

Behavioral Pattern: Visitor



Kevin Dockx

Architect

@Kevindockx | www.kevindockx.com

Coming Up



Describing the visitor pattern

- Calculating discounts for employees and customers

Structure of the visitor pattern

Pattern variation: simplifying the visitor interface



Coming Up



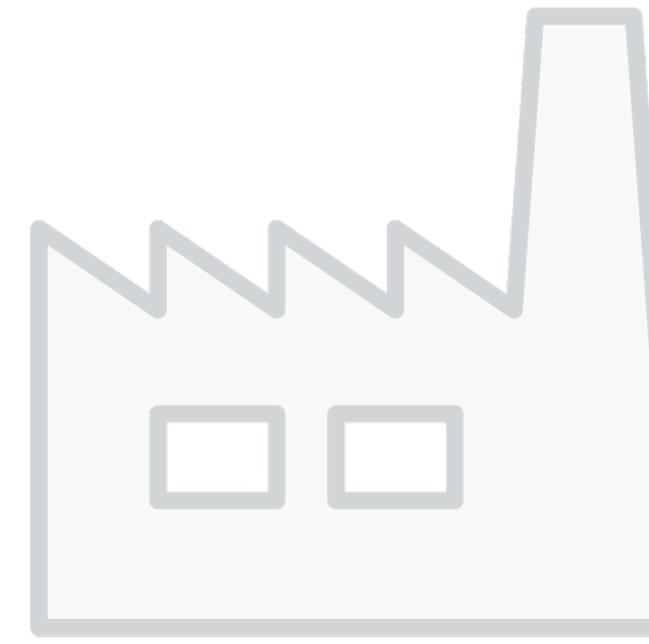
Use cases for this pattern

Pattern consequences

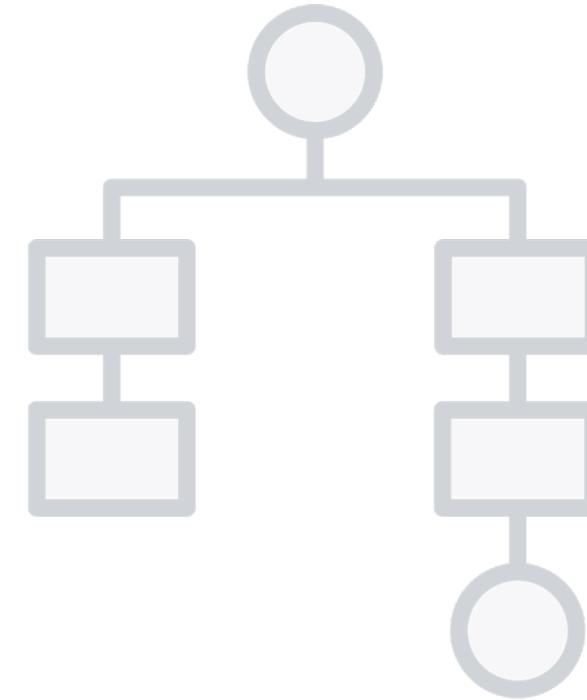
Related patterns



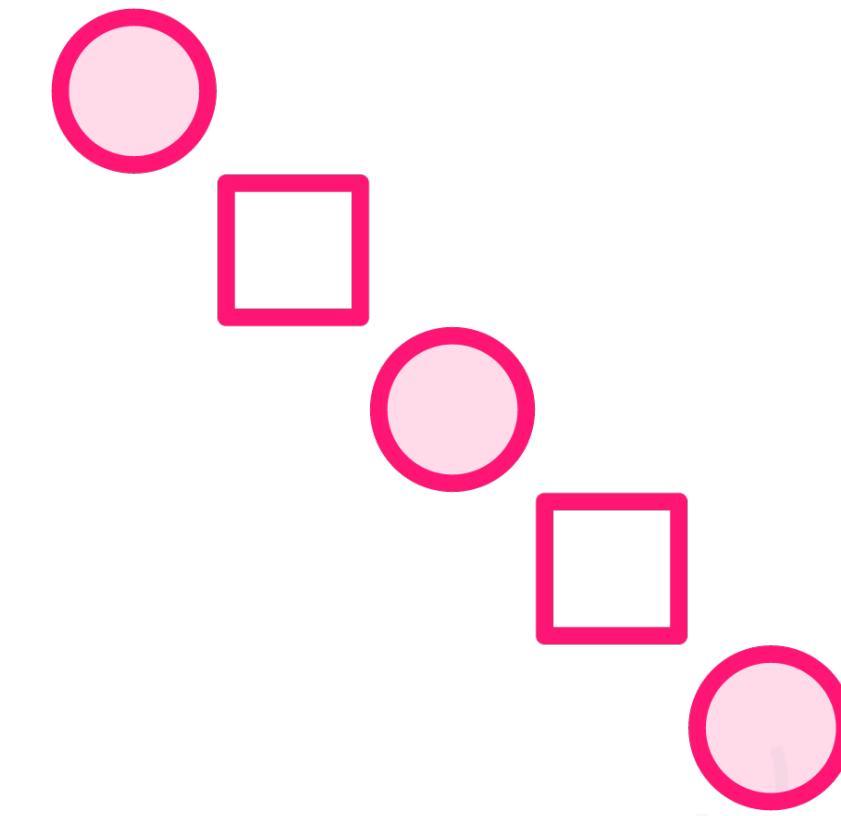
Describing the Visitor Pattern



Creational



Structural



Behavioral



Visitor

The intent of this pattern is to represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.



```
public class Customer  
{  
  
public class InternalCustomer : Customer { }  
  
public class GovernmentCustomer : Customer { }  
  
public class PrivateCustomer : Customer { }
```

Describing the Visitor Pattern



```
public class Customer
{}

public class InternalCustomer : Customer { }

public class GovernmentCustomer : Customer { }

public class PrivateCustomer : Customer { }
```

Describing the Visitor Pattern



```
public class Customer
{}

public class InternalCustomer : Customer { }

public class GovernmentCustomer : Customer { }

public class PrivateCustomer : Customer { }
```

Describing the Visitor Pattern



```
public class Customer
{}

public class InternalCustomer : Customer { }

public class GovernmentCustomer : Customer { }

public class PrivateCustomer : Customer { }
```

Describing the Visitor Pattern



```
public class Customer
{
    public decimal CalculateDiscount()
    { // do calculation }
}

public class InternalCustomer : Customer { }

public class GovernmentCustomer : Customer { }

public class PrivateCustomer : Customer { }
```

Describing the Visitor Pattern



```
public class Customer
{
    public decimal CalculateDiscount()
    { // do calculation }
}

public class InternalCustomer : Customer { }

public class GovernmentCustomer : Customer { }

public class PrivateCustomer : Customer { }

public class Employee
{
    public decimal CalculateDiscount()
    { // do calculation }
}
```

Describing the Visitor Pattern



```
public class Customer
{
    public decimal CalculateDiscount()
    { // do calculation }
}

public class InternalCustomer : Customer { }

public class GovernmentCustomer : Customer { }

public class PrivateCustomer : Customer { }

public class Employee
{
    public decimal CalculateDiscount()
    { // do calculation }
}
```

Describing the Visitor Pattern



```
public class Customer
{
    public decimal CalculateDiscount()
    { // do calculation }
}

public class InternalCustomer : Customer { }

public class GovernmentCustomer : Customer { }

public class PrivateCustomer : Customer { }

public class Employee
{
    public decimal CalculateDiscount()
    { // do calculation }
}
```

Describing the Visitor Pattern



Describing the Visitor Pattern

The more additional requirements come in, the more often these classes need to be changed

- Also, adding all that behavior violates the single responsibility principle



Describing the Visitor Pattern



Describing the Visitor Pattern

Customer

decimal Discount

Employee

decimal Discount



Describing the Visitor Pattern

IElement

void Accept(IVisitor Visitor)

Customer

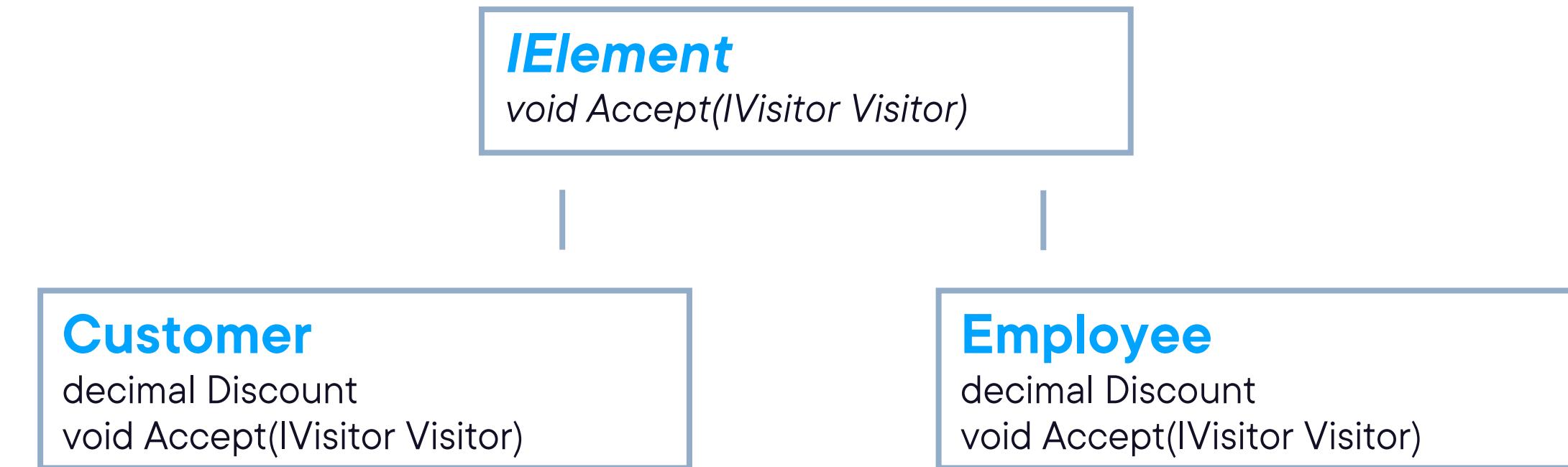
decimal Discount

Employee

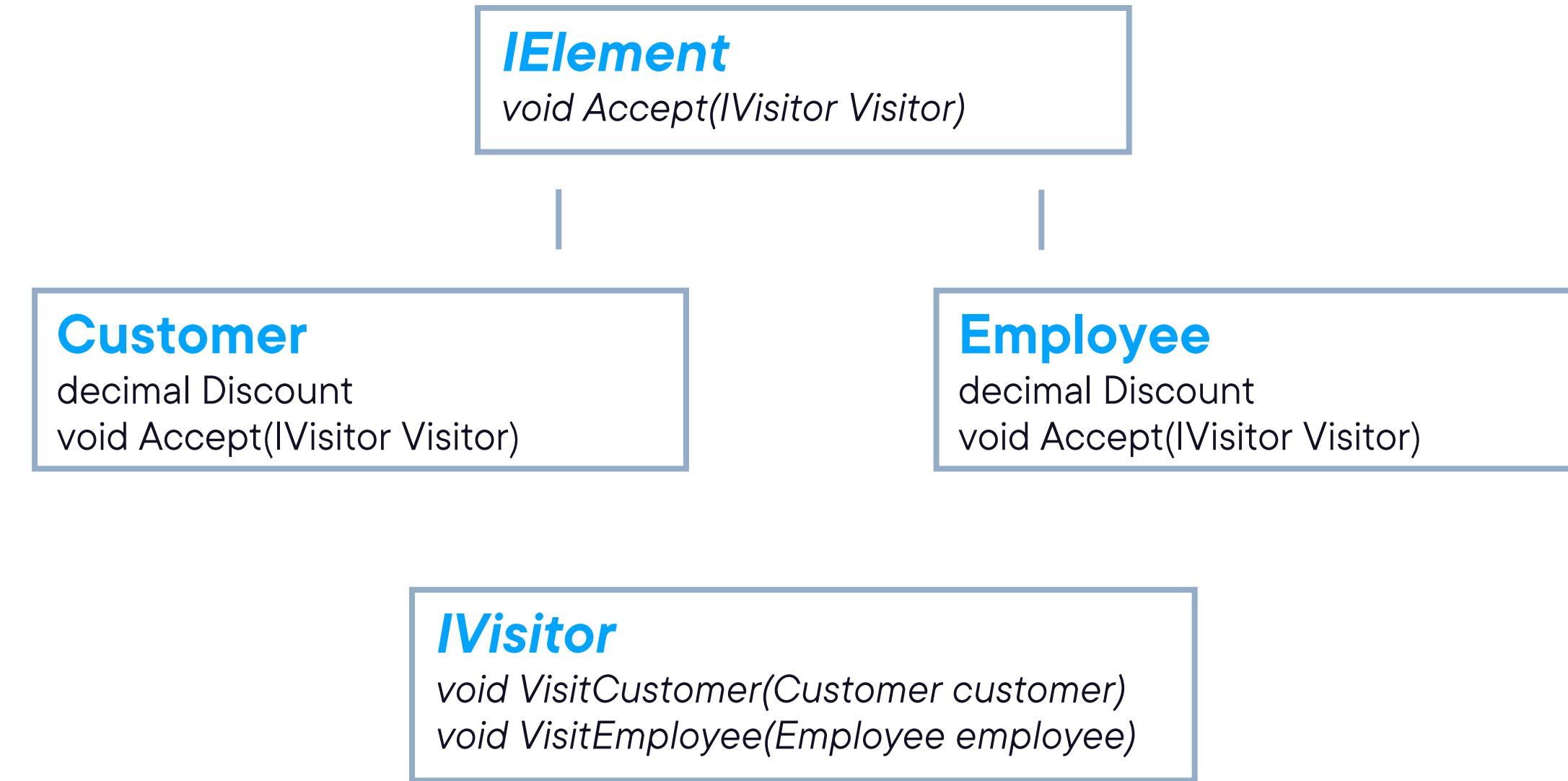
decimal Discount



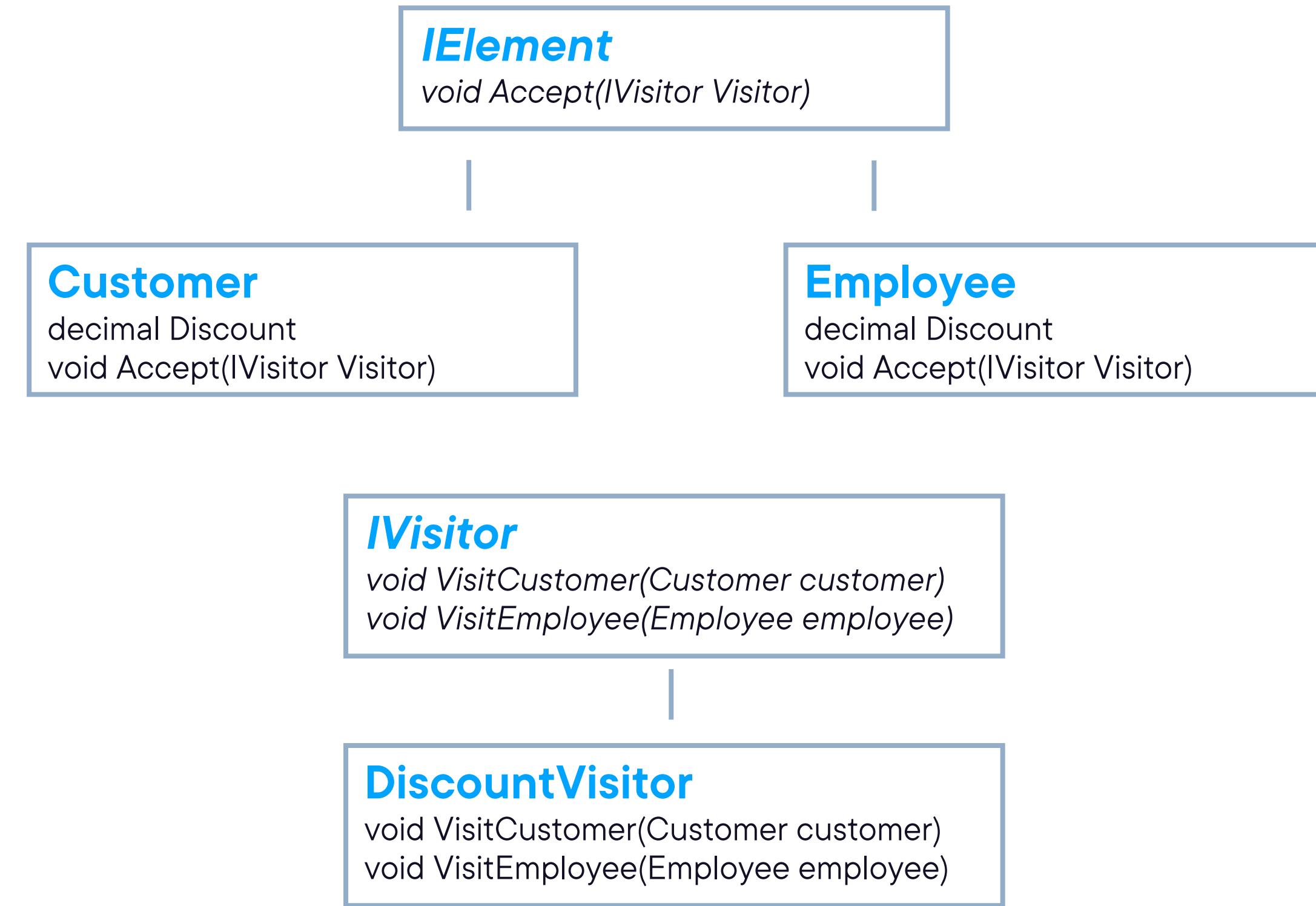
Describing the Visitor Pattern



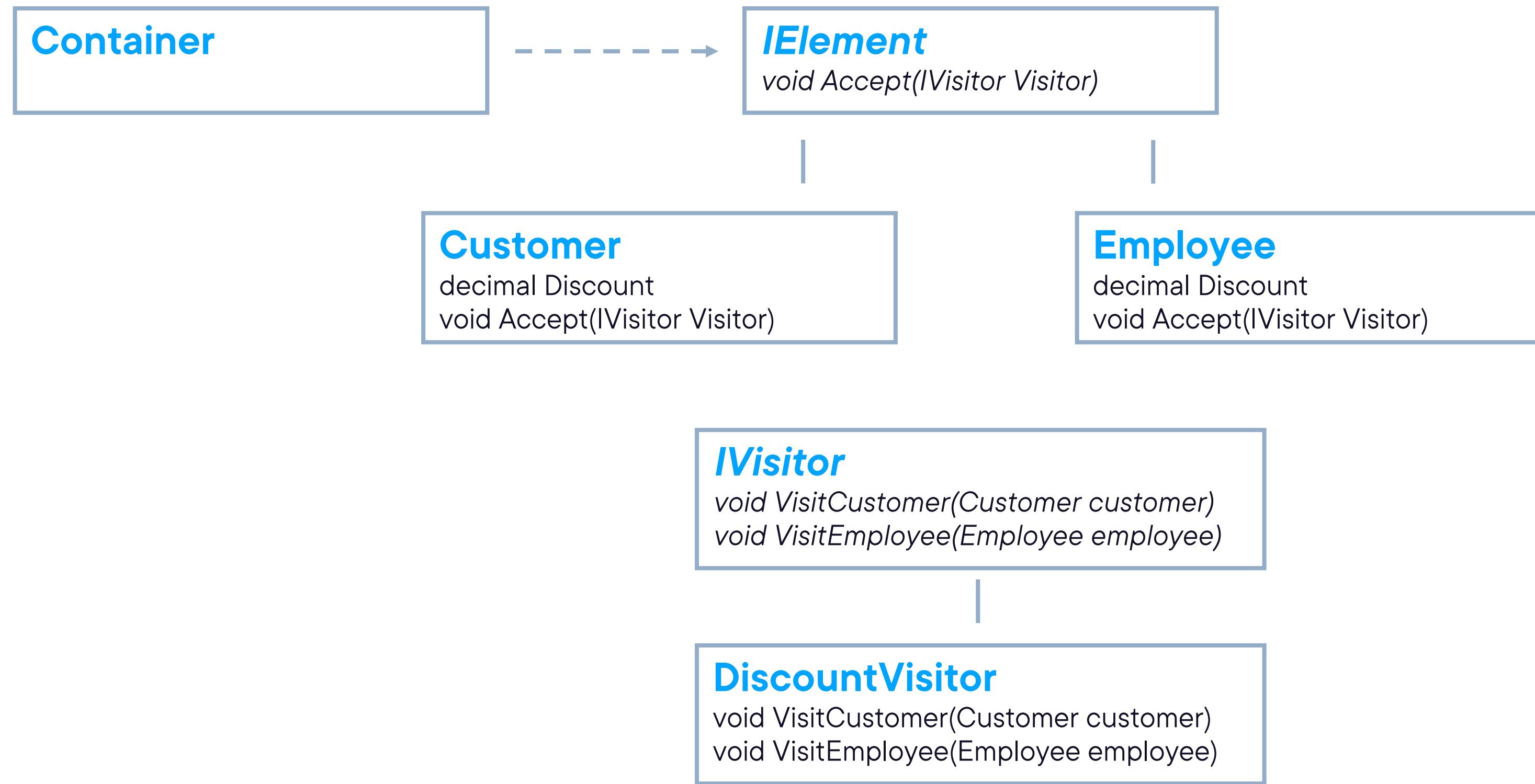
Describing the Visitor Pattern



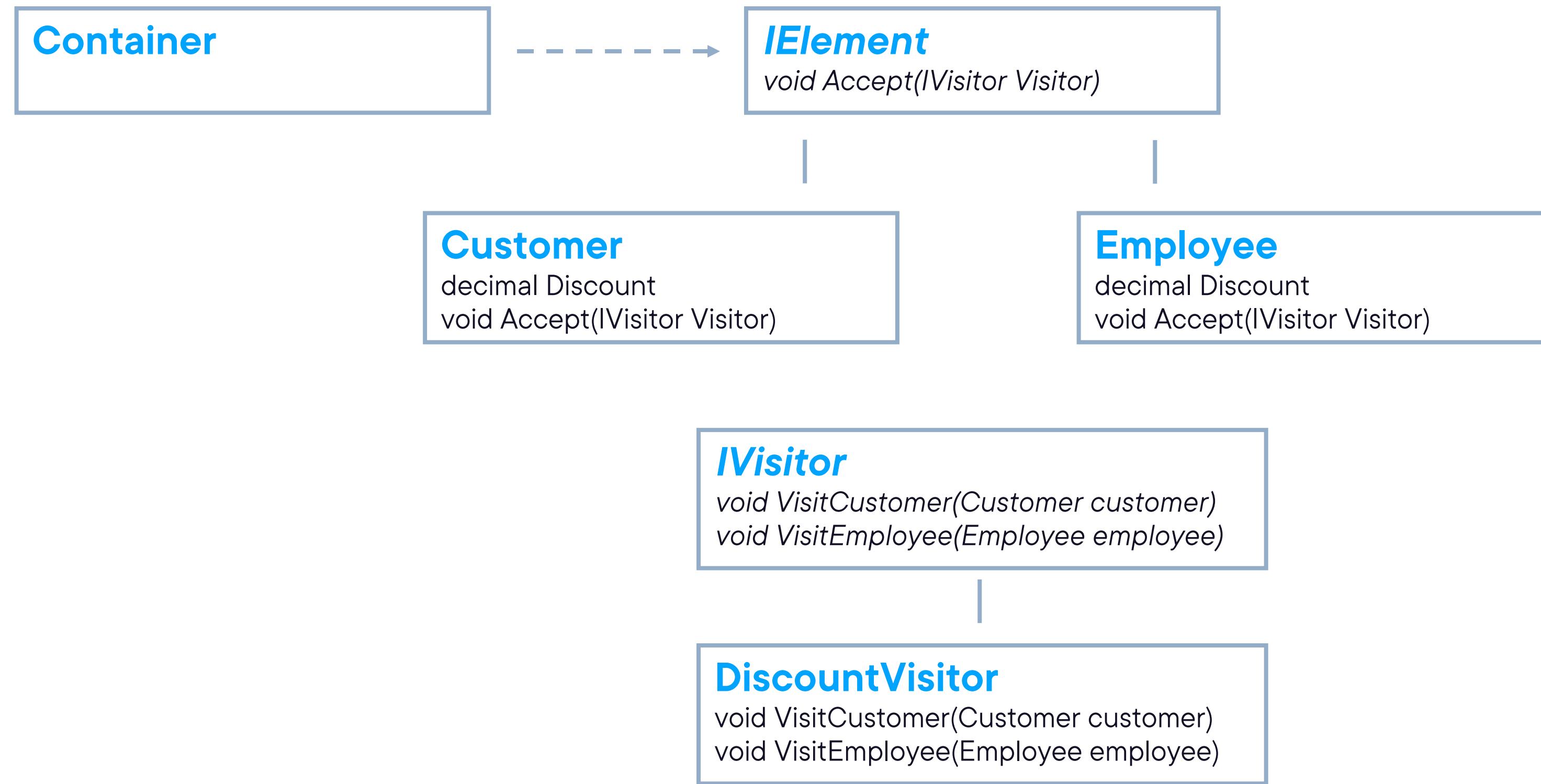
Describing the Visitor Pattern



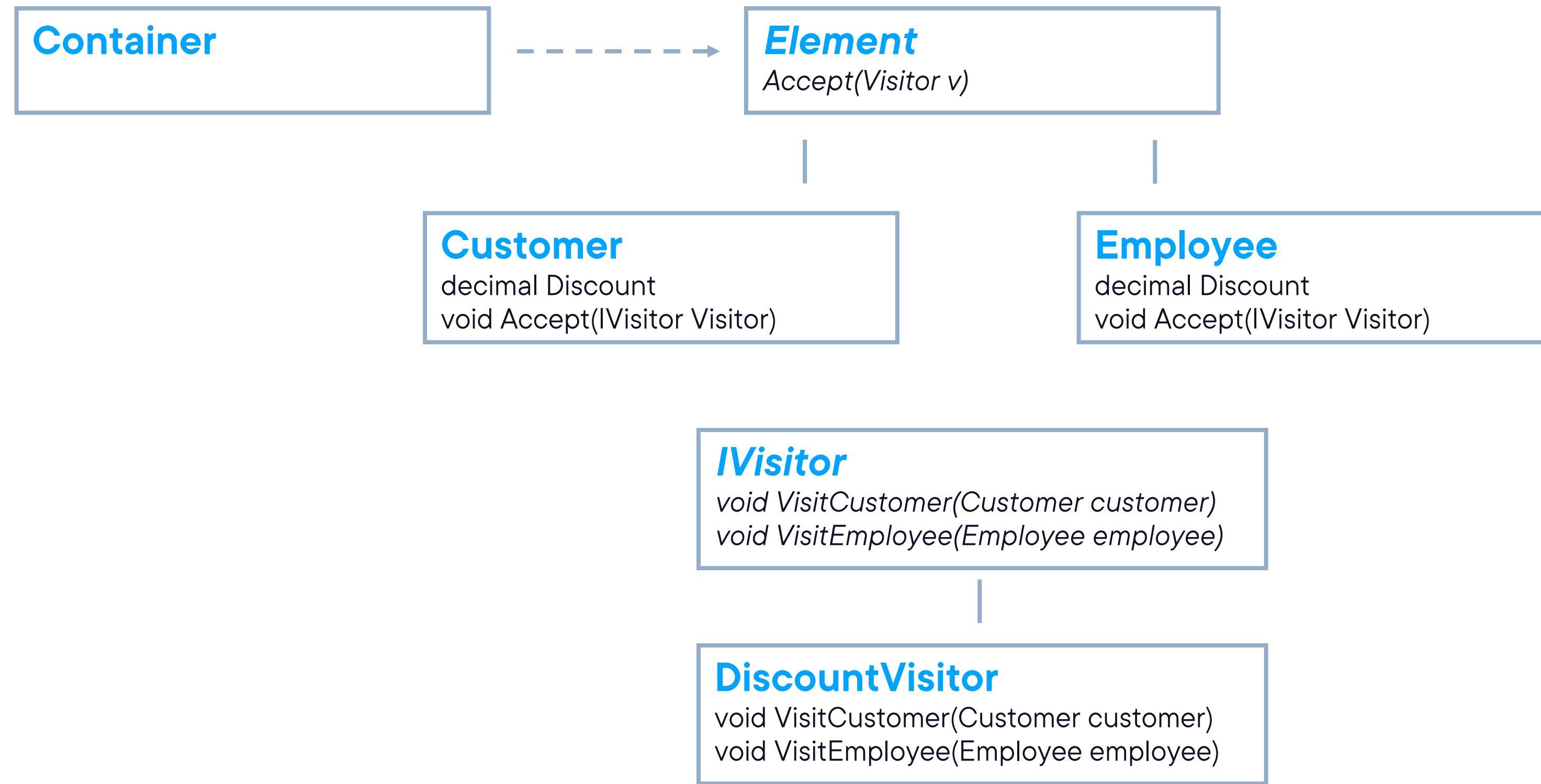
Describing the Visitor Pattern



Structure of the Visitor Pattern



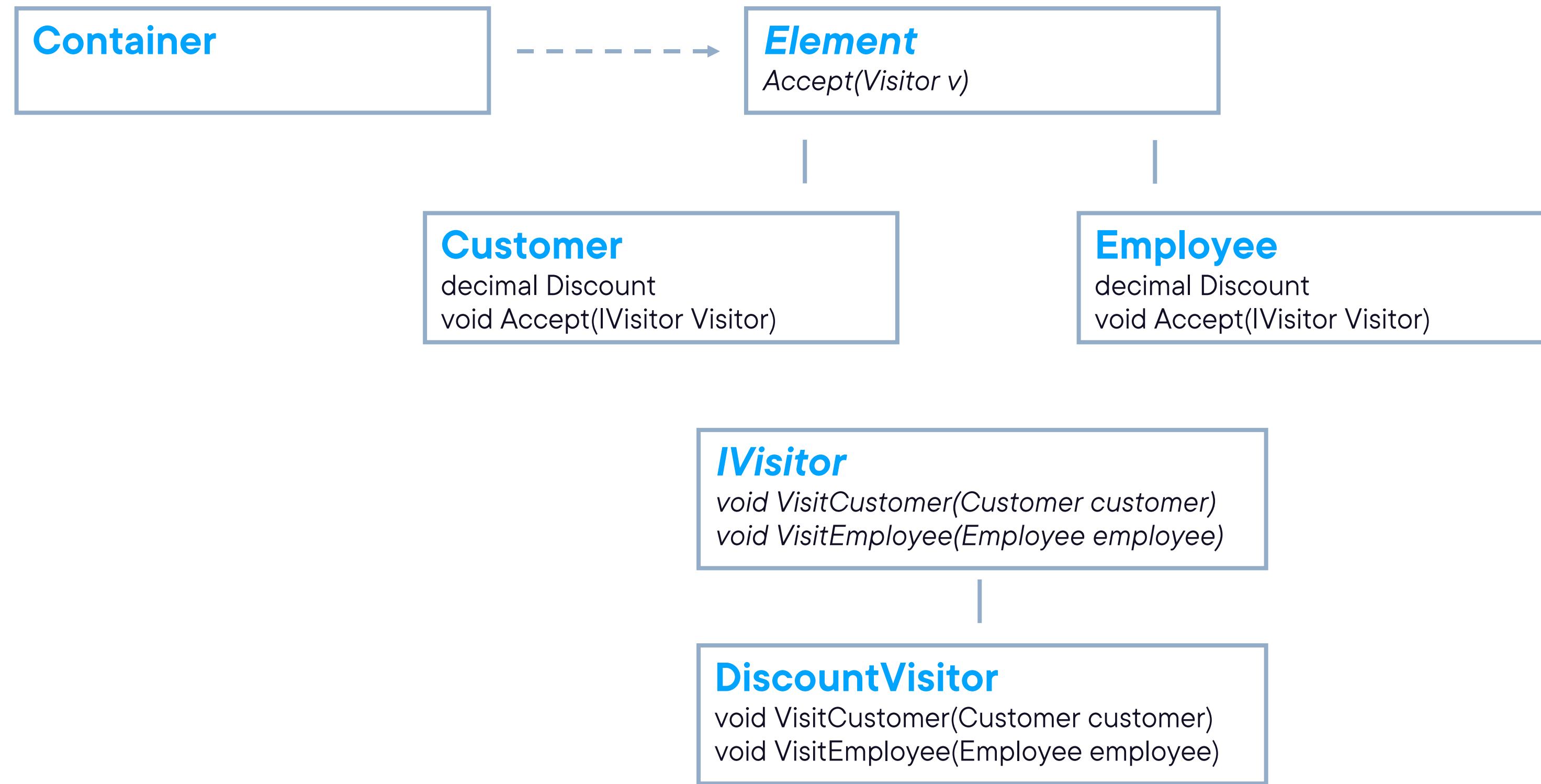
Structure of the Visitor Pattern



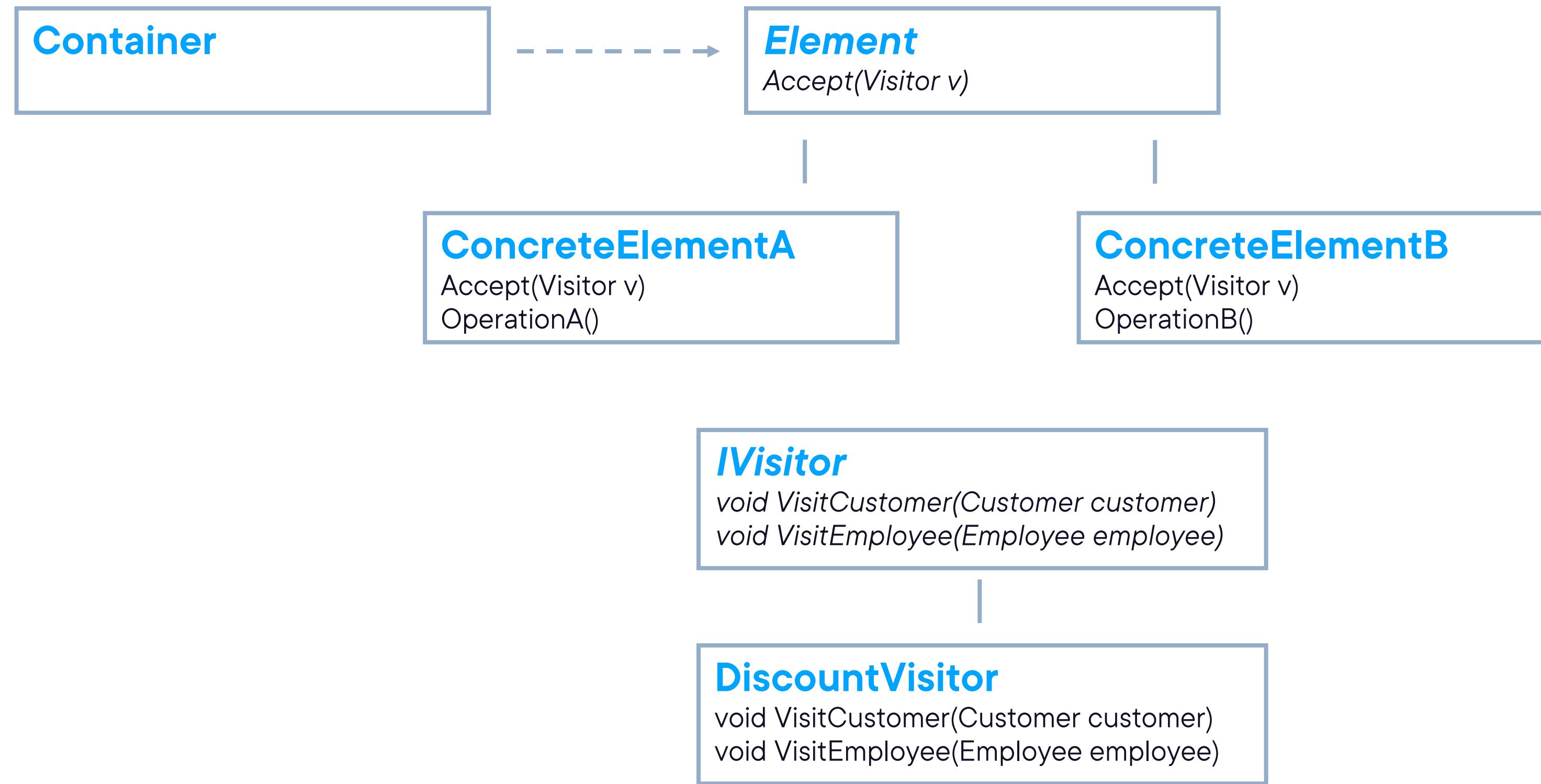
**Element defines an accept
operation that takes a
Visitor as an argument**



Structure of the Visitor Pattern



Structure of the Visitor Pattern



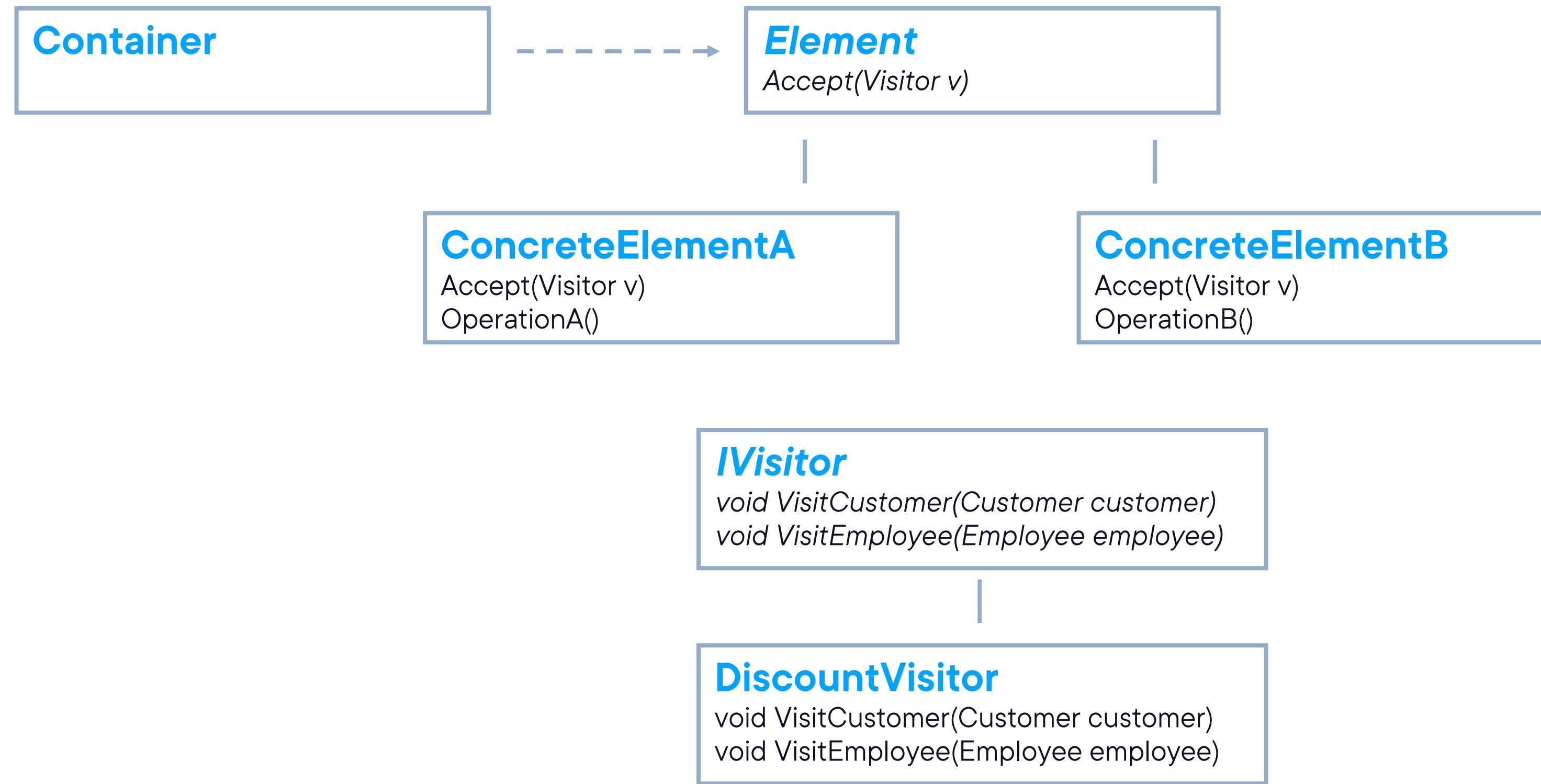
**ConcreteElement
implements the accept
operation that takes a
Visitor as an argument**



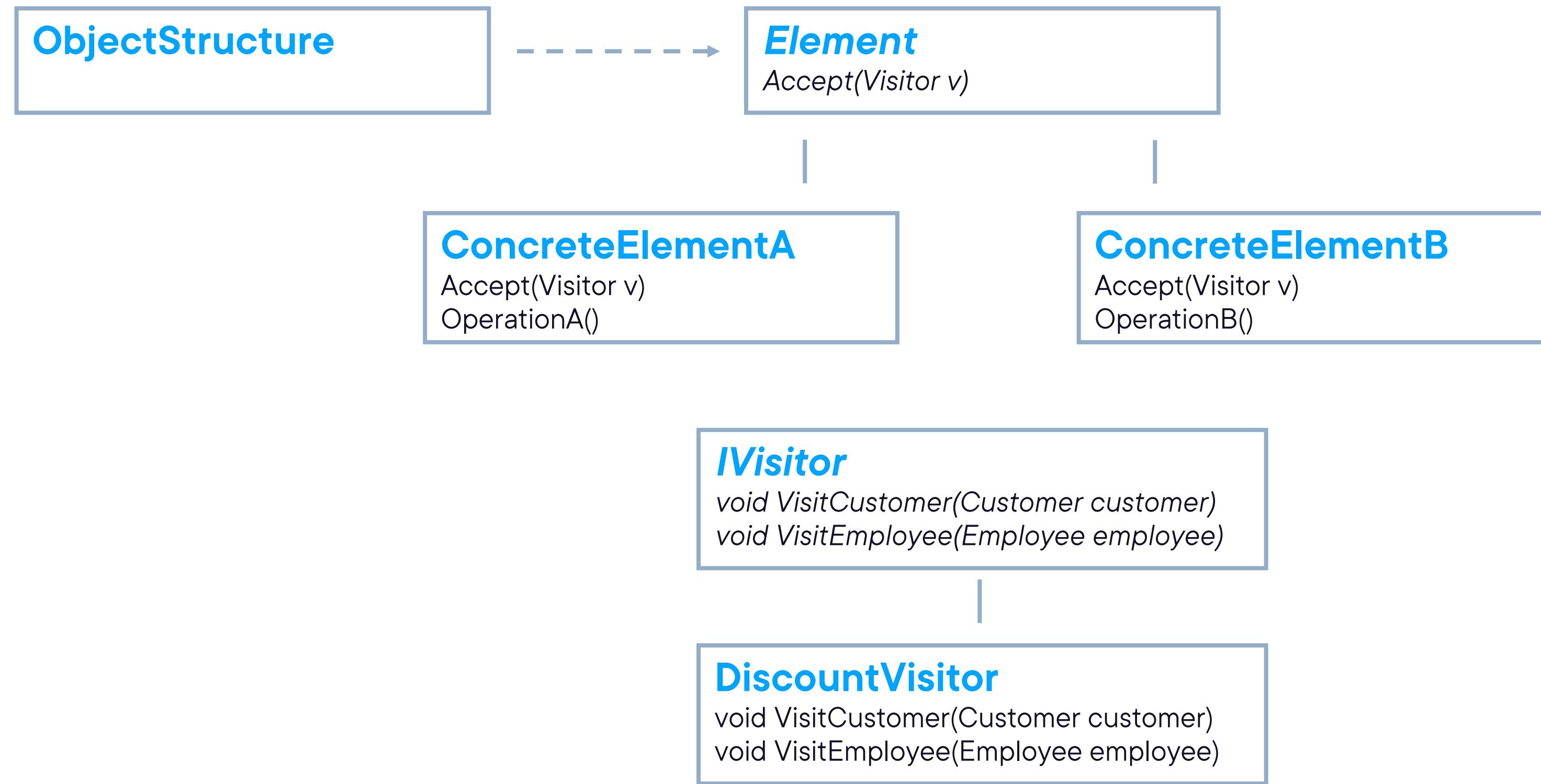
ObjectStructure enumerates its elements. It may provide an interface to allow a Visitor to visit its Elements. It can be a composite or a collection.



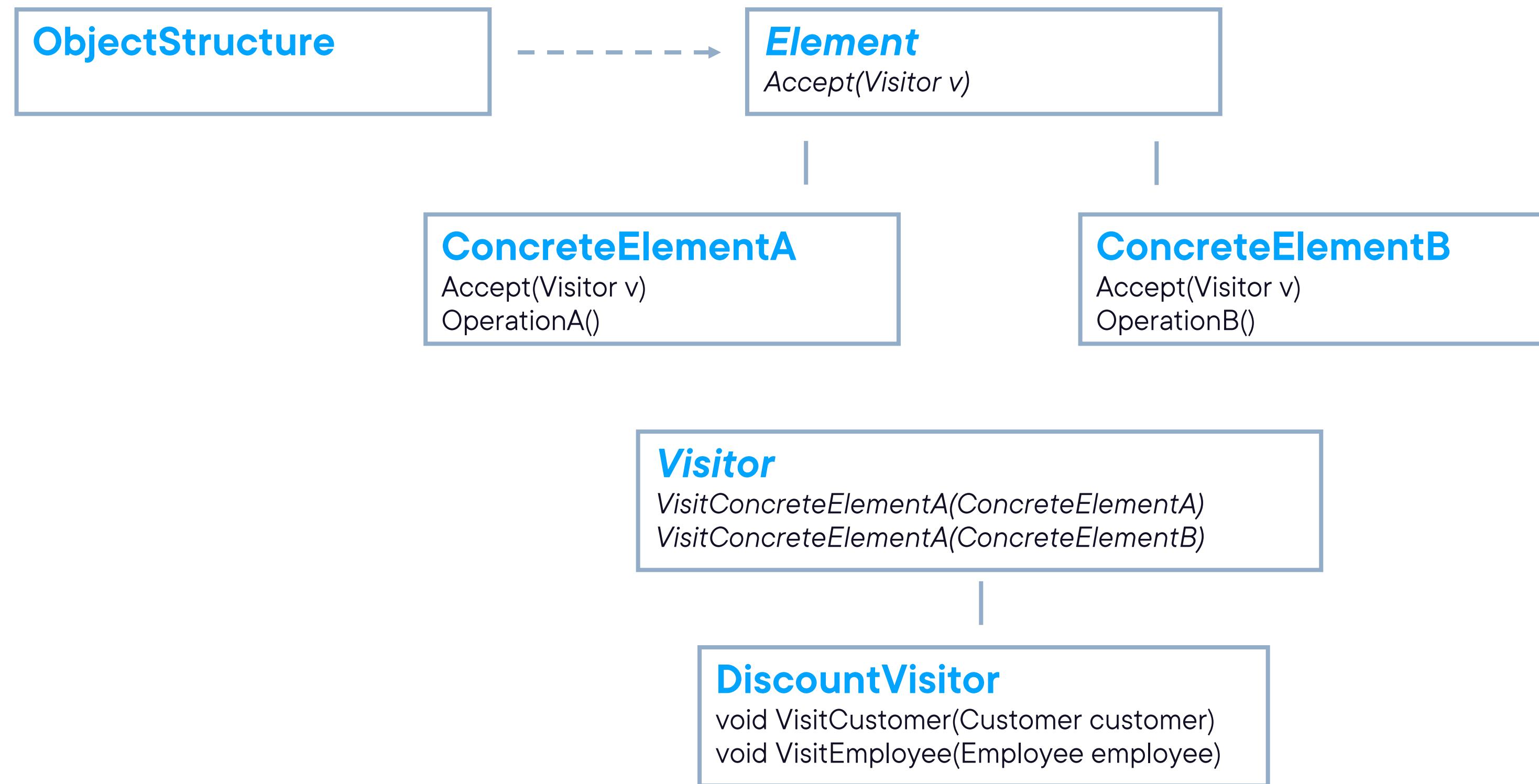
Structure of the Visitor Pattern



Structure of the Visitor Pattern



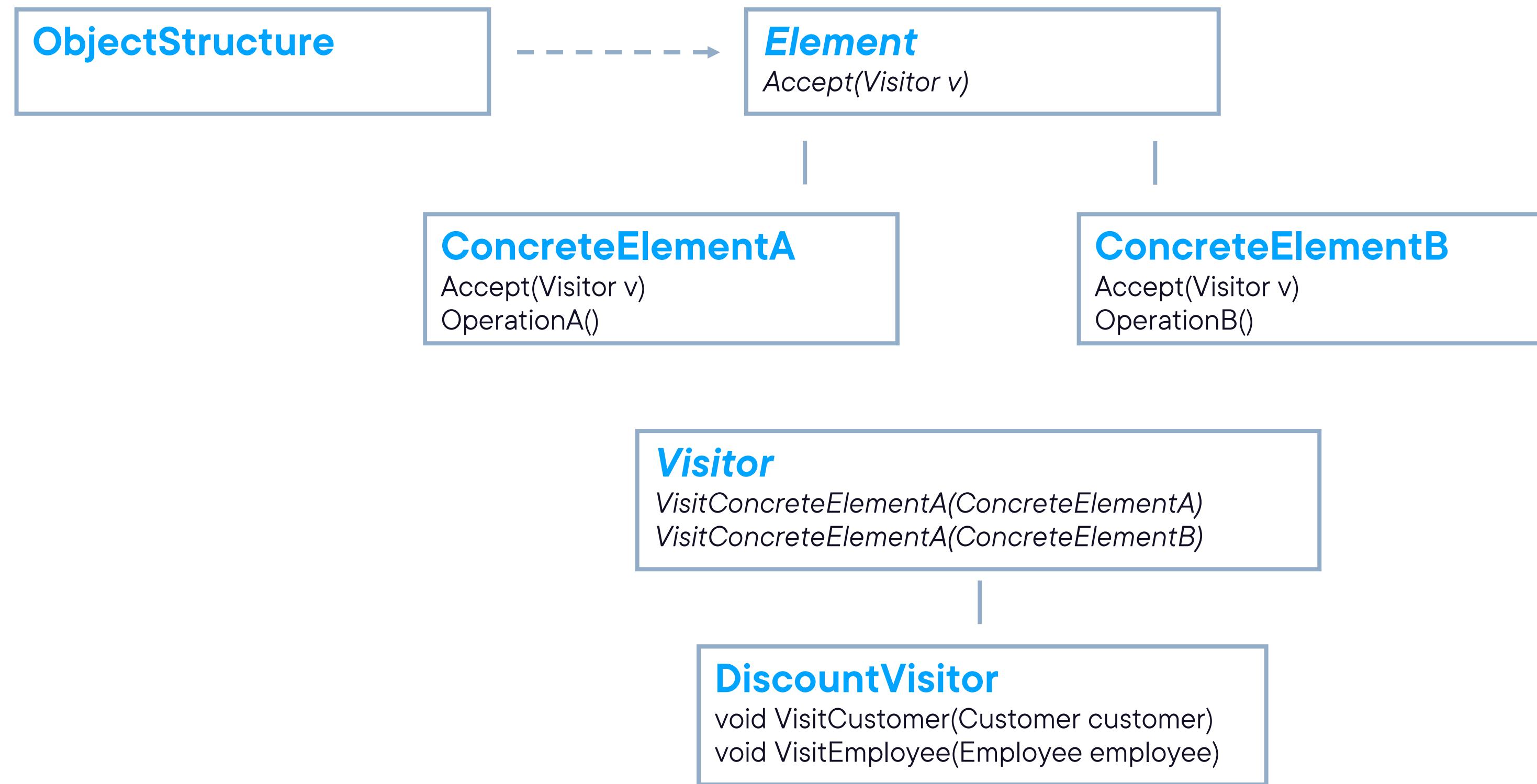
Structure of the Visitor Pattern



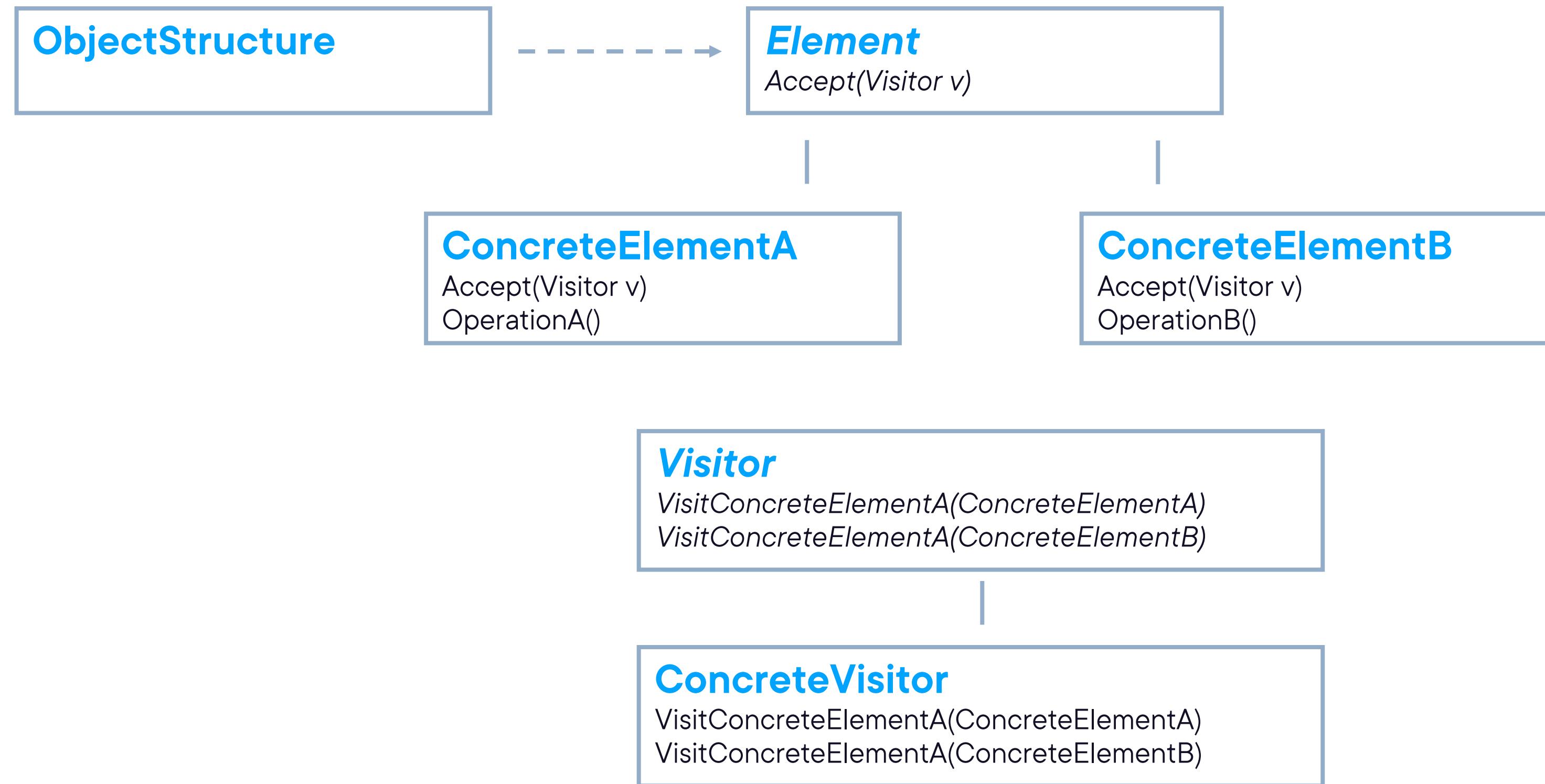
Visitor declares a visit
operation for each class of
ConcreteElement in the
ObjectStructure



Structure of the Visitor Pattern



Structure of the Visitor Pattern



**ConcreteVisitor
implements each operation
declared by Visitor**



Demo



Implementing the visitor pattern



Demo



Simplifying the visitor interface



Use Cases for the Visitor Pattern



When an object structure contains many classes of objects with differing interfaces, and you want to perform operations on them that depend on their concrete classes



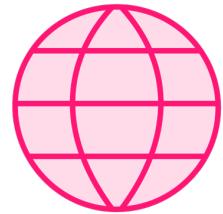
When the classes defining your object structure don't have to change often, but you do often want to define new operations over the structure



When you've got potentially many operations that need to be performed on objects in your object structure, but not necessarily on all of them



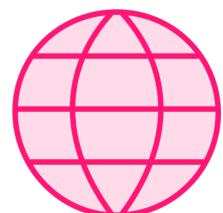
Use Cases for the Visitor Pattern



Document element processing



Medical records system



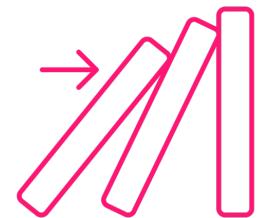
Data structure serialization



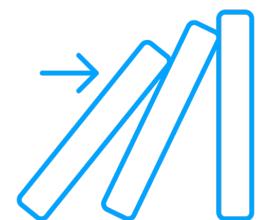
Financial transactions within a financial software system



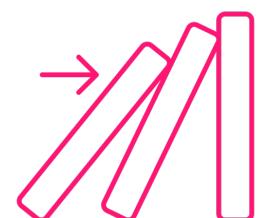
Pattern Consequences



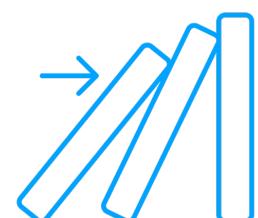
It makes adding new operations easy; you can define a new operation by creating a new visitor: **open/closed principle**



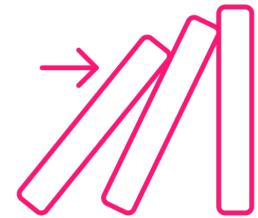
A visitor can accumulate info on the objects it visits



A visitor gathers related operations together (and separates unrelated ones: **single responsibility principle**)



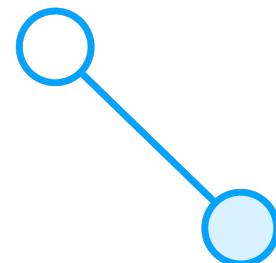
Adding a new **ConcreteElement** class can be hard



It might require you to break encapsulation

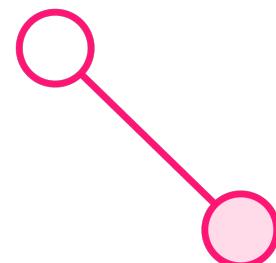


Related Patterns



Composite

A visitor can be used to apply an operation over an object structure defined by the composite pattern



Iterator

You can use an iterator to traverse a potentially complex data structure, and apply logic to the items in that structure with a visitor



Summary



Intent of the visitor pattern:

- To represent an operation to be performed on the elements of an object structure



Summary



Implementation:

- IVisitor interface(s) (and implementations) work on concrete elements



Up Next:

Behavioral Pattern: Interpreter

