C# Design Patterns: Factory and Abstract Factory

FACTORY PATTERN

Filip Ekberg
PRINCIPAL CONSULTANT & CEO

@fekberg fekberg.com

Course Overview

Understanding the characteristics of the factory patterns

Implement the factory patterns in C#

Understanding the benefits and tradeoffs

What Is a Factory?

“A factory is an object for creating objects”

Wikipedia

https://en.wikipedia.org/wiki/Factory_(object-oriented_programming)

Factory Pattern Flavors

Simple Factory

Factory Method

Abstract Factory

“Factory Pattern” is a programming idiom

Factory Pattern Characteristics

Client

Asks for a created product

Creator

Facilitates a creation

Product

The product of the creation

Factory Pattern Characteristics

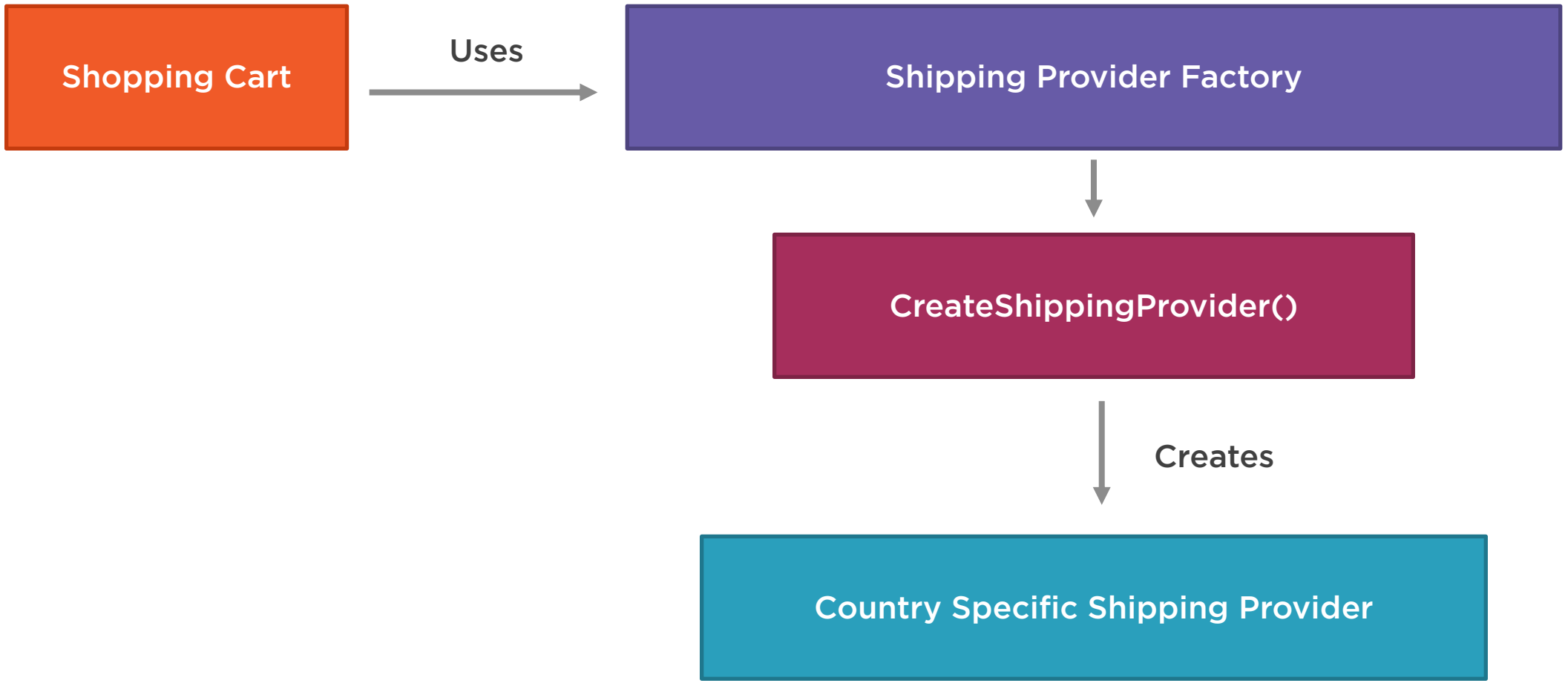
Client	Creator	Product
Shopping Cart	ShippingProviderFactory	ShippingProvider Instance

The client **no longer needs to know** how to create an object or exactly what flavor of that class it will use

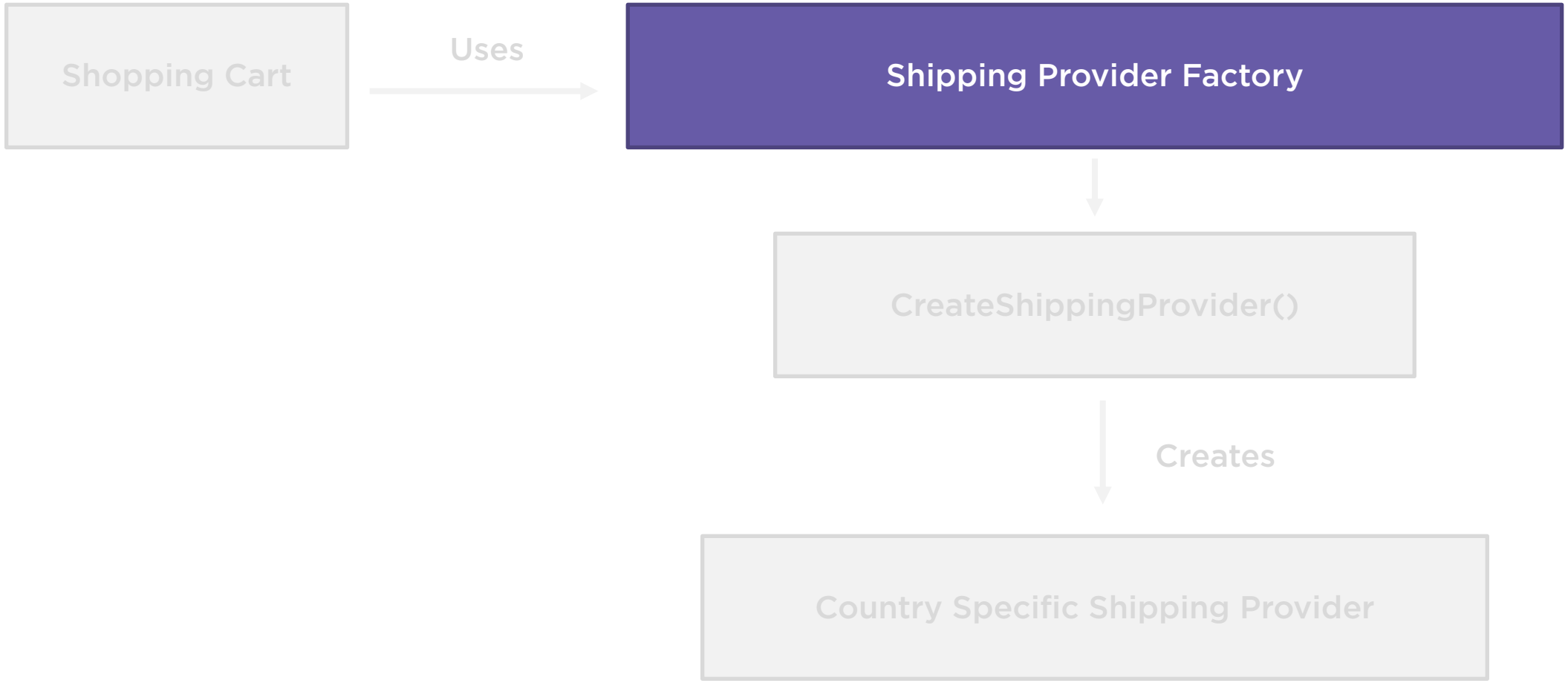
Simple Factory

Using factory method is a more extensible alternative

Example: Simple Factory



Example: Simple Factory



Demo

Factory Pattern: First Look

Introduce a factory to make code in the application more reusable

Factory Method Pattern

The factory pattern is introduced to allow for a flexible and extensible application

Example: Factory Method Pattern

```
public abstract class ShippingProviderFactory
{
    public abstract ShippingProvider CreateShippingProvider(string country);

    public ShippingProvider GetShippingProvider(string country)
    {
        var provider = CreateShippingProvider(country);

        return provider;
    }
}
```

Example: Factory Method Pattern

```
public abstract class ShippingProviderFactory
{
    public abstract ShippingProvider CreateShippingProvider(string country);

    public ShippingProvider GetShippingProvider(string country)
    {
        var provider = CreateShippingProvider(country);

        return provider;
    }
}
```

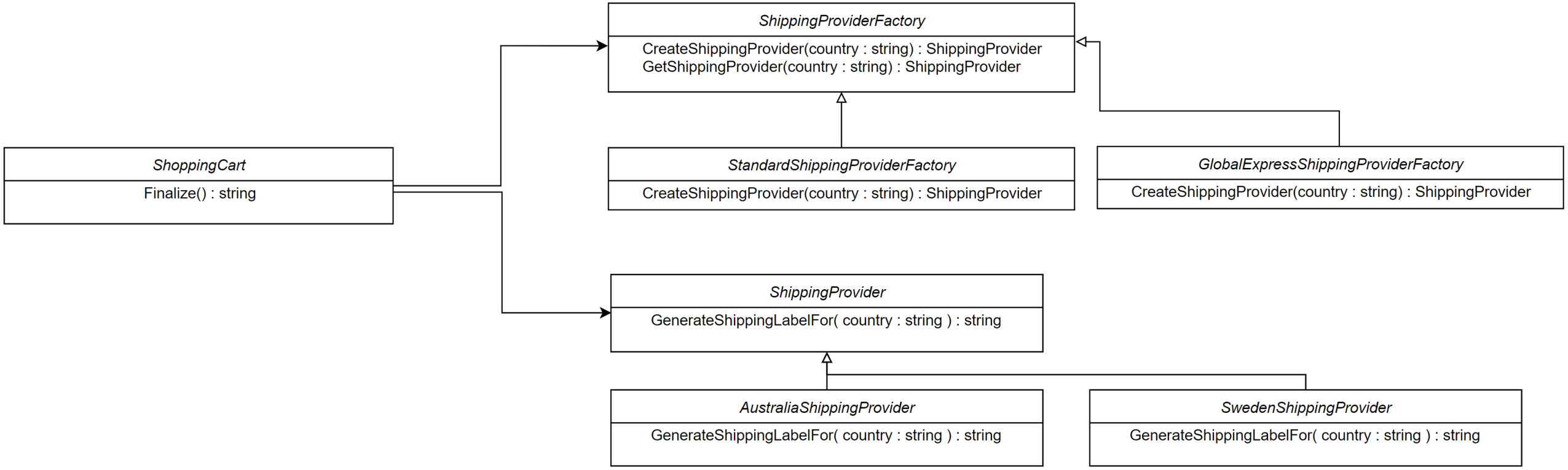
Example: Factory Method Pattern

```
public abstract class ShippingProviderFactory
{
    public abstract ShippingProvider CreateShippingProvider(string country);

    public ShippingProvider GetShippingProvider(string country)
    {
        var provider = CreateShippingProvider(country);

        return provider;
    }
}
```

Example: Factory Method Pattern

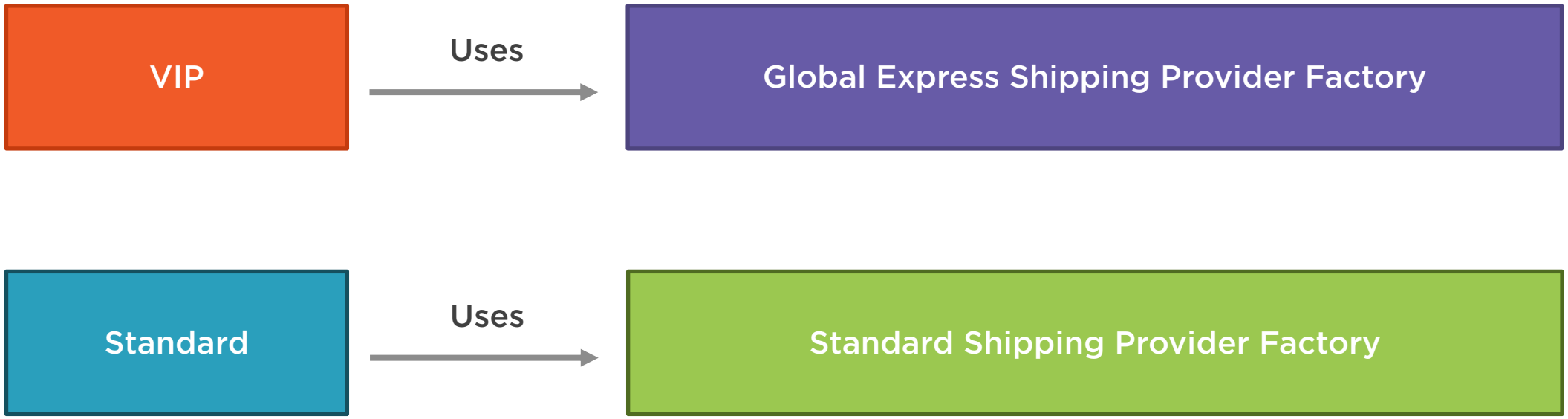


Extend the creator to override the default factory method

Demo

Example: Factory Method Pattern

VIP vs Standard User



A powerful pattern that makes your application code more reusable and extensible

In many cases the **simple factory** and **factory method** will be sufficient

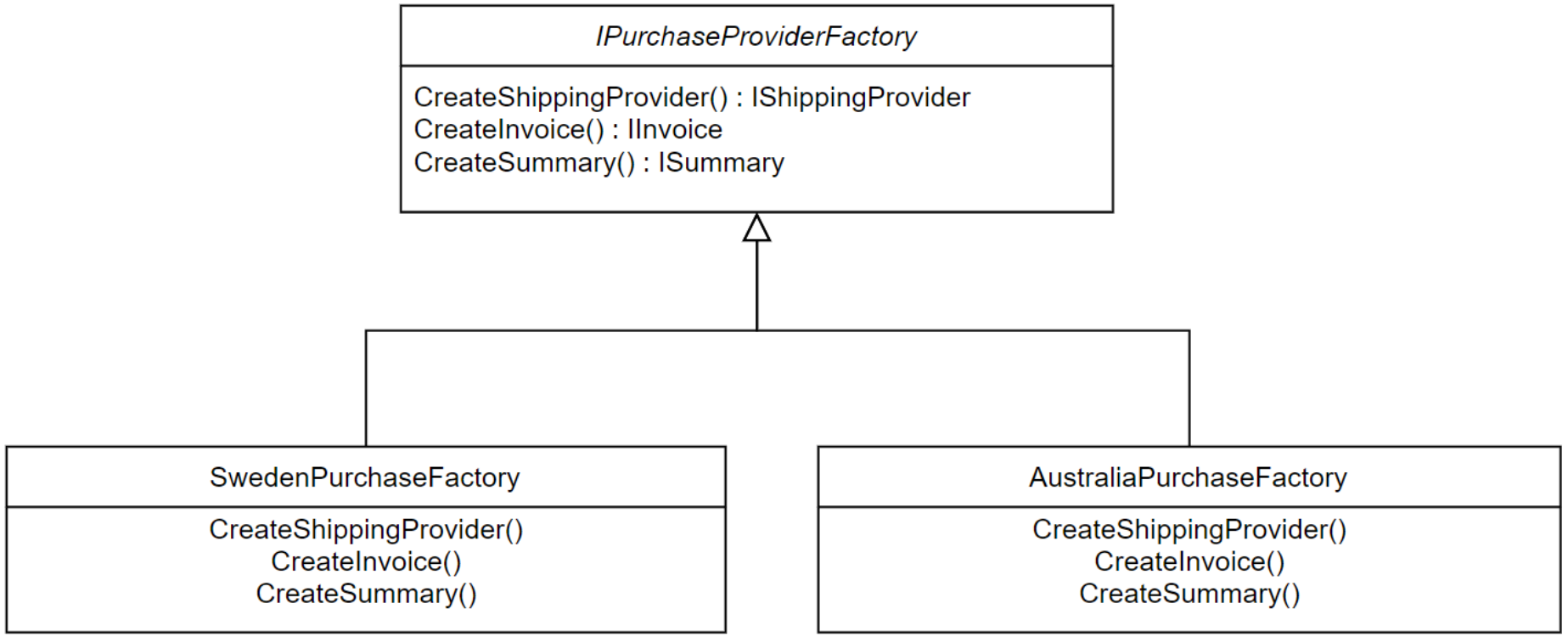
Abstract Factory Pattern

“The **abstract factory** pattern provides a way to **encapsulate** a **group of** individual **factories** that have a **common theme** without specifying their concrete classes”

Wikipedia

https://en.wikipedia.org/wiki/Abstract_factory_pattern

Example: Abstract Factory



Example: Using the Abstract Factory

```
var shippingProvider = purchaseProviderFactory.CreateShippingProvider(order);

var invoice = purchaseProviderFactory.CreateInvoice(order);
```

Example: Abstract Factory Pattern

```
PurchaseProviderFactory purchaseProviderFactory;

if (order.Sender.Country == "Australia")
{
    purchaseProviderFactory = new AustraliaPurchaseProviderFactory();
}

else if (order.Sender.Country == "Sweden")
{
    purchaseProviderFactory = new SwedenPurchaseProviderFactory();
}

var cart = new ShoppingCart(purchaseProviderFactory);
```

Demo

Example: Abstract Factory Pattern

Demo

Example: Factory Pattern in Testing

Extract creation of mocked, faked or commonly used instances in tests

Demo

Example: Adding a Factory Provider

A factory of factories

Summary

- Separates the client from the creation
- Less duplication of code
- Introduce subclasses and concrete implementations to add functionality
- Very common when writing tests
- Code becomes easier to maintain and navigate
- Factory Method and Abstract Factory are very common patterns

Thank You!