"Factory"

Factory

- confusion around many "factories"
- hiding/abstracting object creation
- ideally no `new()` as all constructors should be private...
- ...and objects created only from such factories

(Static) "factory method"

- confusing term for (typically **static**) helper methods creating objects

```
public class TaxPolicy
{
   public static TaxPolicy CreateForeignPolicy(string country)
   {
      // dodatkowa logika tworzenia obiektu
      // return new TaxPolicy(...)
   }
}
```

(Static) "factory method"

- confusing term for (typically **`static`**) helper methods creating objects

```
public class TaxPolicy
{
   public static TaxPolicy CreateForeignPolicy(string country)
   {
      // dodatkowa logika tworzenia obiektu
      // return new TaxPolicy(...)
   }
}
```

- kind of "named constructor"/helper

(Static) "factory method"

- confusing term for (typically **static**) helper methods creating objects

```
public class TaxPolicy
{
   public static TaxPolicy CreateForeignPolicy(string country)
   {
      // dodatkowa logika tworzenia obiektu
      // return new TaxPolicy(...)
   }
}
```

- kind of "named constructor"/helper
- better name than just a constructor, with some parameters

```
new TaxPolicy("PL") vs TaxPolicyFactory.CreateForeignPolicy("PL")
new TaxPolicy(810317..., true) vs TaxPolicyFactory.CreatePersonal(...)
```

(Static) "factory method"

- confusing term for (typically **`static`**) helper methods creating objects

```
public class TaxPolicy
{
   public static TaxPolicy CreateForeignPolicy(string country)
   {
      // dodatkowa logika tworzenia obiektu
      // return new TaxPolicy(...)
   }
}
```

- kind of "named constructor"/helper
- better name than just a constructor, with some parameters

```
new TaxPolicy("PL") vs TaxPolicyFactory.CreateForeignPolicy("PL")
new TaxPolicy(810317..., true) vs TaxPolicyFactory.CreatePersonal(...)
```

- allows to not create object, like renting

(Static) "factory method"

- confusing term for (typically **static**) helper methods creating objects

```
public class TaxPolicy
{
   public static TaxPolicy CreateForeignPolicy(string country)
   {
      // dodatkowa logika tworzenia obiektu
      // return new TaxPolicy(...)
   }
}
```

- kind of "named constructor"/helper
- better name than just a constructor, with some parameters

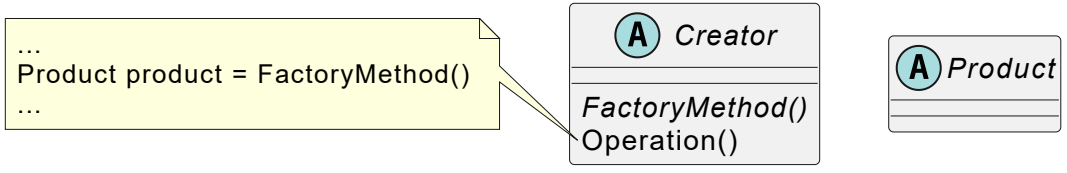
```
new TaxPolicy("PL") vs TaxPolicyFactory.CreateForeignPolicy("PL")
new TaxPolicy(810317..., true) vs TaxPolicyFactory.CreatePersonal(...)
```

- allows to not create object, like renting
- it can be called "factory method" because it creates objects, but it is not a Factory Method pattern - no inheritance/abstraction

Factory Method

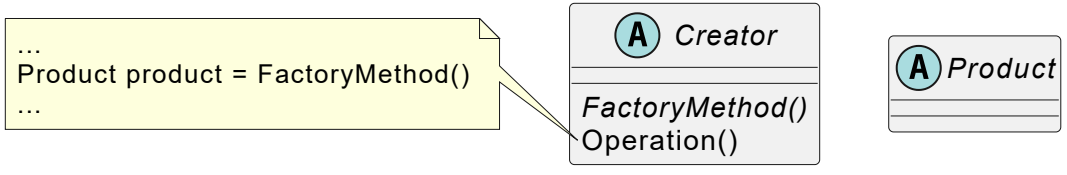
Factory Method

- GoF: "*Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.*" aka *Virtual Constructor*
- provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created



Factory Method

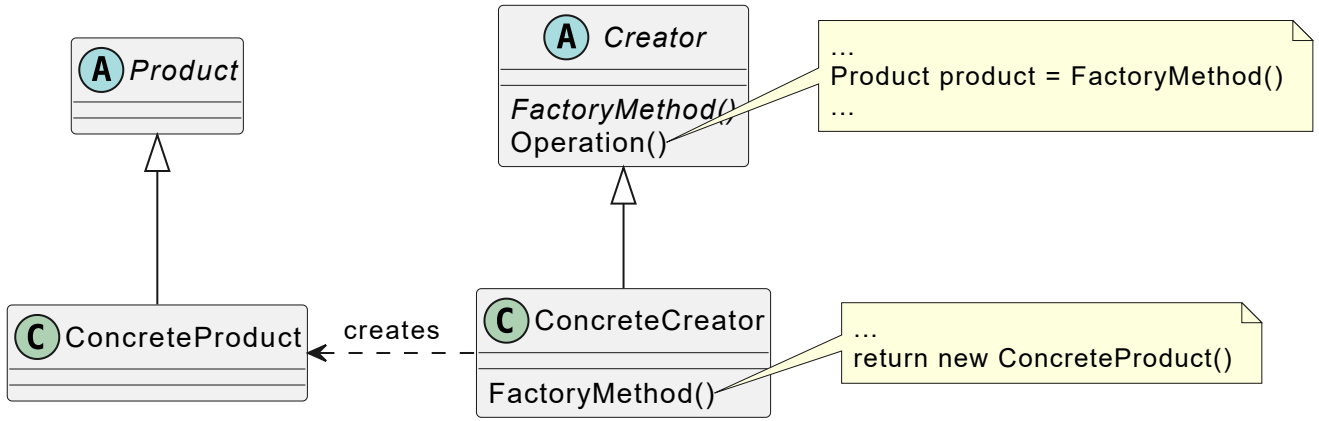
- GoF: "*Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.*" aka *Virtual Constructor*
- provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created



- we encapsulate the exact type being created by **a single** *factory method*

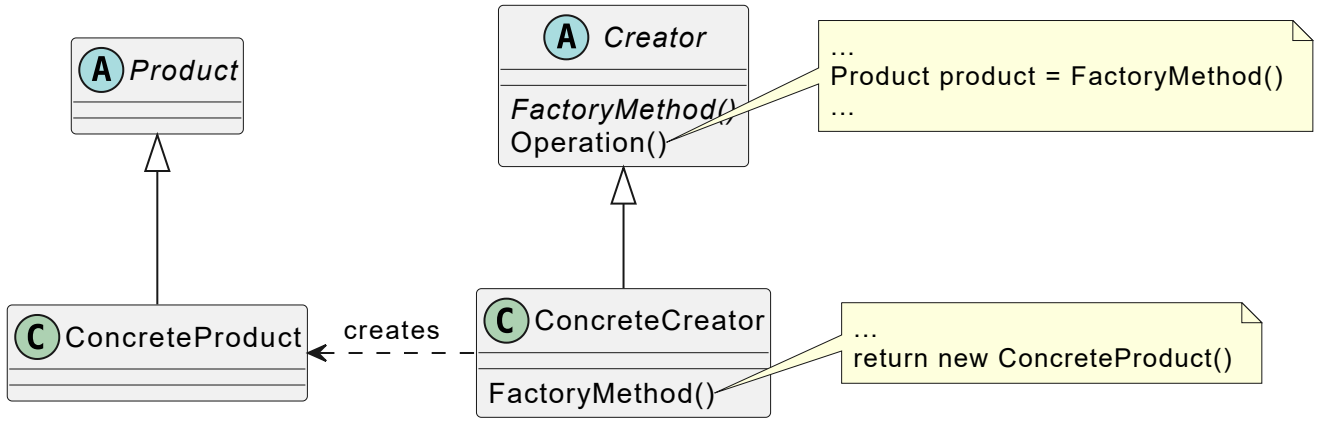
Factory Method

- GoF: "*Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.*" aka *Virtual Constructor*
- provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created



Factory Method

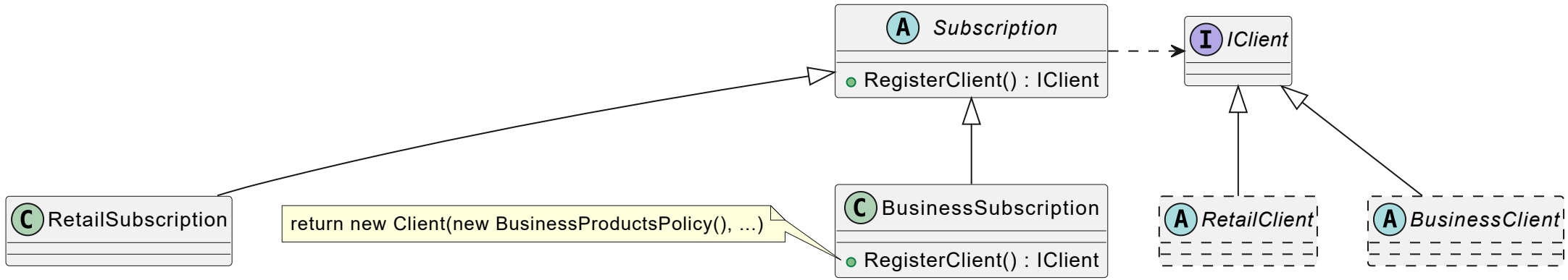
- GoF: "*Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.*" aka *Virtual Constructor*
- provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created



`Creator` is NOT a Factory itself! It just have a Factory Method inside! A... method!

Factory Method

USE WHEN we don't know in advance what exact subtype will be created - like **`Create/SignupCustomer() : ICustomer`** returns various types (retail/business)



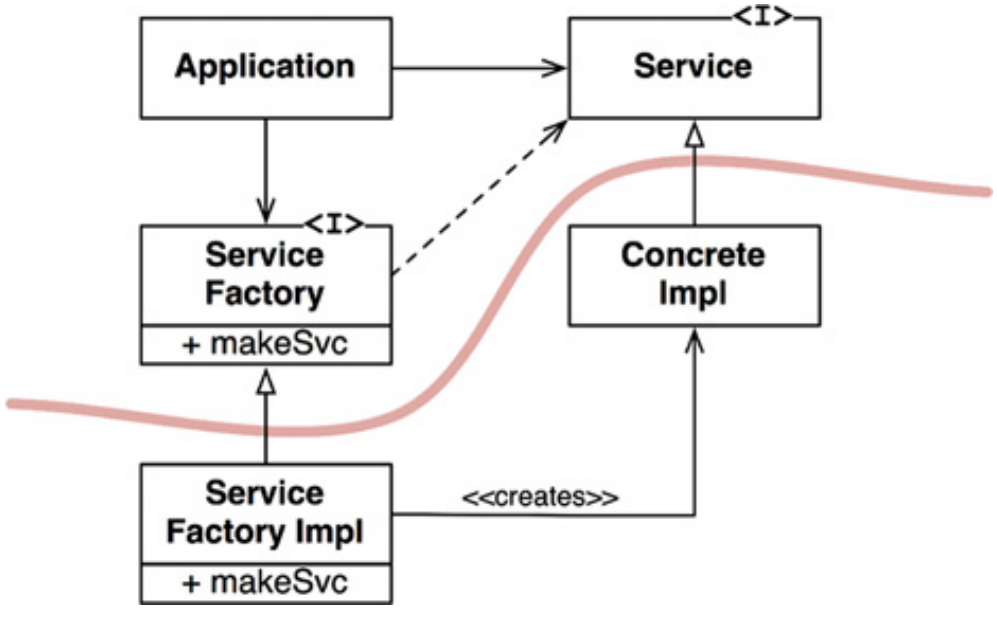
Factory Method

- subclassing is fine when the client has to subclass the Creator class **anyway**, but otherwise subclass the Creator class just to create a particular product seems counter-productive
- IOW - the best case scenario is when you're introducing the pattern into an existing hierarchy of creator classes
- USE WHEN we don't know in advance what exact subtype will be created
- beware of similarity to Abstract Factory

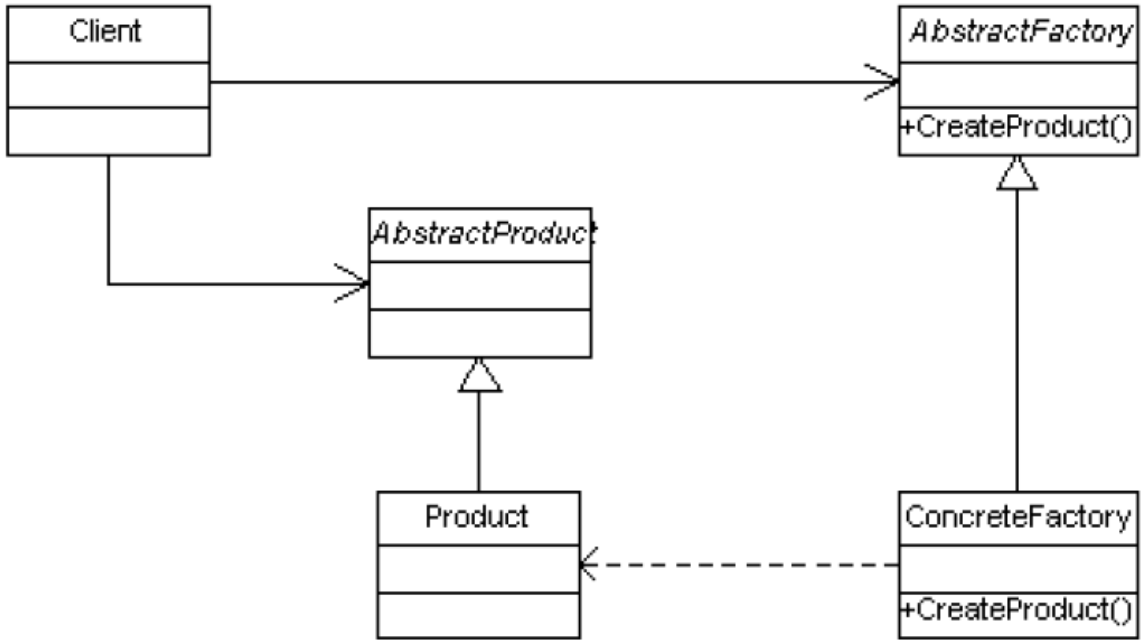
Factory Method

Various **`Create...`** methods in DDD's Factory are Factory Methods

```
async Task Handle(AddCartItem command,
   IPromotionsFactory promotionsFactory,
   ...)
{
 ...
 var pricing = promotionsFactory.CreatePricingStrategy
```

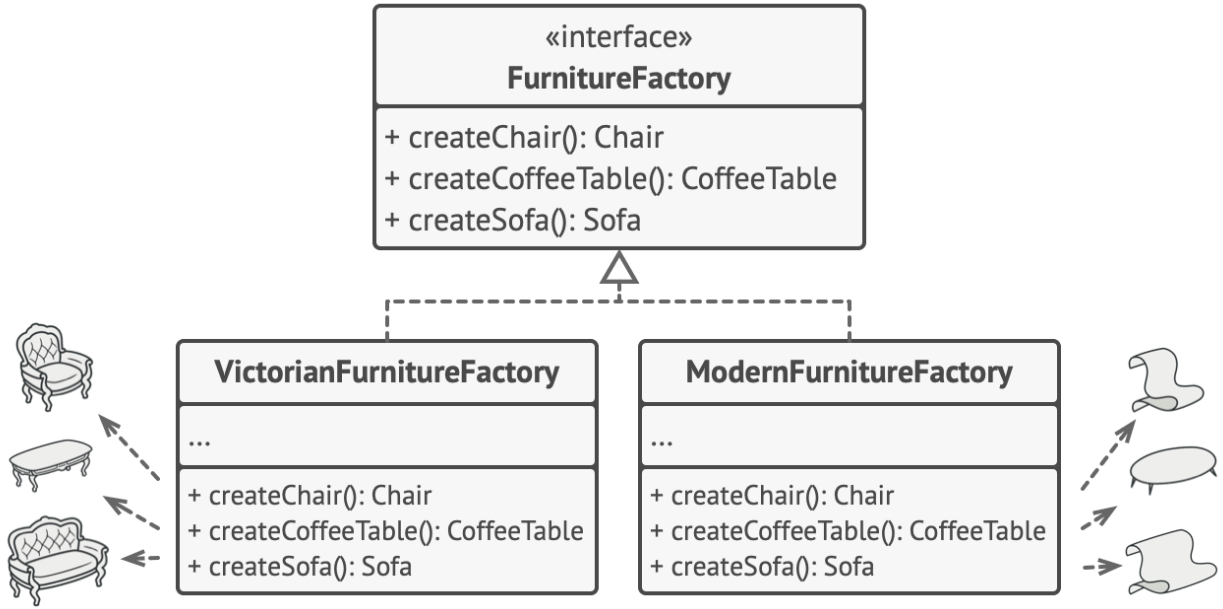


Factory Method vs Abstract Factory



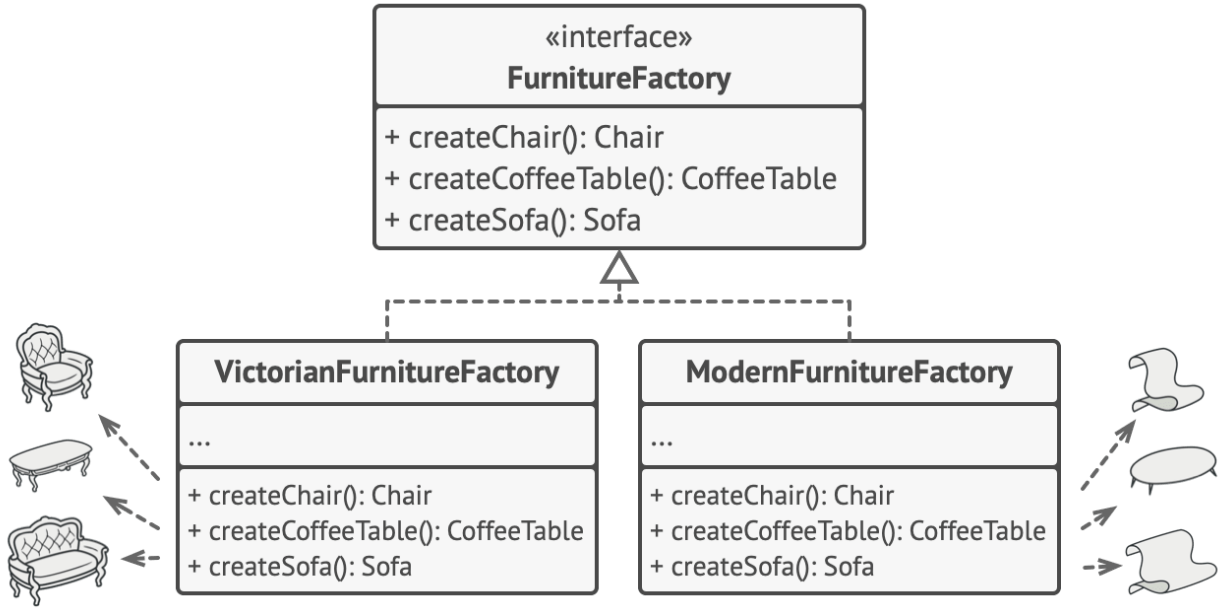
Abstract Factory

- a creational design pattern that lets you produce families of related objects without specifying their concrete classes
- in general, it is a set of Factory Methods



Abstract Factory

- a creational design pattern that lets you produce families of related objects without specifying their concrete classes
- in general, it is a set of Factory Methods



- while Factory Method abstracts creation of a single type, Abstract Factory pattern abstracts multiple (families of types)

Abstract Factory

```
public interface OrderPoliciesFactory
{
   public TaxPolicy createTaxPolicy();
   public RebatePolicy createRebatePolicy();
}
```

Factory Method vs Abstract Factory

"the main difference between a "factory method" and an "abstract factory" is that the factory method is a method, and an abstract factory is an object"

Factory Method vs Abstract Factory

"the main difference between a "factory method" and an "abstract factory" is that the factory method is a method, and an abstract factory is an object"

Various **`Create...`** methods in DDD's Factory are Factory Methods, and those **`I...Factory`** are (degraded) Abstract Factories 😇😅

```
async Task Handle(AddCartItem command,
   IPromotionsFactory promotionsFactory,
   ...)
{
 ...
 var pricing = promotionsFactory.CreatePricingStrategy
```