

DESIGN PATTERNS IN C# MADE SIMPLE

MODULE 2 Adding Behavior to an Object with the Decorator Pattern



ZORAN HORVAT
CEO AT CODING HELMET

<http://codinghelmet.com>

zh@codinghelmet.com

 zoranh75

Buyer.cs Length.cs Size.cs Book.cs Clip01Demo.cs X

C# Demo Demo.Clip01.Clip01Demo Implementation()

```
namespace Demo.Clip01
{
    1 reference
    class Clip01Demo : Common.Demo
    {
        2 references
        protected override int ClipNumber { get; } = 1;
        private readonly Length mm = Length.Millimeter;

        2 references
        protected override void Implementation()
        {
            Book product = new Book("Design Patterns", new Size(188*mm, 239*mm, 28*mm));

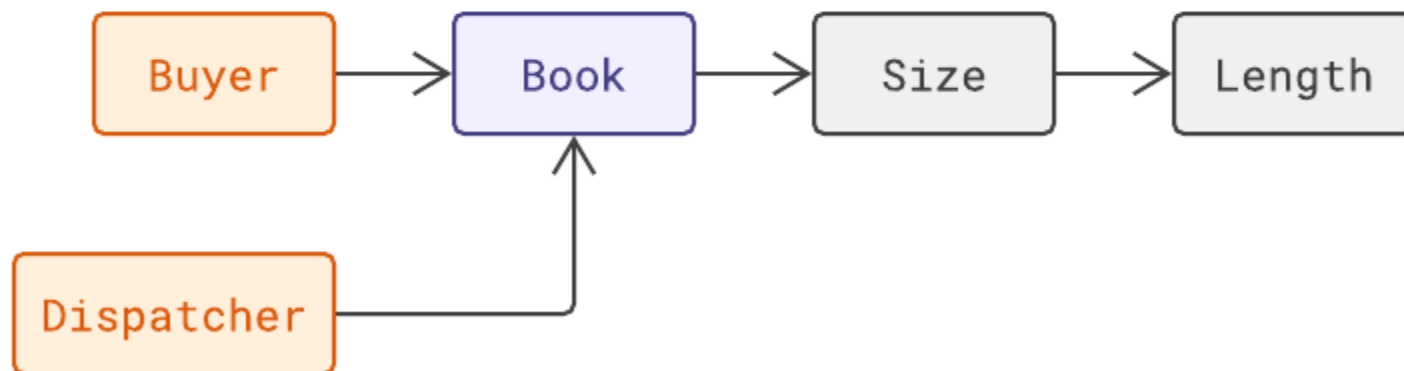
            Buyer customer = new Buyer();
            customer.Handle(product);
        }
    }
}
```

```
graph LR
    Buyer[Buyer] --> Book[Book]
    Book --> Size[Size]
    Size --> Length[Length]
```

```
namespace Demo.Clip01
{
    2 references
    class Dispatcher
    {
        1 reference
        public void Handle(Book product)
        {
            Size originalSize = product.Dimensions;

            Size packagedSize = originalSize.Add(new Size(
                7 * Length.Millimeter,
                7 * Length.Millimeter,
                7 * Length.Millimeter));

            Console.WriteLine($"Dispatching book \"{product.Title}\" of size {packagedSize}");
        }
    }
}
```

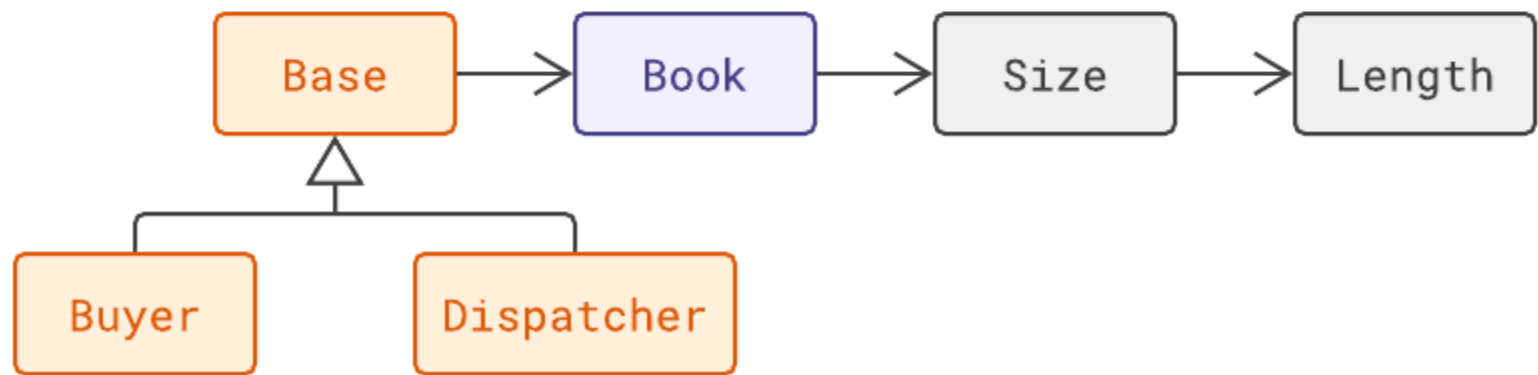


```

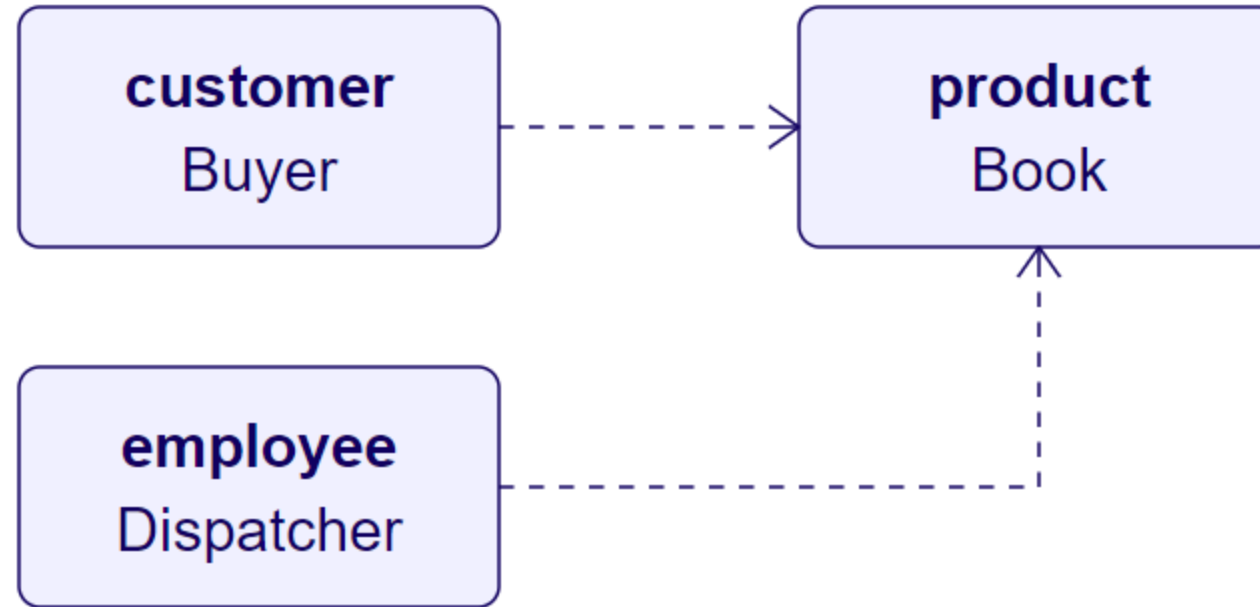
namespace Demo.Clip01
{
    2 references
    class Dispatcher
    {
        1 reference
        public void Handle(Book product)
        {
            Size originalSize = product.Dimensions;

            Size packagedSize = originalSize.Add(new Size(
                7 * Length.Millimeter,
                7 * Length.Millimeter,
                7 * Length.Millimeter));

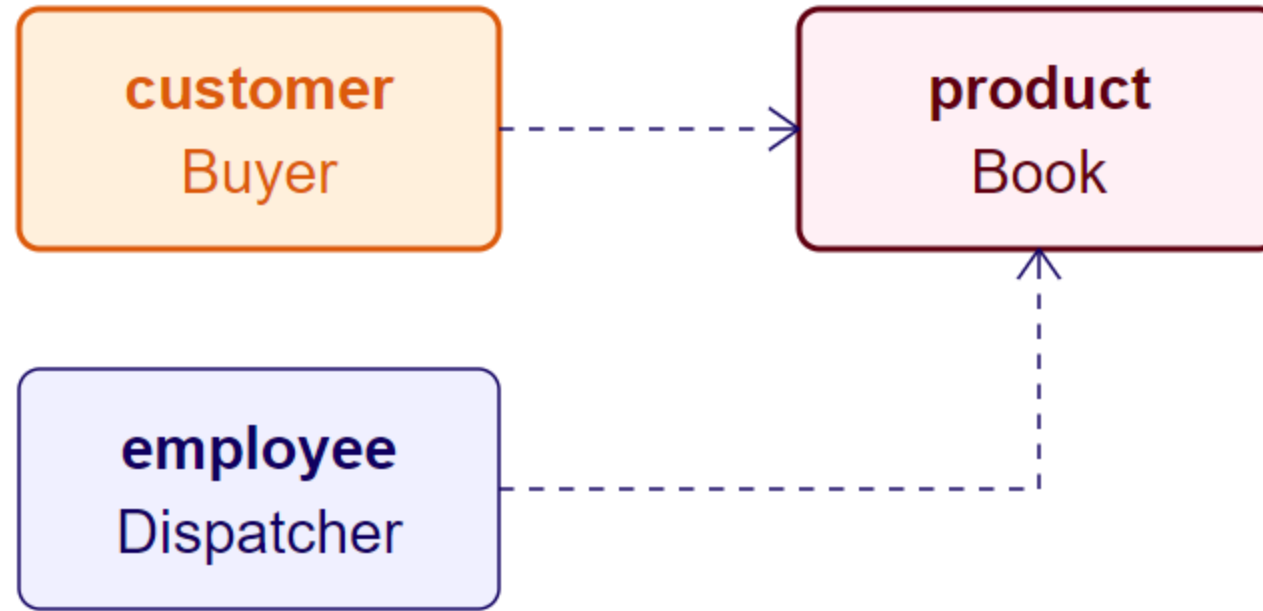
            Console.WriteLine($"Dispatching book \"{product.Title}\" of size {packagedSize}");
        }
    }
}
    
```



Motivation for the Decorator Pattern

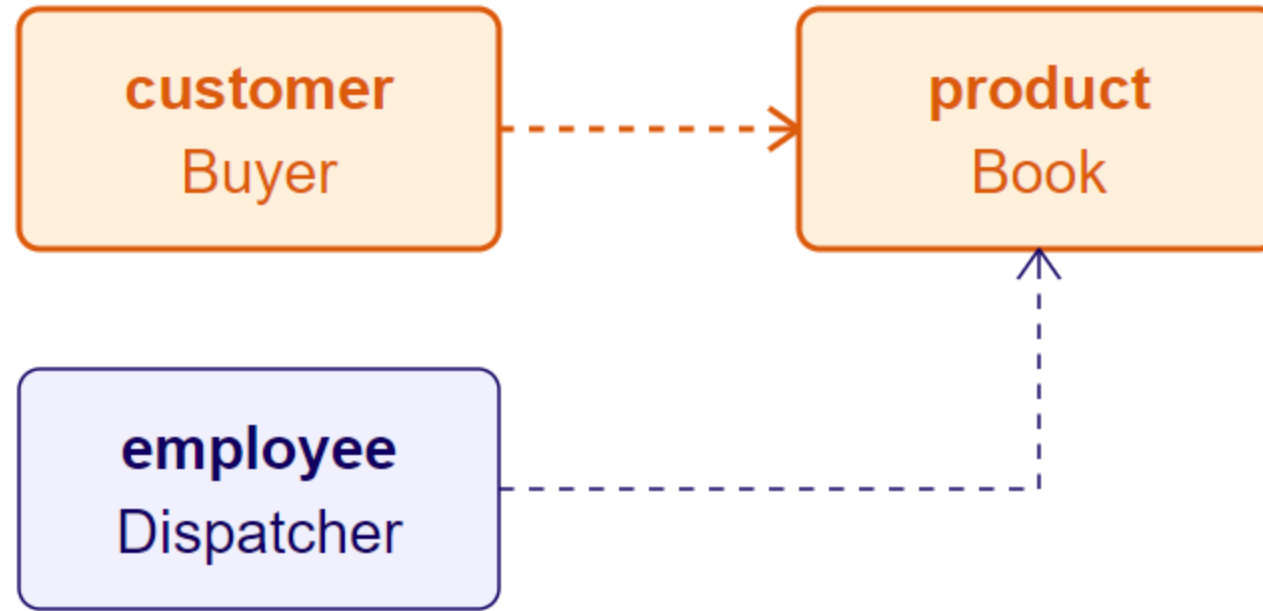


Motivation for the Decorator Pattern



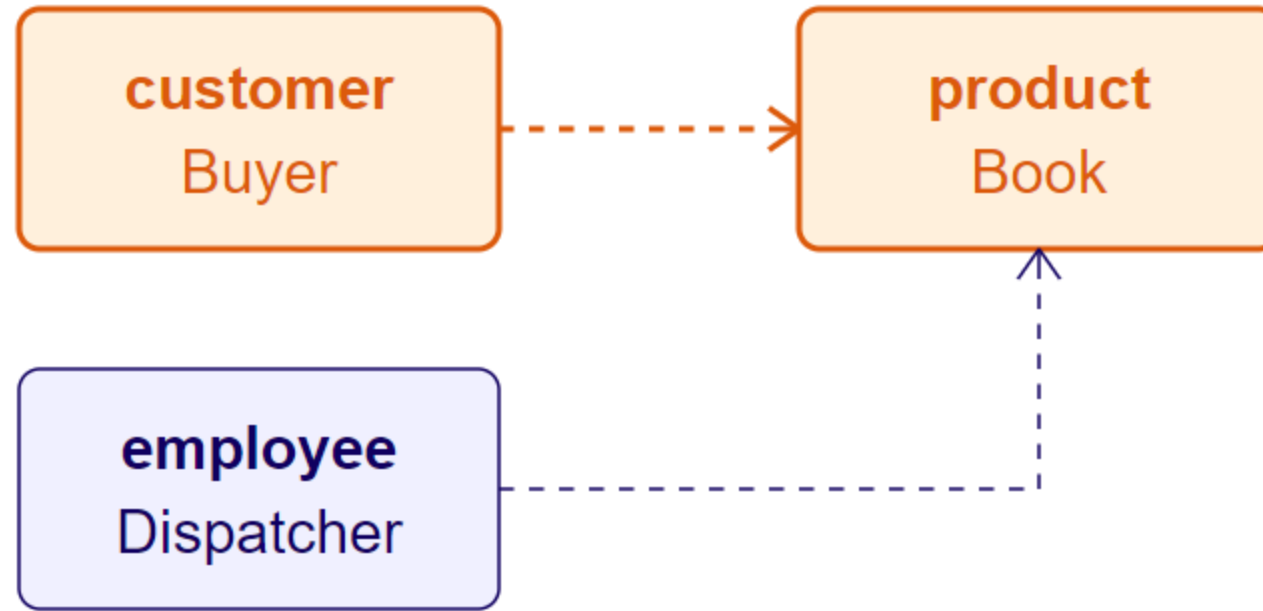
External code calls `customer.Handle(product)`

Motivation for the Decorator Pattern



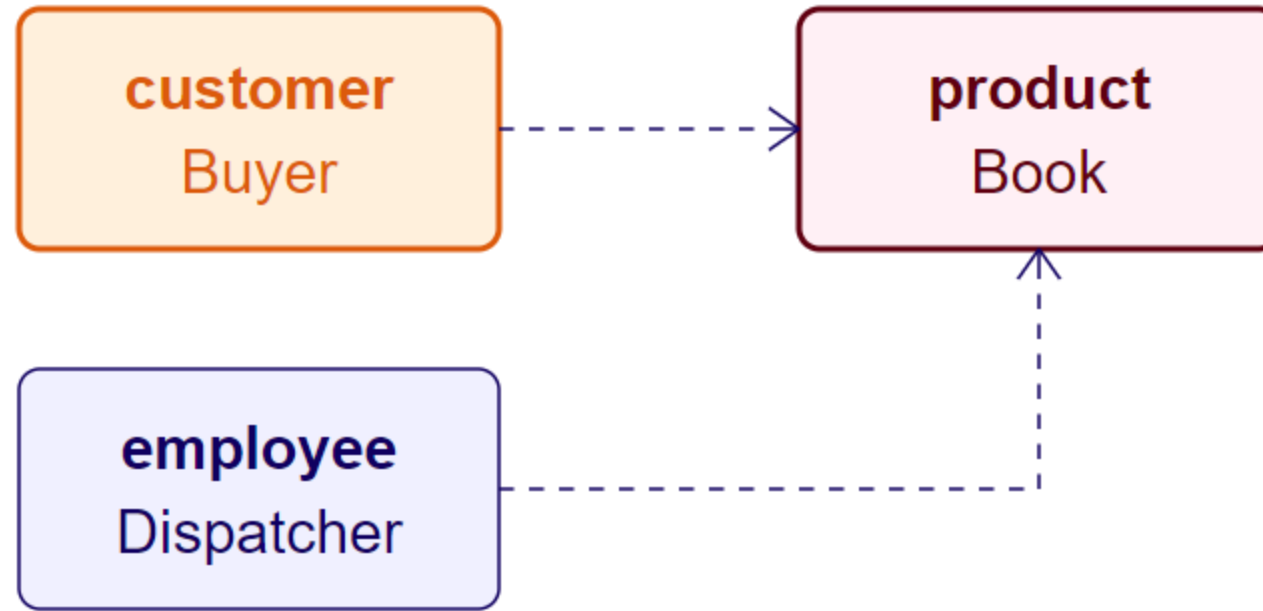
customer calls product.Dimensions

Motivation for the Decorator Pattern

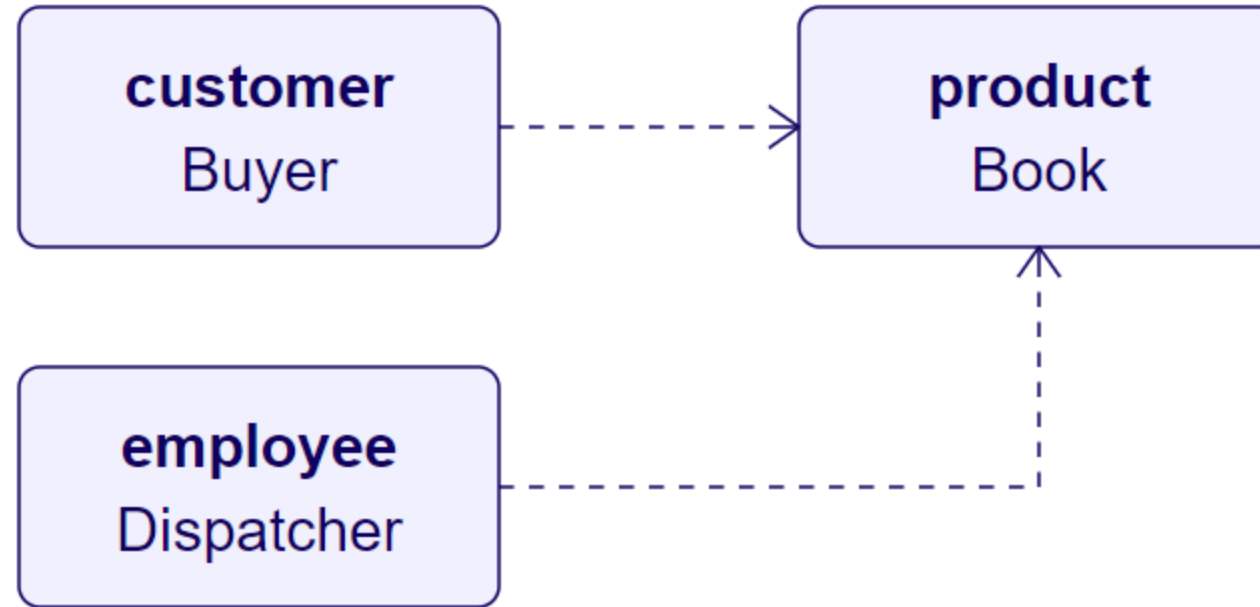


product.Dimensions returns 18.8 x 23.9 x 2.8 cm

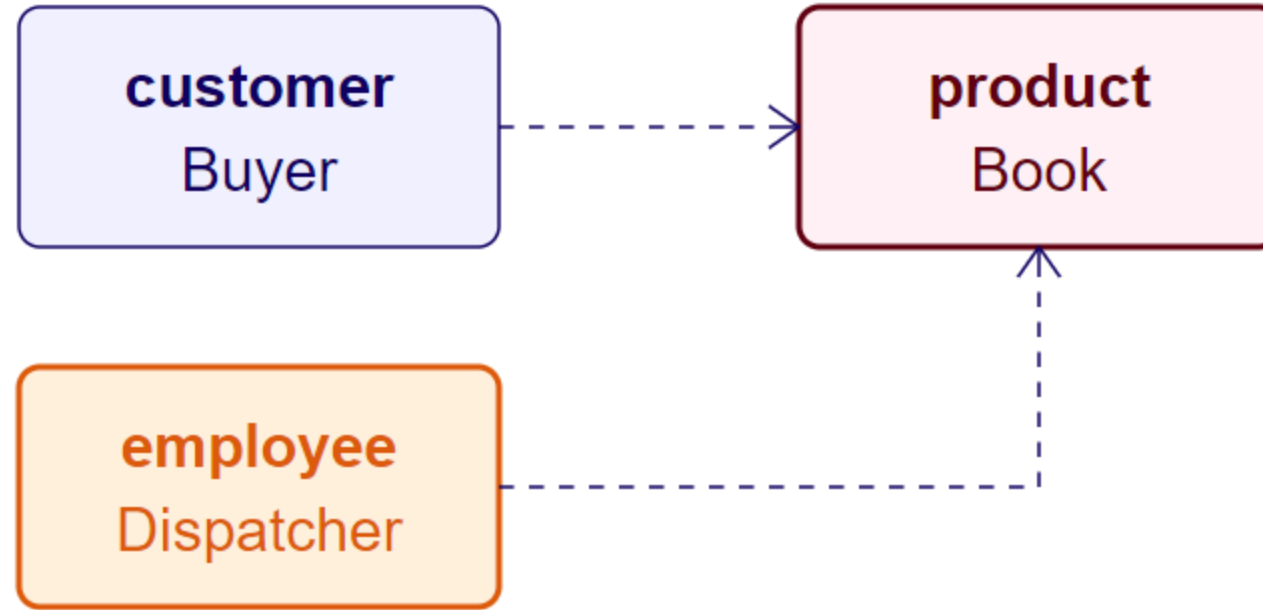
Motivation for the Decorator Pattern



Motivation for the Decorator Pattern

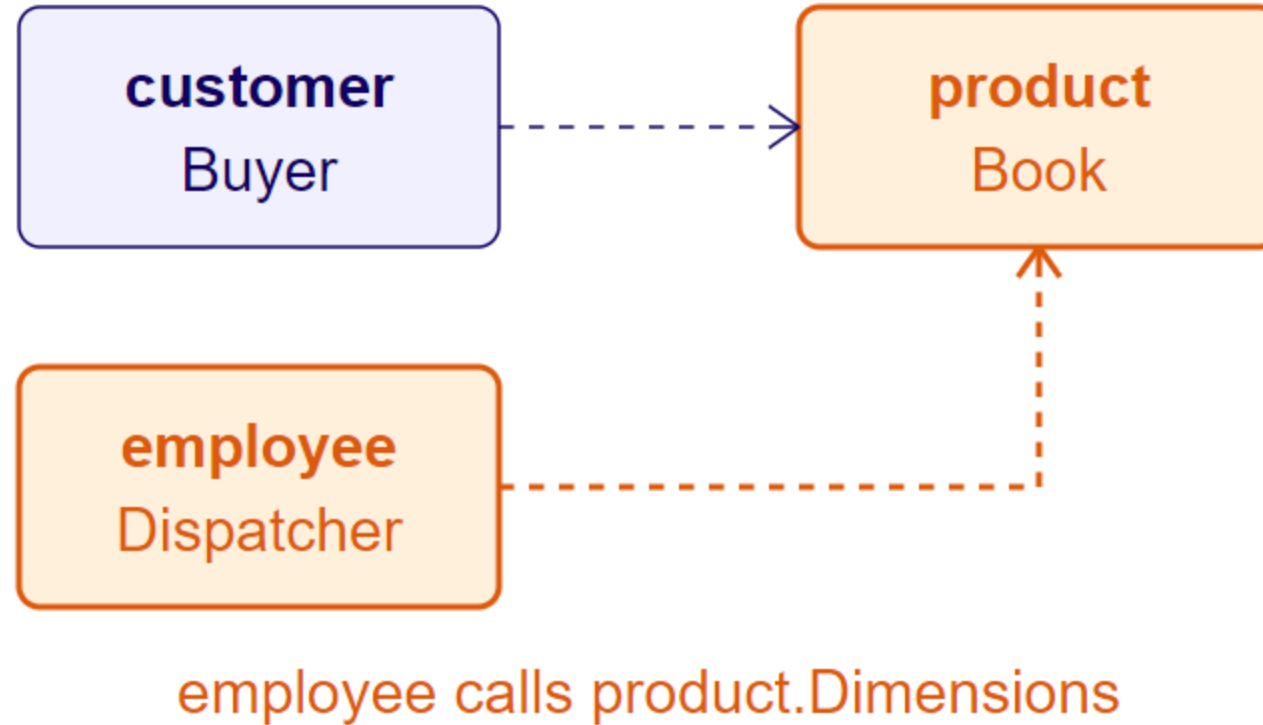


Motivation for the Decorator Pattern

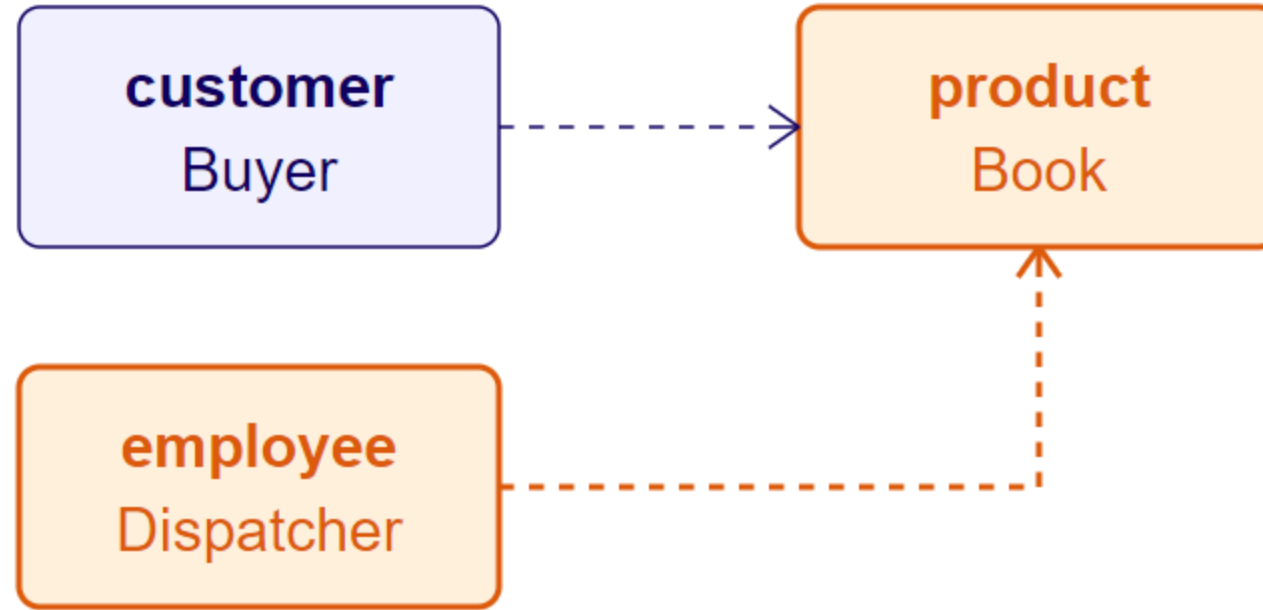


External code calls `employee.Handle(product)`

Motivation for the Decorator Pattern

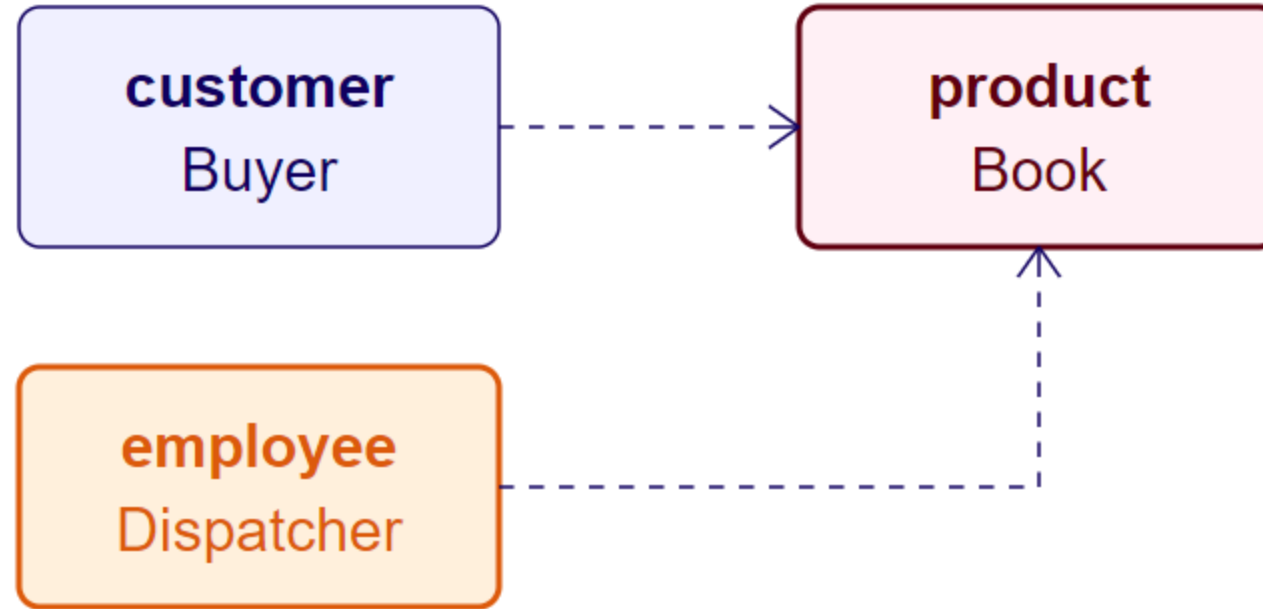


Motivation for the Decorator Pattern

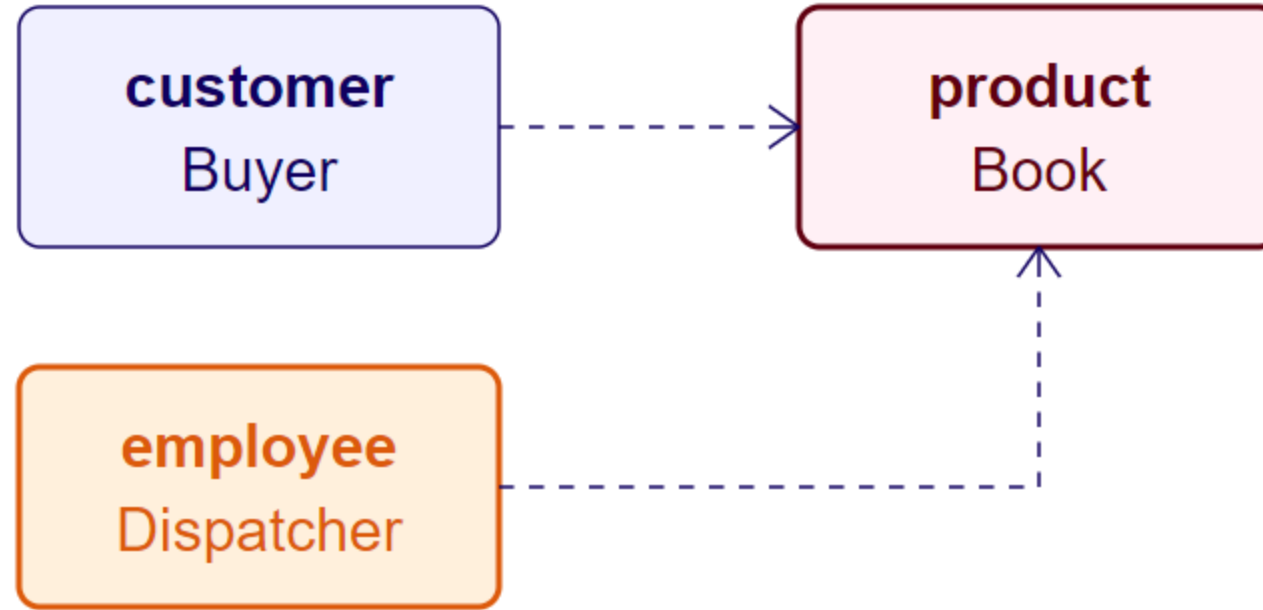


product.Dimensions returns 18.8 x 23.9 x 2.8 cm

Motivation for the Decorator Pattern

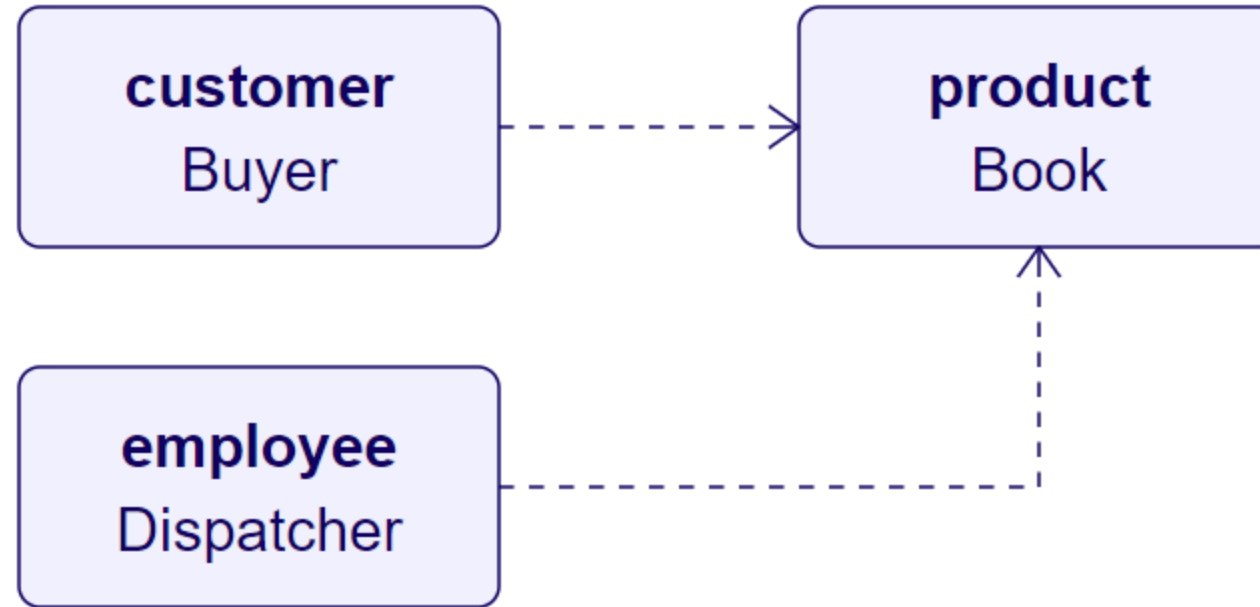


Motivation for the Decorator Pattern



employee adds 7 mm to each dimension: 19.5 x 24.6 x 3.5 cm

Motivation for the Decorator Pattern



BookHandler.cs* Book.cs Clip02Demo.cs* X

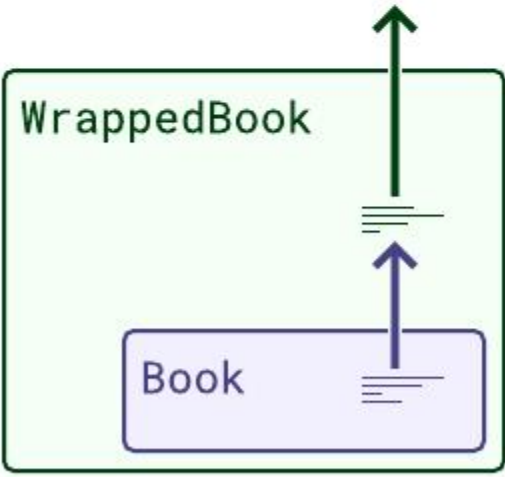
C# Demo Demo.Clip02.Clip02Demo Implementation()

```
namespace Demo.Clip02
{
    1 reference
    class Clip02Demo : Common.Demo
    {
        3 references
        protected override int ClipNumber { get; } = 2;
        private readonly Length mm = Length.Millimeter;

        3 references
        protected override void Implementation()
        {
            Book product = new Book("Design Patterns", new Size(188*mm, 239*mm, 28*mm));

            BookHandler buyer = new BookHandler();
            buyer.Handle(product);

            Book wrappedProduct = new WrappedBook(product);
            BookHandler dispatcher = new BookHandler();
            dispatcher.Handle(wrappedProduct);
        }
    }
}
```



```
graph TD
    subgraph WrappedBook
        Book
    end
    Book --> WrappedBook
    Book --> Book
```

BookHandler.cs* Book.cs Clip02Demo.cs* X

C# Demo Demo.Clip02.Clip02Demo Implementation()

```
namespace Demo.Clip02
{
    1 reference
    class Clip02Demo : Common.Demo
    {
        3 references
        protected override int ClipNumber { get; } = 2;
        private readonly Length mm = Length.Millimeter;

        3 references
        protected override void Implementation()
        {
            Book product = new Book("Design Patterns", new Size(188*mm, 239*mm, 28*mm));

            BookHandler buyer = new BookHandler();
            buyer.Handle(product);

            Book wrappedProduct = new WrappedBook(product);
            BookHandler dispatcher = new BookHandler();
            dispatcher.Handle(wrappedProduct);
        }
    }
}
```

```
classDiagram
    class Book
    class WrappedBook
    Book <|-- WrappedBook
```

```

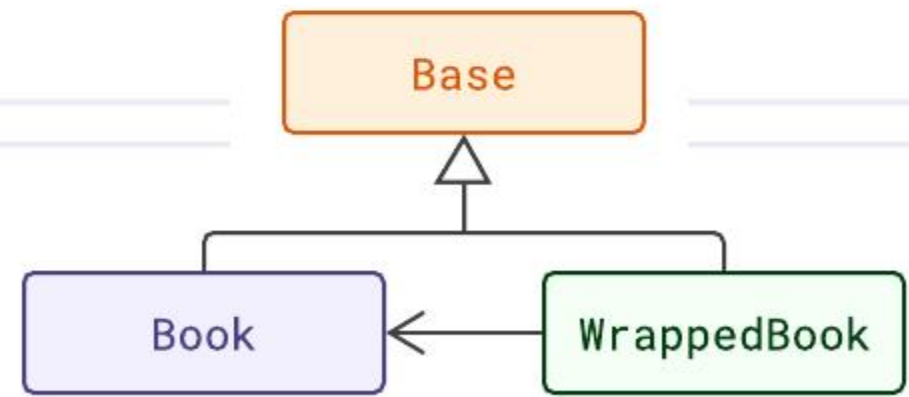
namespace Demo.Clip02
{
    1 reference
    class Clip02Demo : Common.Demo
    {
        3 references
        protected override int ClipNumber { get; } = 2;
        private readonly Length mm = Length.Millimeter;

        3 references
        protected override void Implementation()
        {
            Book product = new Book("Design Patterns", new Size(188*mm, 239*mm, 28*mm));

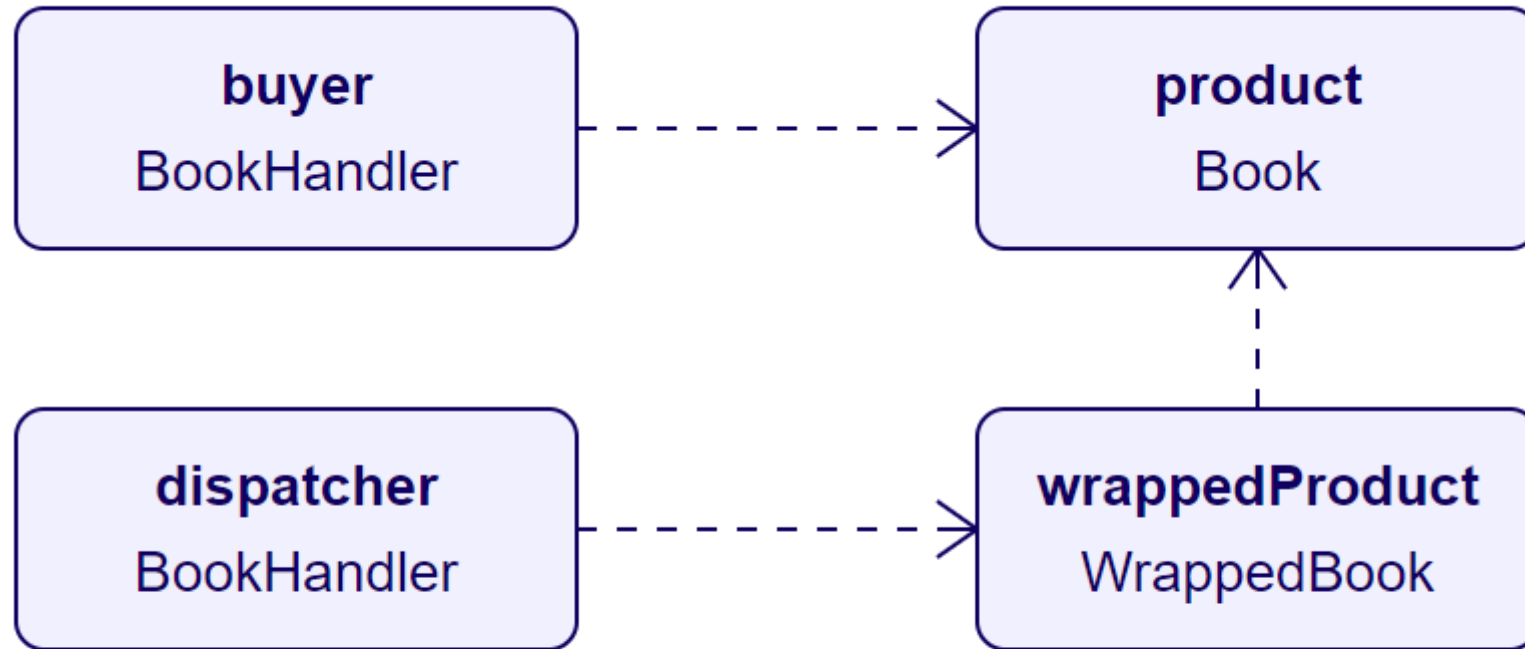
            BookHandler buyer = new BookHandler();
            buyer.Handle(product);

            Book wrappedProduct = new WrappedBook(product);
            BookHandler dispatcher = new BookHandler();
            dispatcher.Handle(wrappedProduct);
        }
    }
}

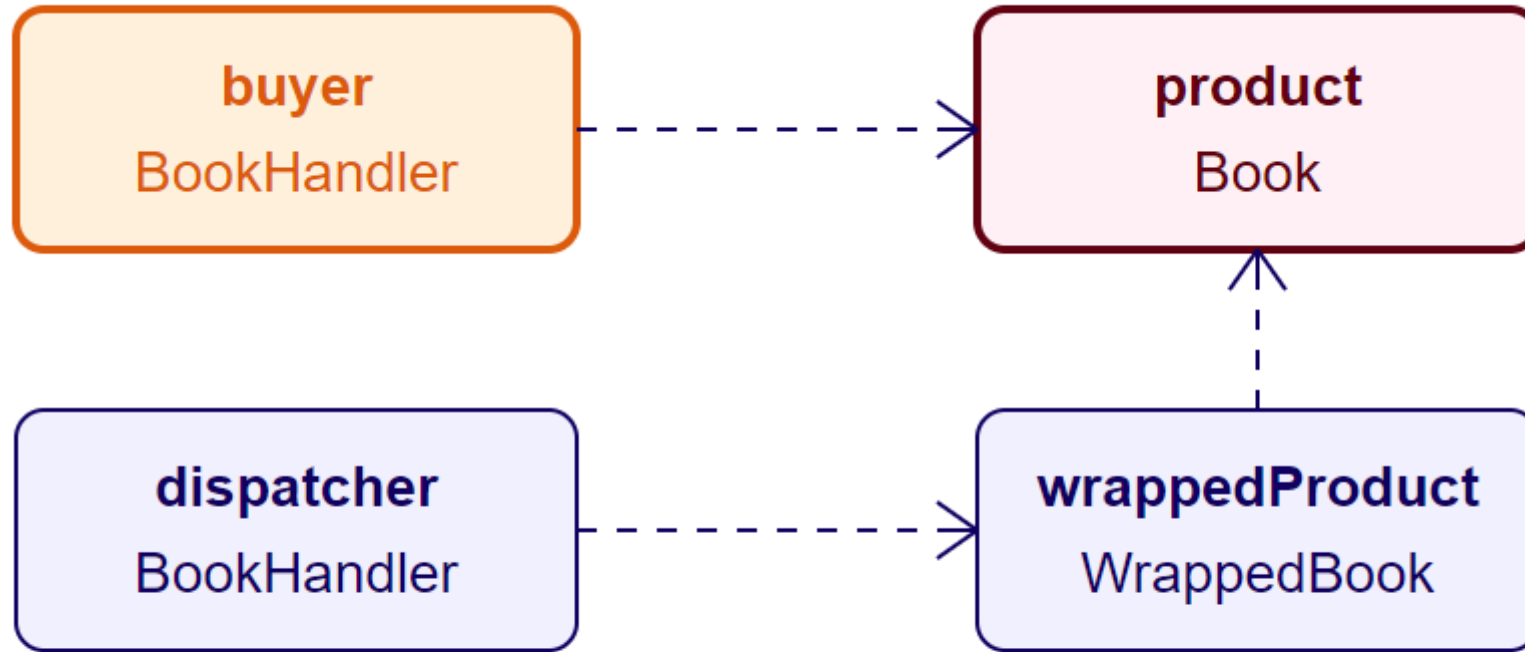
```



Decorating Through Subclassing

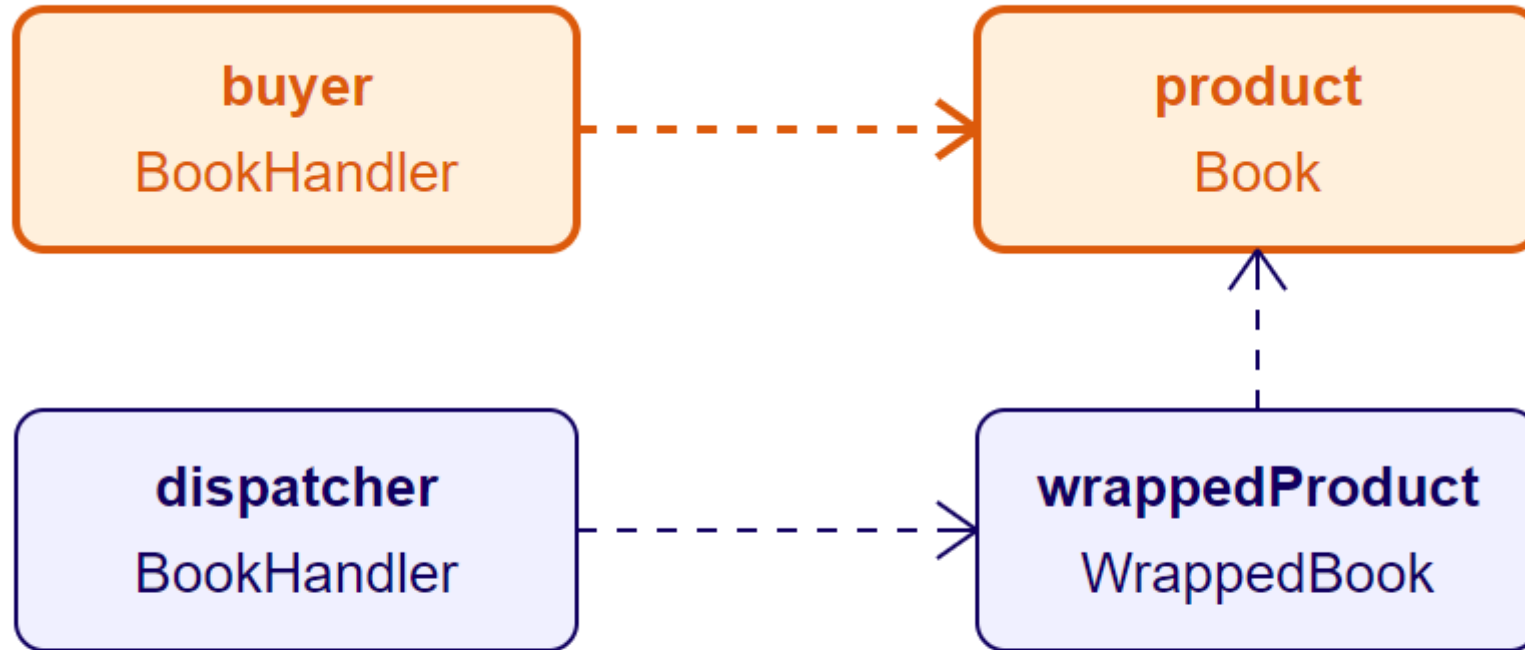


Decorating Through Subclassing



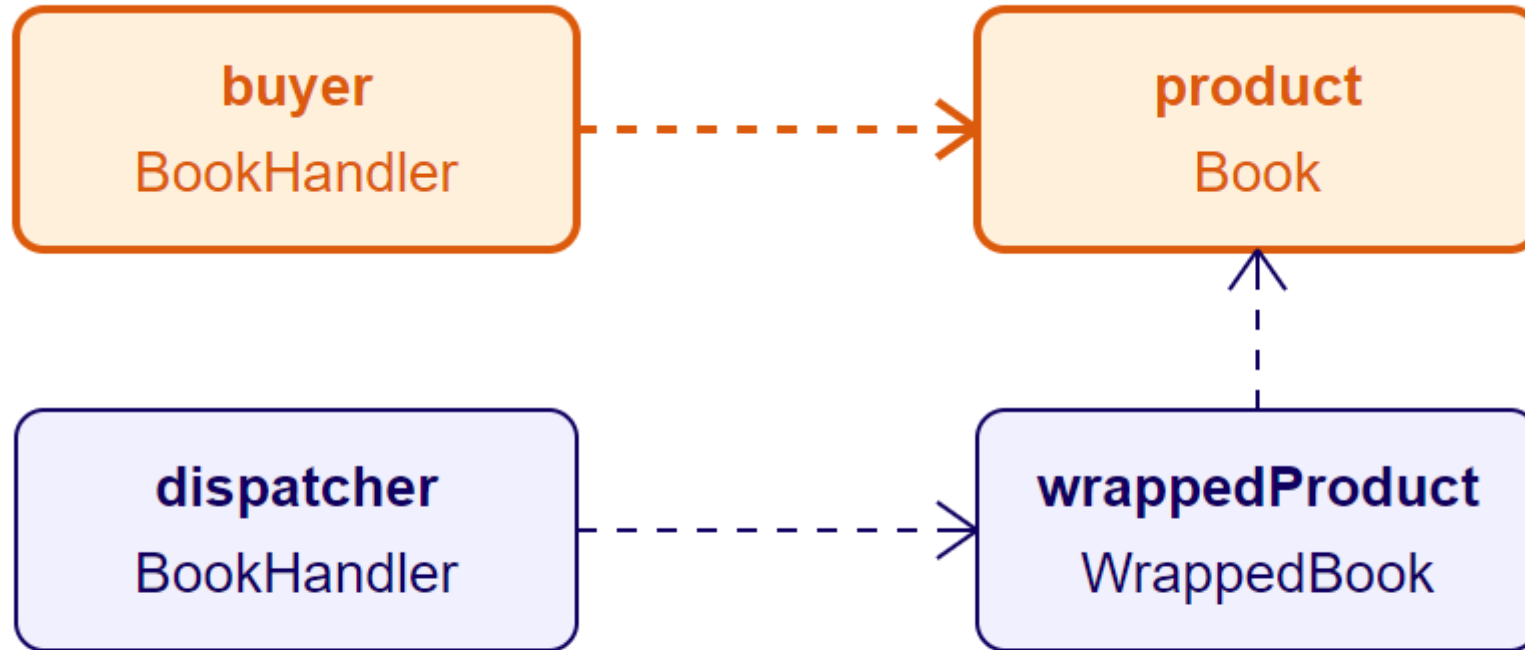
External code calls `buyer.Handle(product)`

Decorating Through Subclassing



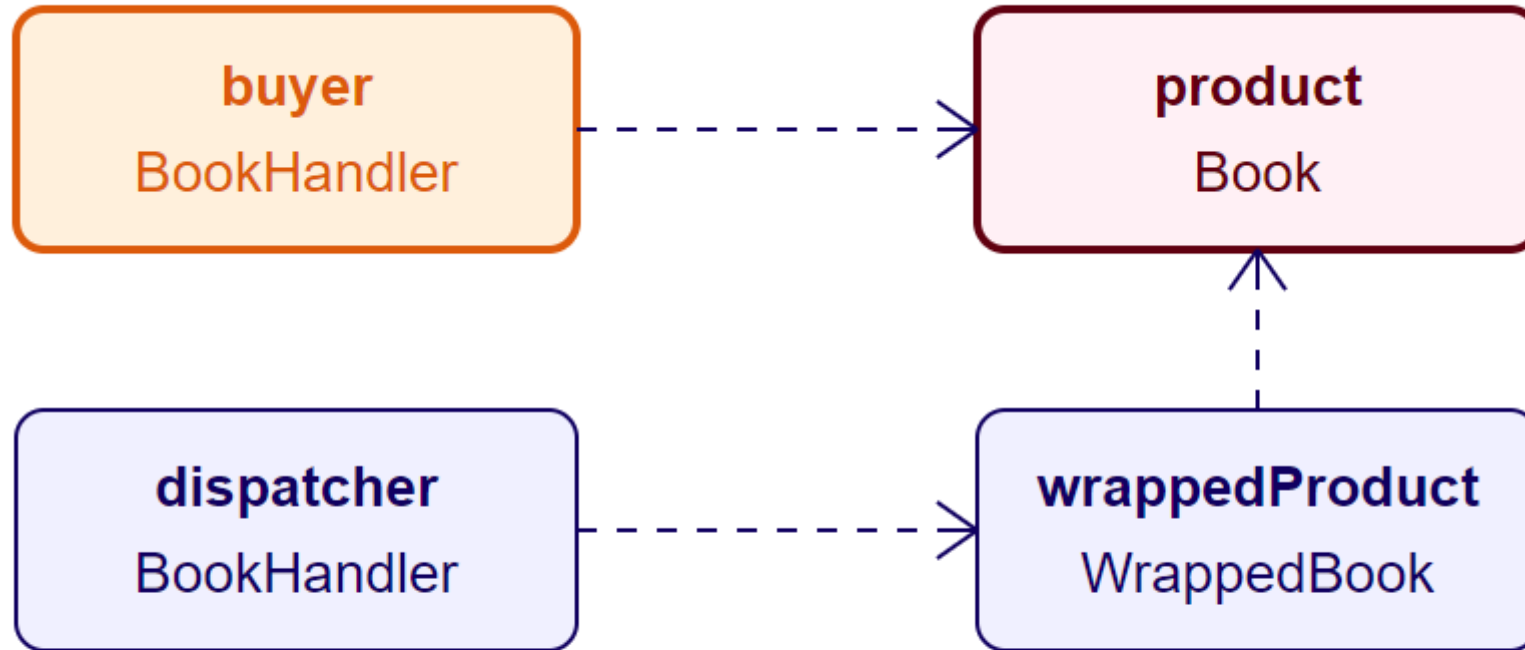
buyer calls product.Dimensions

Decorating Through Subclassing

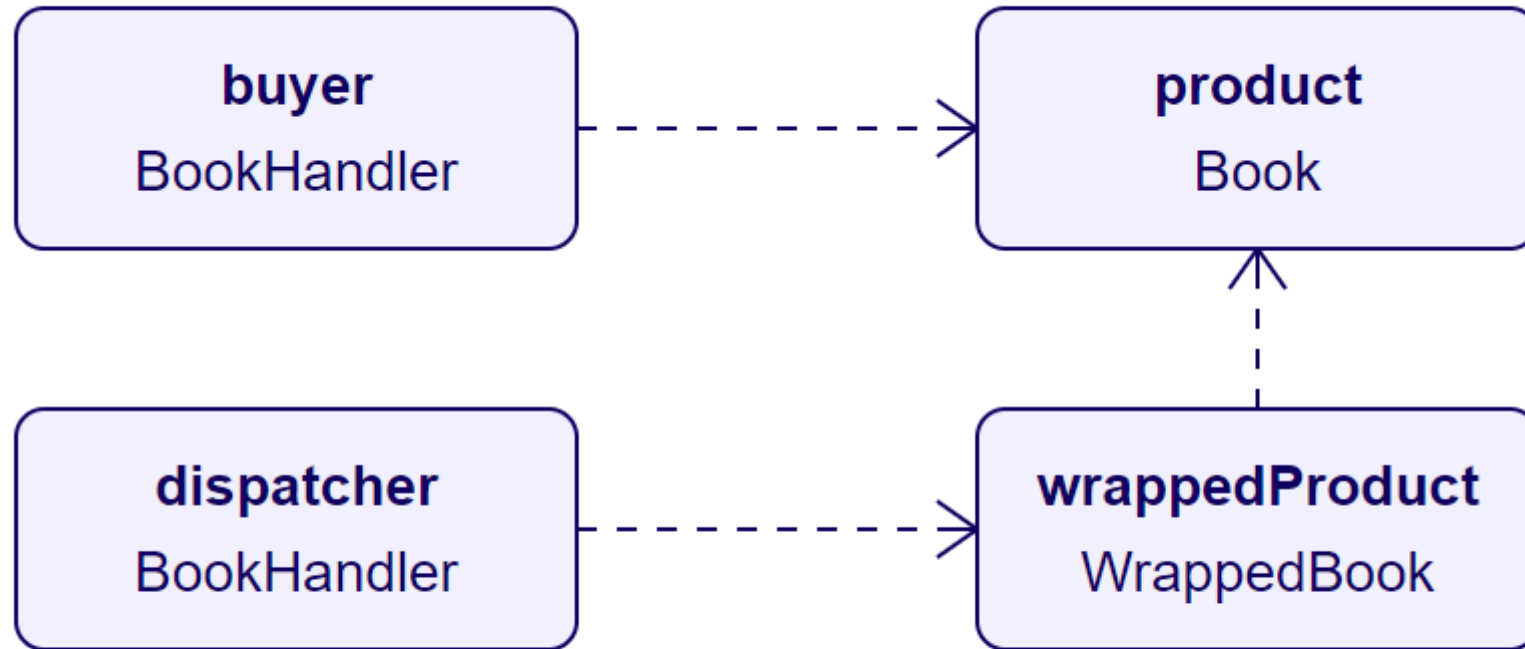


product.Dimensions returns 18.8 x 23.9 x 2.8 cm

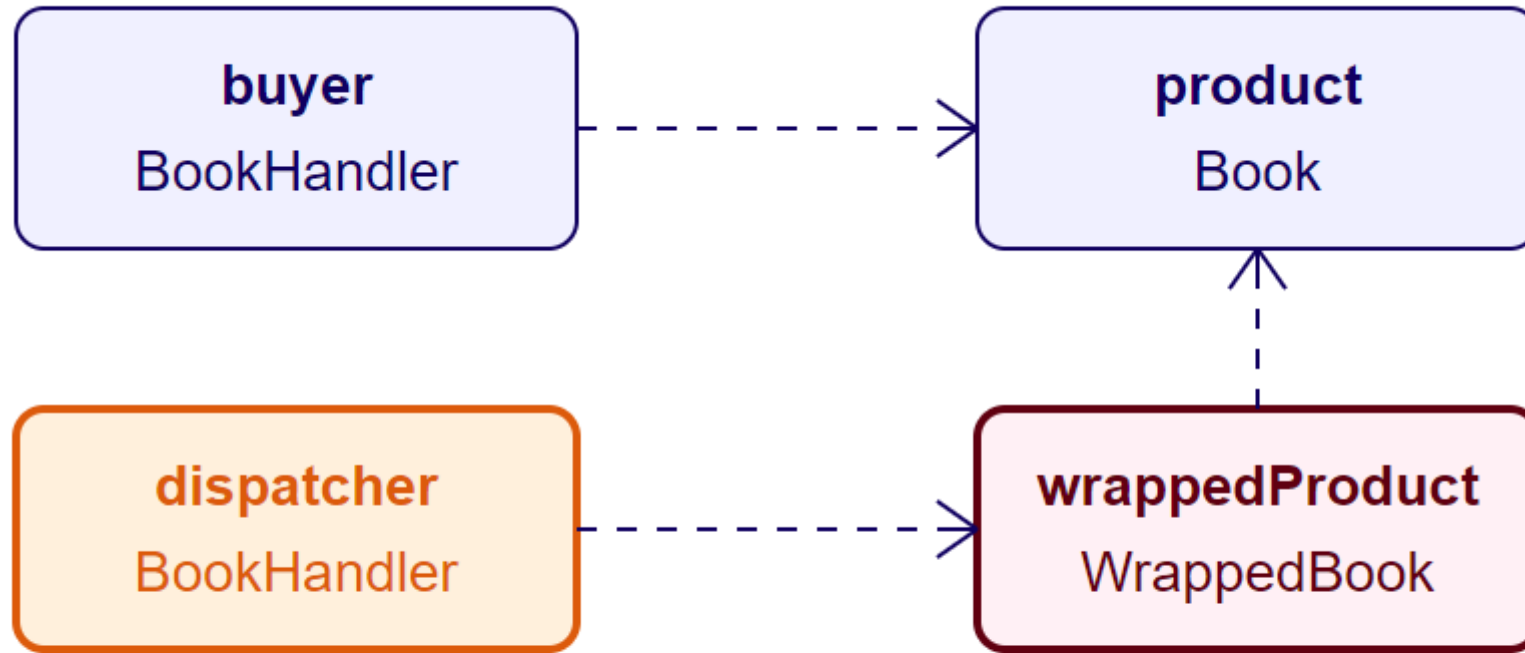
Decorating Through Subclassing



Decorating Through Subclassing

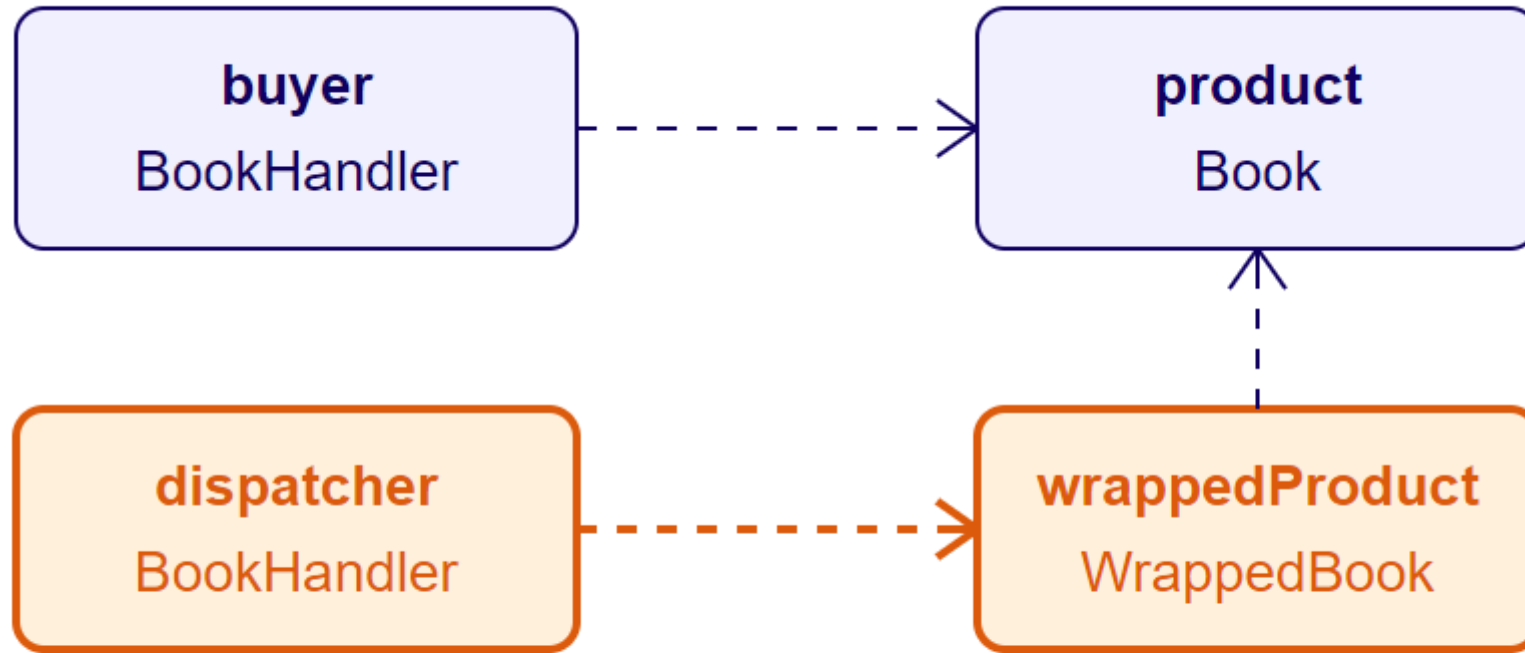


Decorating Through Subclassing



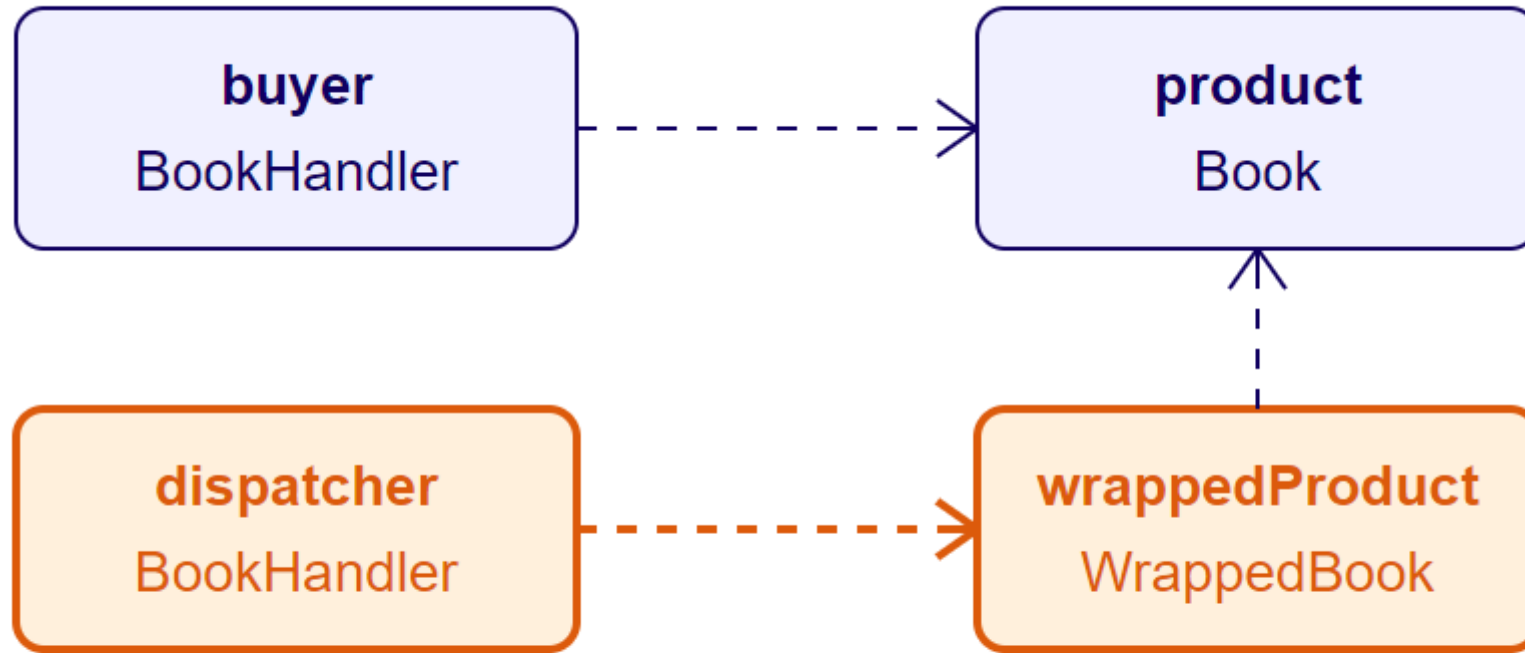
External code calls `dispatcher.Handle(wrappedProduct)`

Decorating Through Subclassing



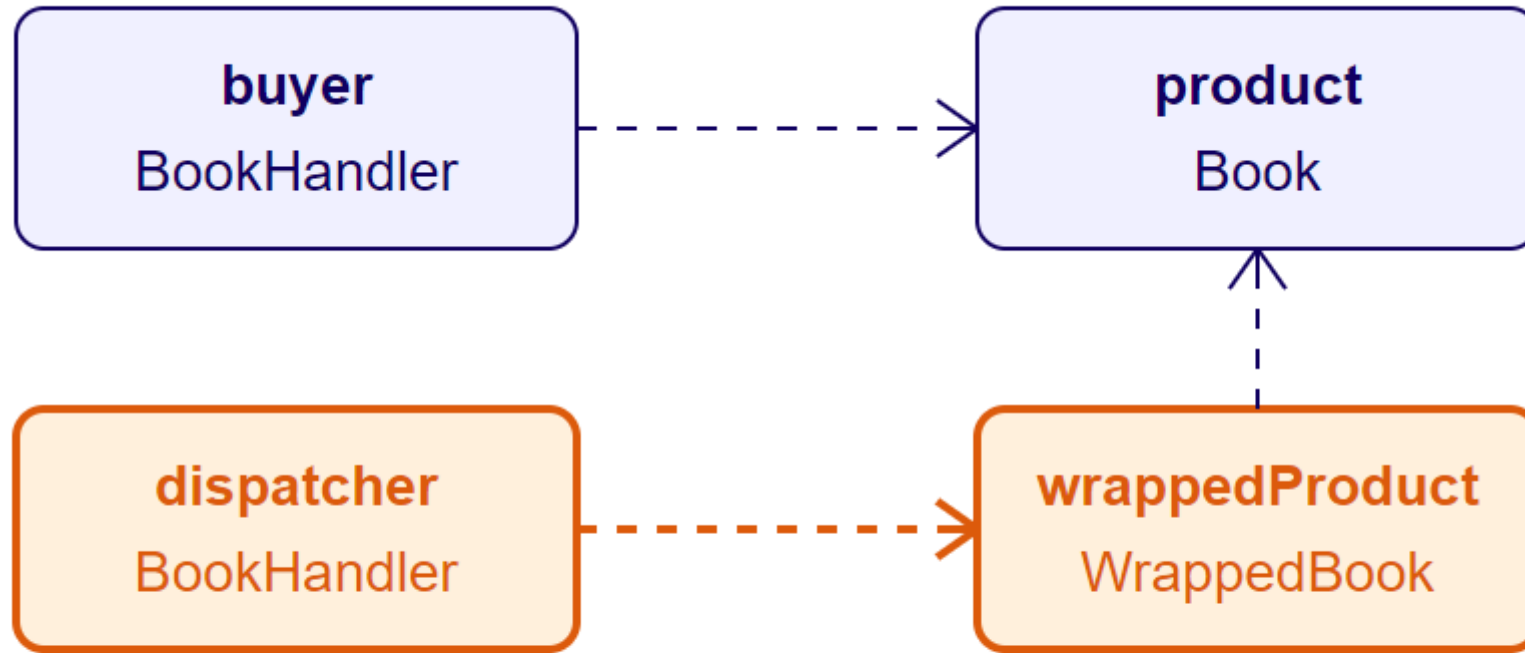
dispatcher calls wrappedProduct.Dimensions

Decorating Through Subclassing



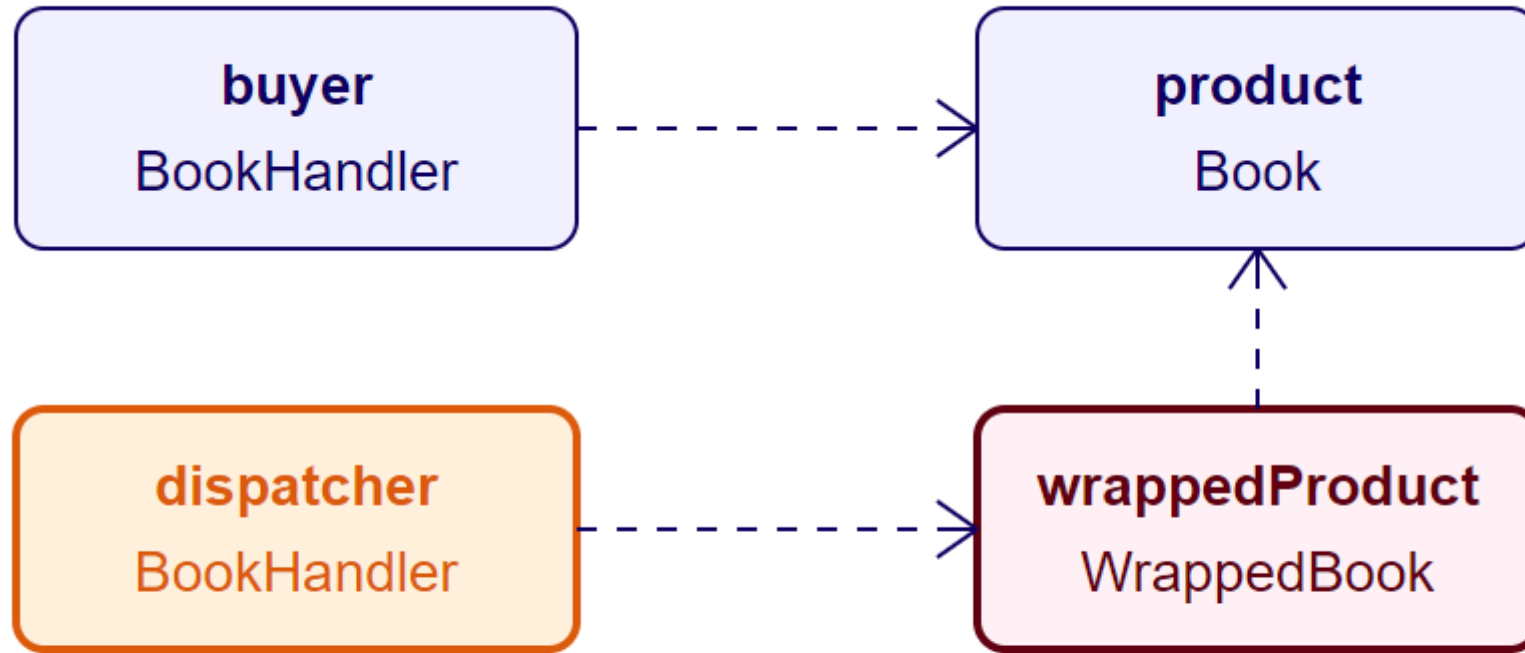
Adding 7 mm to base dimensions 18.8 x 23.9 x 2.8 cm

Decorating Through Subclassing

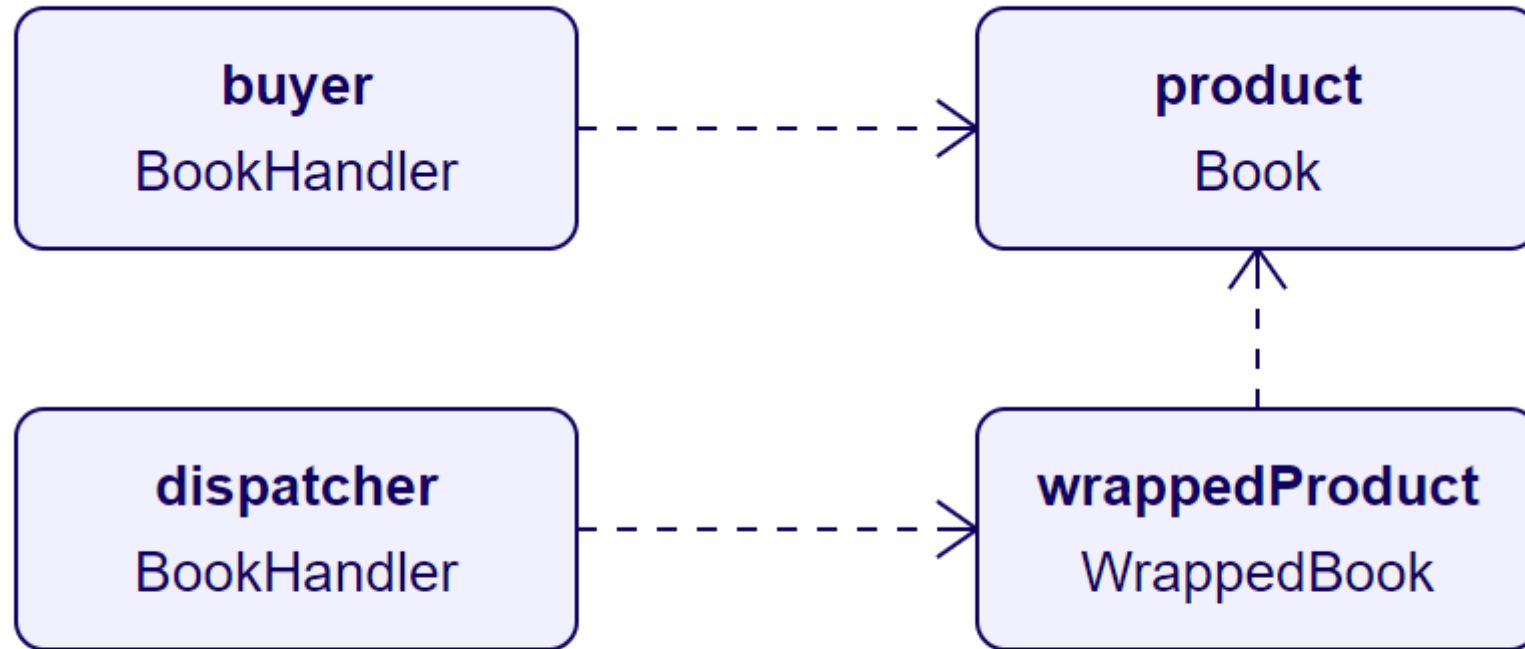


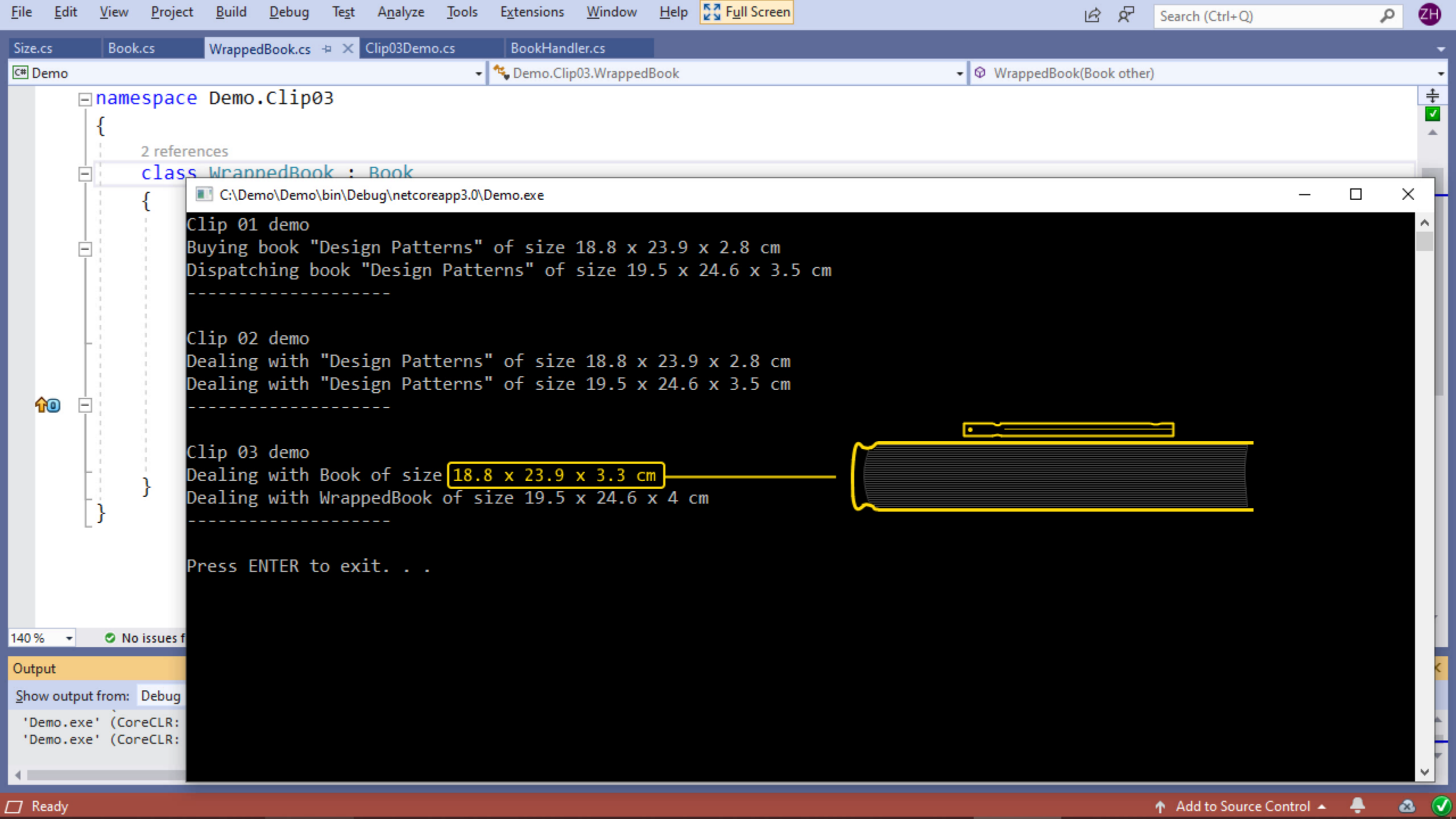
wrappedProduct.Dimensions returns 19.5 x 24.6 x 3.5 cm

Decorating Through Subclassing



Decorating Through Subclassing



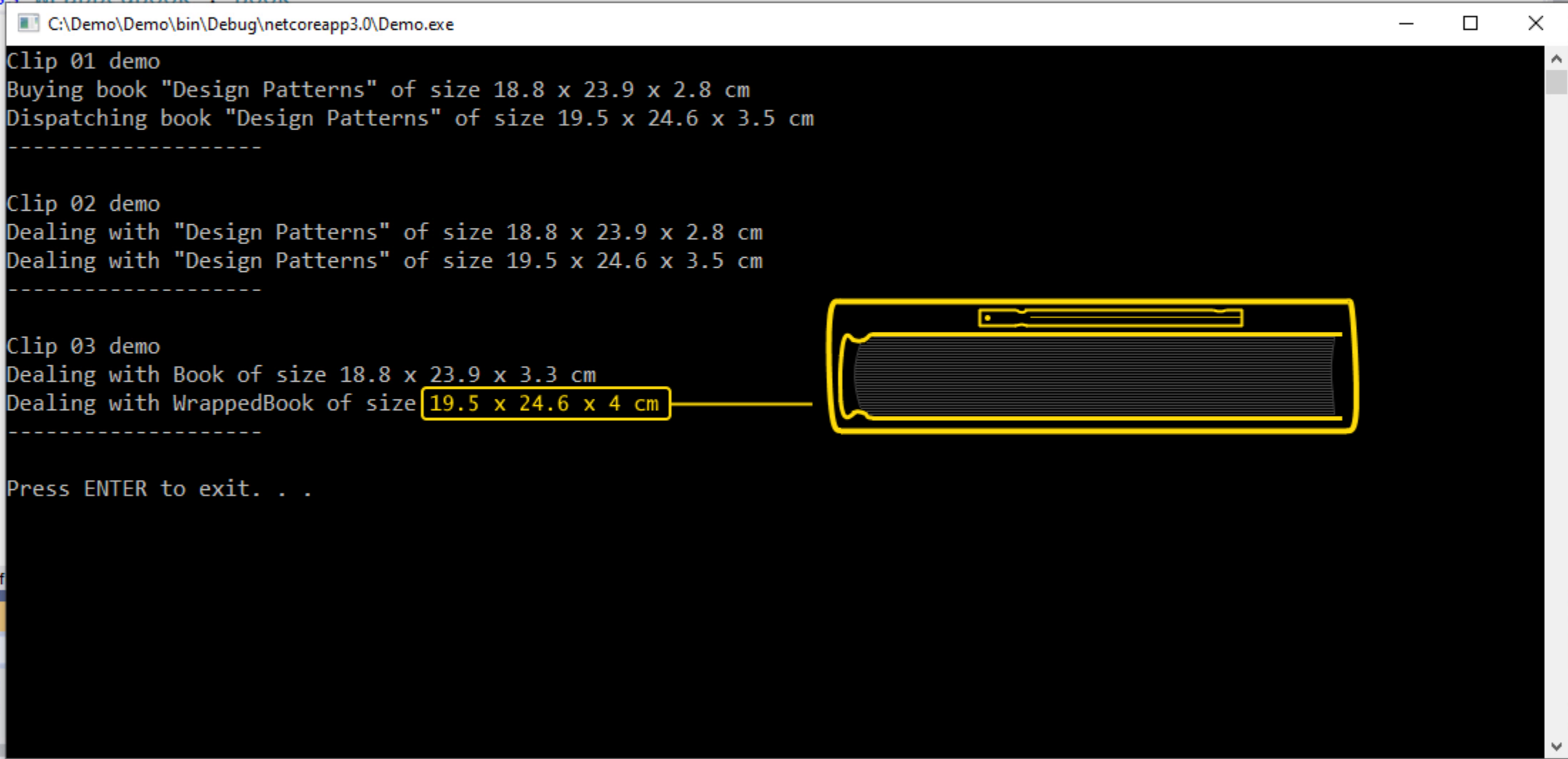


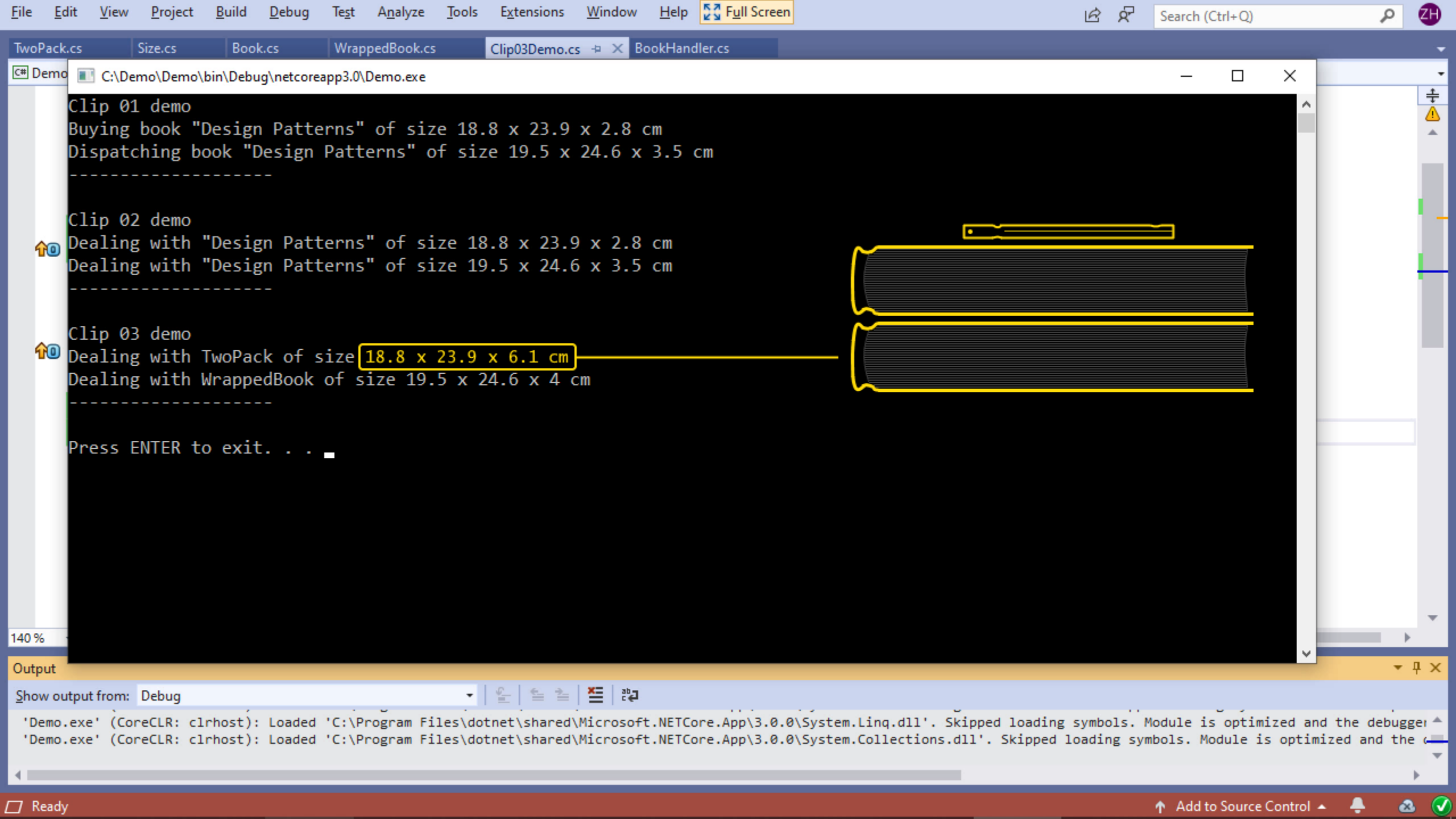

```
namespace Demo.Clip03
{
    2 references
    class WrappedBook : Book
    {
        Clip 01 demo
        Buying book "Design Patterns" of size 18.8 x 23.9 x 2.8 cm
        Dispatching book "Design Patterns" of size 19.5 x 24.6 x 3.5 cm
        -----

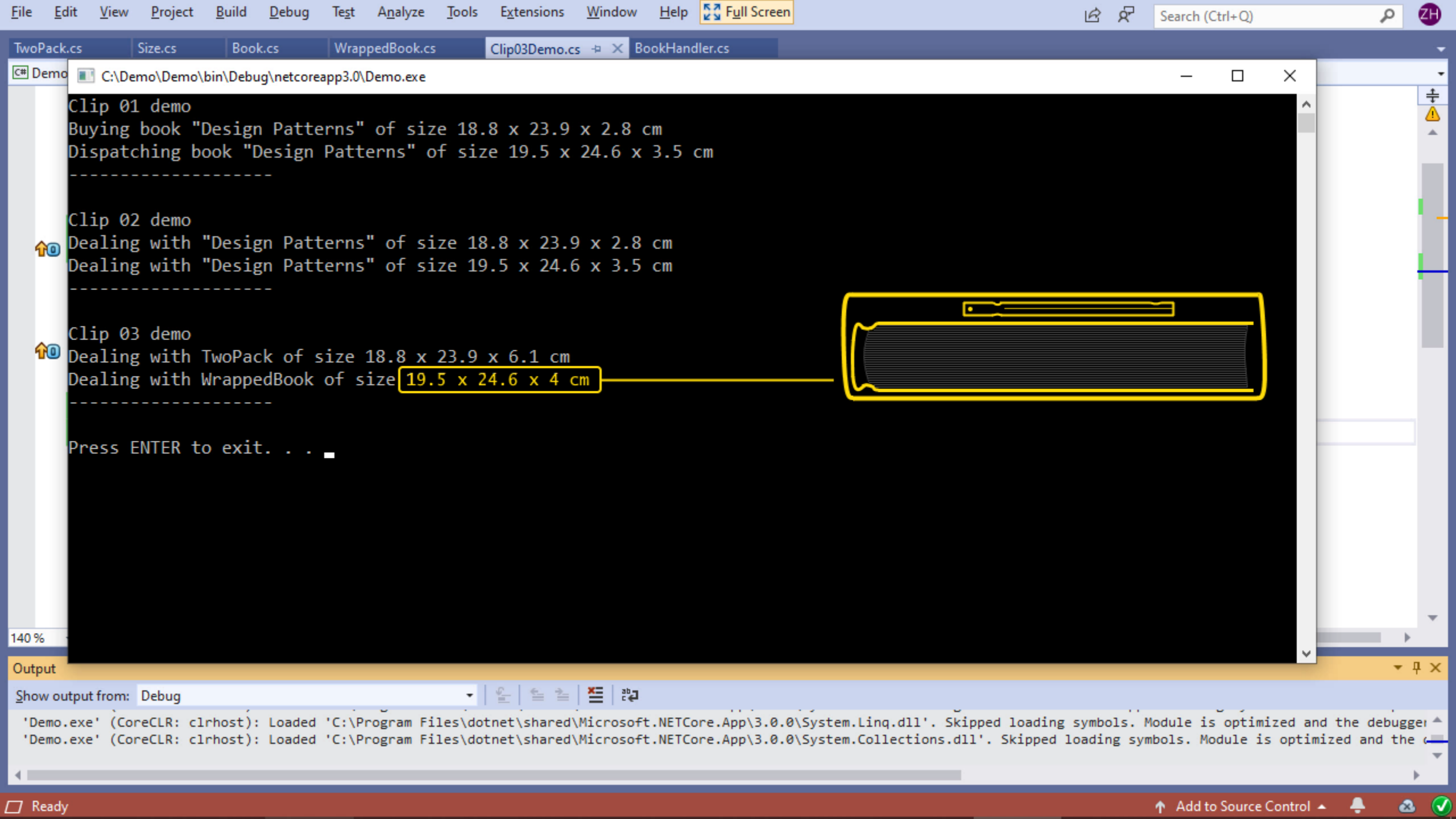
        Clip 02 demo
        Dealing with "Design Patterns" of size 18.8 x 23.9 x 2.8 cm
        Dealing with "Design Patterns" of size 19.5 x 24.6 x 3.5 cm
        -----

        Clip 03 demo
        Dealing with Book of size 18.8 x 23.9 x 3.3 cm
        Dealing with WrappedBook of size 19.5 x 24.6 x 4 cm
        -----

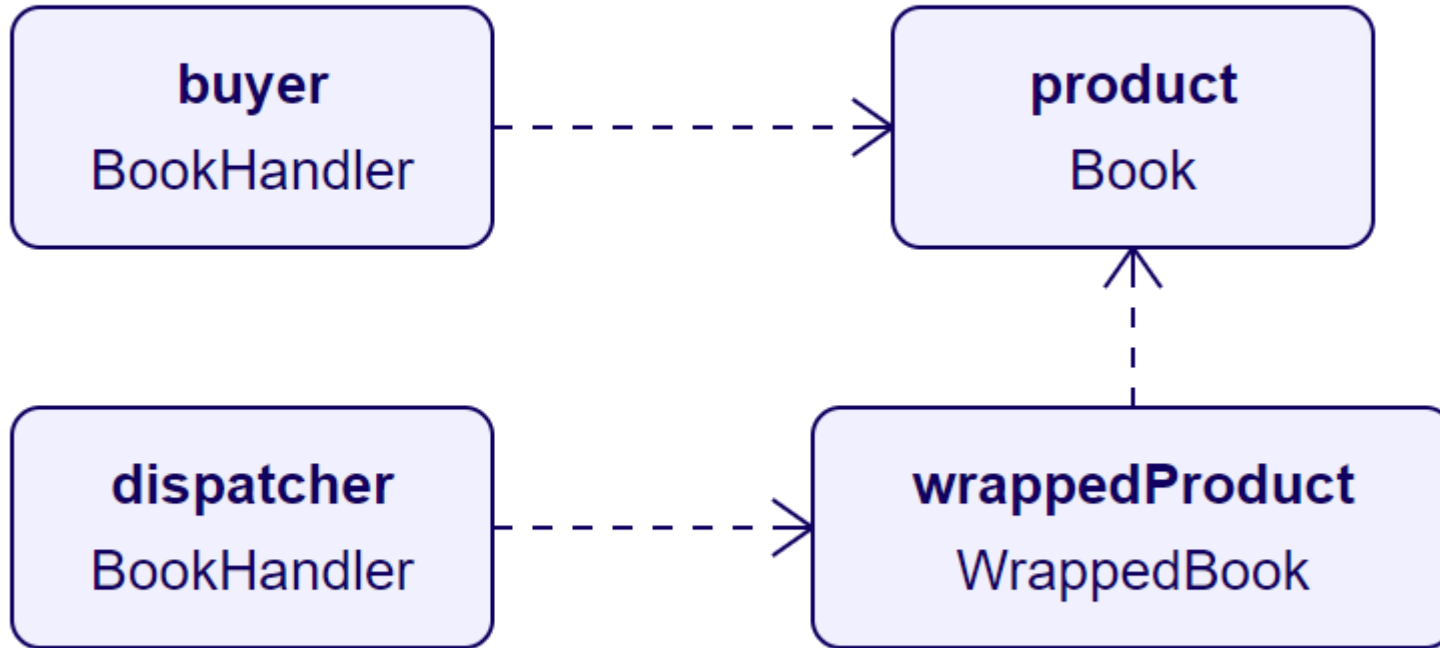
        Press ENTER to exit. . .
    }
}
```



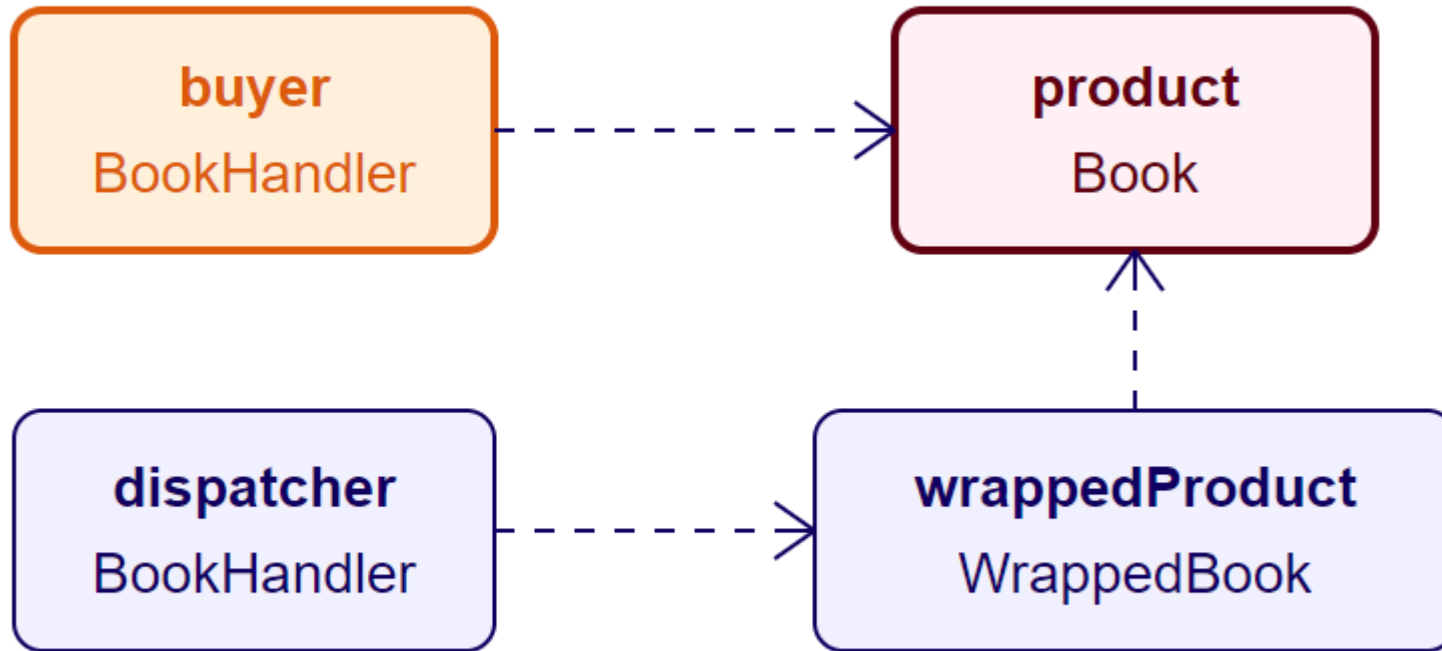




Issues with Subclassing Decorator

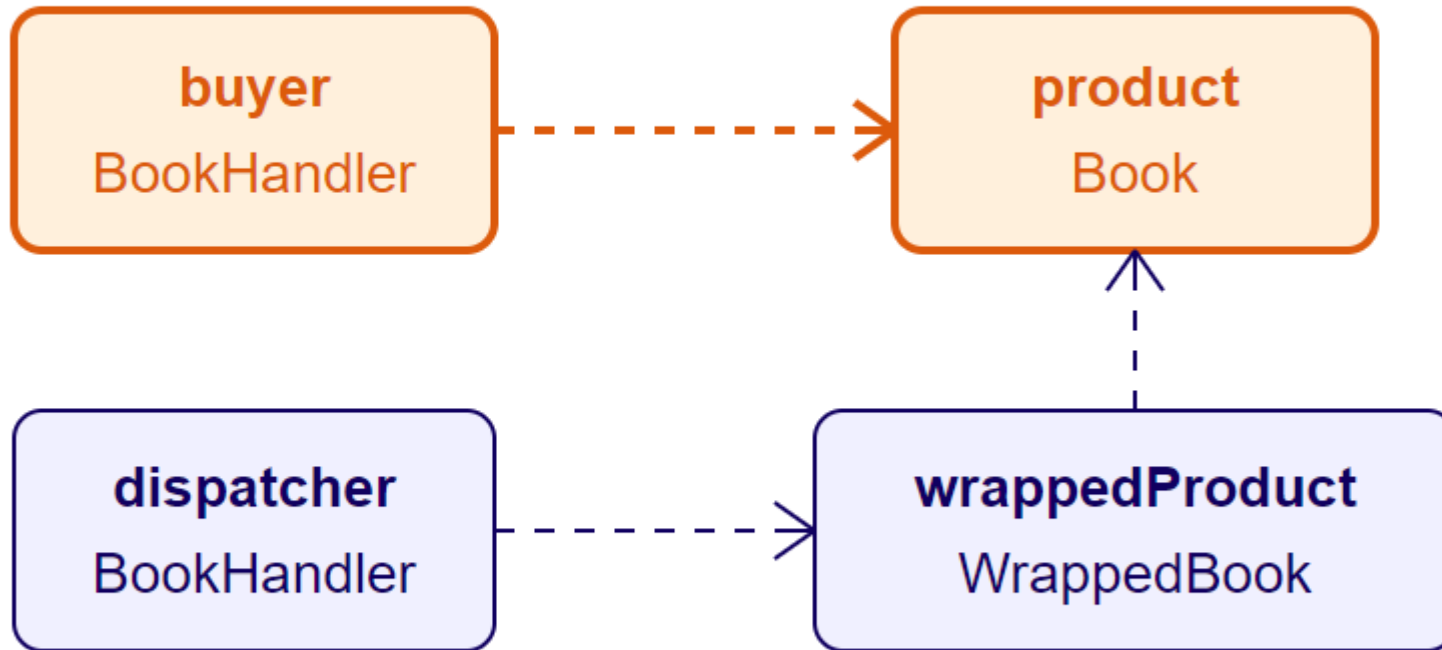


Issues with Subclassing Decorator



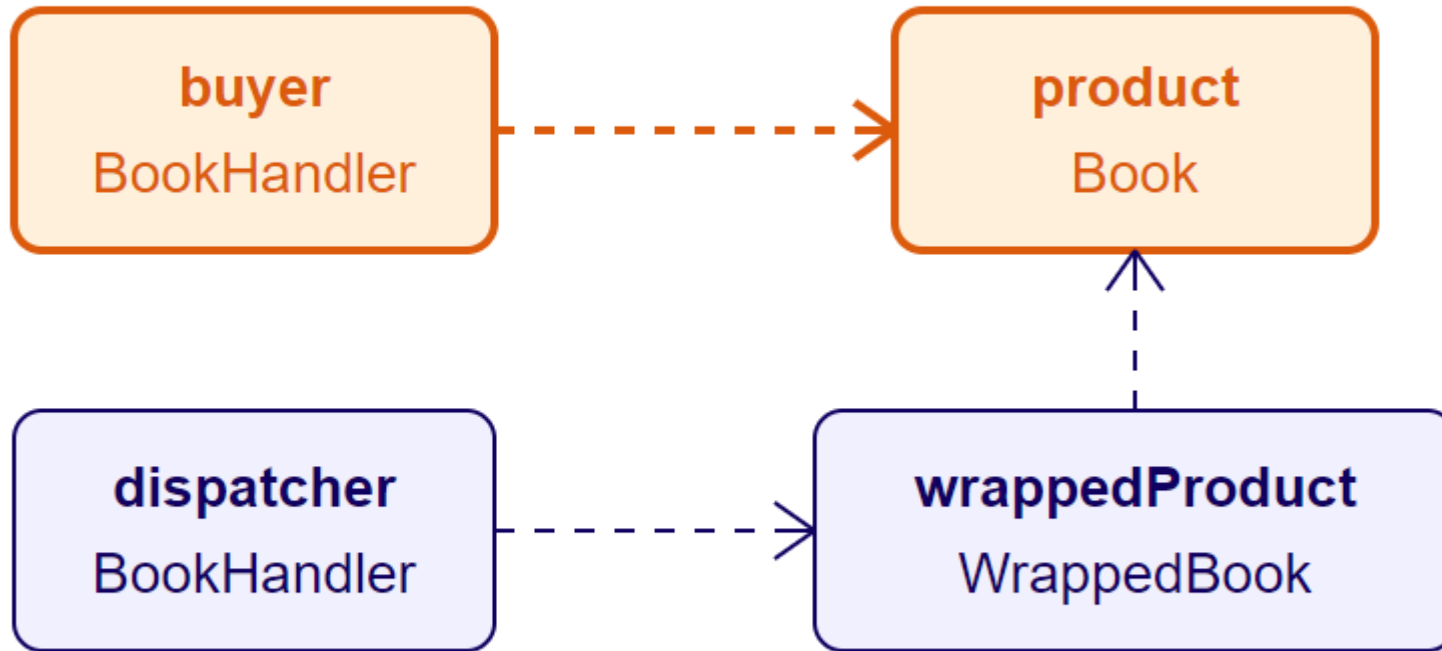
External code calls `buyer.Handle(product)`

Issues with Subclassing Decorator



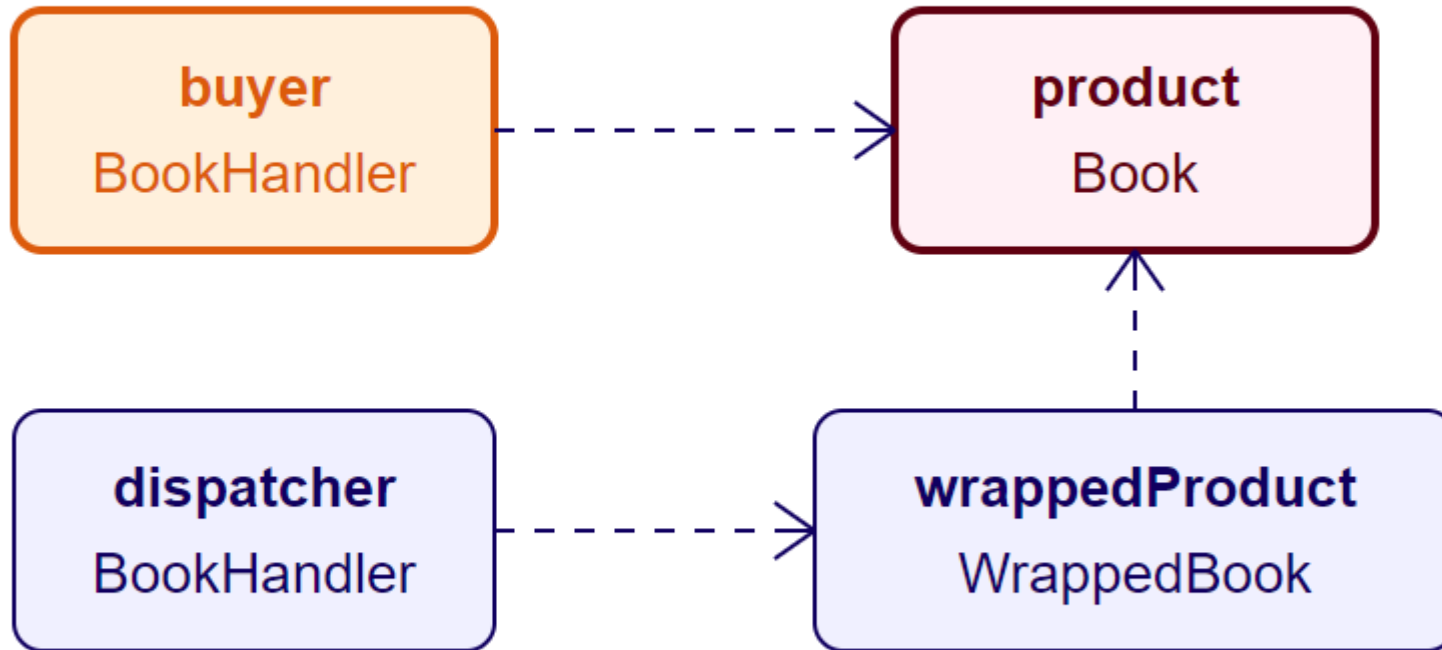
buyer calls product.GetDimensions(142 x 125 x 5 mm)

Issues with Subclassing Decorator

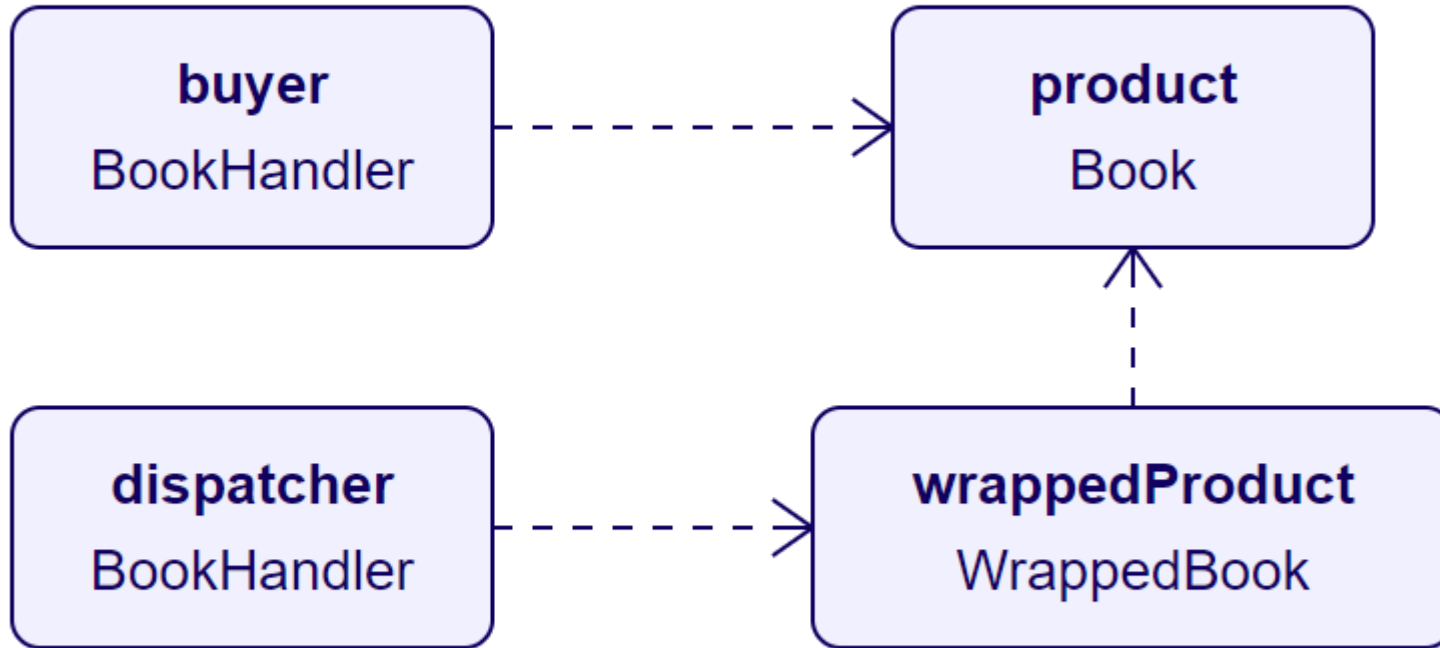


`product.GetDimensions()` returns 18.8 x 23.9 x 3.3 cm

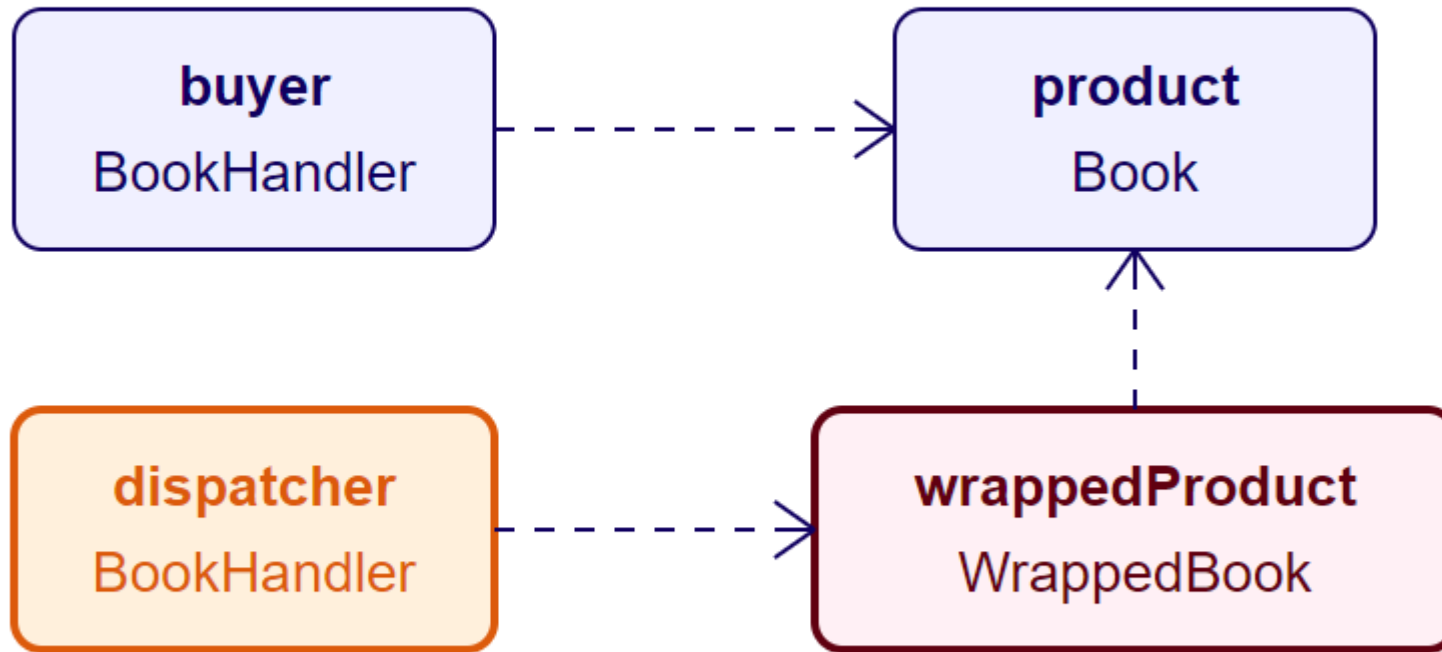
Issues with Subclassing Decorator



Issues with Subclassing Decorator

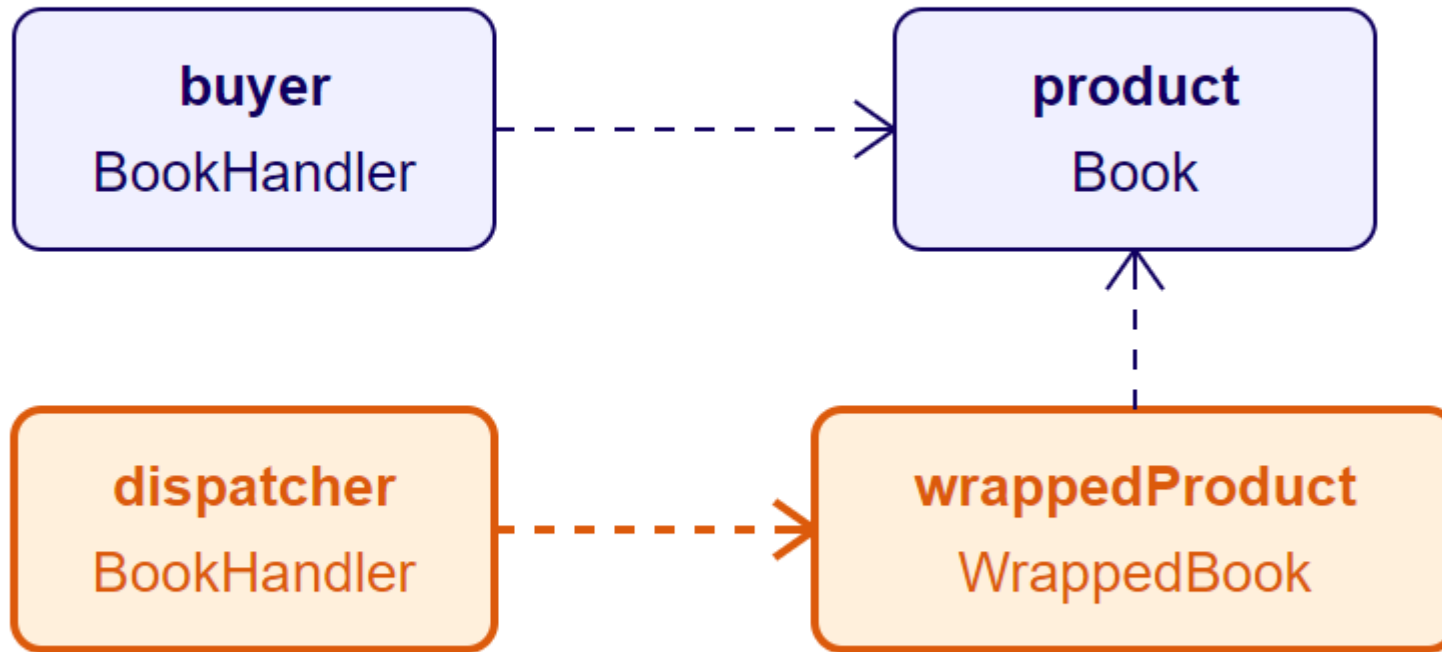


Issues with Subclassing Decorator



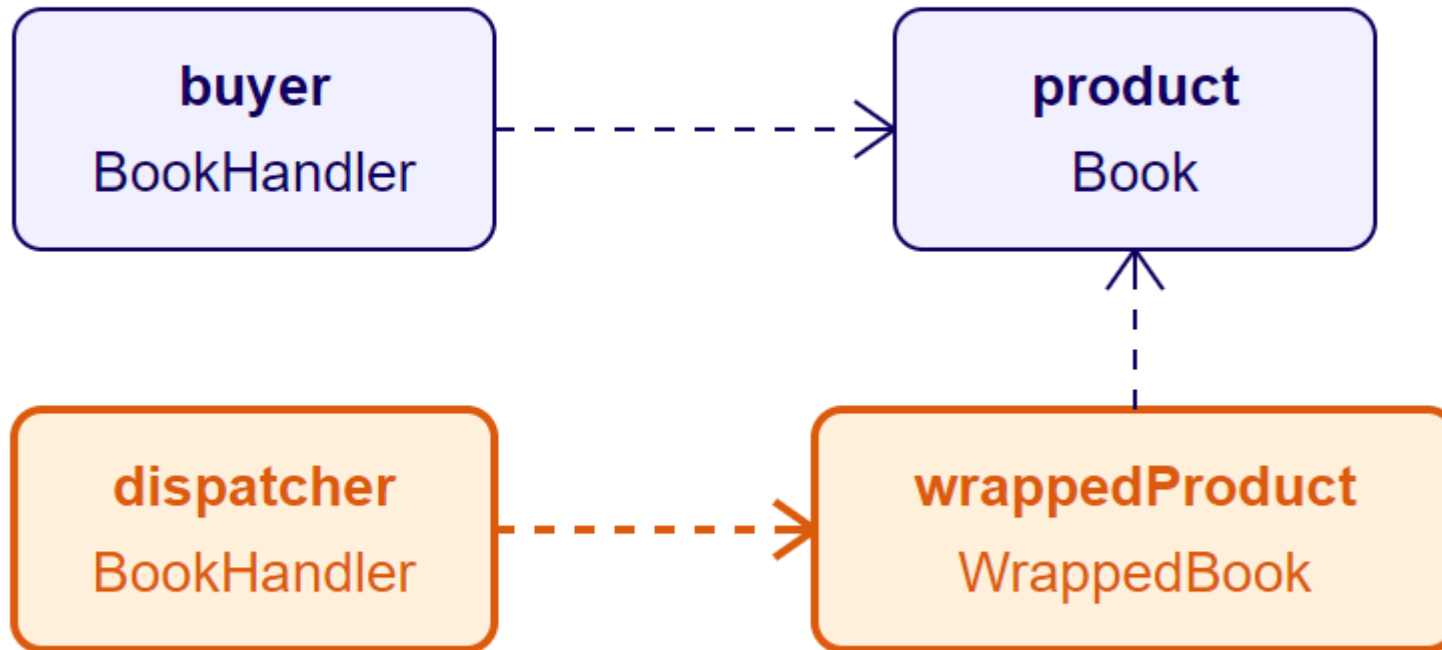
External code calls `dispatcher.Handle(wrappedProduct)`

Issues with Subclassing Decorator



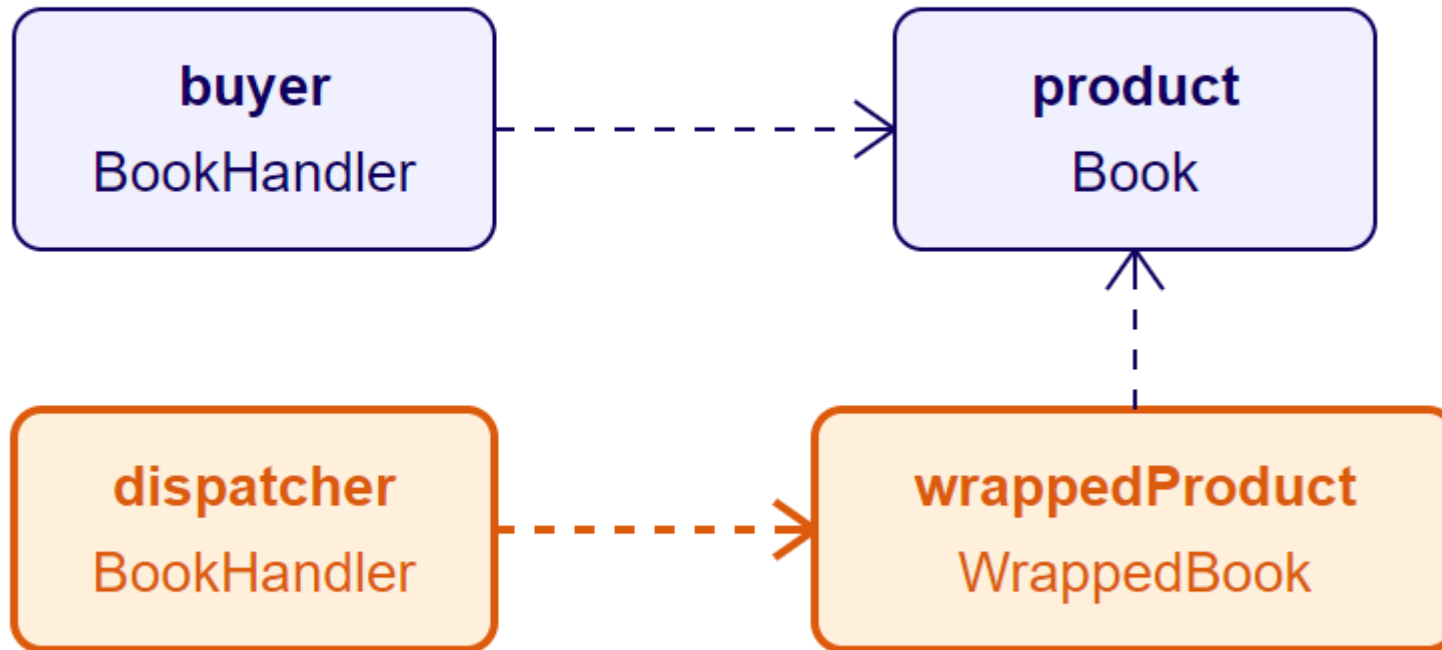
dispatcher calls wrappedProduct.GetDimensions(142 x 125 x 5 mm)

Issues with Subclassing Decorator



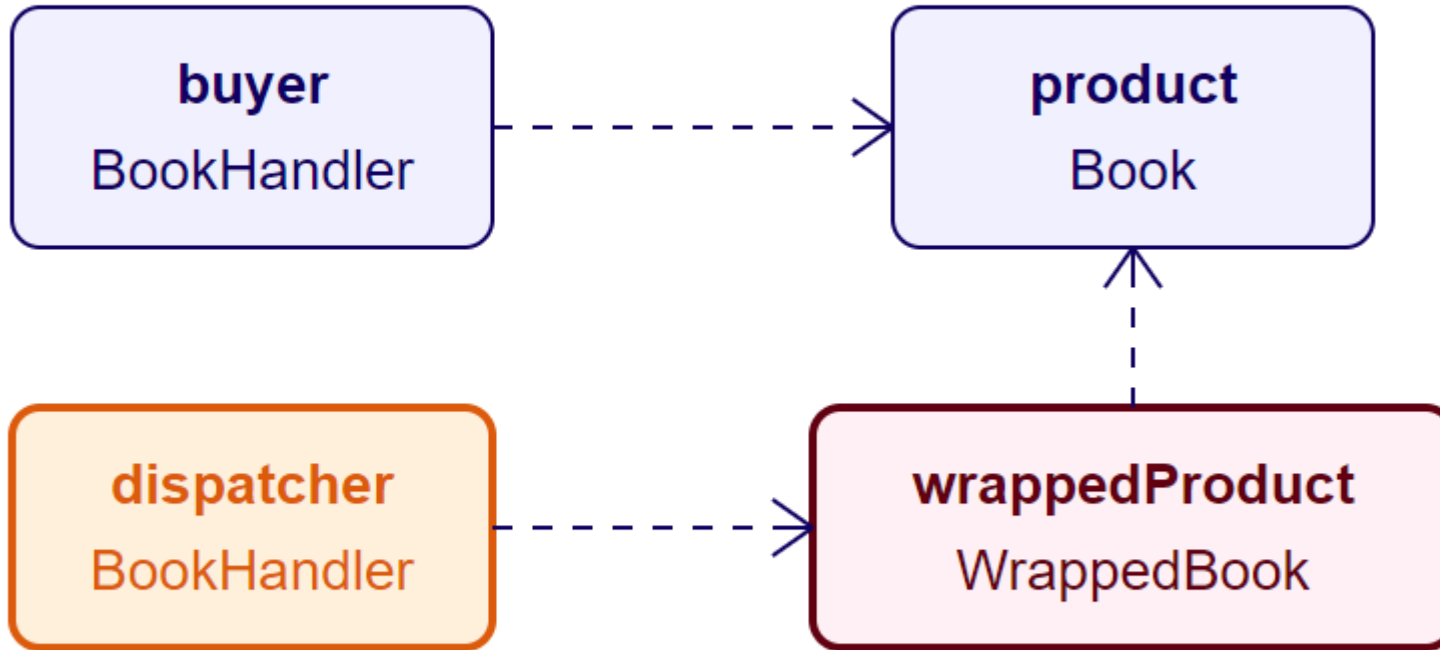
Adding 7 mm to base dimensions 18.8 x 23.9 x 3.3 cm

Issues with Subclassing Decorator

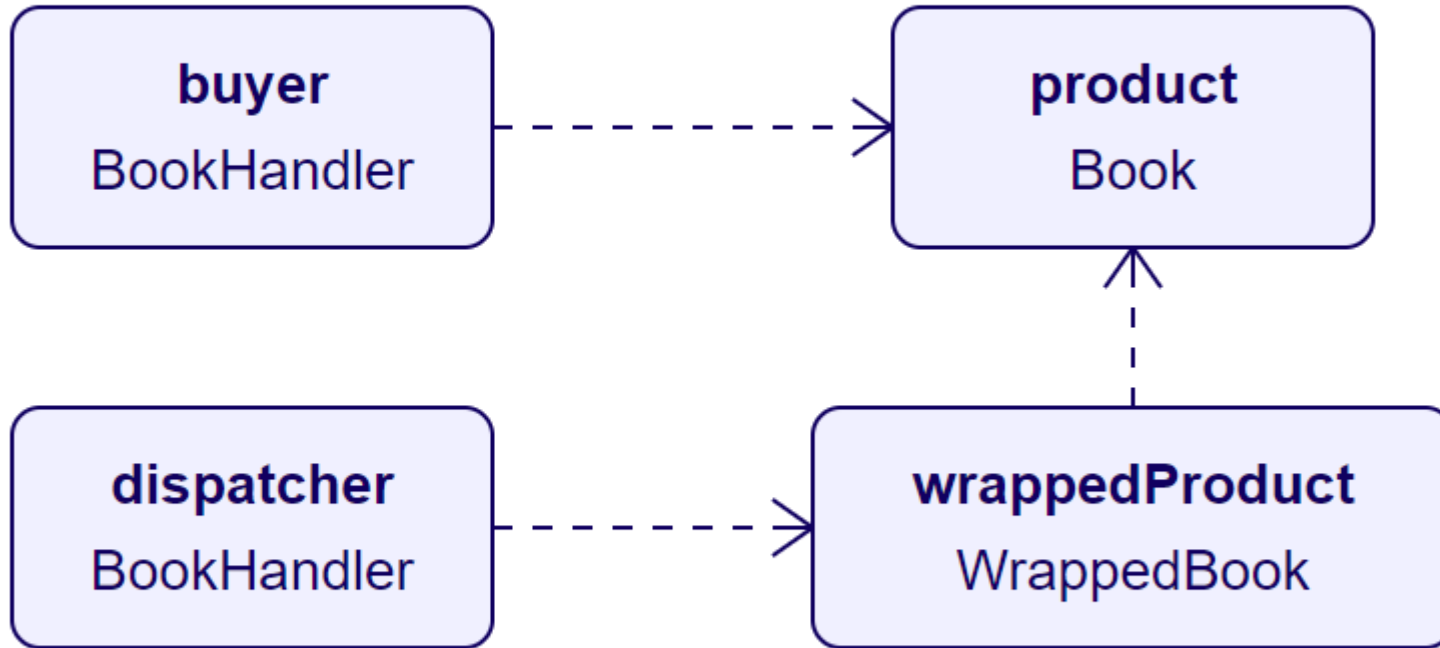


wrappedProduct.GetDimensions() returns 19.5 x 24.6 x 4 cm

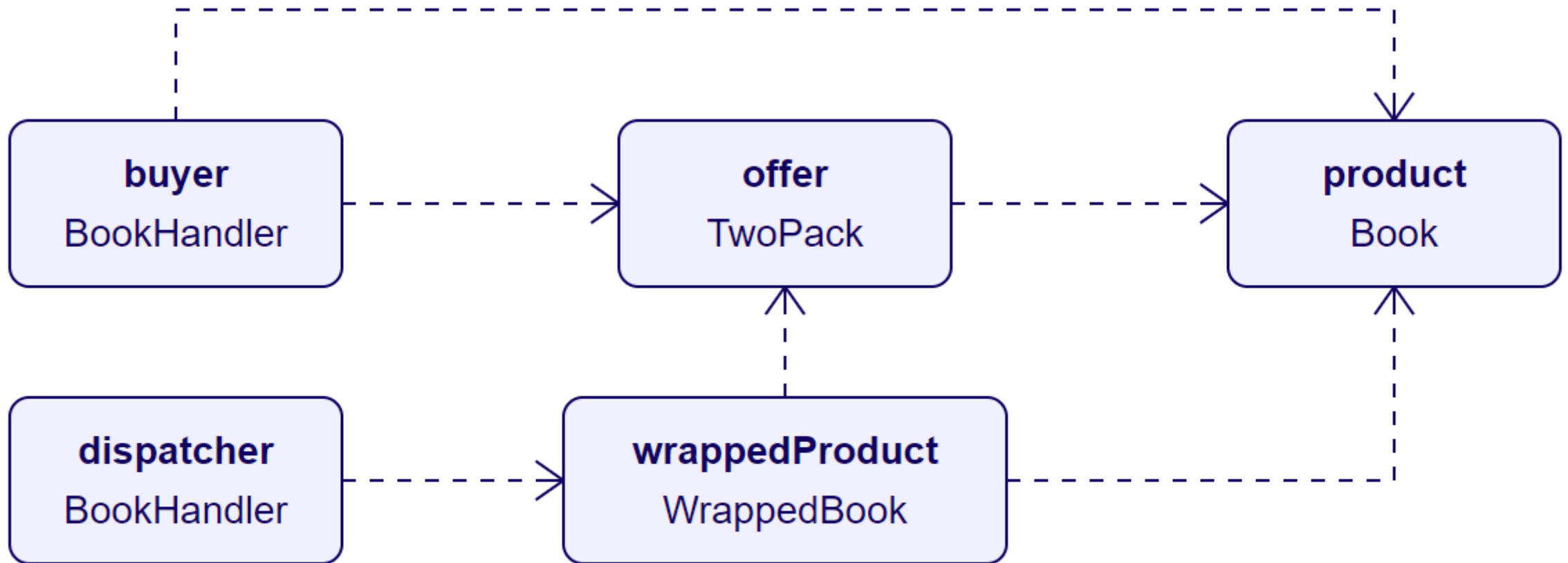
Issues with Subclassing Decorator



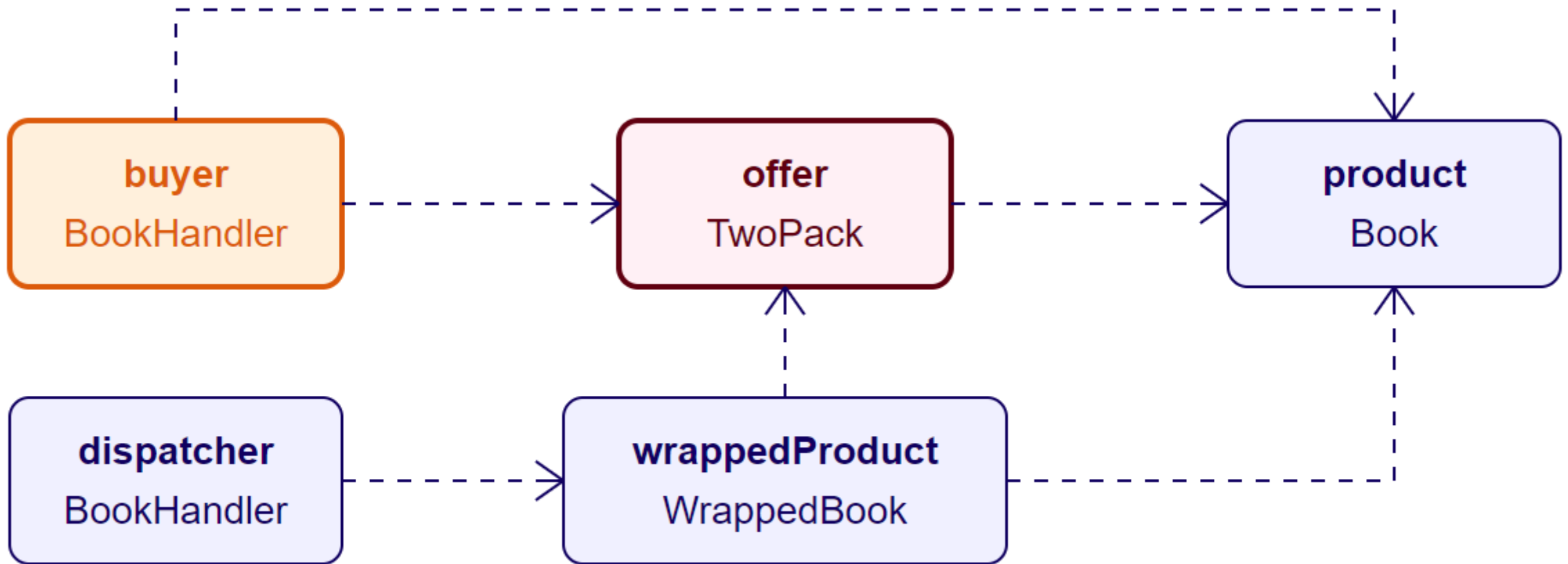
Issues with Subclassing Decorator



Issues with Subclassing Decorator

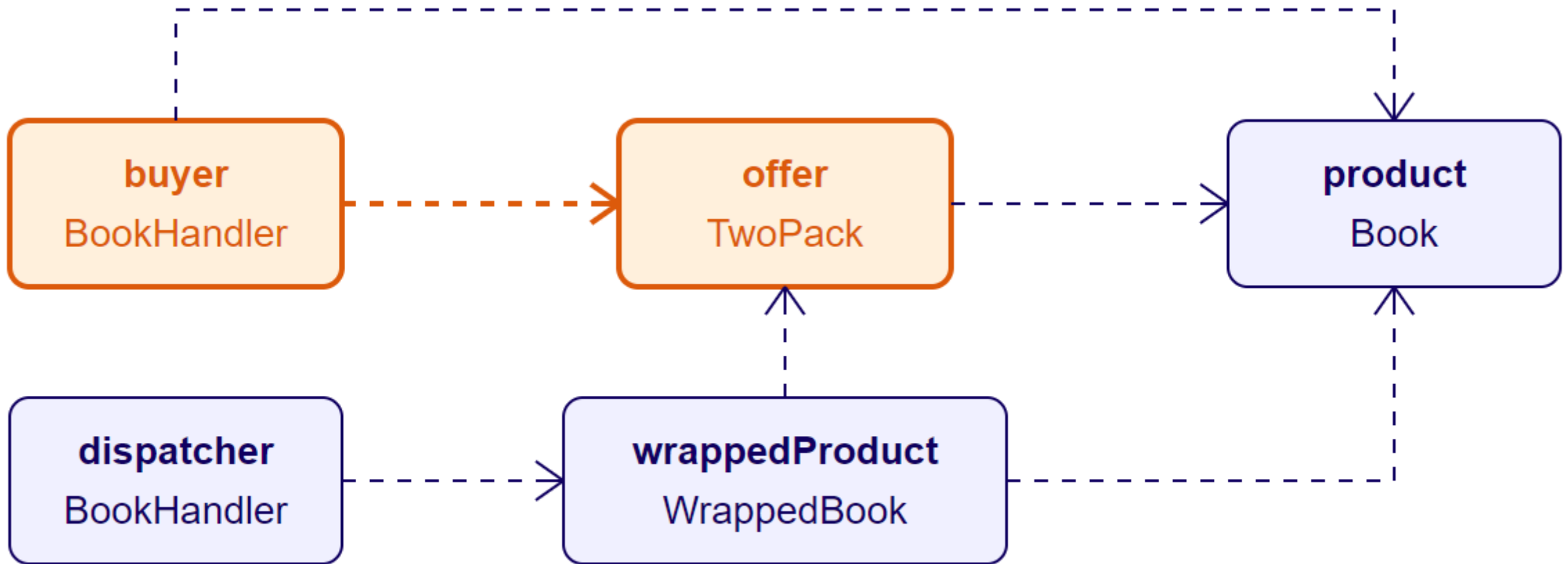


Issues with Subclassing Decorator



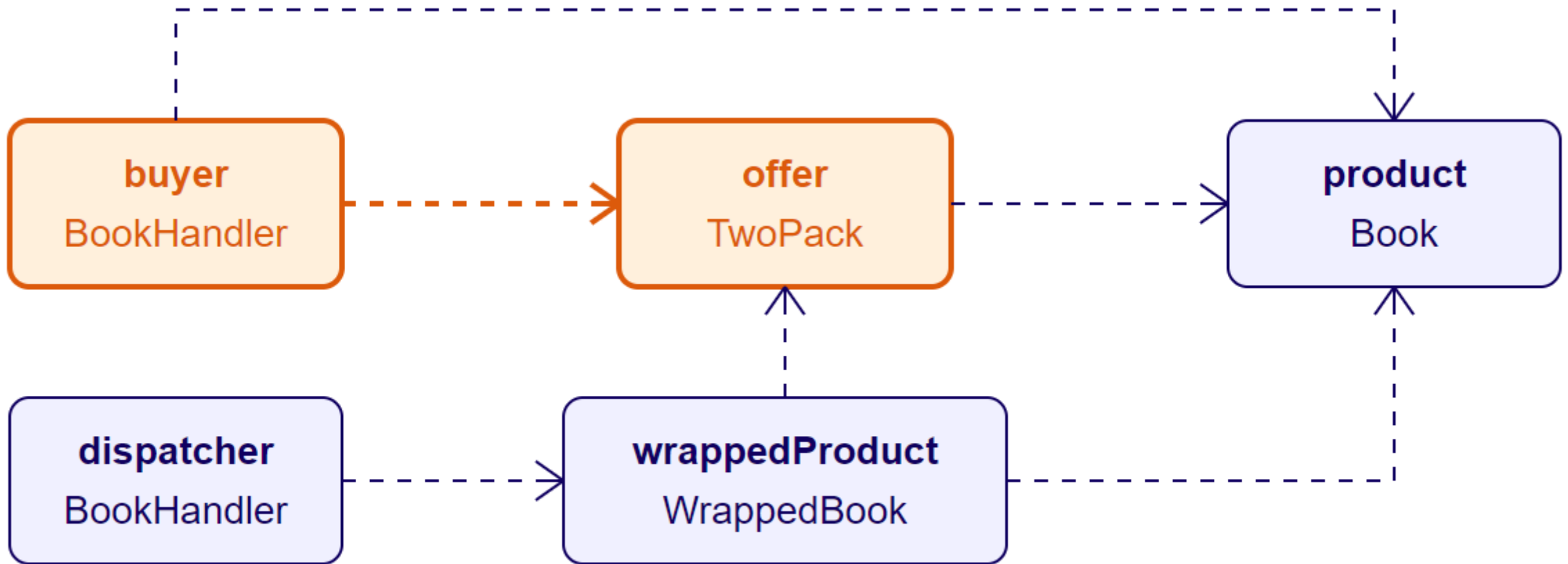
External code calls `buyer.Handle(offer)`

Issues with Subclassing Decorator



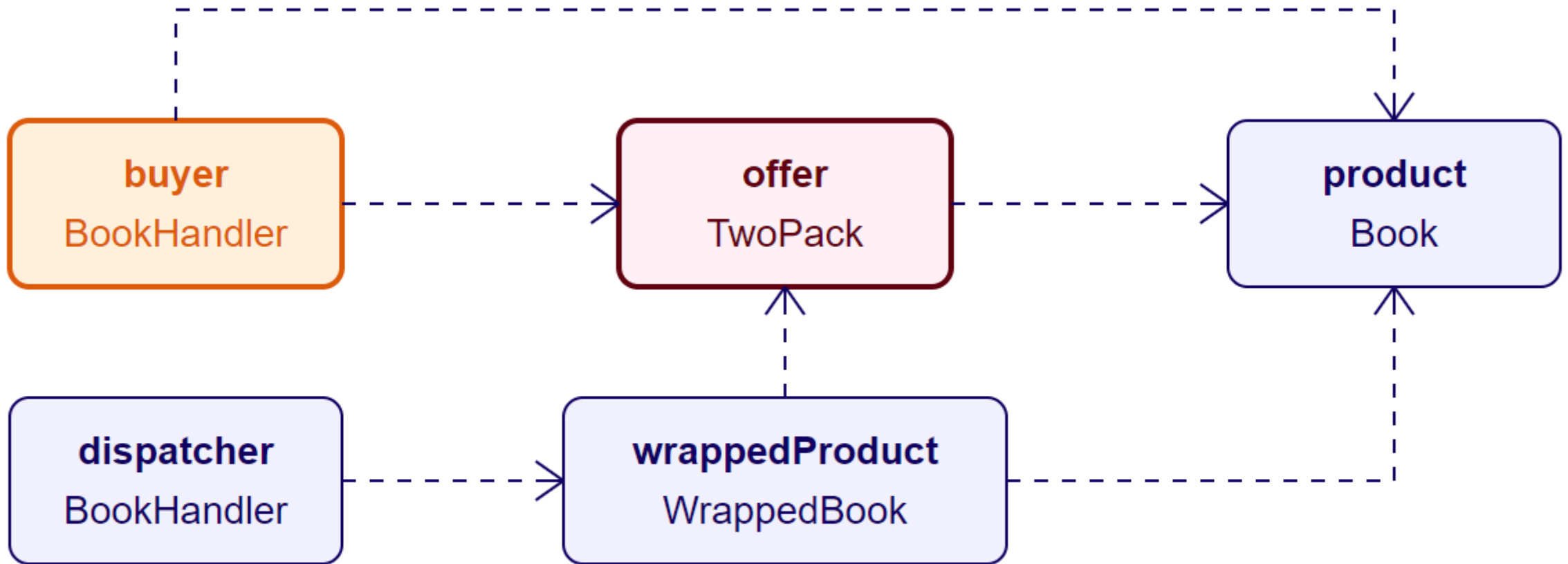
buyer calls offer.GetDimensions(142 x 125 x 5 mm)

Issues with Subclassing Decorator

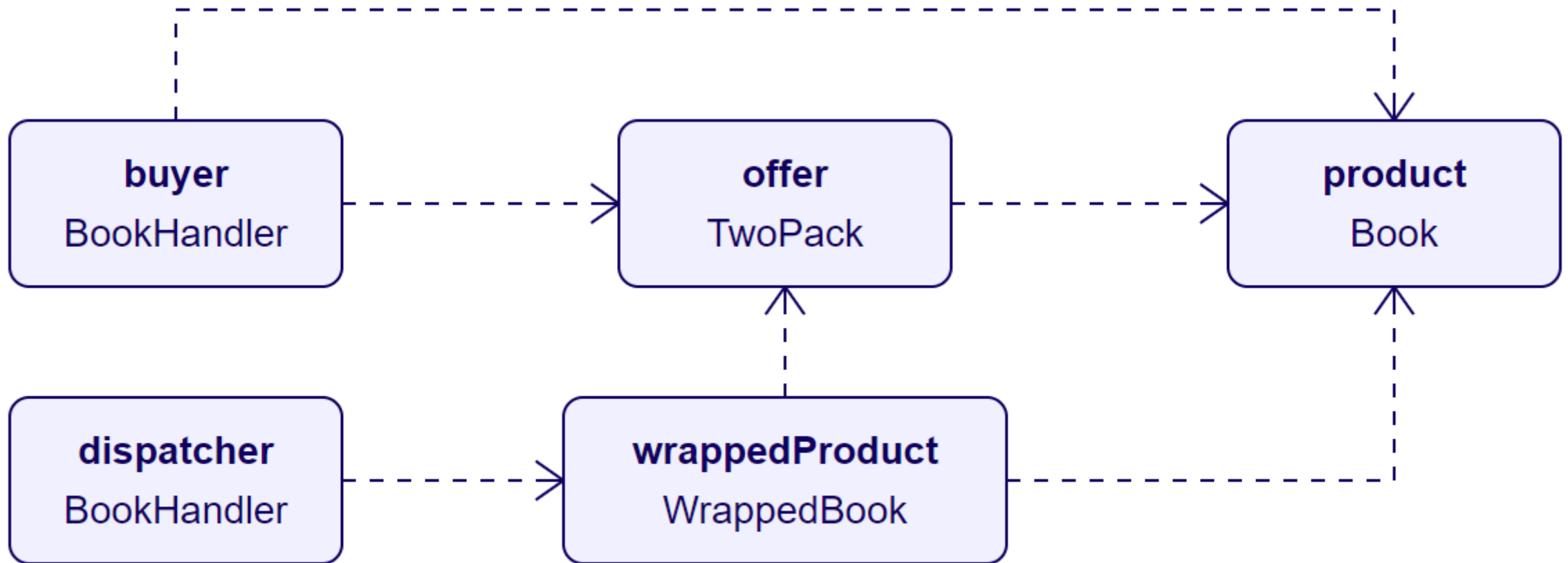


`offer.GetDimensions()` returns 18.8 x 23.9 x 6.1 cm

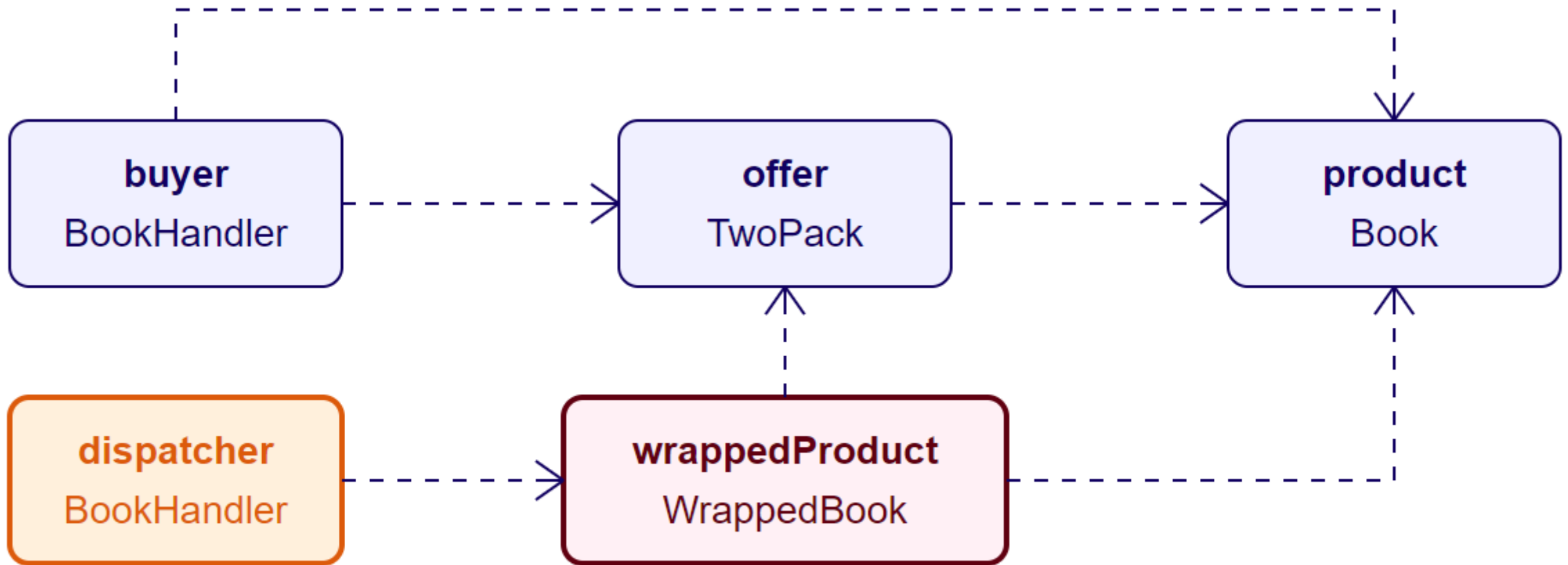
Issues with Subclassing Decorator



Issues with Subclassing Decorator

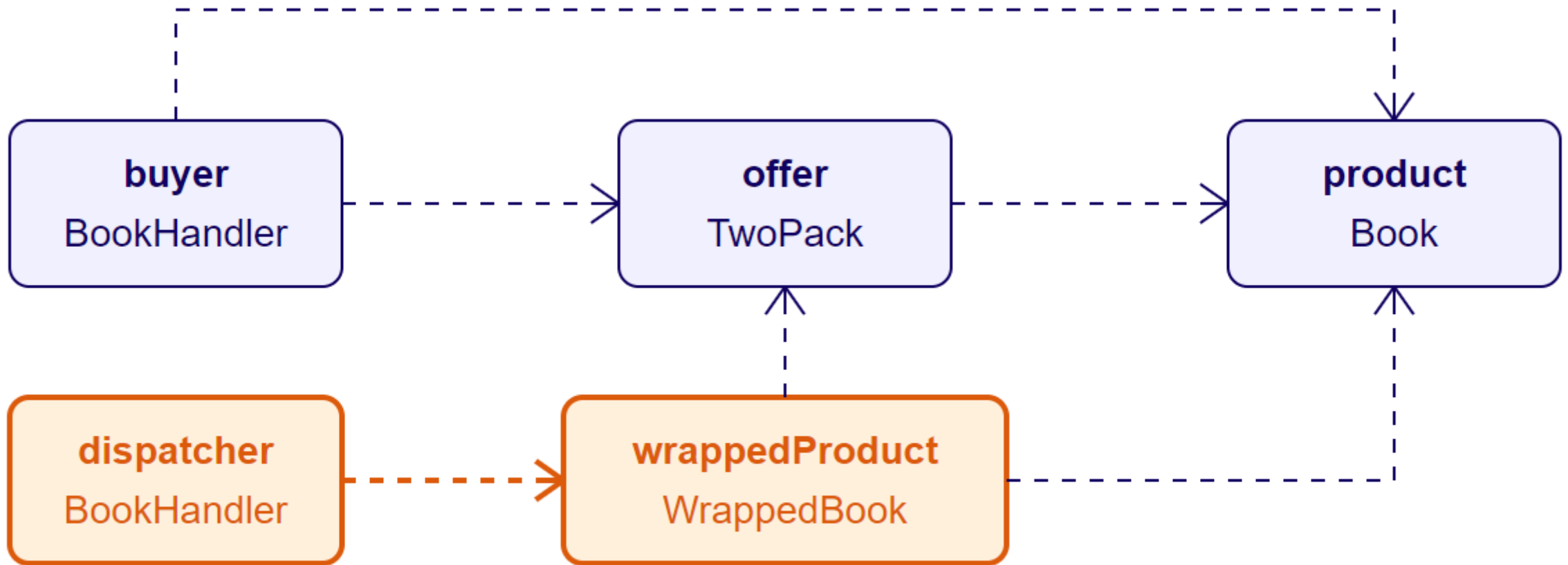


Issues with Subclassing Decorator



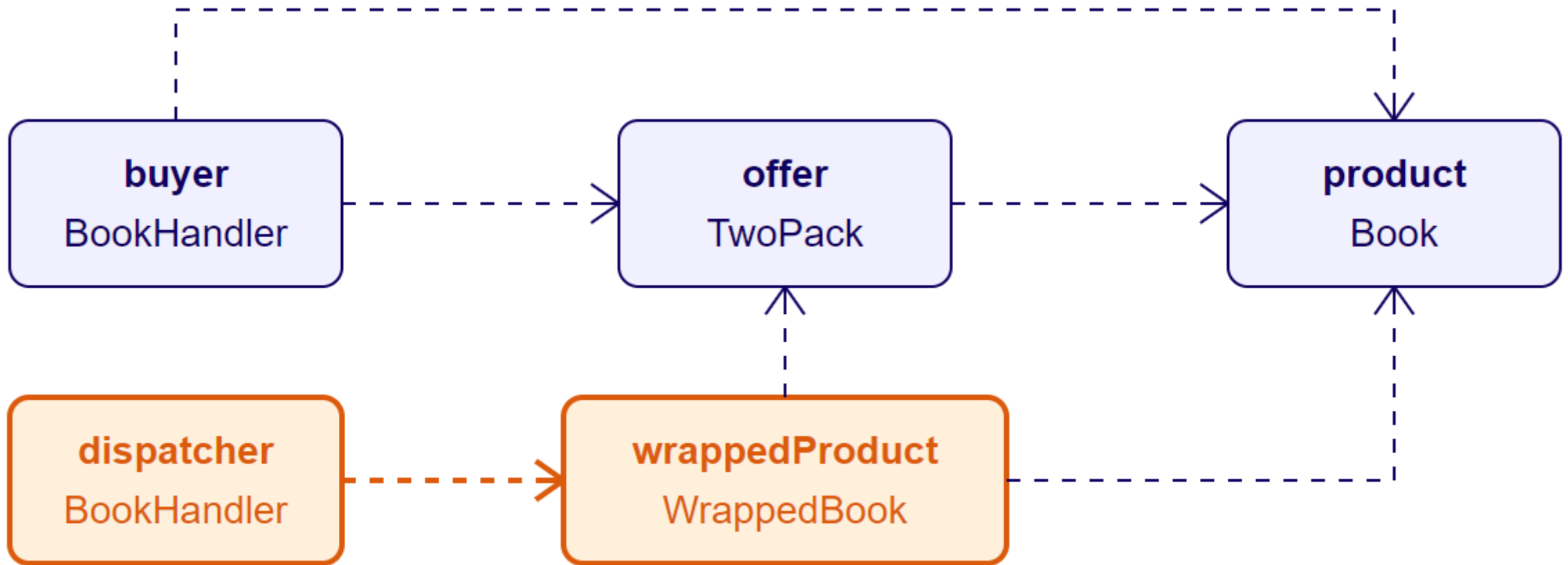
External code calls `dispatcher.Handle(wrappedProduct)`

Issues with Subclassing Decorator



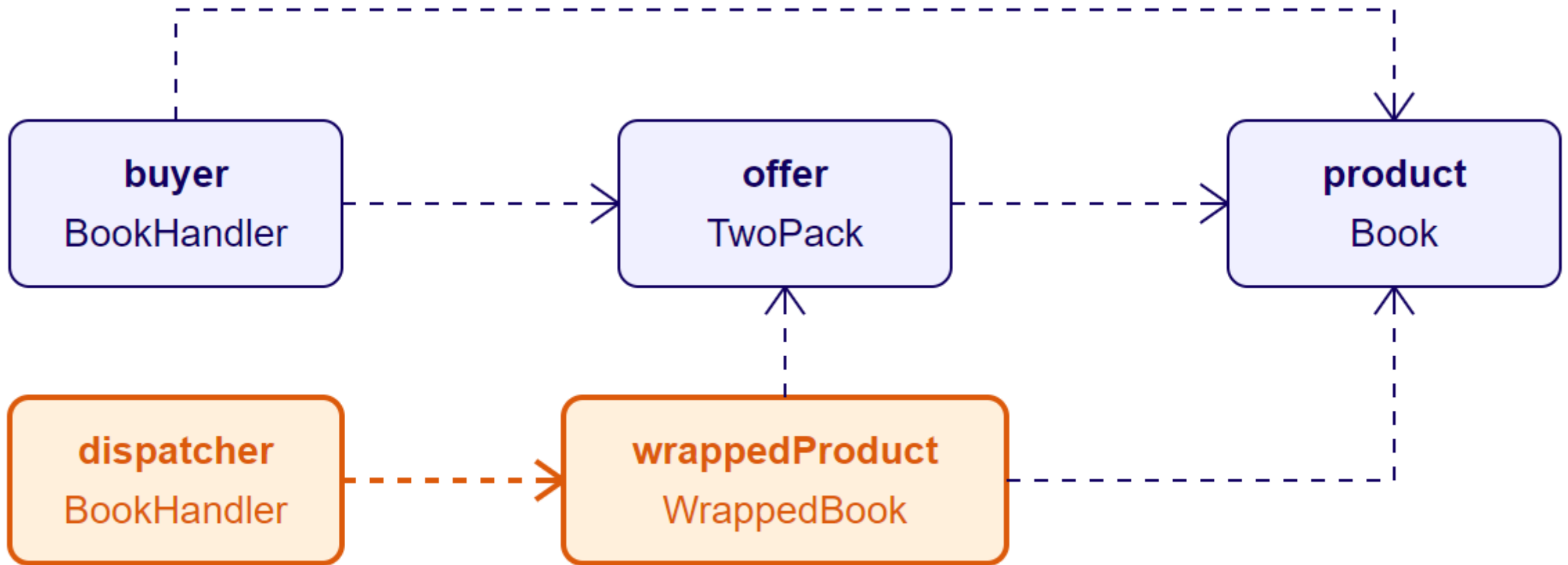
dispatcher calls wrappedProduct.GetDimensions(142 x 125 x 5 mm)

Issues with Subclassing Decorator



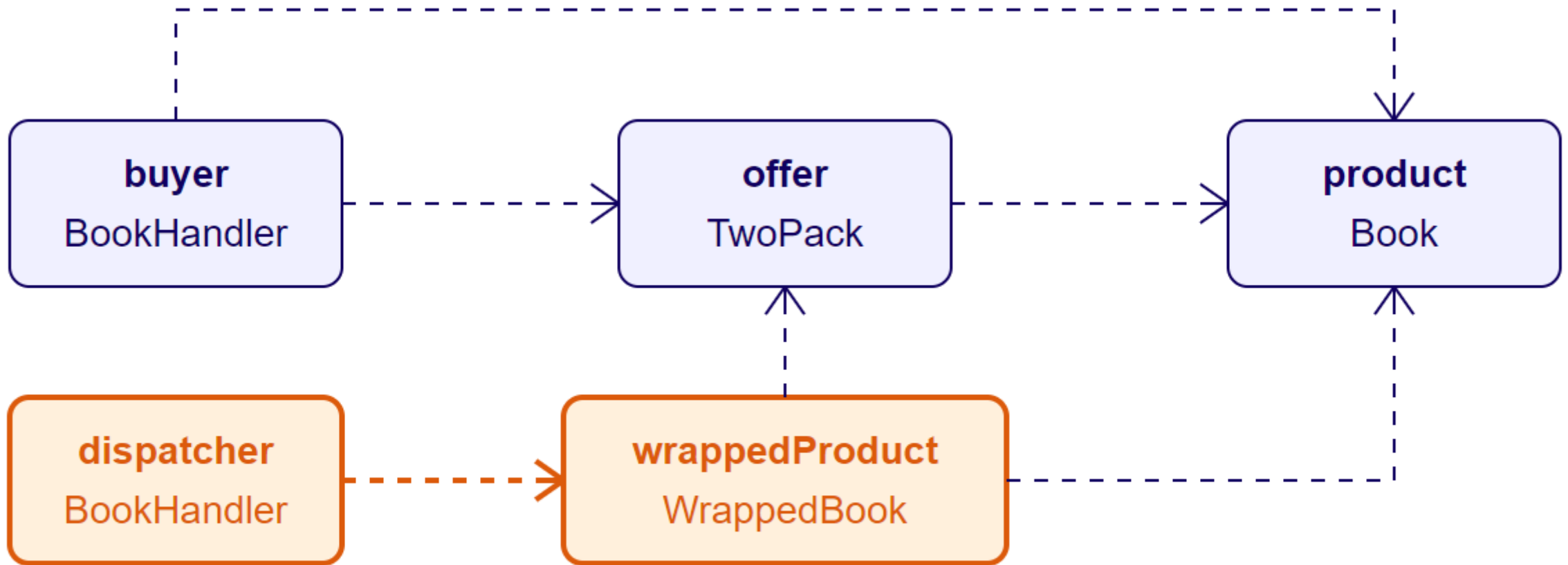
wrappedProduct calls Book.GetDimensions(142 x 125 x 5 mm) implementation

Issues with Subclassing Decorator



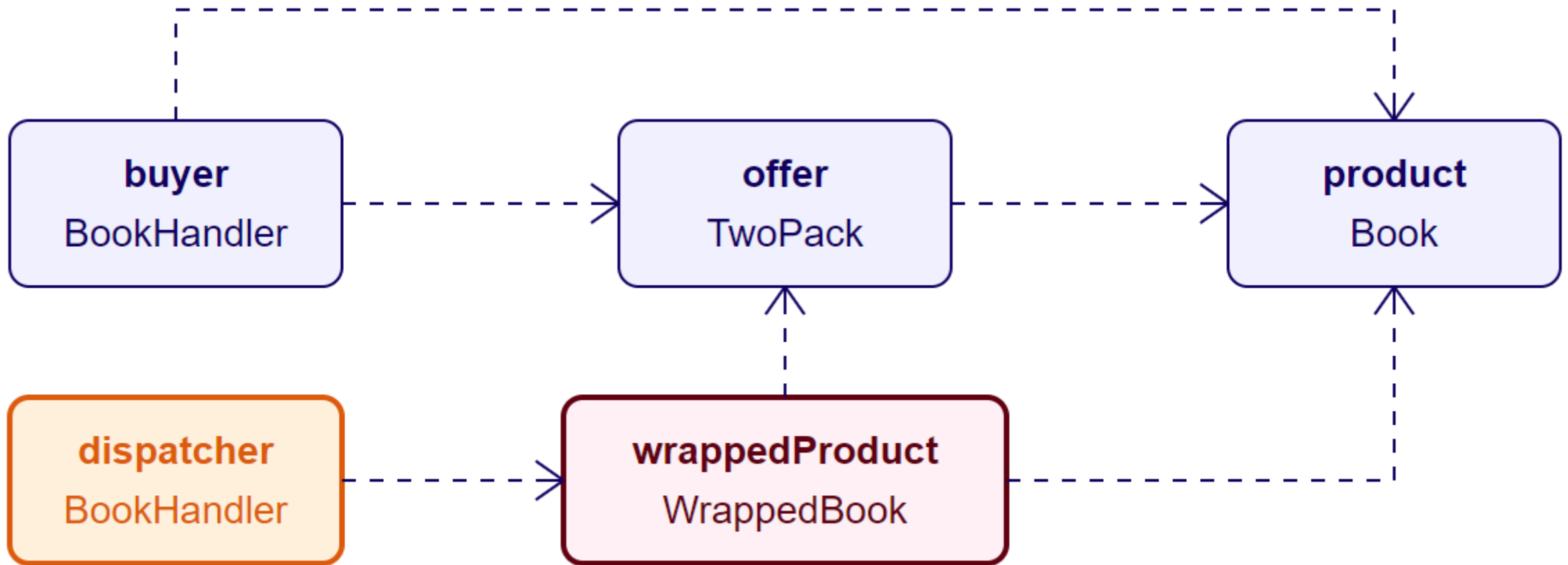
Adding 7 mm to base dimensions 18.8 x 23.9 x 3.3 cm

Issues with Subclassing Decorator

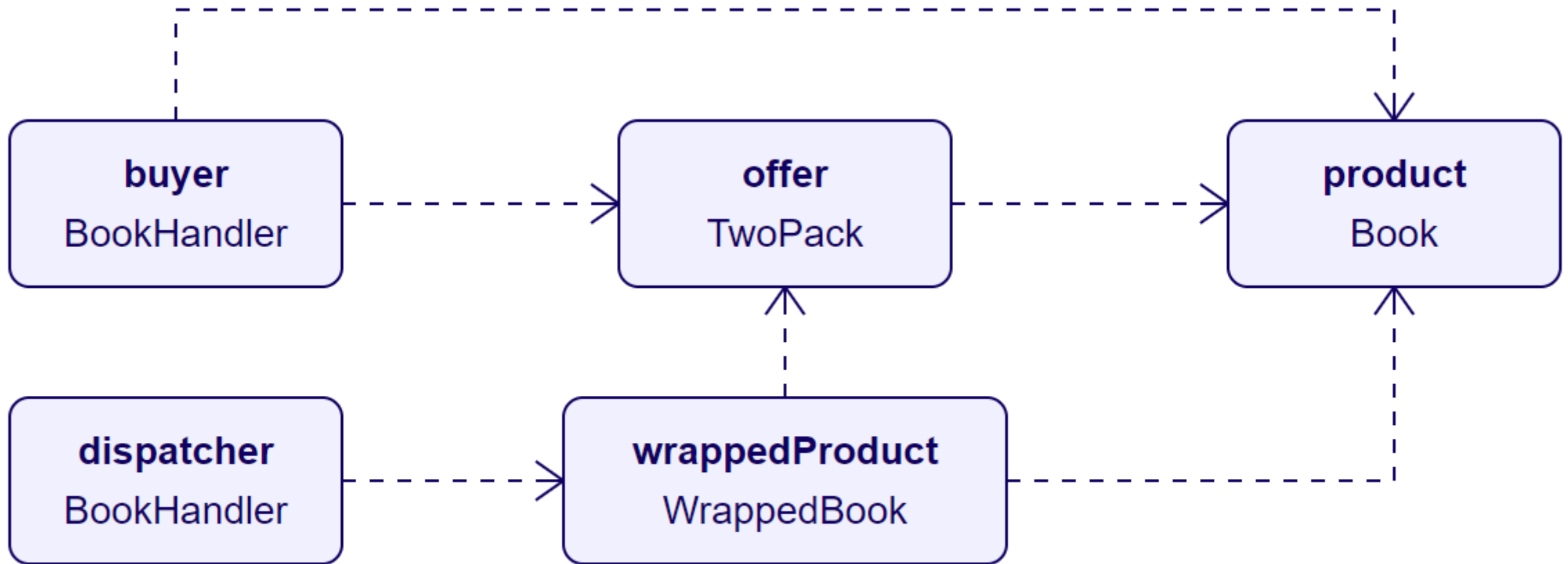


wrappedProduct.GetDimensions() returns 19.5 x 24.6 x 4 cm

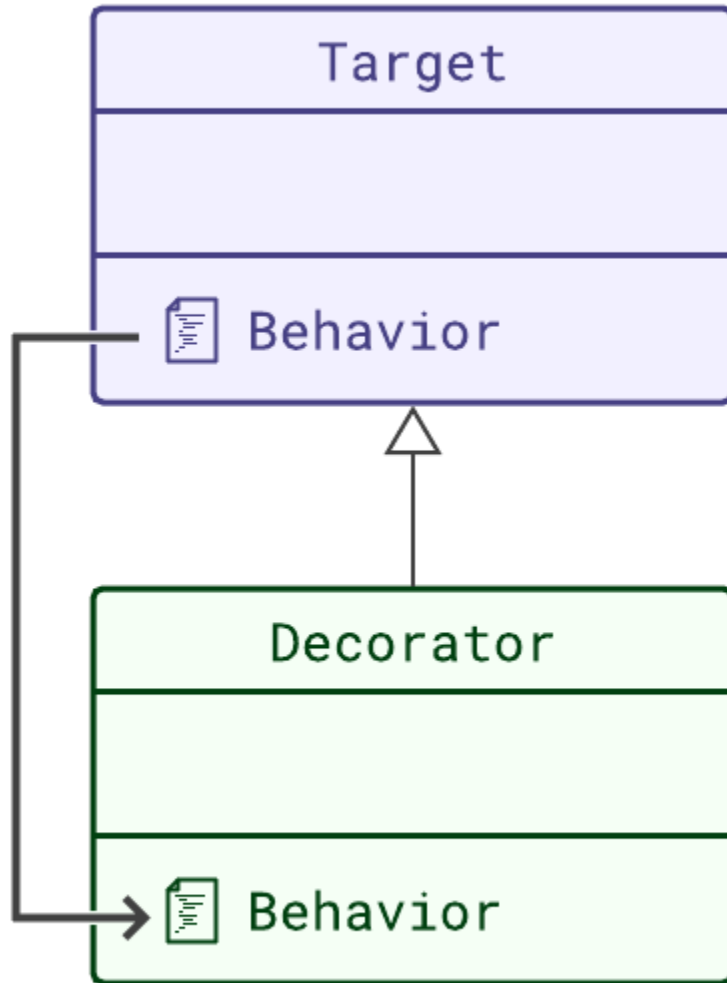
Issues with Subclassing Decorator



Issues with Subclassing Decorator

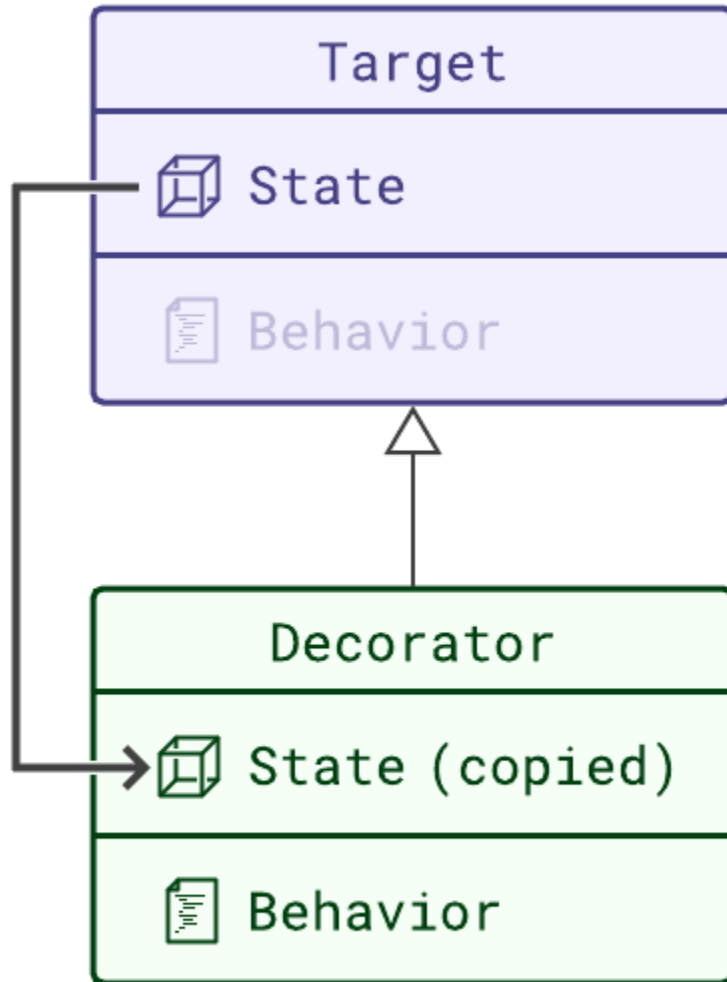


About Derived Decorator



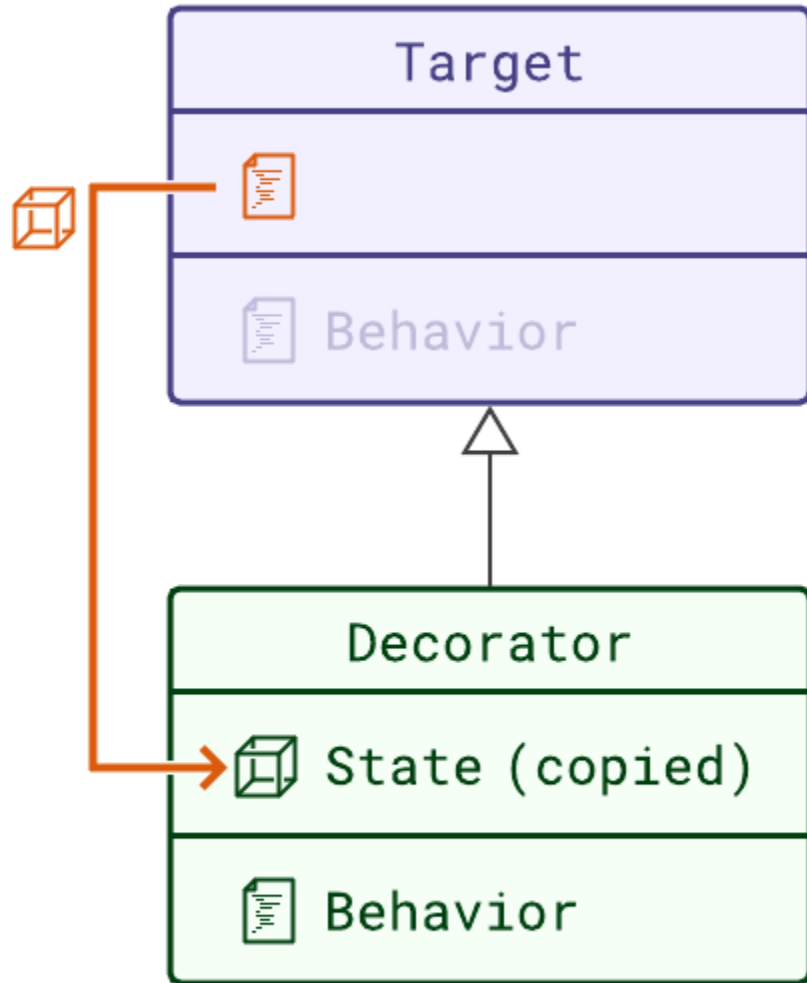
Don't inherit
the decorator
unless you only
inherit the state

About Derived Decorator



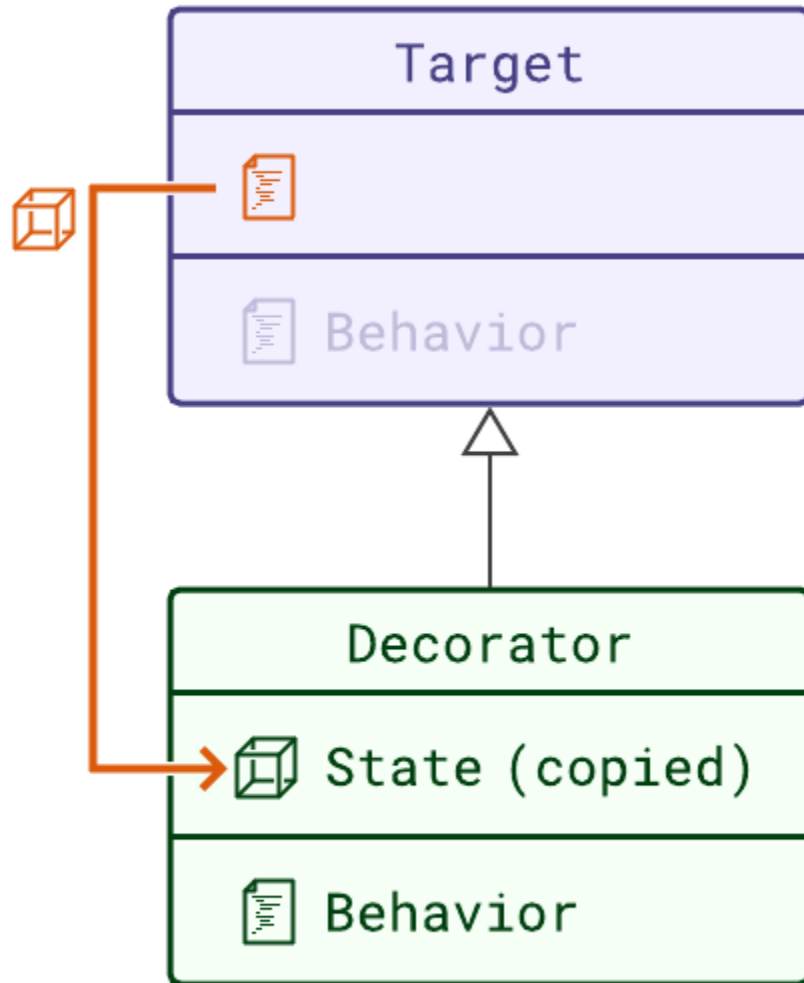
Don't inherit
the decorator
unless you only
inherit the state

About Derived Decorator



Don't inherit
the decorator
unless you only
inherit the state

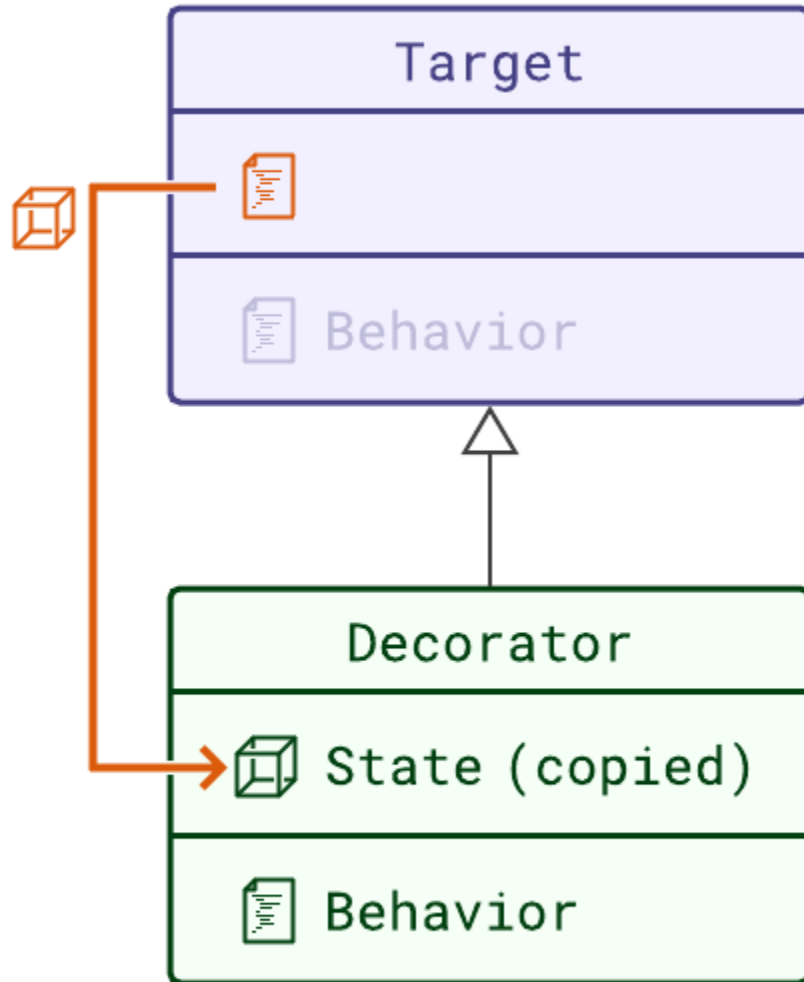
About Derived Decorator



Don't inherit
the decorator
unless you only
inherit the state

**Property getters
are behavior!**

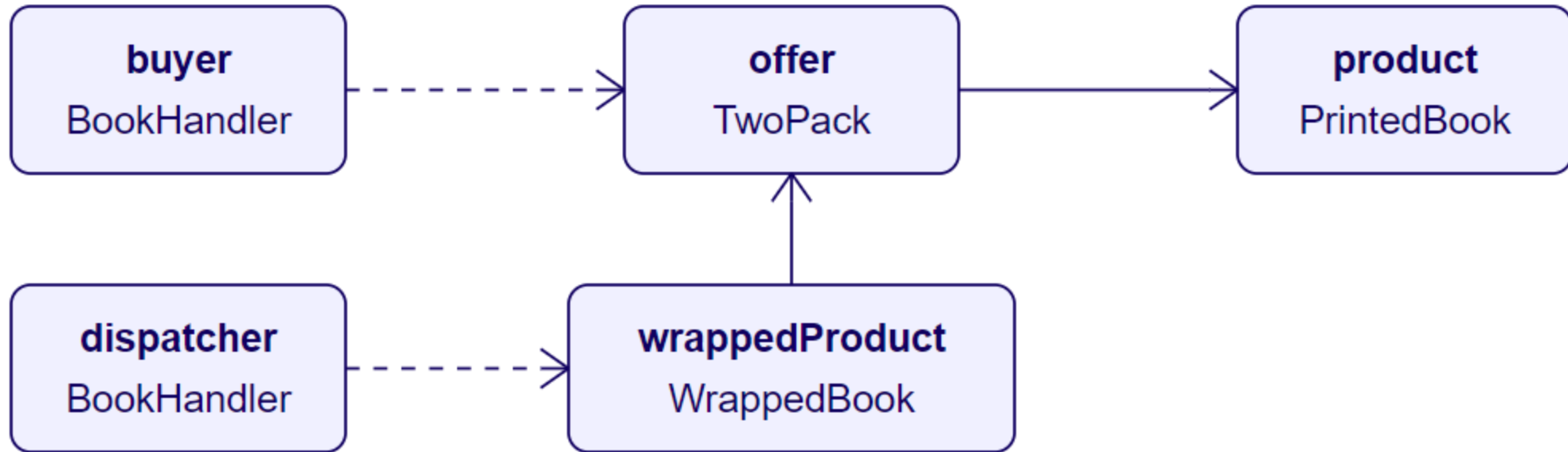
About Derived Decorator



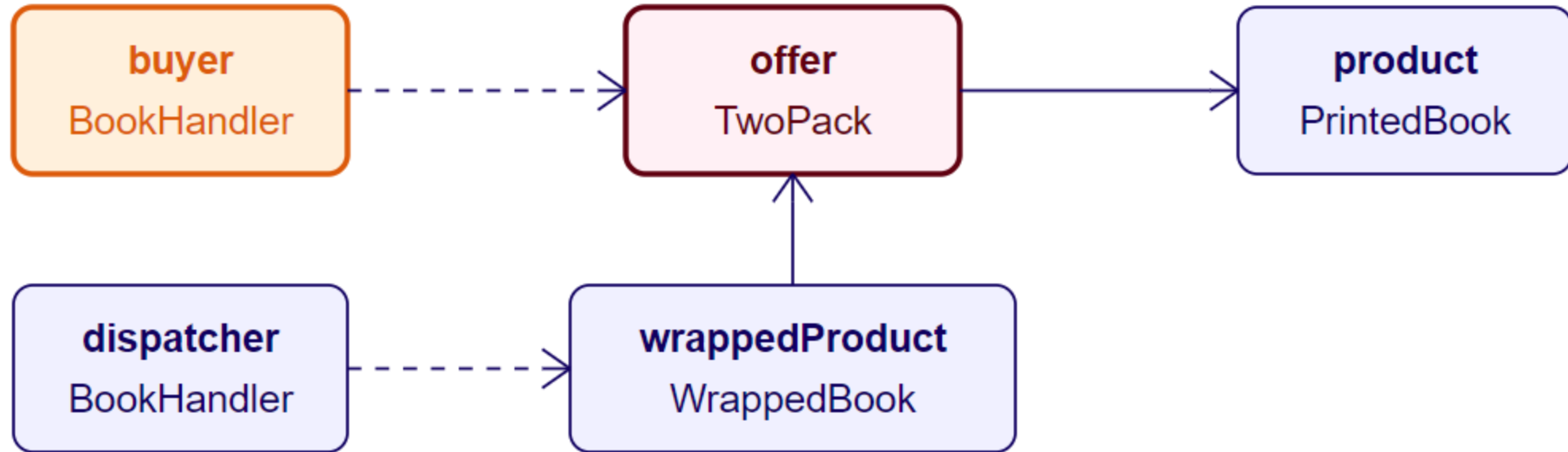
Don't inherit
the decorator
unless you only
inherit the state

Property getters
are behavior!

Using the Delegating Decorator

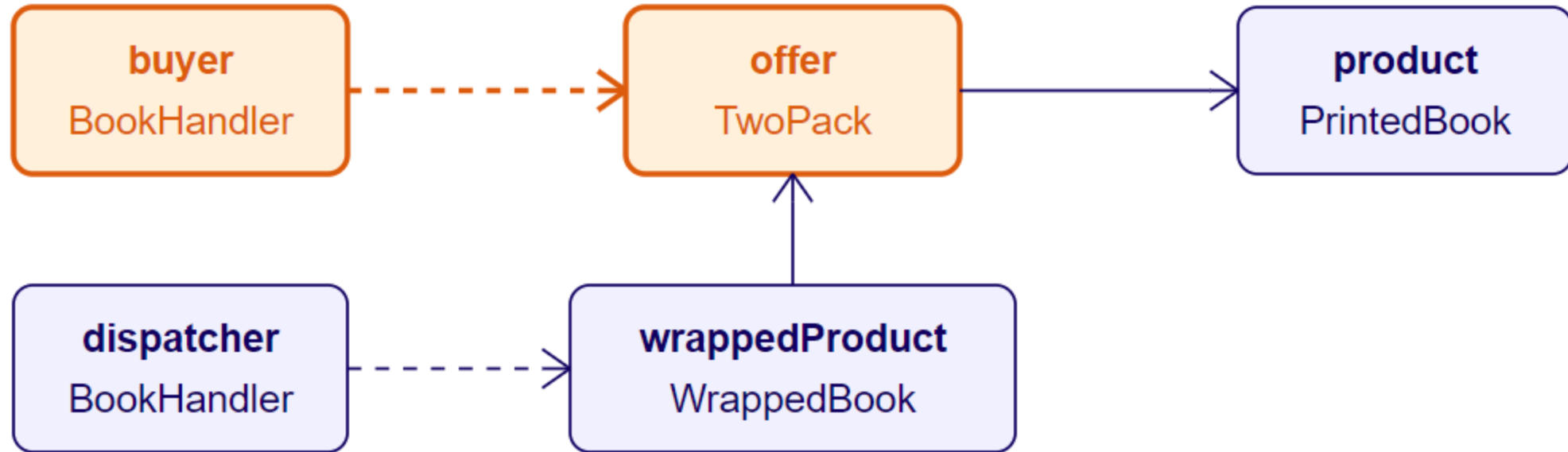


Using the Delegating Decorator



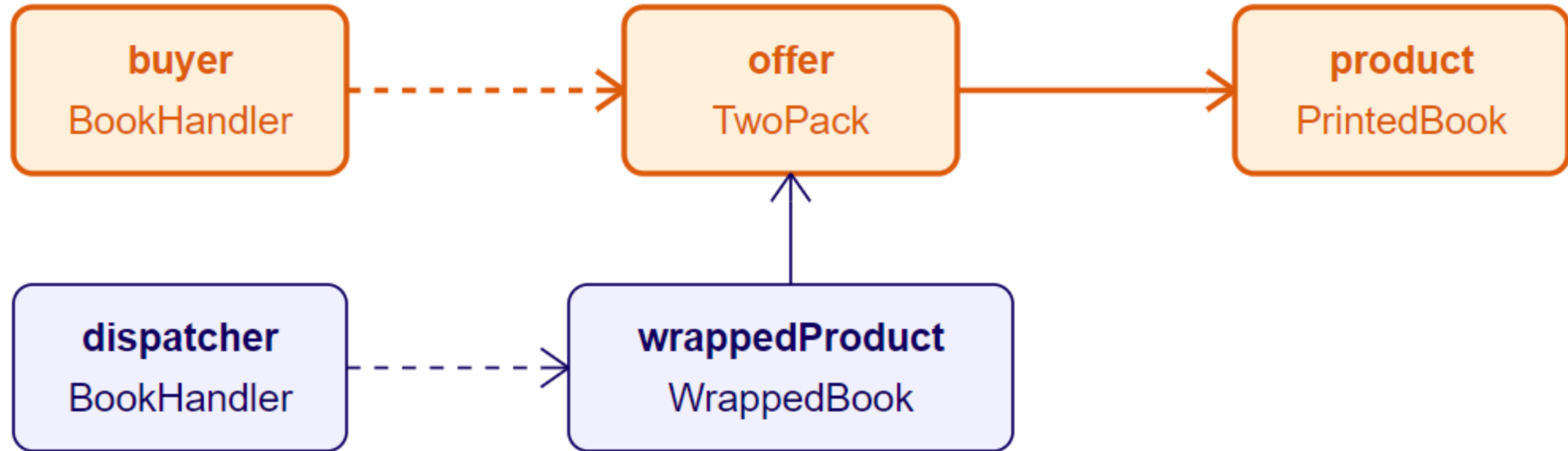
External code calls `buyer.Handle(offer)`

Using the Delegating Decorator



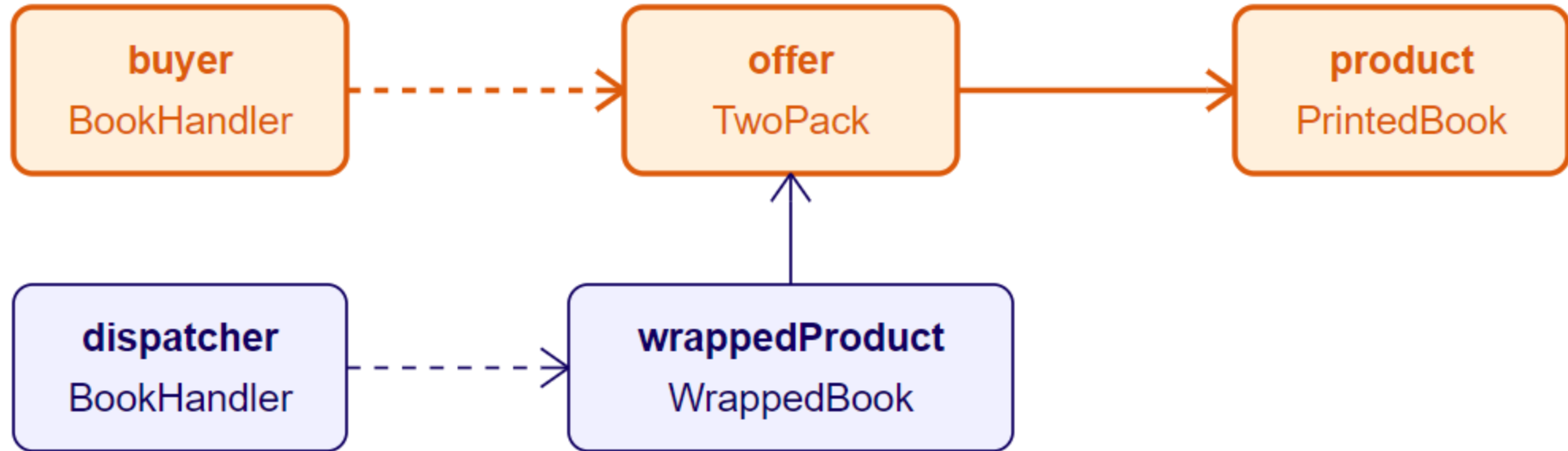
buyer calls offer.GetDimensions(142 x 125 x 5 mm)

Using the Delegating Decorator



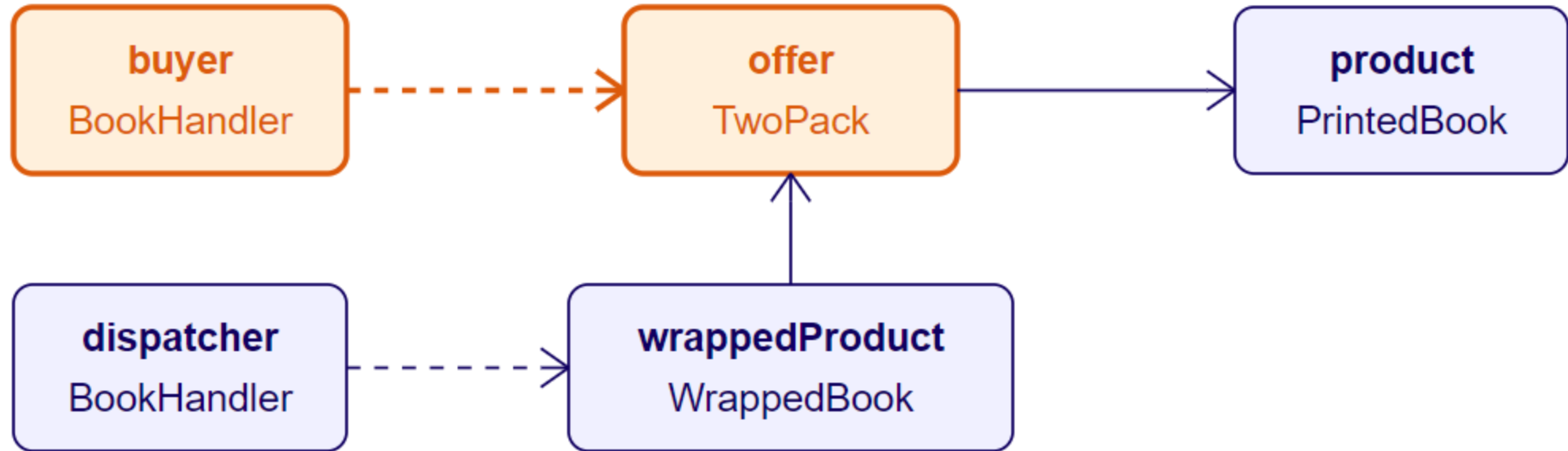
offer calls `product.GetDimensions(0 x 0 x 0 mm)`

Using the Delegating Decorator

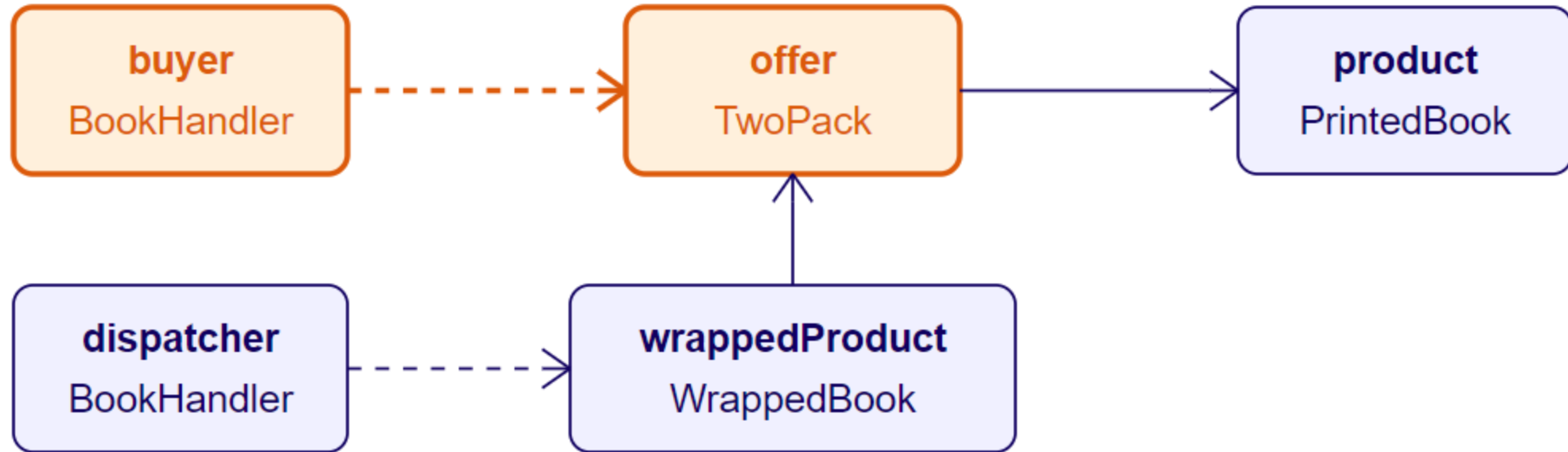


`product.GetDimensions()` returns 18.8 x 23.9 x 2.8 cm

Using the Delegating Decorator

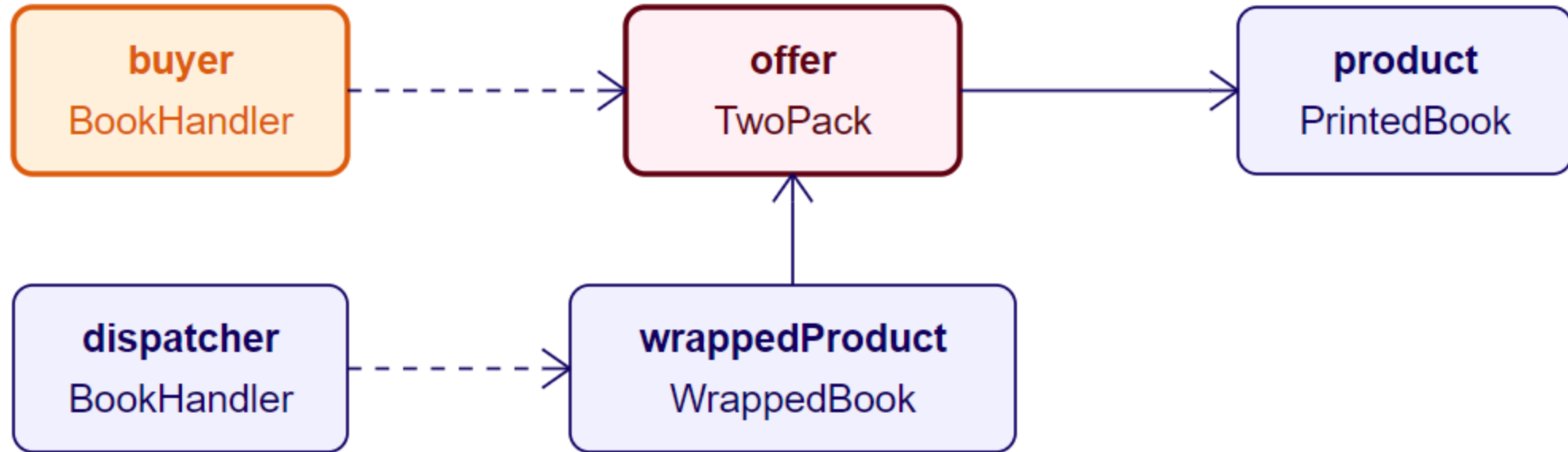


Using the Delegating Decorator

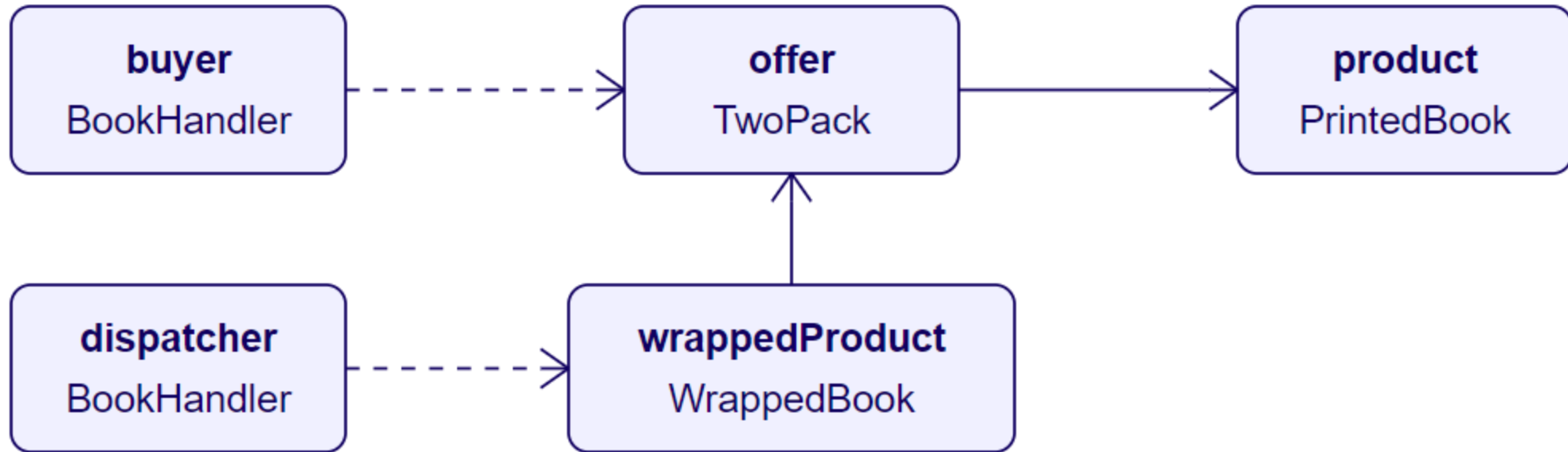


`offer.GetDimensions()` returns 18.8 x 23.9 x 6.1 cm

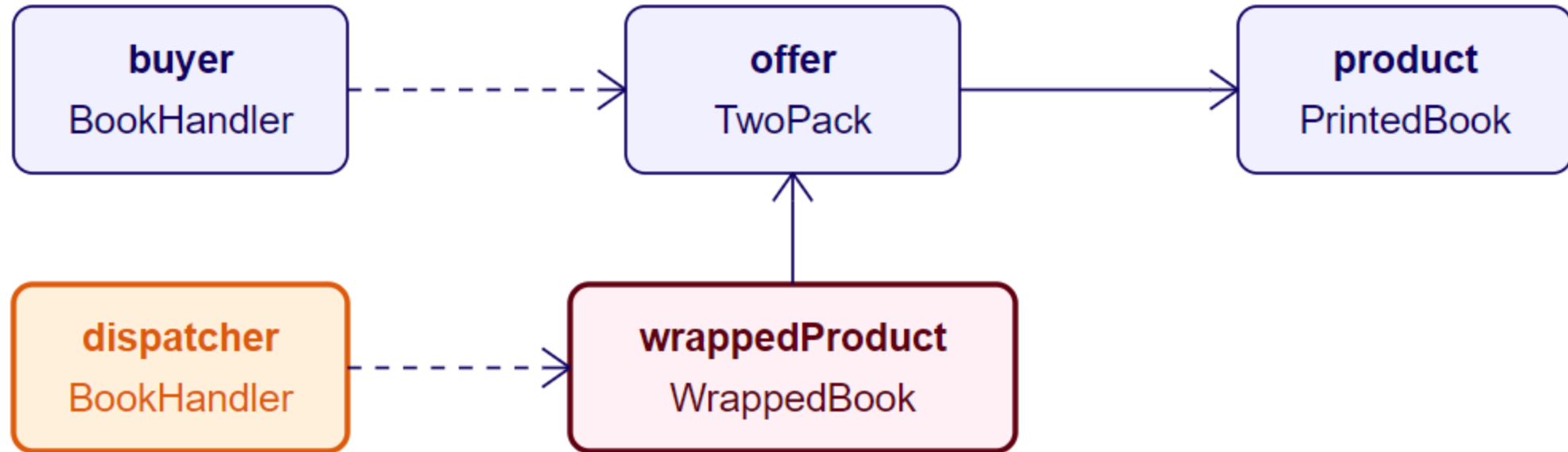
Using the Delegating Decorator



Using the Delegating Decorator

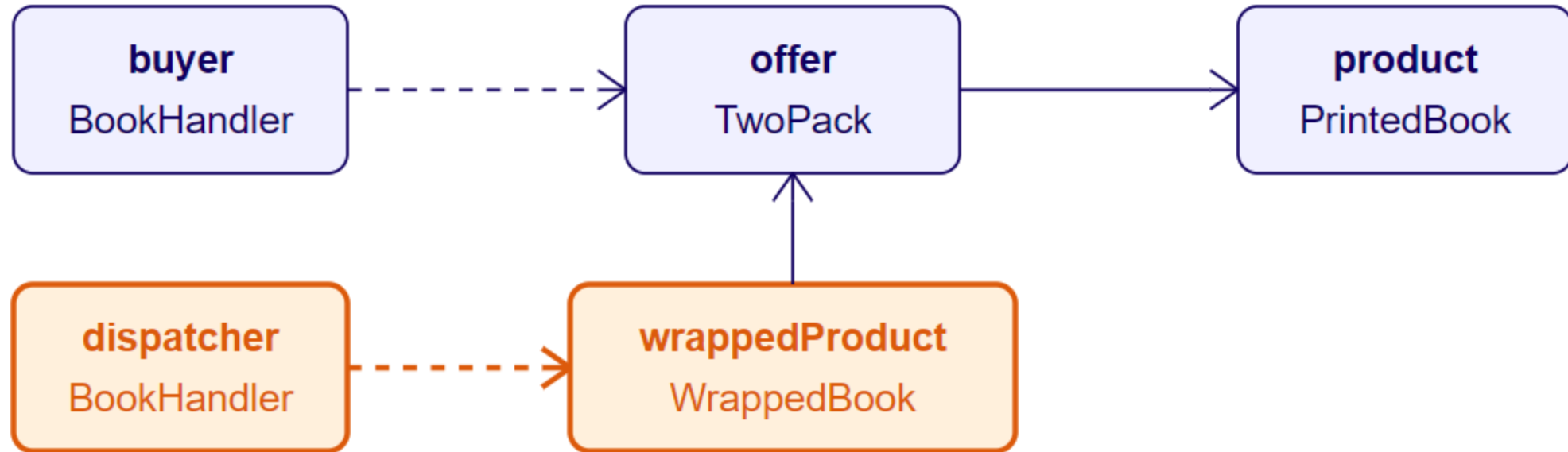


Using the Delegating Decorator



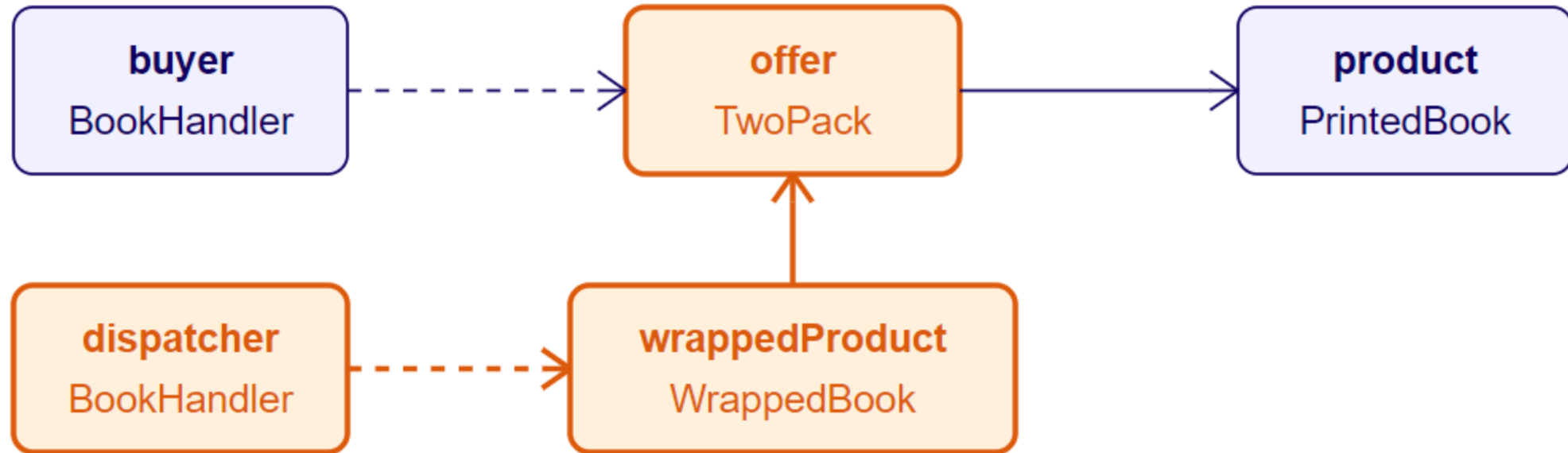
External code calls `dispatcher.Handle(wrappedProduct)`

Using the Delegating Decorator



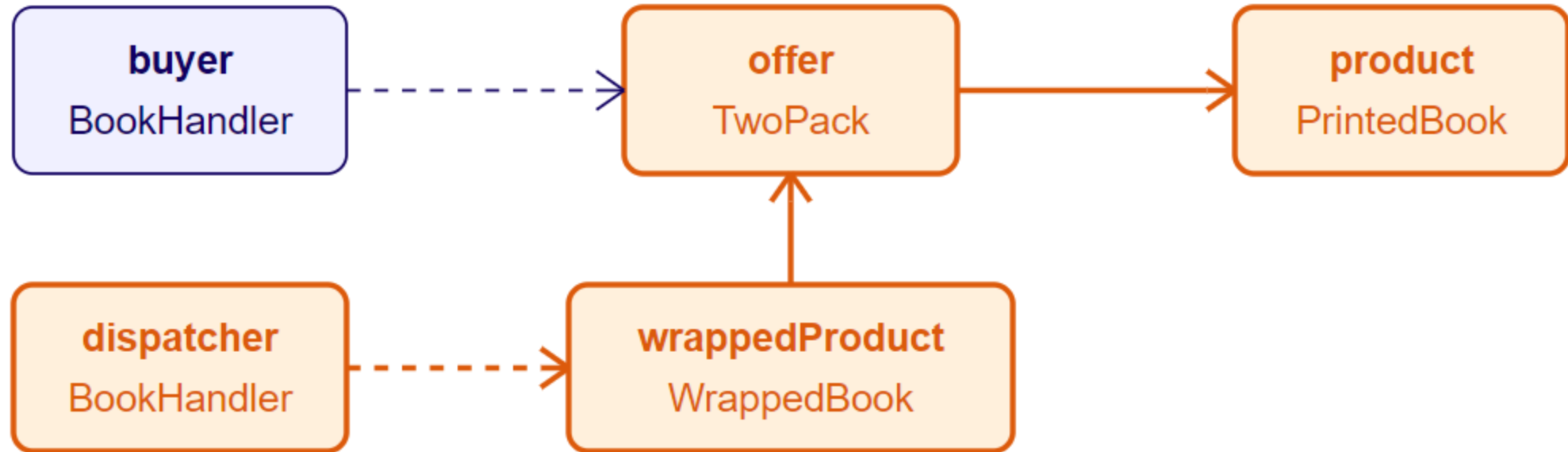
dispatcher calls wrappedProduct.GetDimensions(142 x 125 x 5 mm)

Using the Delegating Decorator



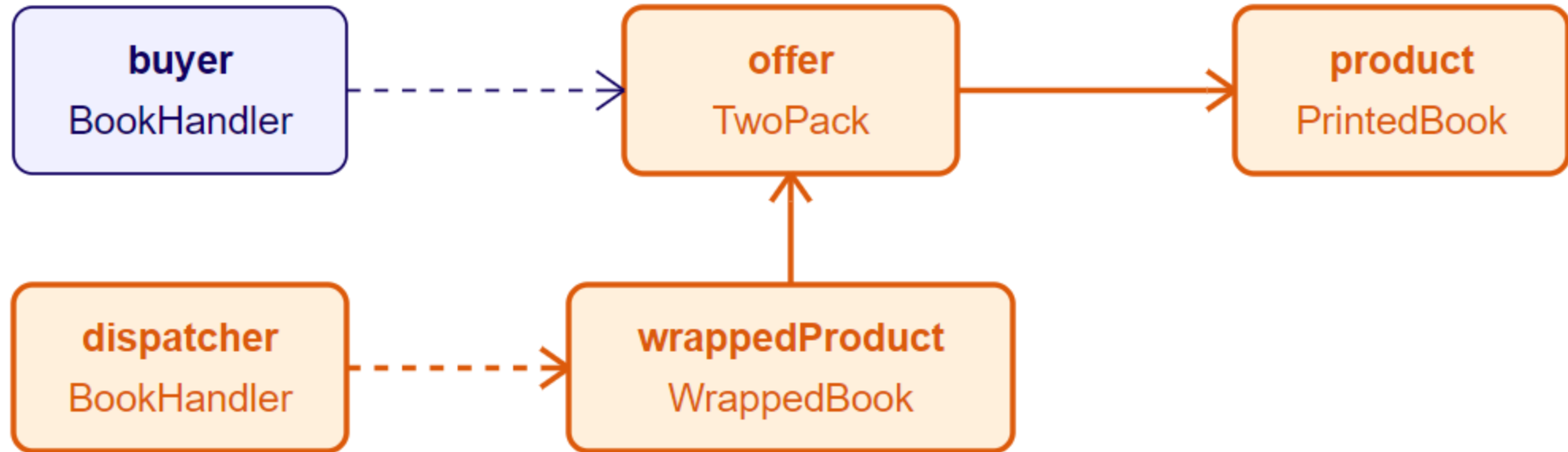
wrappedProduct calls offer.GetDimensions(142 x 125 x 5 mm)

Using the Delegating Decorator



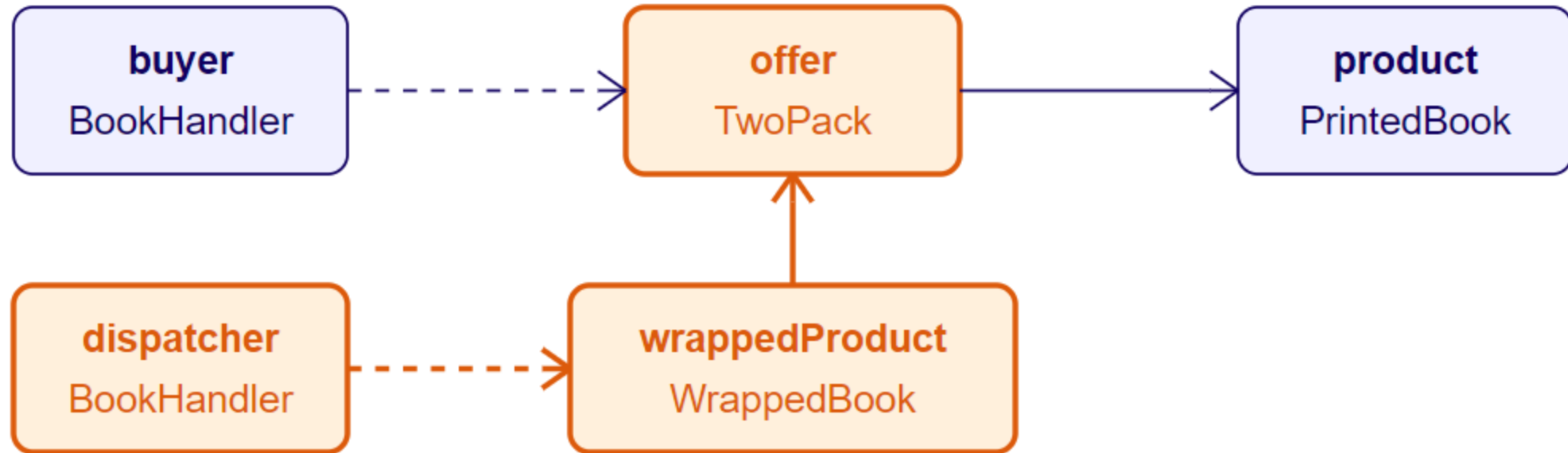
offer calls product.GetDimensions(0 x 0 x 0 mm)

Using the Delegating Decorator

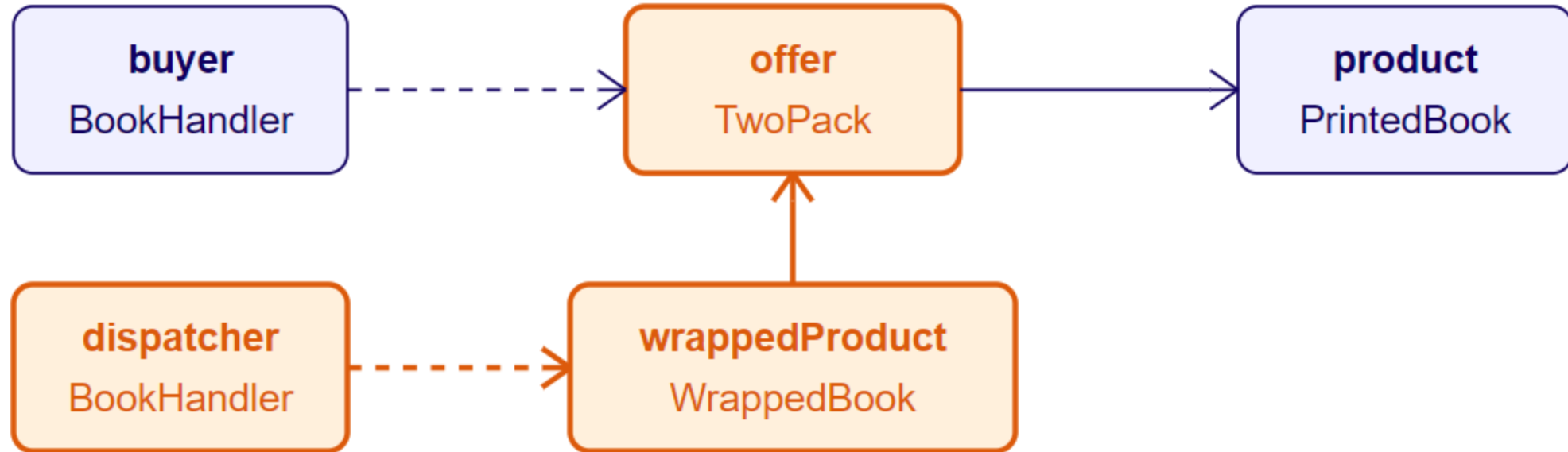


`product.GetDimensions()` returns 18.8 x 23.9 x 2.8 cm

Using the Delegating Decorator

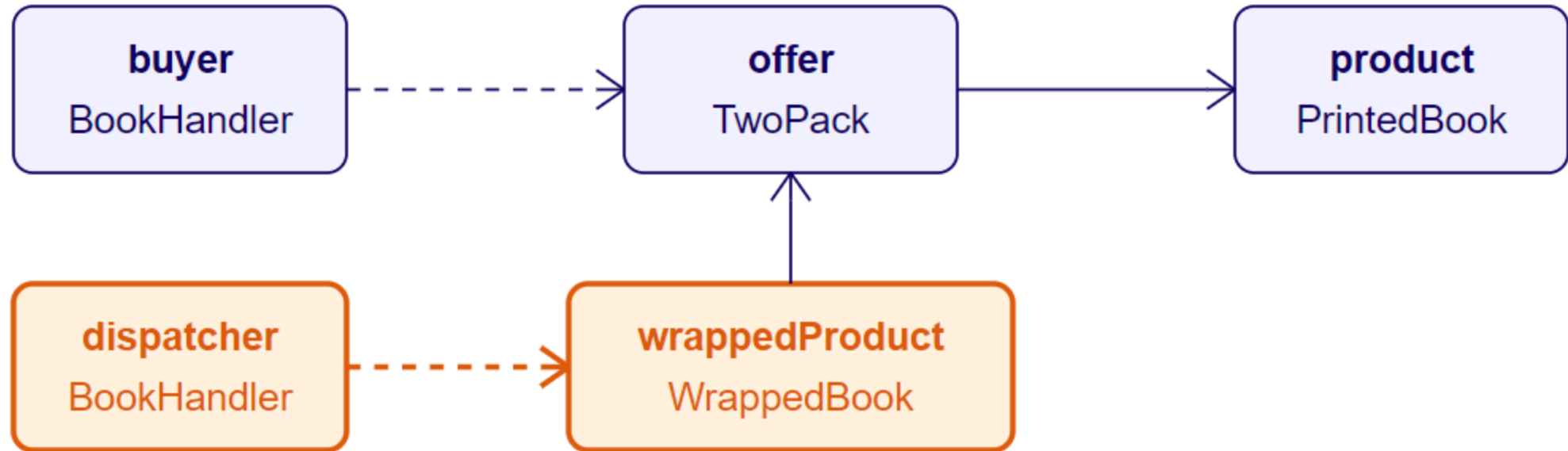


Using the Delegating Decorator

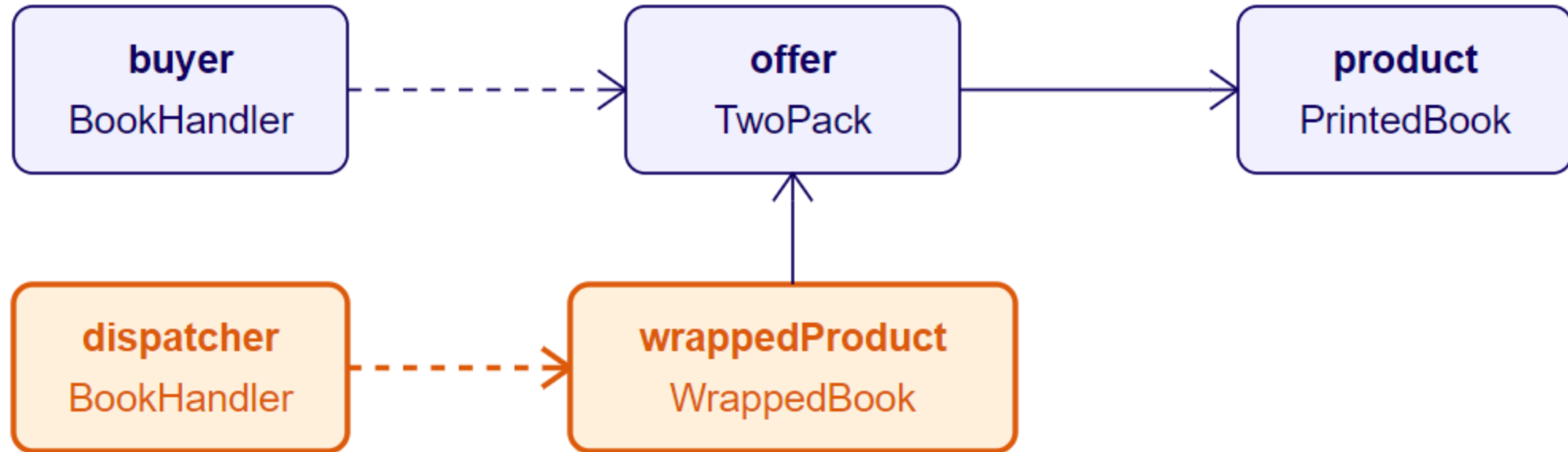


`offer.GetDimensions()` returns 18.8 x 23.9 x 6.1 cm

Using the Delegating Decorator

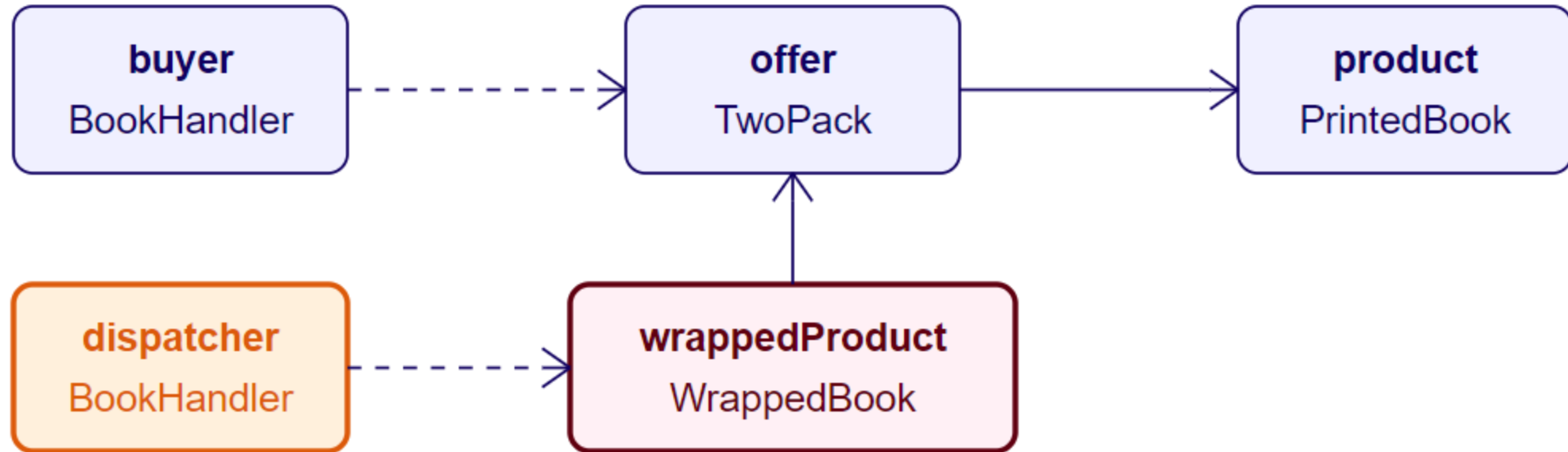


Using the Delegating Decorator

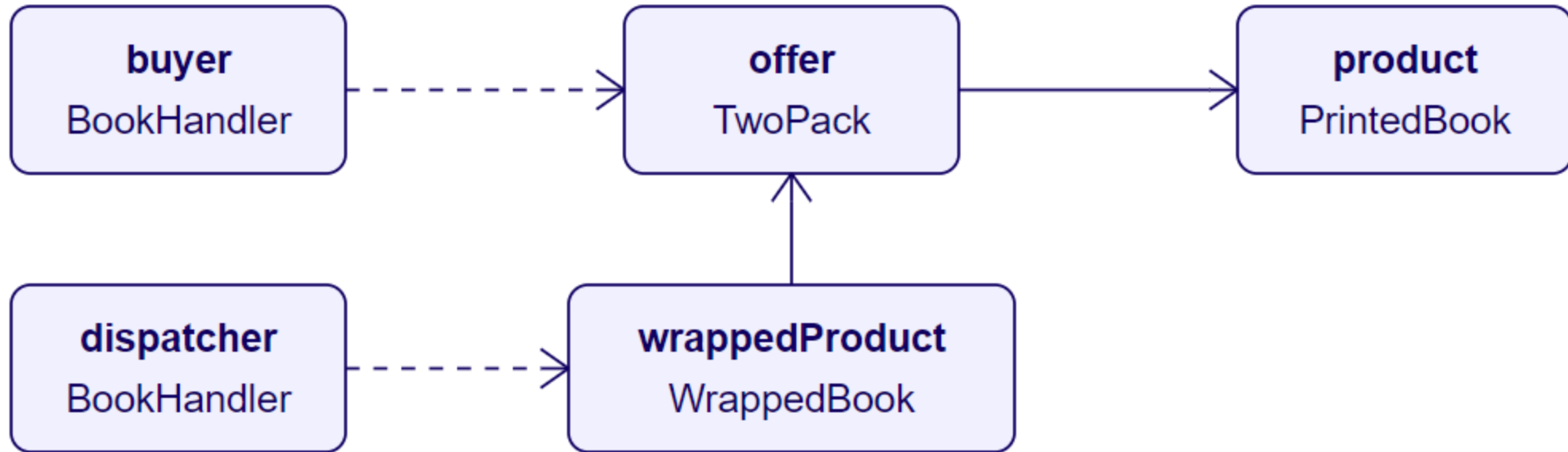


`wrappedProduct.GetDimensions()` returns 19.5 x 24.6 x 6.8 cm

Using the Delegating Decorator



Using the Delegating Decorator



Summary

Decorator design pattern

- Decorator wraps around another object
- Exposes the same interface
- Decorator can substitute the decorated object
- Allowed by the object substitution principle
- Decorator can add behavior to the calls
- Decorator simplifies the consumer

Summary

Implementing the Decorator

- Either derive from the decorated class, or
- Implement the same interface
- Subclassing causes many issues
- Implement the same interface in both classes
- Keep a reference to the decorated object

Summary

Abstract Decorator supports different decorators

- Only implement added behavior in concrete decorators



Next module:

Adapting to a Different Interface
with the Adapter Pattern