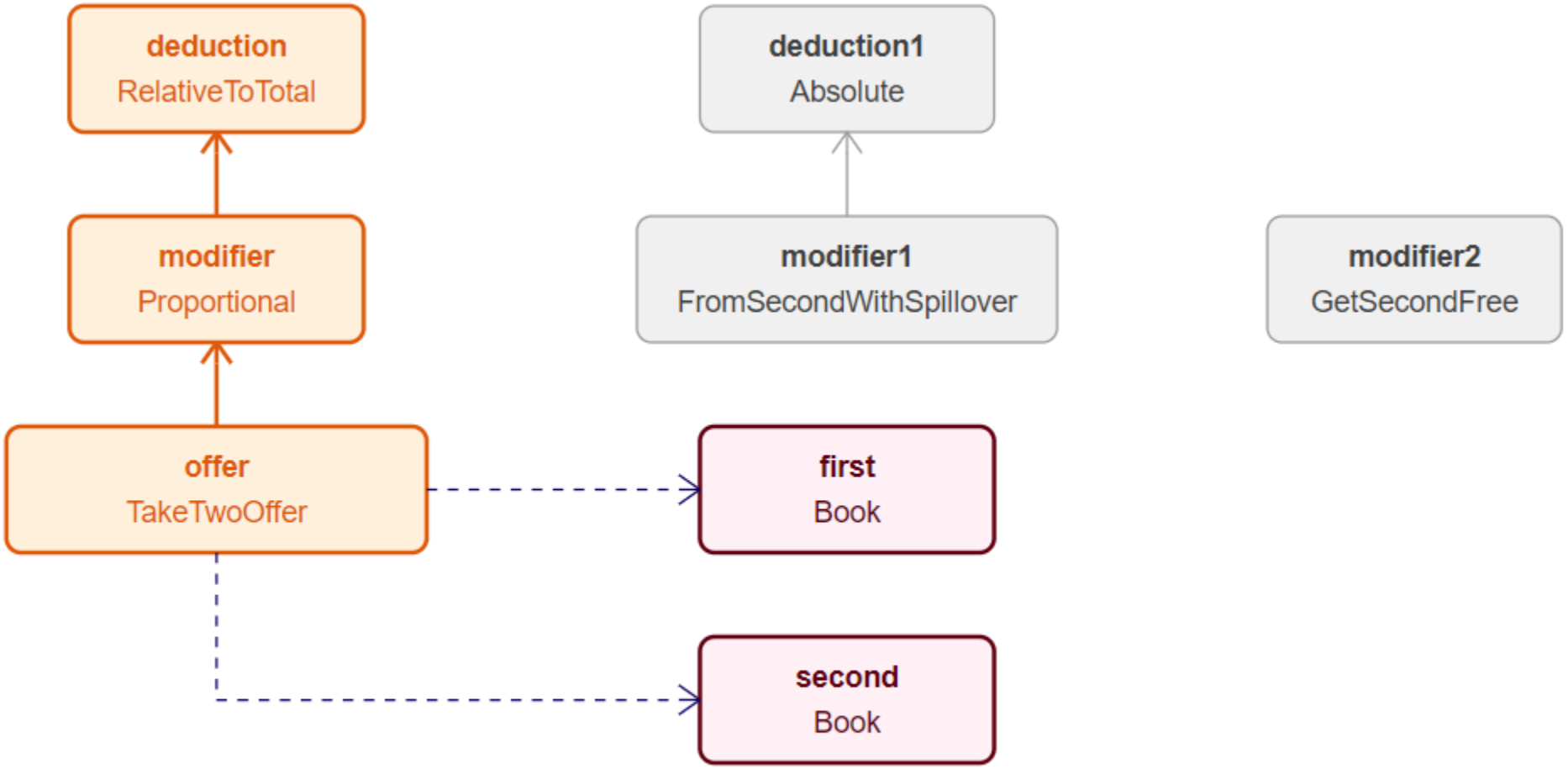
Nesting Strategies



Nesting Strategies

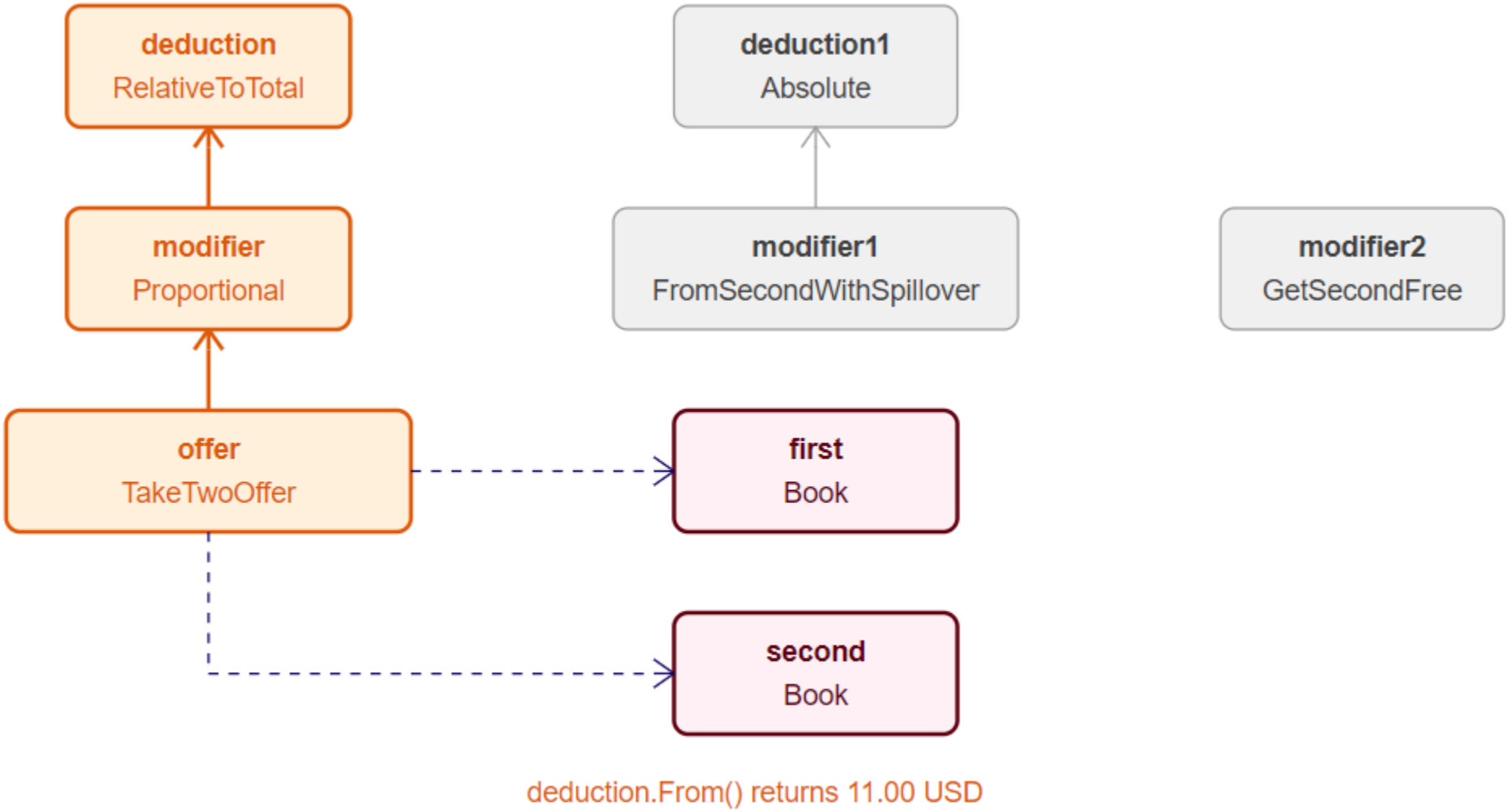


Nesting Strategies

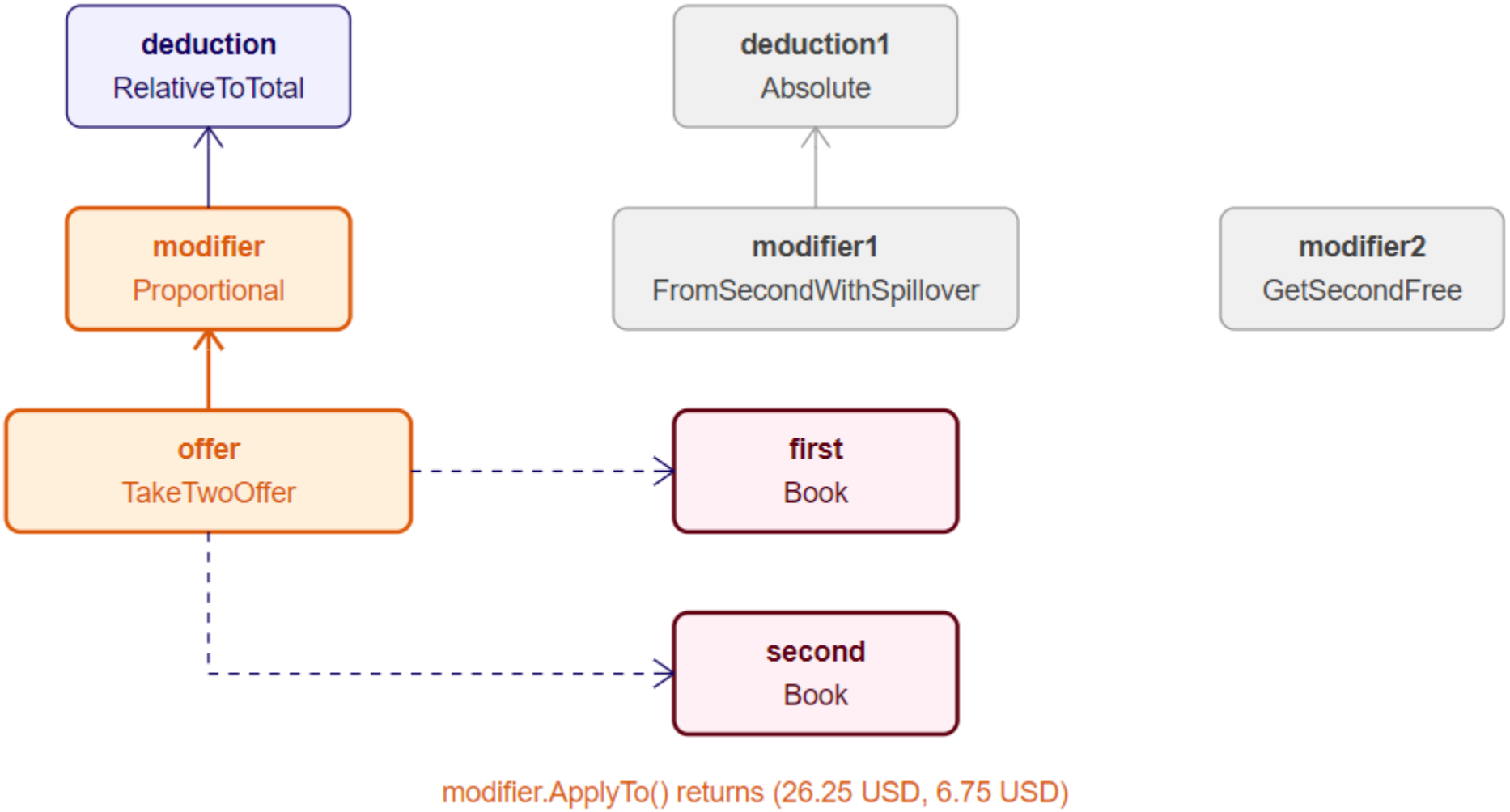


modifier calls deduction.From(35.00 USD, 9.00 USD)

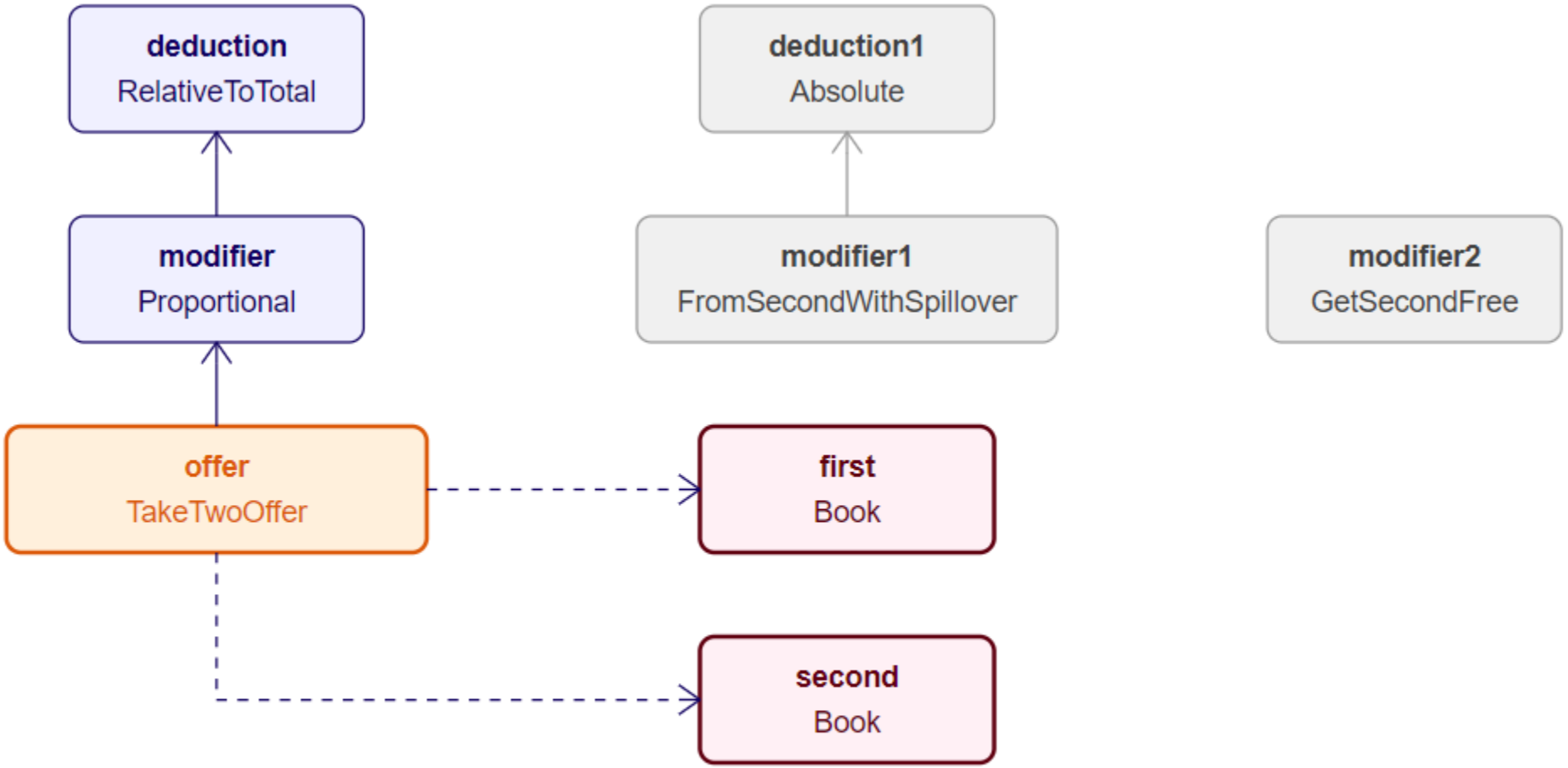
Nesting Strategies



Nesting Strategies

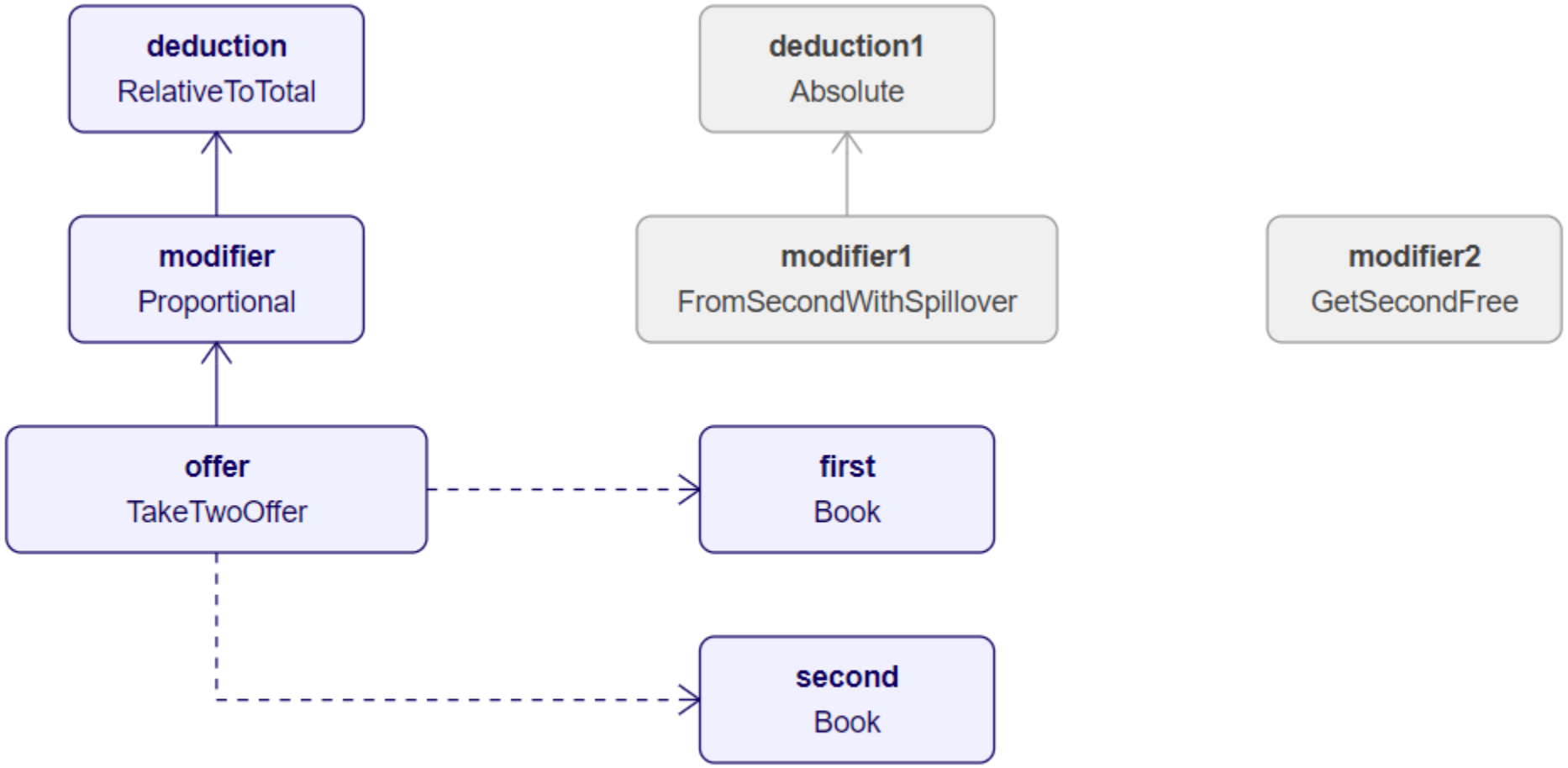


Nesting Strategies



offer.ApplyTo() returns (Design Patterns 26.25 USD (Was 35.00 USD), The Little Prince 6.75 USD (Was 9.00 USD))

Nesting Strategies



Summary

The Strategy pattern

- Parameterizes a method with behavior
- Class is extended without modification
- More concrete strategies can be implemented later

Summary

Implementing strategies

- Use a Func delegate as a lightweight strategy
- No abstract type and derived types for every strategy
- Complex strategies can have their own strategies
- Strategies with strategies avoid combinatorial explosion
- Effectively prevent power law growth in number of classes

Summary

Strategies in the .NET Framework

- Many classes let us inject strategies
- Collections accept strategies for indexing, sorting, etc.
- LINQ offers extensibility via Func delegates as strategies
- Makes Strategy one of the most widely applicable patterns