

Creational Pattern: Abstract Factory



Kevin Dockx

Architect

@Kevindockx | www.kevindockx.com

Coming Up



Describing the abstract factory pattern

Structure of the abstract factory pattern

Implementation

- Real-life sample: shopping cart

**Comparing abstract factory and
factory method patterns**



Coming Up



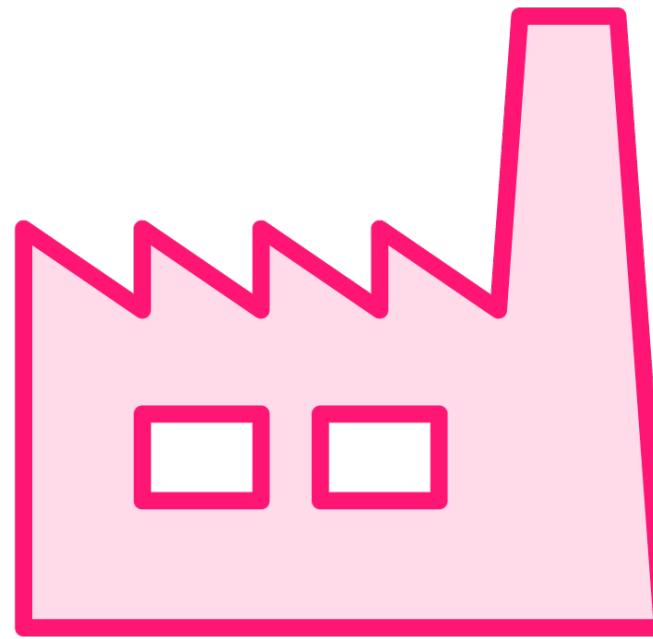
Use cases for this pattern

Pattern consequences

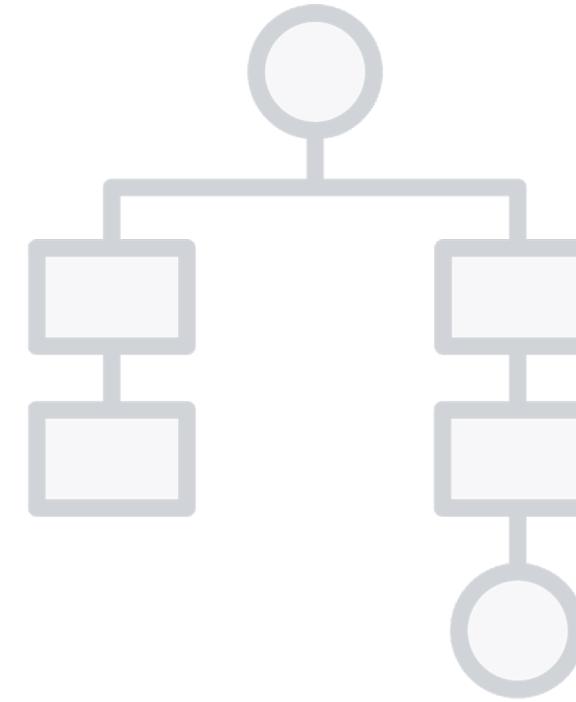
Related patterns



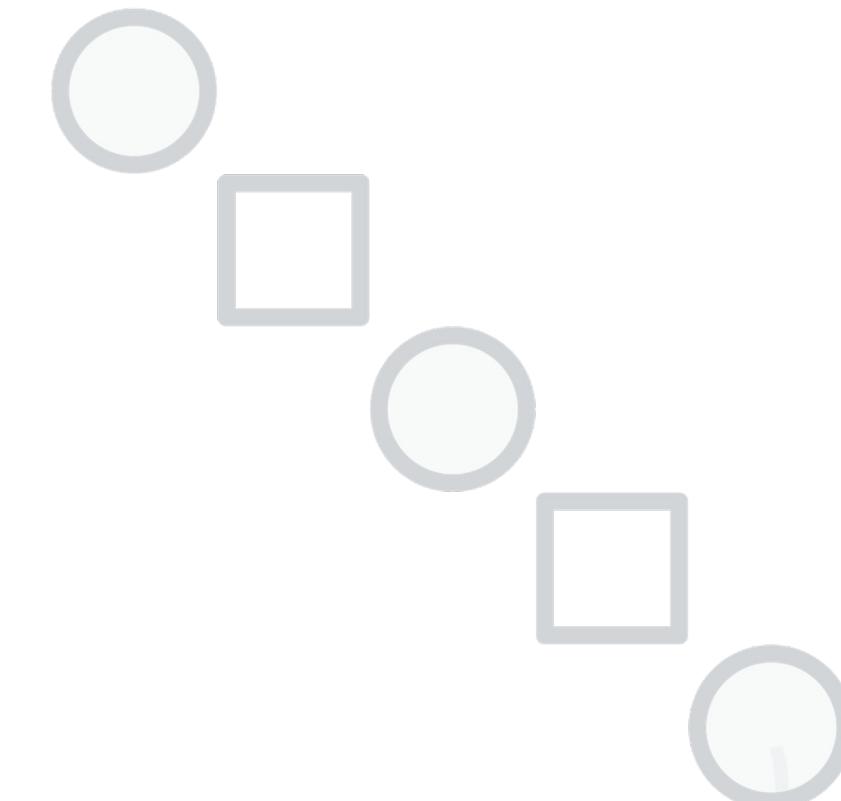
Describing the Abstract Factory Pattern



Creational



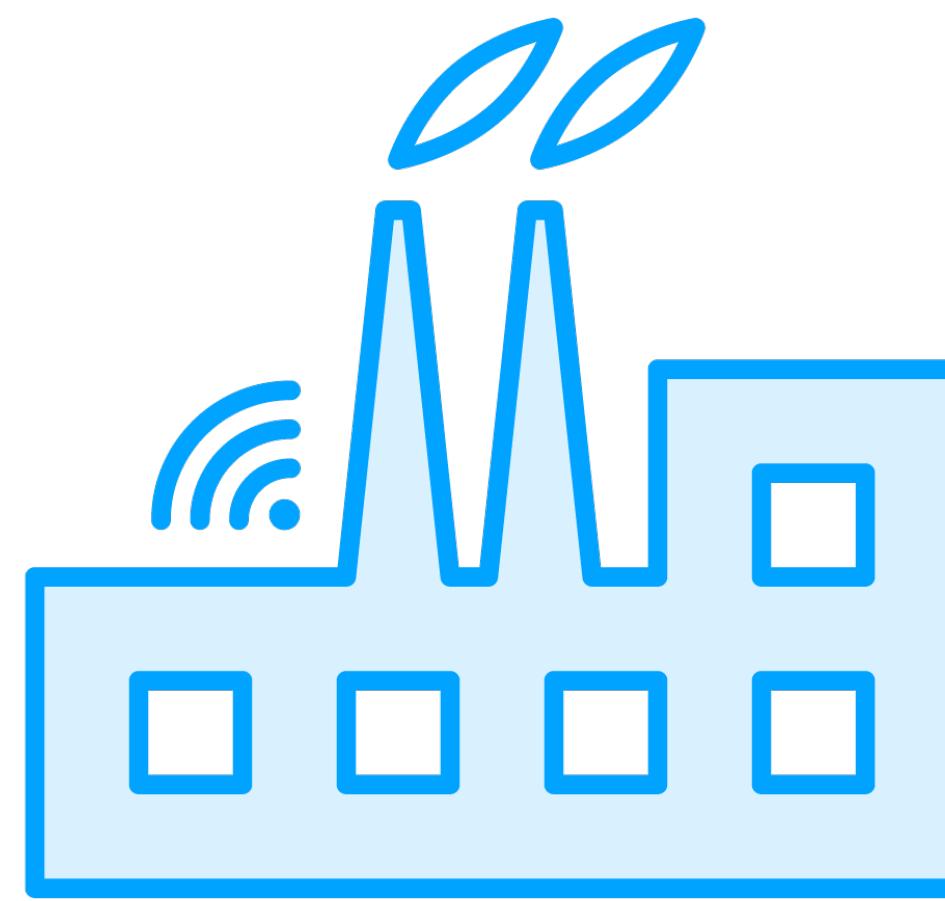
Structural



Behavioral



Describing the Abstract Factory Pattern



Factory method



Abstract factory



Abstract Factory

The intent of the abstract factory pattern is to provide an interface for creating families of related or dependent objects without specifying their concrete classes



Describing the Abstract Factory Pattern

Shopping cart scenario

- `DiscountService` by country
- `ShippingCostsService` by country

Family of objects that belong together



```
var belgiumDiscountService = new BelgiumDiscountService();
var discount = belgiumDiscountService.DiscountPercentage;

var belgiumShippingCostsService = new BelgiumShippingCostsService();
var shippingCosts = belgiumShippingCostsService.ShippingCosts;
```

Describing the Abstract Factory Pattern



```
var belgiumDiscountService = new BelgiumDiscountService();  
var discount = belgiumDiscountService.DiscountPercentage;  
  
var belgiumShippingCostsService = new BelgiumShippingCostsService();  
var shippingCosts = belgiumShippingCostsService.ShippingCosts;
```

Describing the Abstract Factory Pattern

Tight coupling

Doesn't convey the intent that these objects belong together

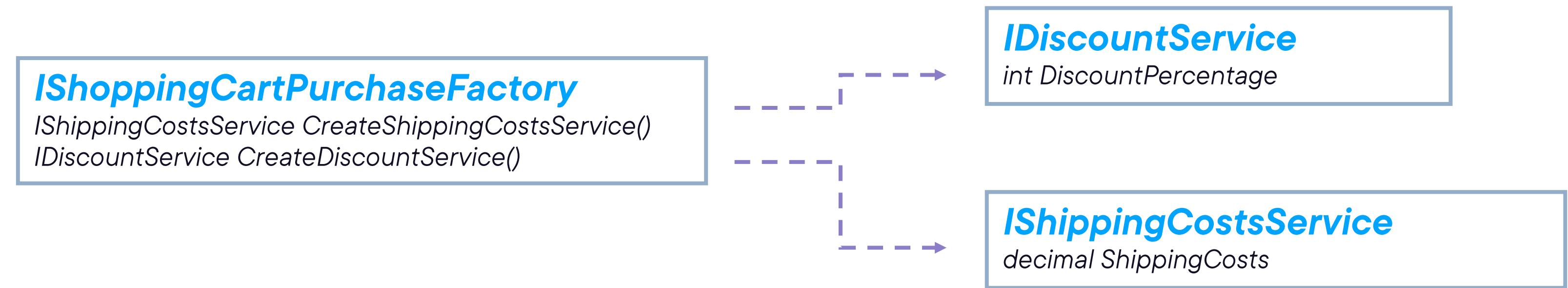


Describing the Abstract Factory Pattern

I ShoppingCartPurchaseFactory
IShippingCostsService CreateShippingCostsService()
IDiscountService CreateDiscountService()



Describing the Abstract Factory Pattern



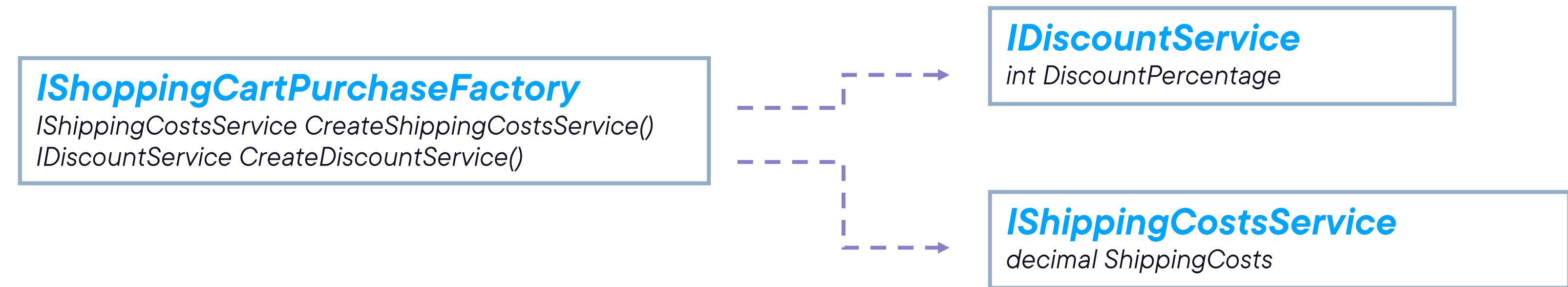
**Use an abstract base class
when you need to provide
some basic functionality
that can potentially be
overridden**



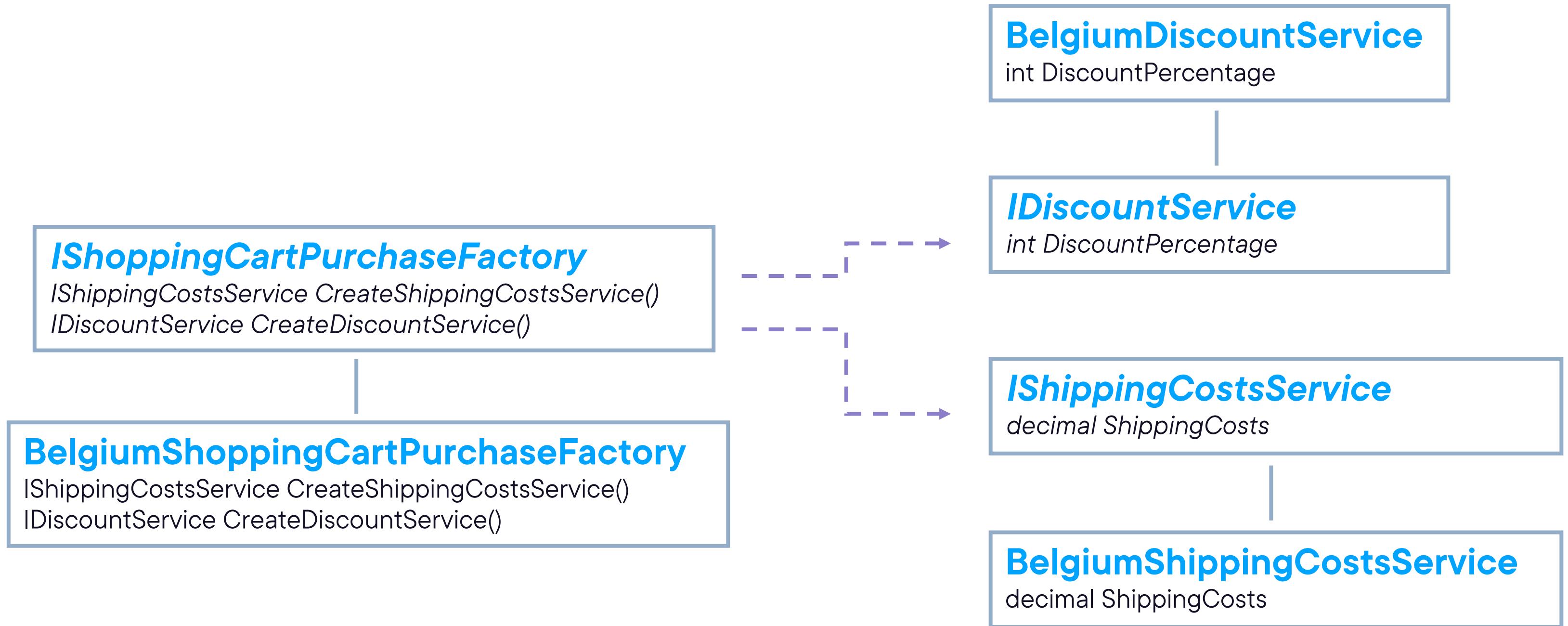
**Use an interface when you
only need to specify the
expected functionality of a
class**



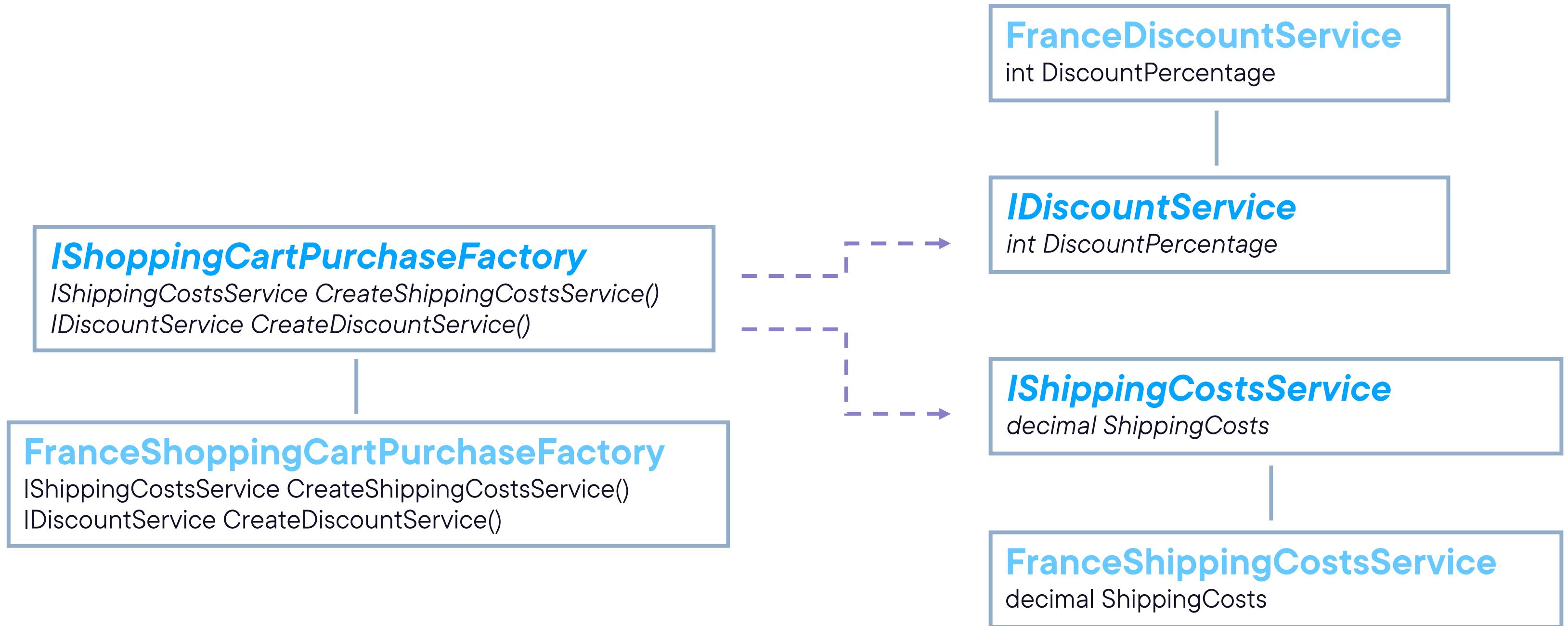
Describing the Abstract Factory Pattern



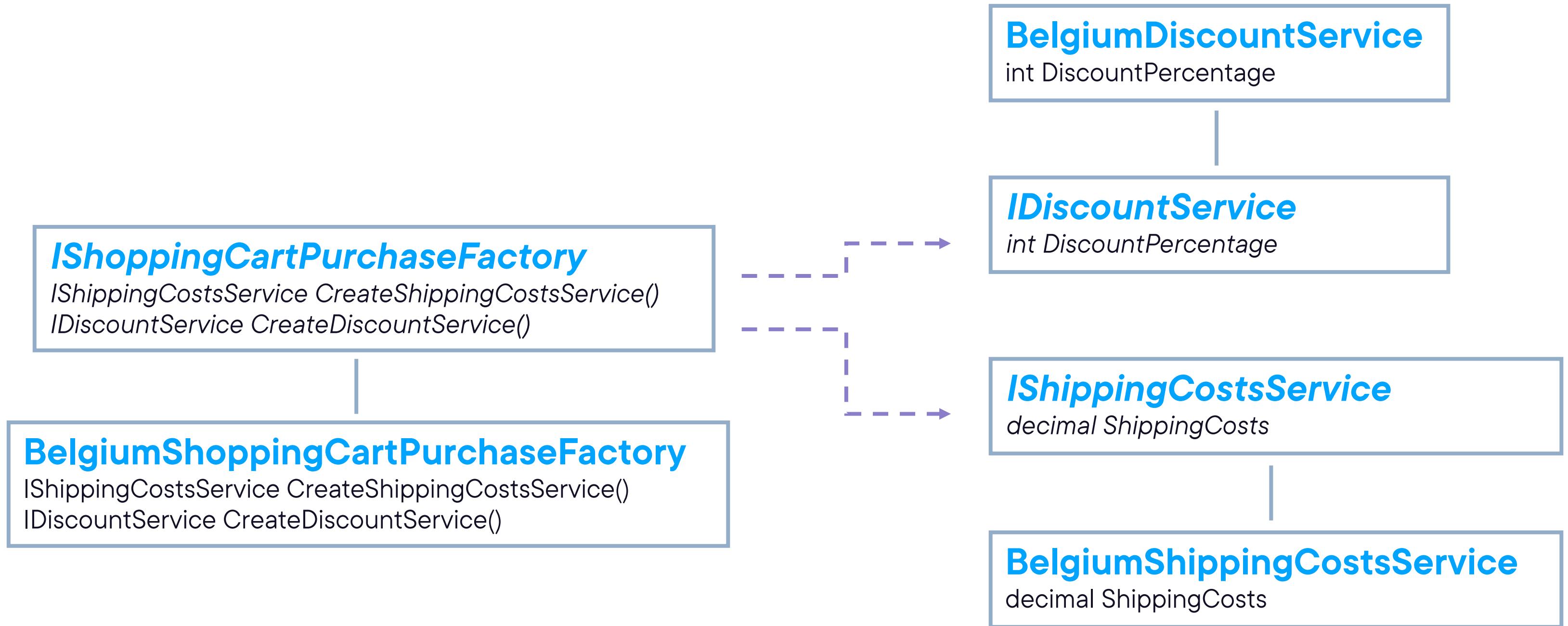
Describing the Abstract Factory Pattern



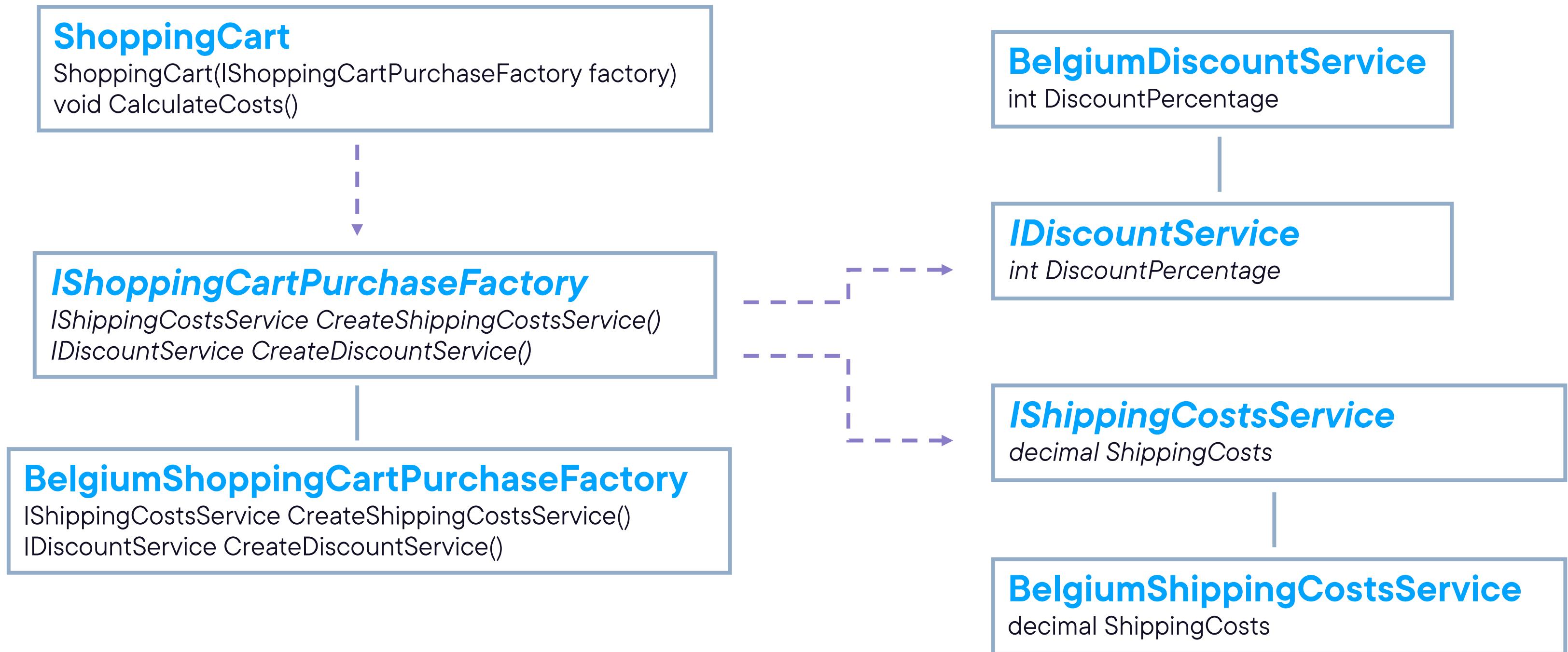
Describing the Abstract Factory Pattern



Describing the Abstract Factory Pattern



Describing the Abstract Factory Pattern



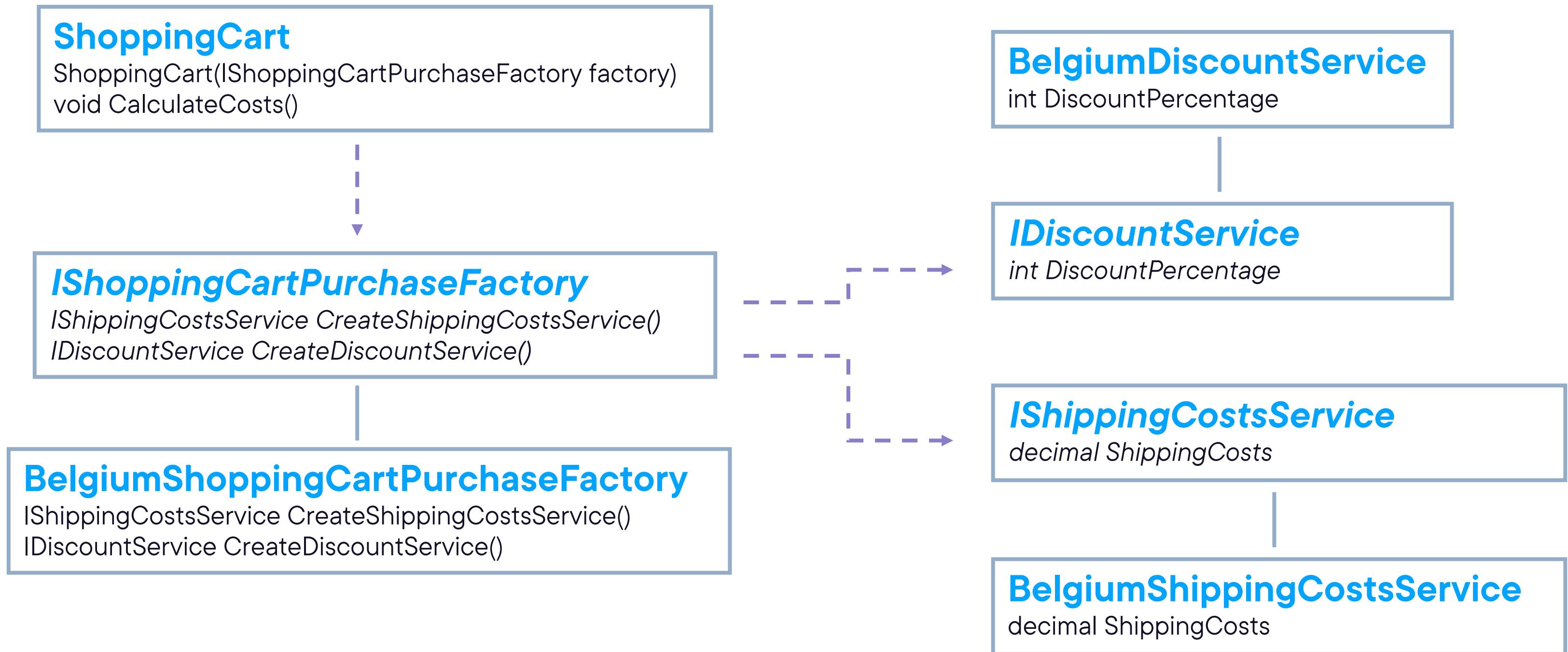
Provide a concrete factory implementation via the constructor



Optionally add methods
that use the concrete
service implementations



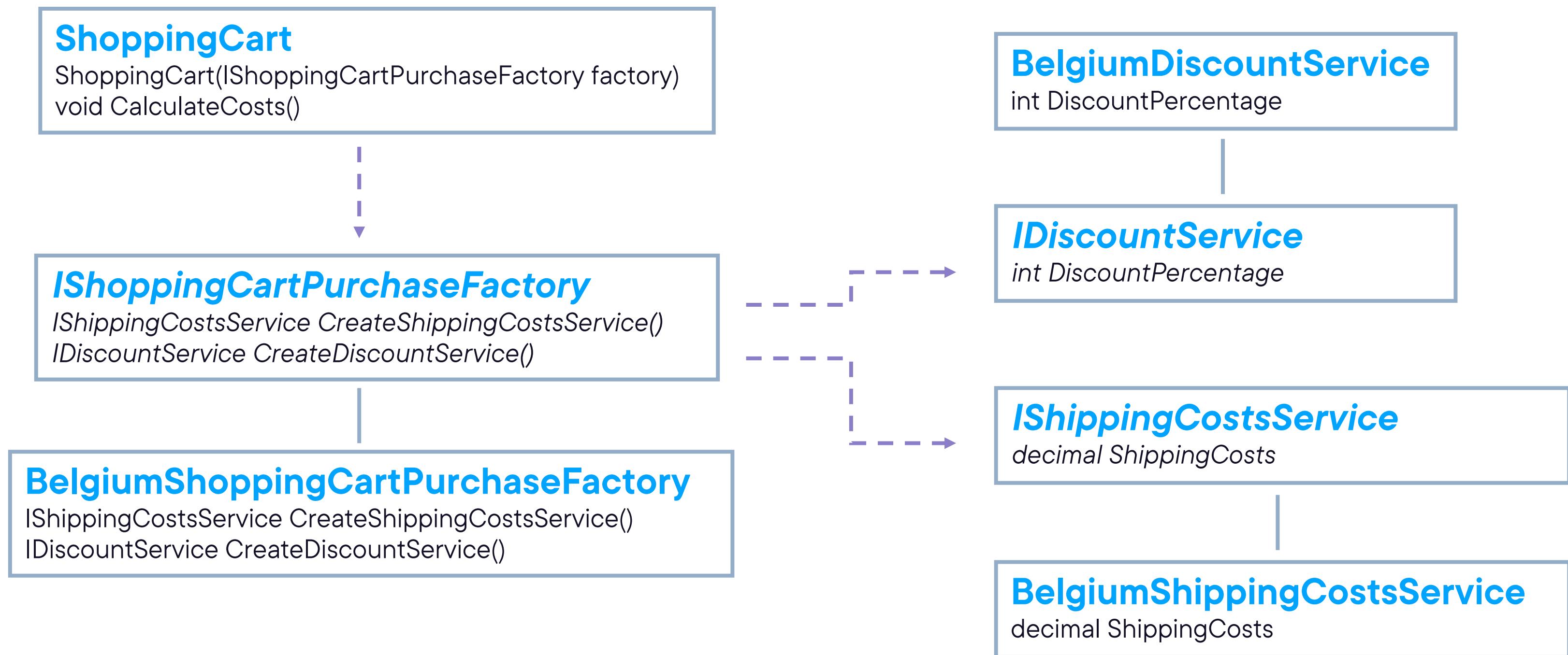
Describing the Abstract Factory Pattern



The client is decoupled
from the concrete factory
implementation



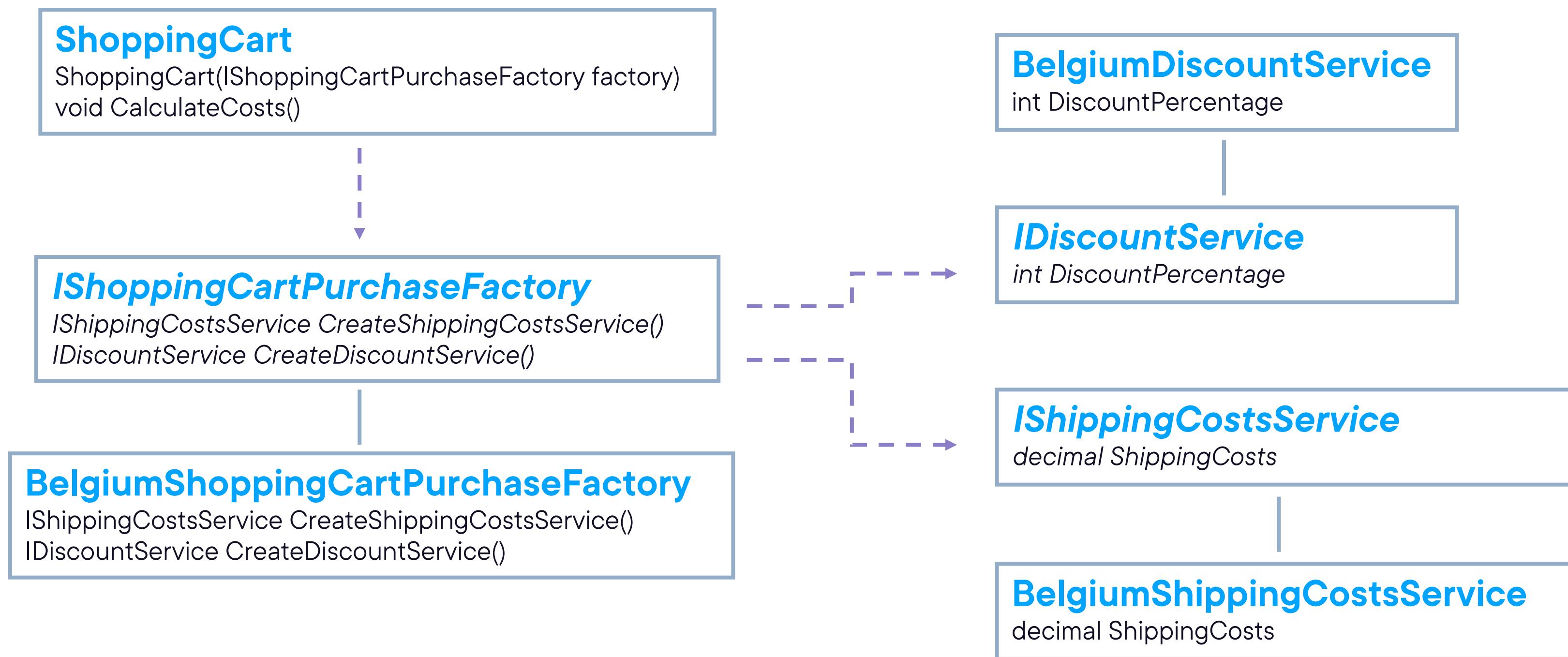
Abstract Factory Pattern Structure



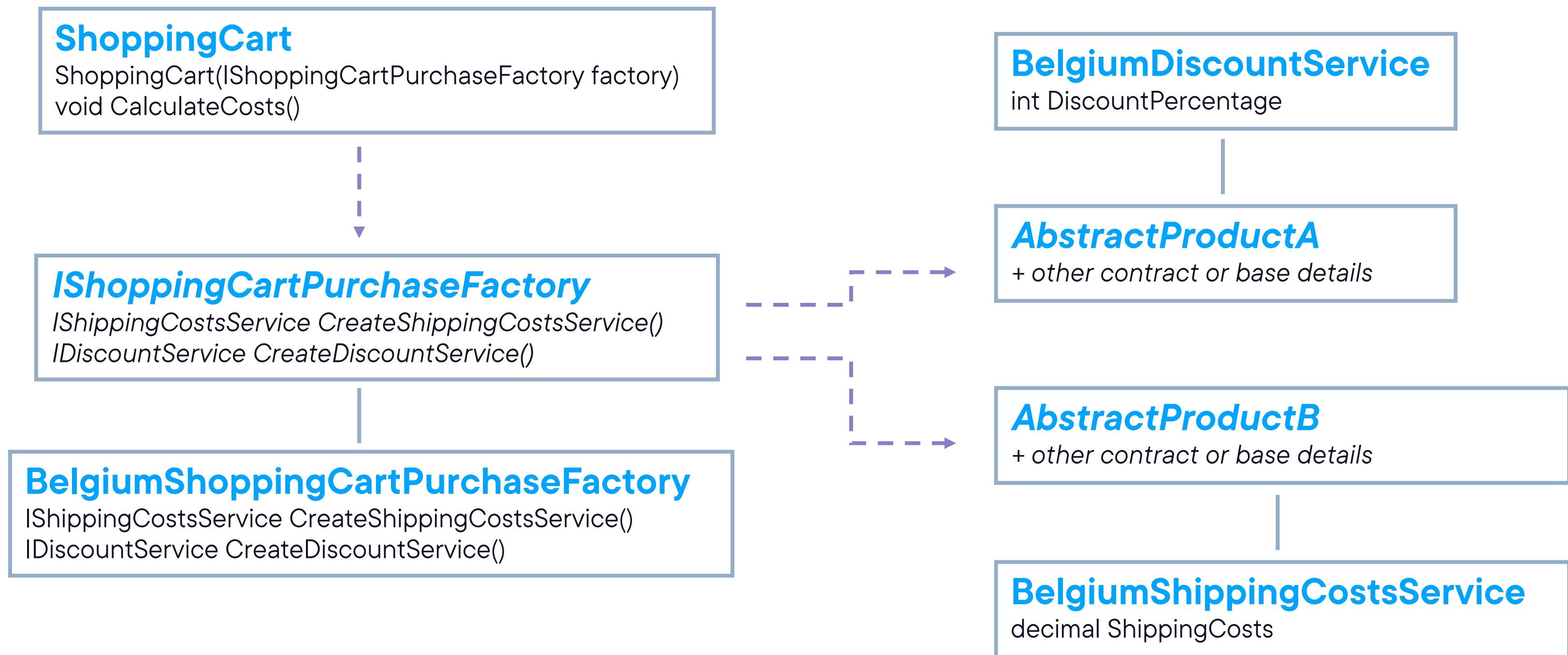
**AbstractProduct declares
an interface for a type of
product object**



Abstract Factory Pattern Structure



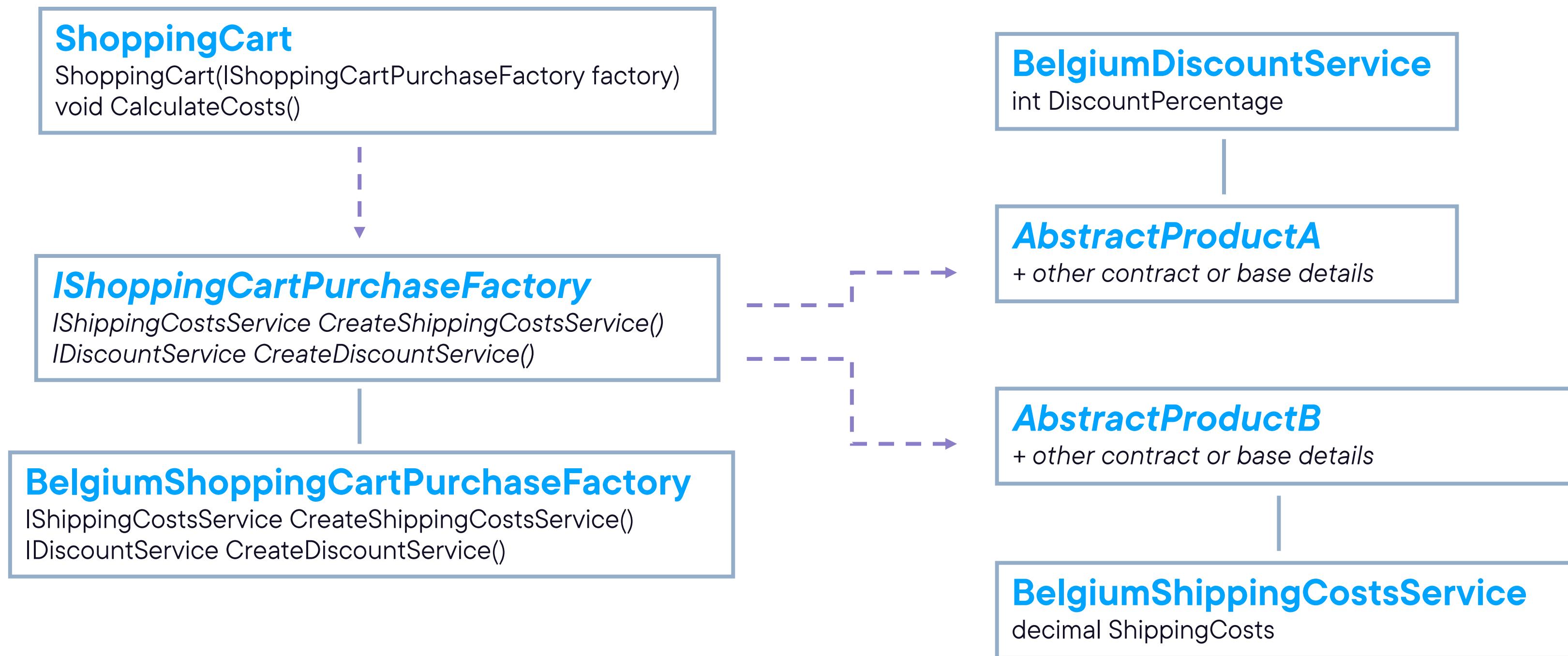
Abstract Factory Pattern Structure



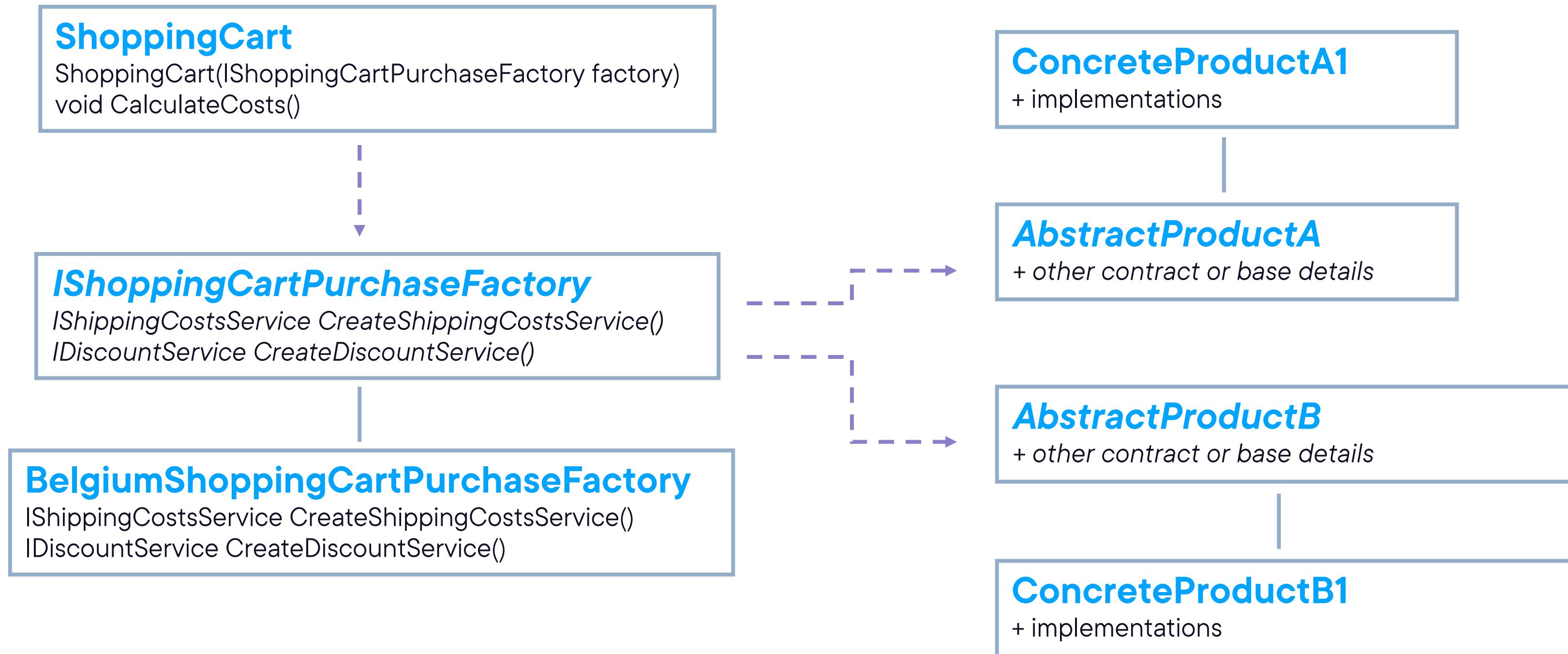
**ConcreteProduct defines
the product that has to be
created by a corresponding
factory, and implements the
AbstractProduct interface**



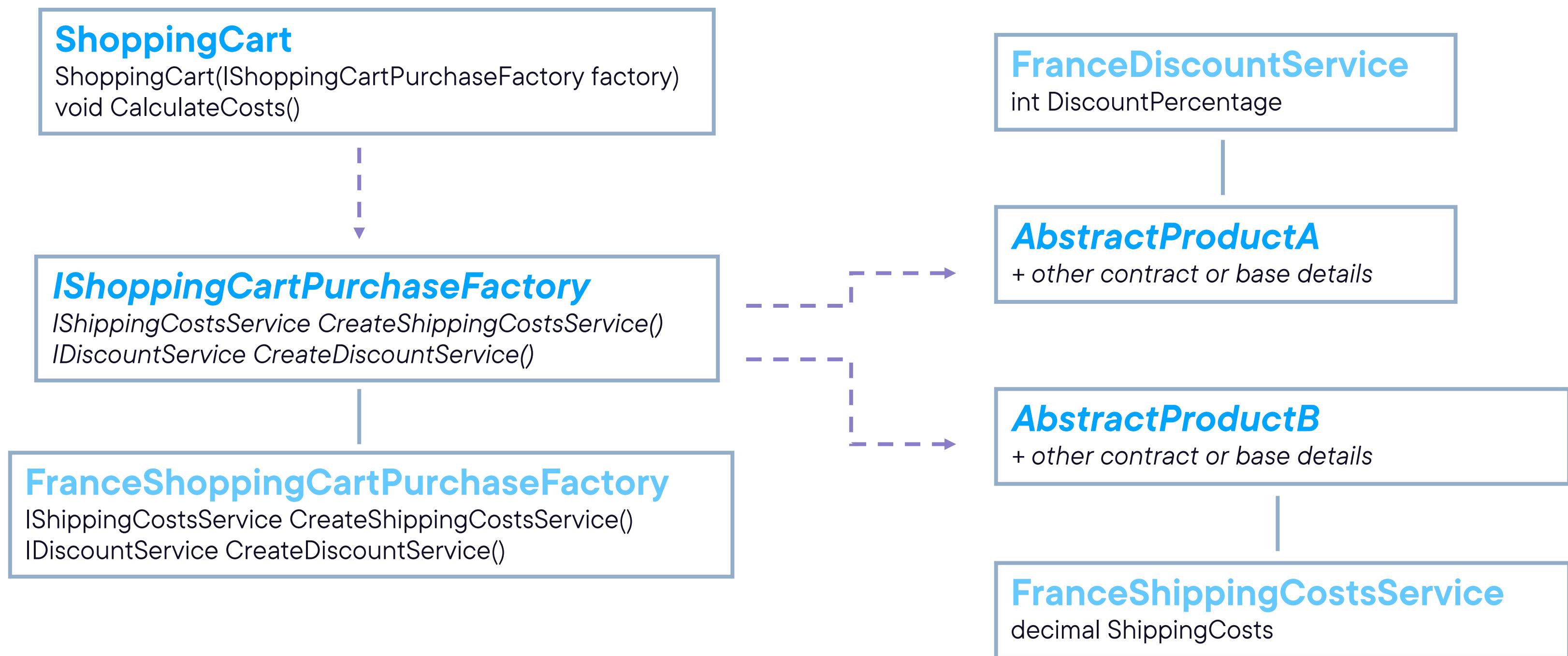
Abstract Factory Pattern Structure



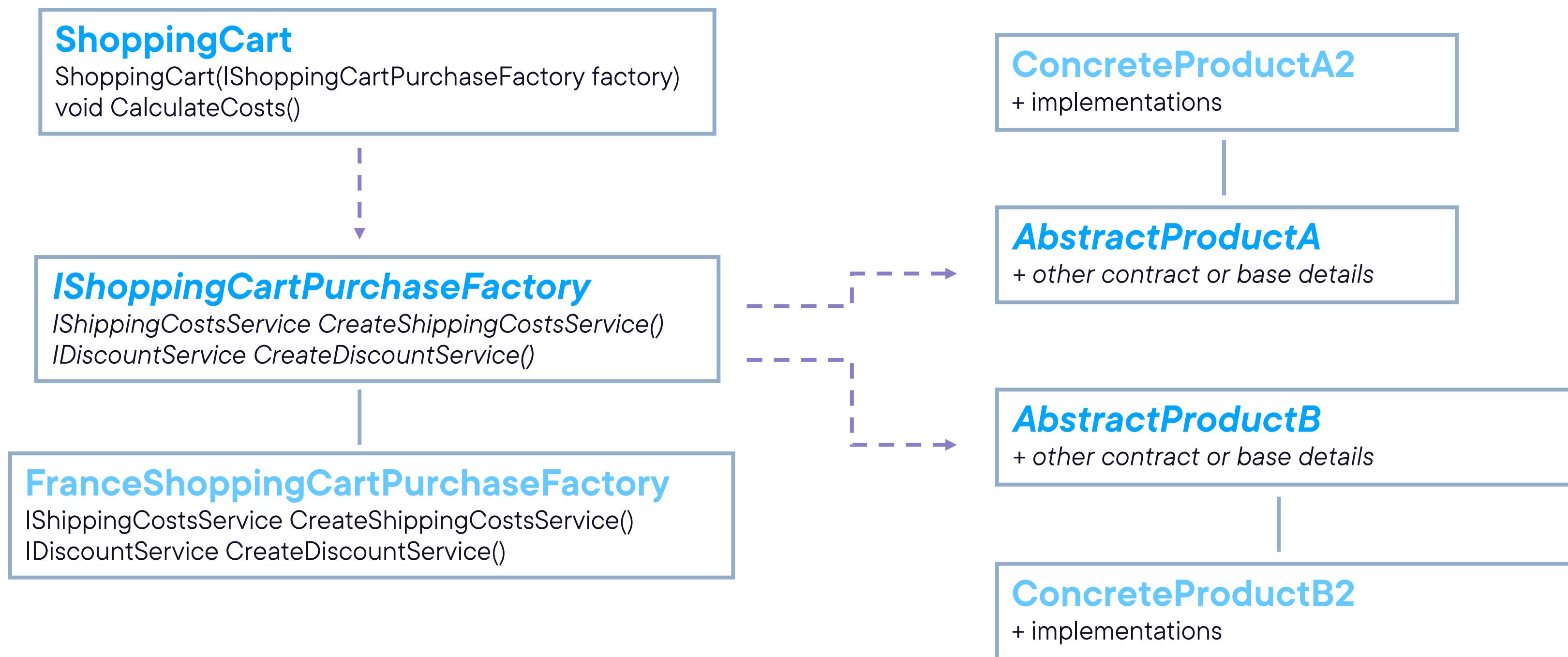
Abstract Factory Pattern Structure



Abstract Factory Pattern Structure



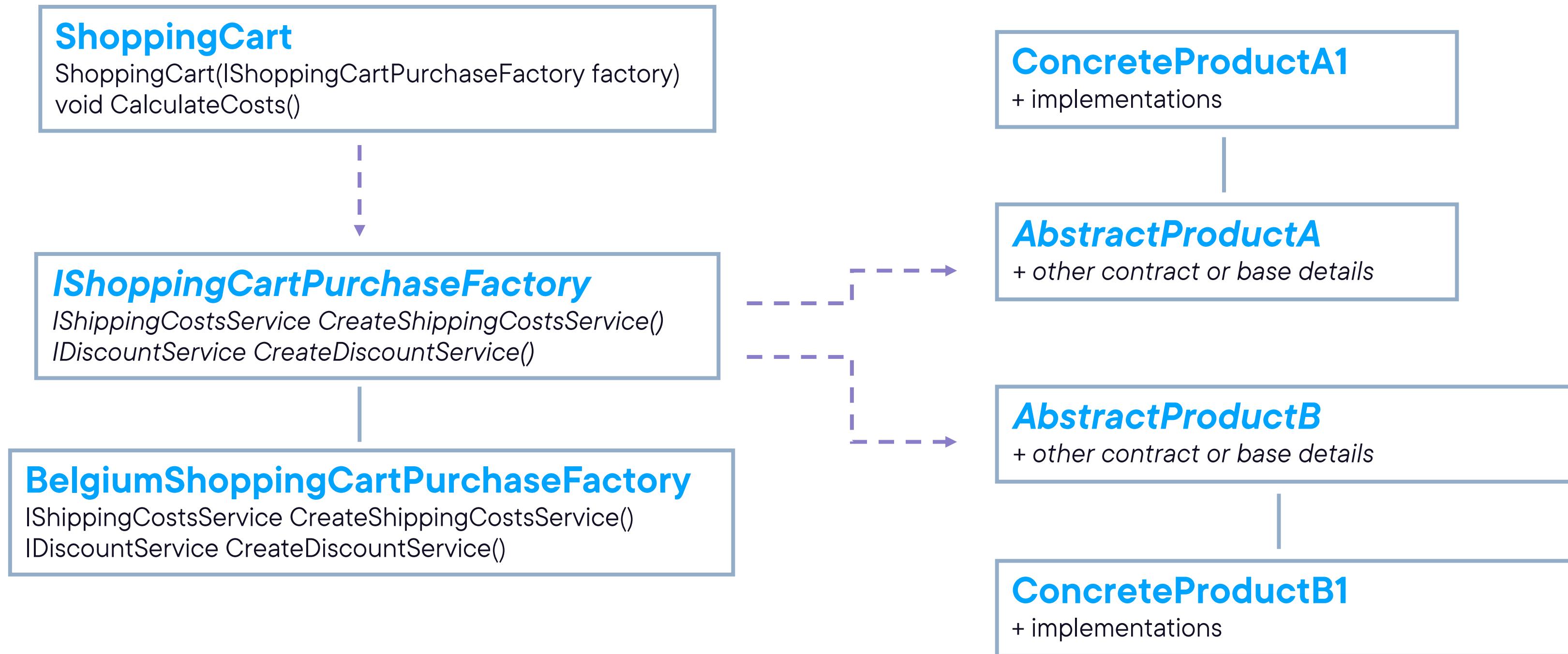
Abstract Factory Pattern Structure



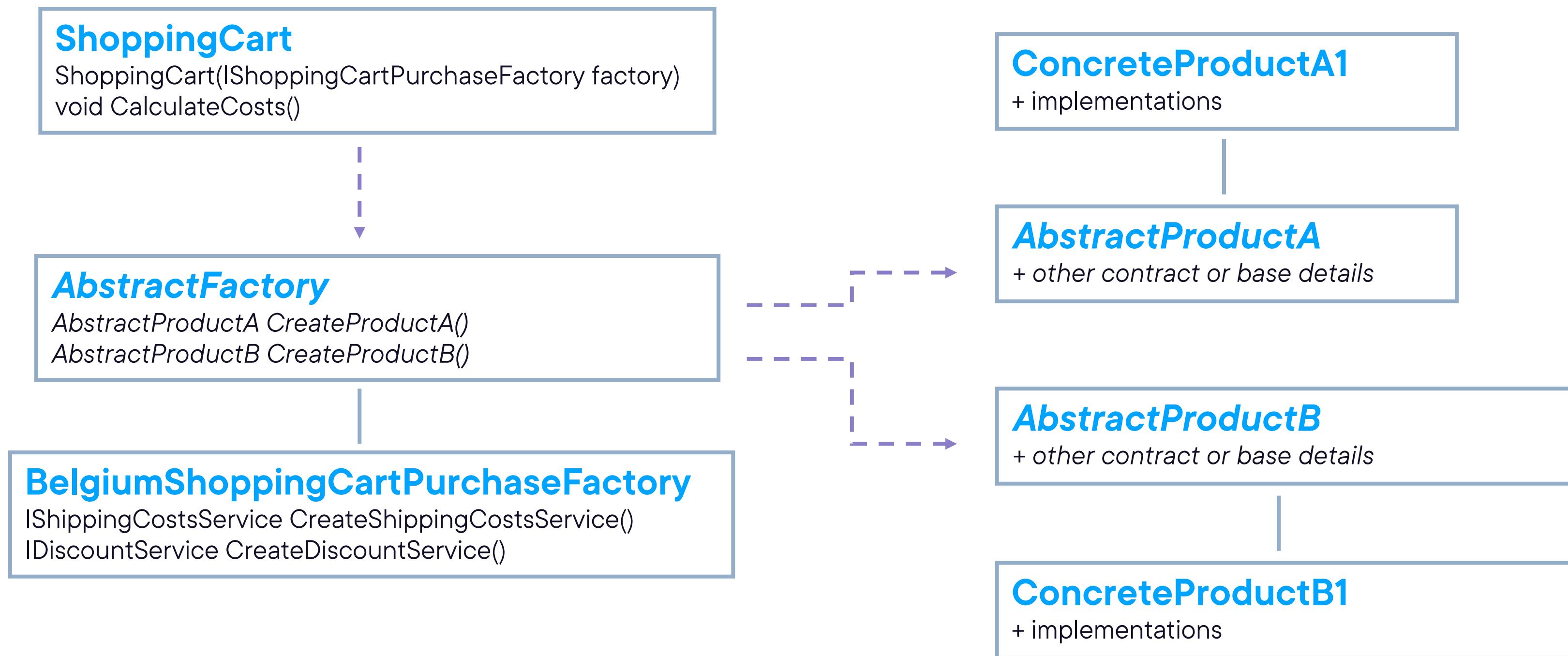
**AbstractFactory declares
an interface for operations
that create
AbstractProduct objects**



Abstract Factory Pattern Structure



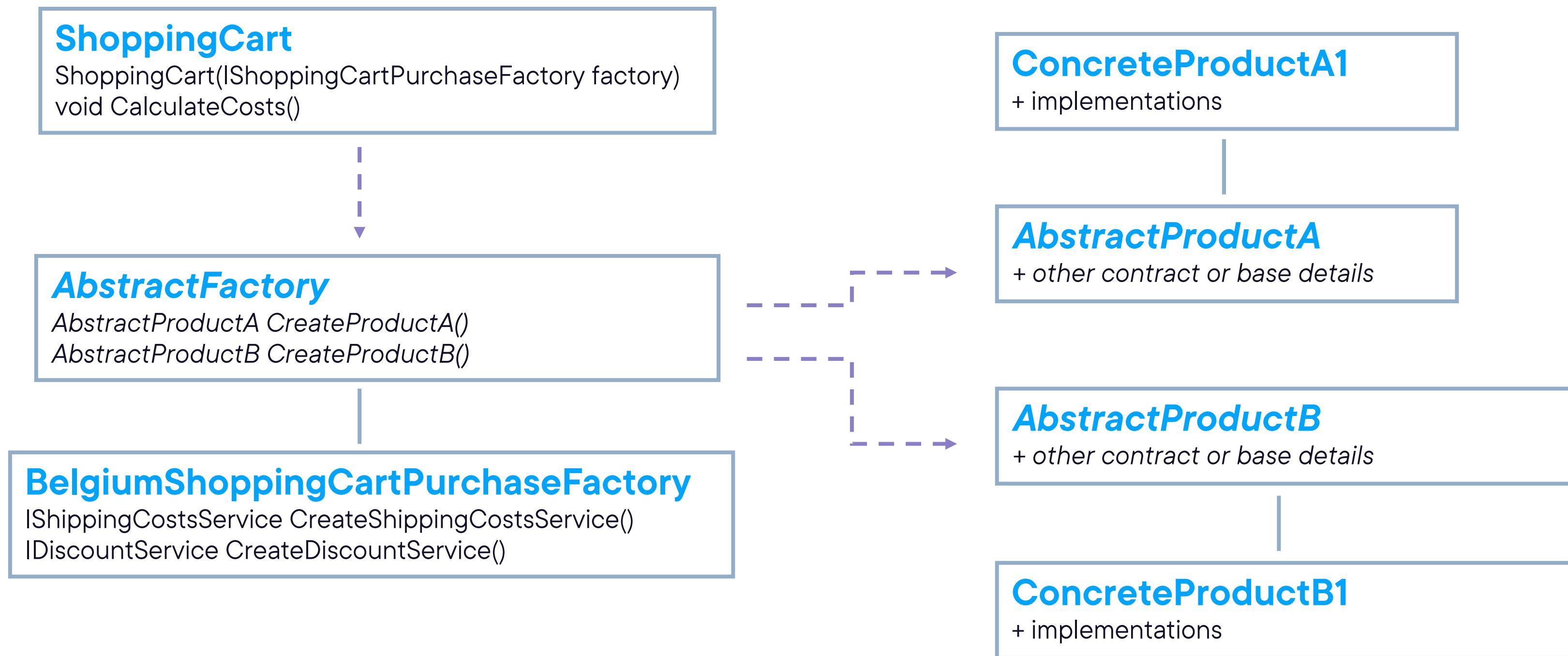
Abstract Factory Pattern Structure



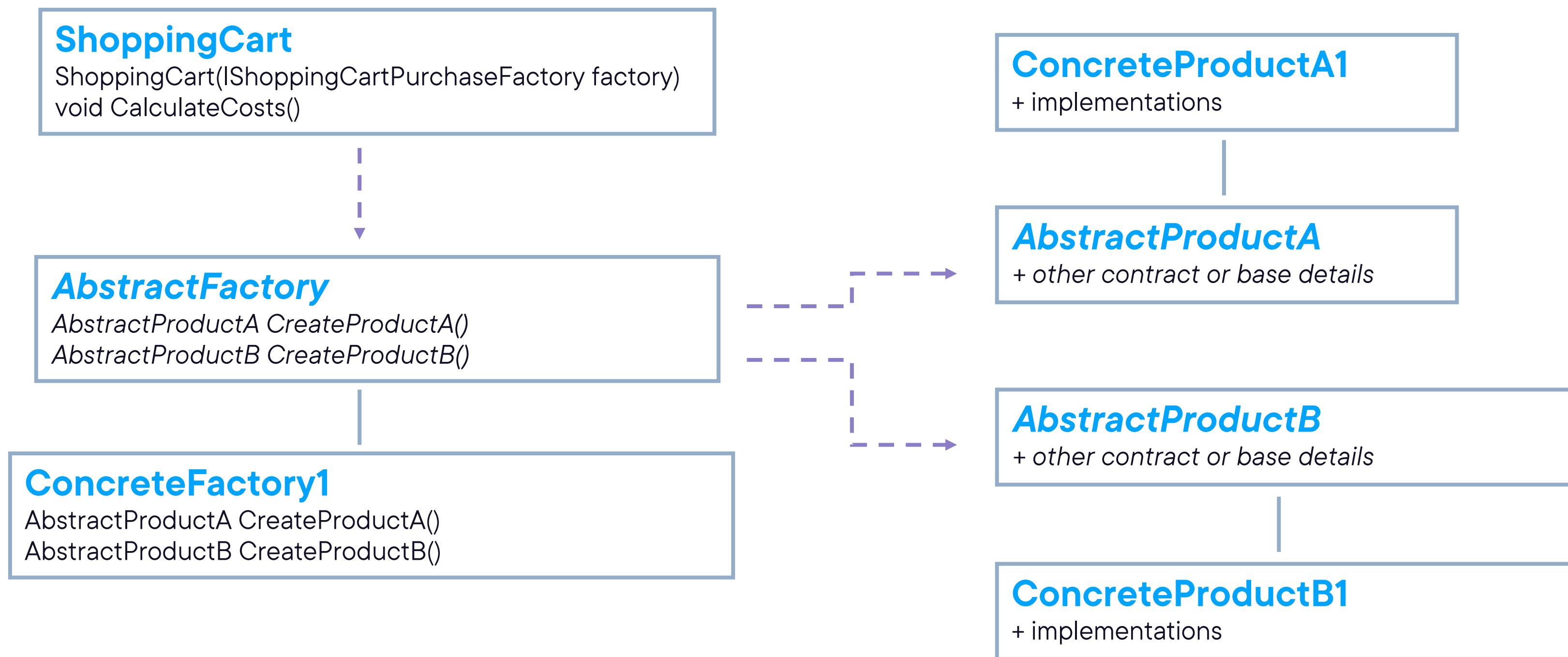
**ConcreteFactory
implements the operations
to create ConcreteProduct
objects**



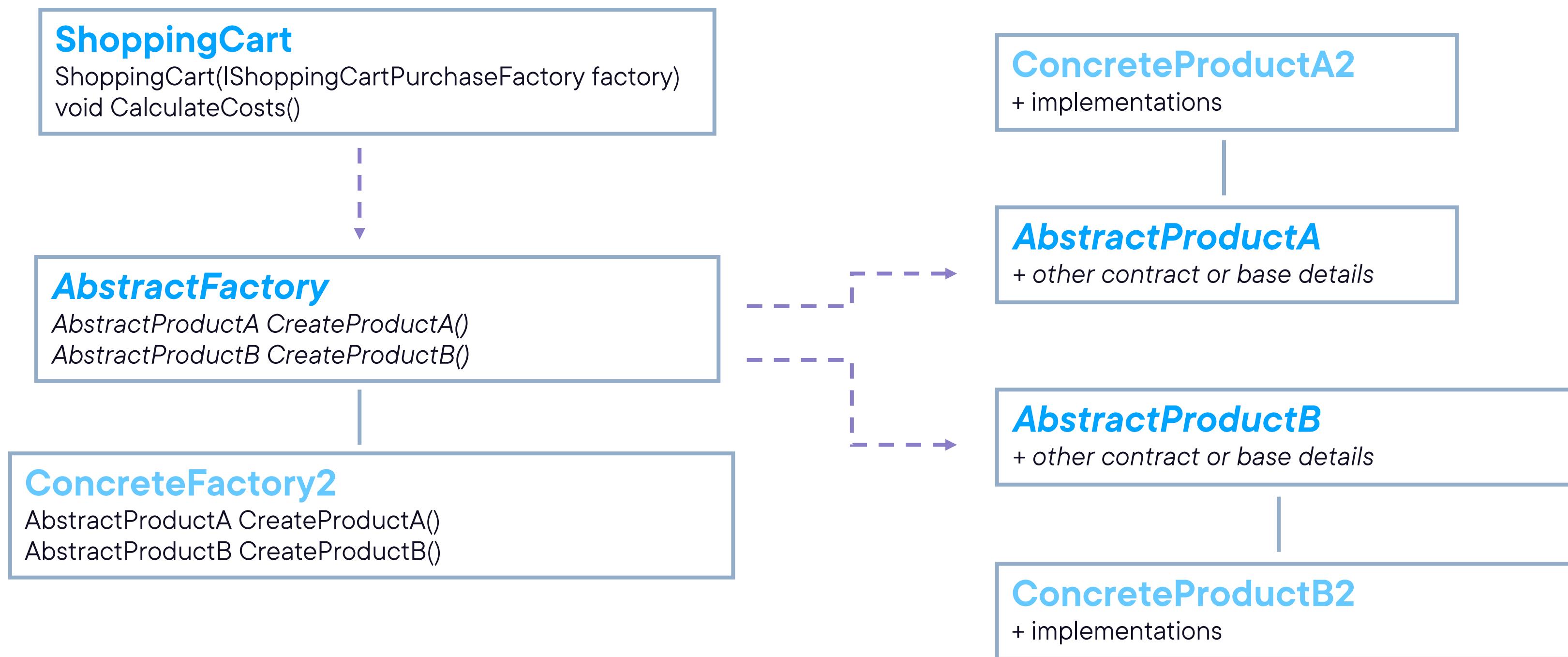
Abstract Factory Pattern Structure



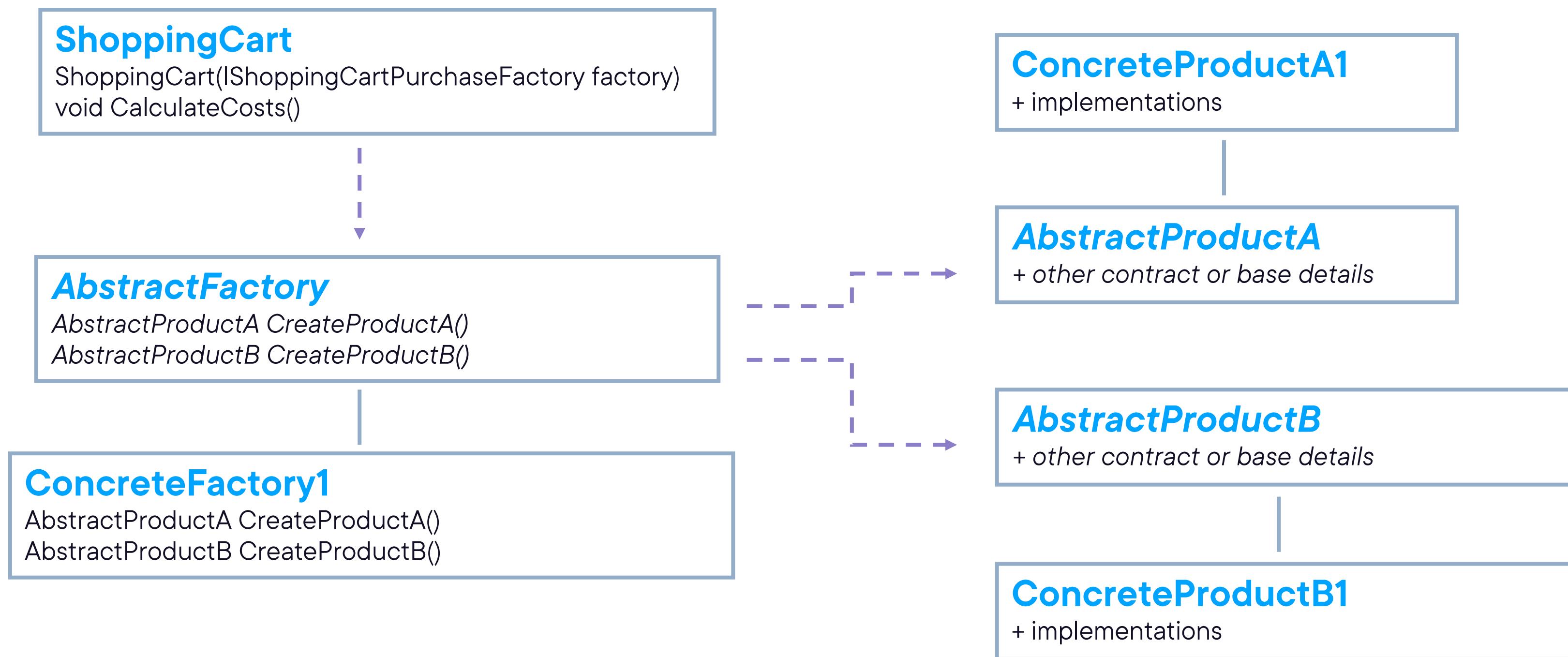
Abstract Factory Pattern Structure



Abstract Factory Pattern Structure



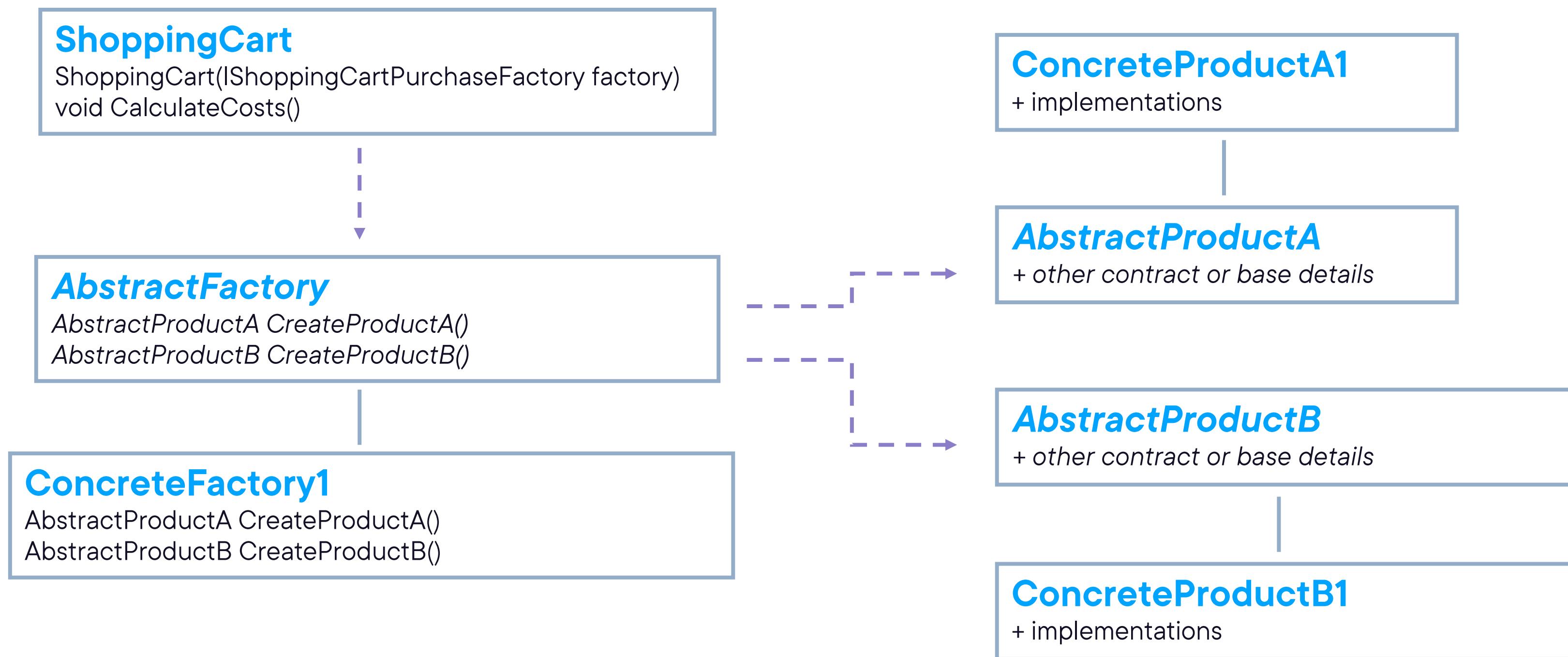
Abstract Factory Pattern Structure



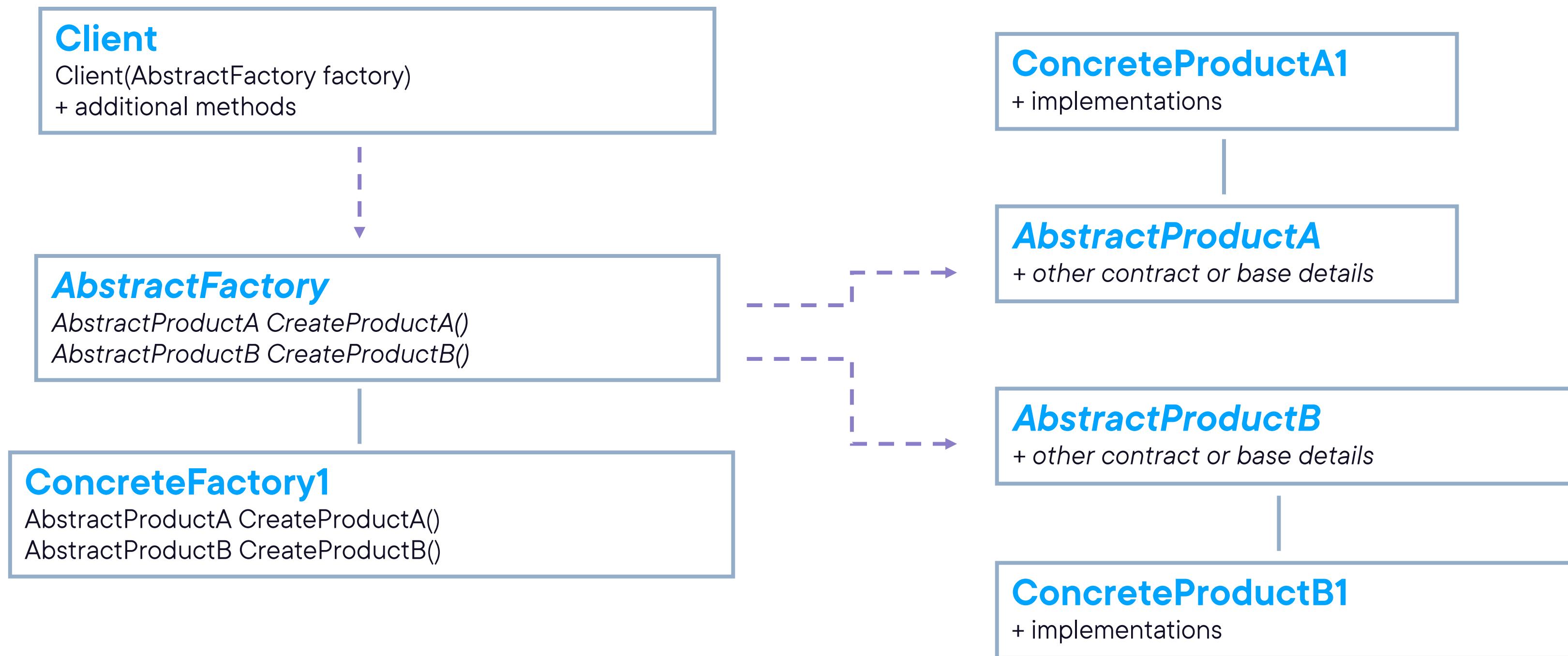
**Client uses only interfaces
declared by
AbstractFactory and
AbstractProduct**



Abstract Factory Pattern Structure



Abstract Factory Pattern Structure



Demo



Implementing the abstract factory pattern



Use Cases for the Abstract Factory Pattern



When a system should be independent of how its products are created, composed and represented



When you want to provide a class library of products and you only want to reveal their interfaces, not their implementations



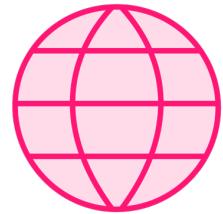
When a system should be configured with one of multiple families of products



When a family of related product objects is designed to be used together and you want to enforce this constraint



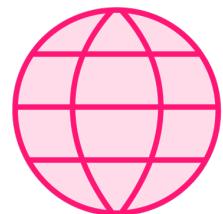
Use Cases for the Abstract Factory Pattern



Supporting multiple languages



Converting documents to different formats



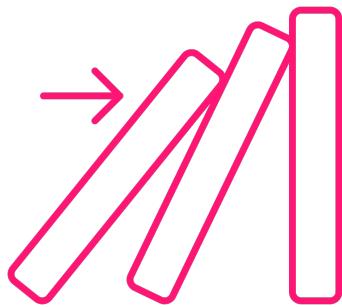
Abstracting away your database access layer



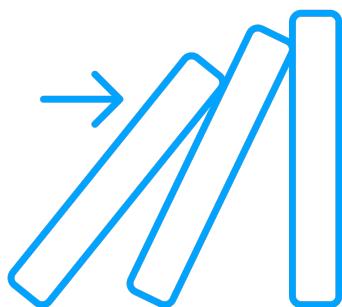
Supporting different application themes or styles



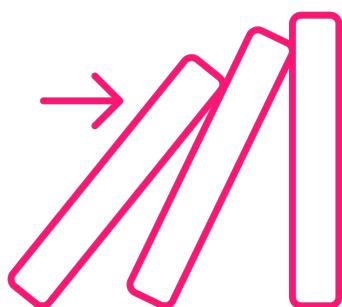
Pattern Consequences



It isolates concrete classes, because it encapsulates the responsibility and the process of creating product objects



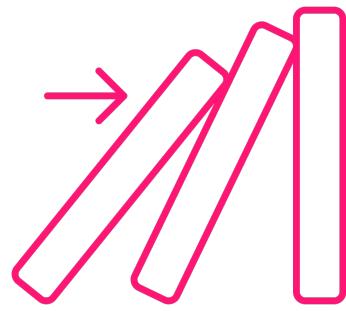
New products can easily be introduced without breaking client code: **open/closed principle**



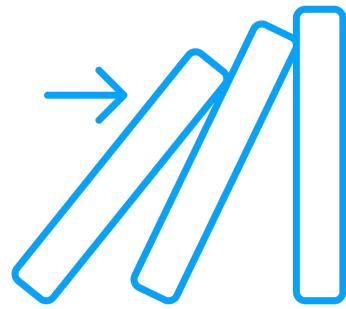
Code to create products is contained in one place: **single responsibility principle**



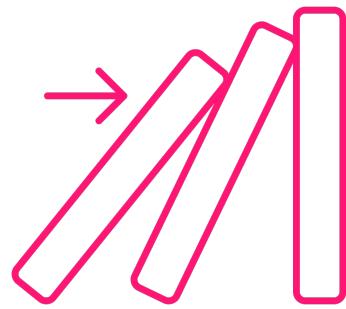
Pattern Consequences



It makes exchanging product families easy



It promotes consistency among products



Supporting new kinds of products is rather difficult



Comparing Abstract Factory to Factory Method

Factory method

vs

Abstract factory

Exposes an interface with a method on it, the factory method, to create an object of a certain type

Produces one product

Creates objects through inheritance

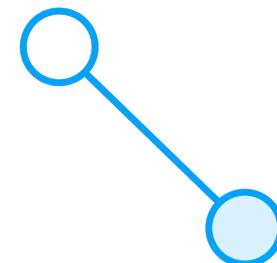
Exposes an interface to create related objects: families of objects

Produces families of products

Creates families of objects through composition

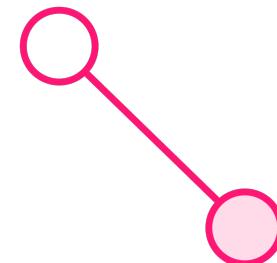


Related Patterns



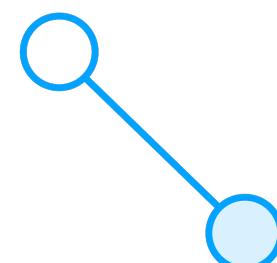
Factory method

Abstract factory can be implemented using factory methods



Prototype

Abstract factory can be implemented using prototypes



Singleton

A concrete factory is often implemented as a singleton



Summary



Intent of the abstract factory pattern:

- To provide an interface for creating families of related or dependent objects without specifying their concrete classes

Clients are isolated from implementation classes: decoupling



Summary



Implementation:

- Return ConcreteProduct via the underlying interface



Up Next:

Creational Pattern: Builder

