

# Creational Pattern: Factory Method

---



**Kevin Dockx**

Architect

@KevinDockx <https://www.kevindockx.com>



Coming Up



**Describing the factory method pattern**

**Structure of the factory method pattern**

**Implementation**

- Real-life sample: shopping cart discount service



Coming Up



**Use cases for this pattern**

**Pattern consequences**

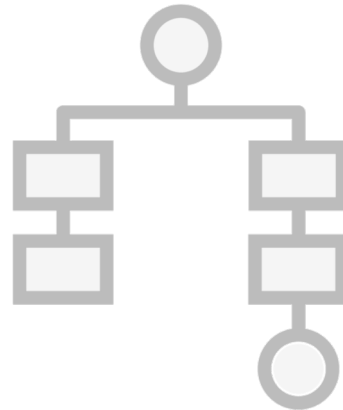
**Related patterns**



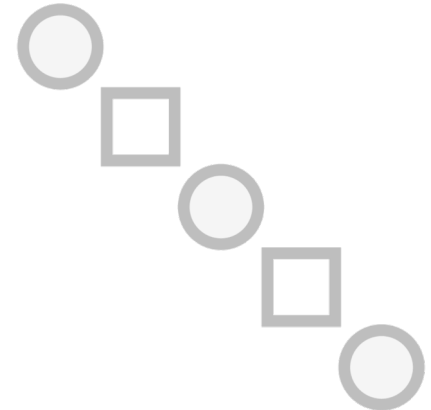
# Describing the Factory Method Pattern



**Creational**



**Structural**



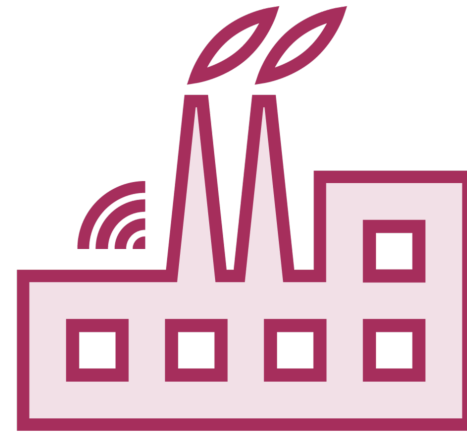
**Behavioral**



# Describing the Factory Method Pattern



**Factory method**



**Abstract factory**



# Factory Method

**The intent of the factory method pattern is to define an interface for creating an object, but to let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses.**



```
var codeDiscountService = new CodeDiscountService(Guid.NewGuid());  
var discount = codeDiscountService.DiscountPercentage;
```

```
// or
```

```
var countryDiscountService = new CountryDiscountService("BE");  
var discount = countryDiscountService.DiscountPercentage;
```

## Describing the Factory Method Pattern

```
var codeDiscountService = new CodeDiscountService(Guid.NewGuid());  
var discount = codeDiscountService.DiscountPercentage;  
  
// or  
  
var countryDiscountService = new CountryDiscountService("BE");  
var discount = countryDiscountService.DiscountPercentage;
```

## Describing the Factory Method Pattern



```
var codeDiscountService = new CodeDiscountService(Guid.NewGuid());  
var discount = codeDiscountService.DiscountPercentage;  
  
// or  
  
var countryDiscountService = new CountryDiscountService("BE");  
var discount = countryDiscountService.DiscountPercentage;
```

## Describing the Factory Method Pattern

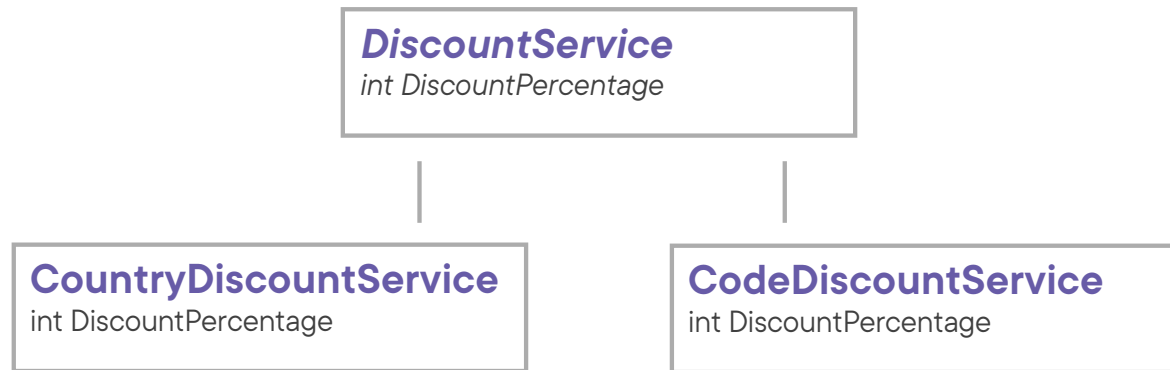
# Describing the Factory Method Pattern

**CountryDiscountService**  
int DiscountPercentage

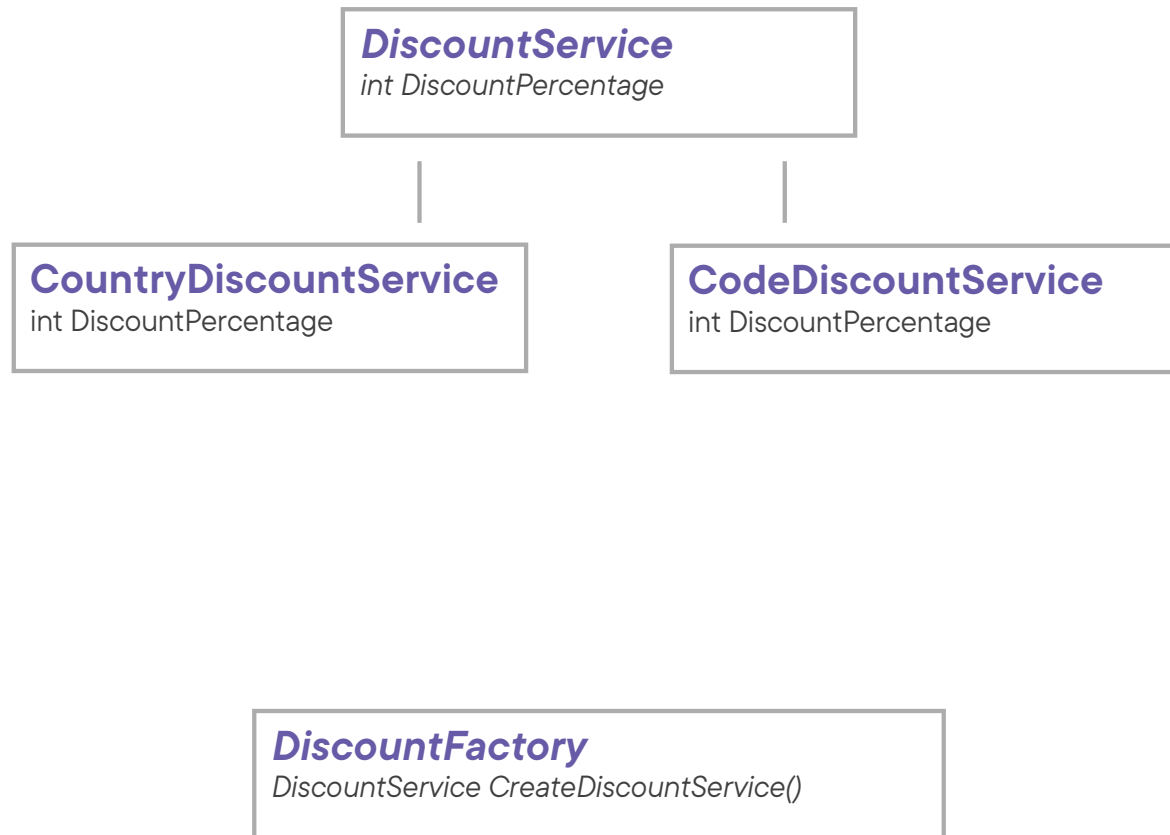
**CodeDiscountService**  
int DiscountPercentage



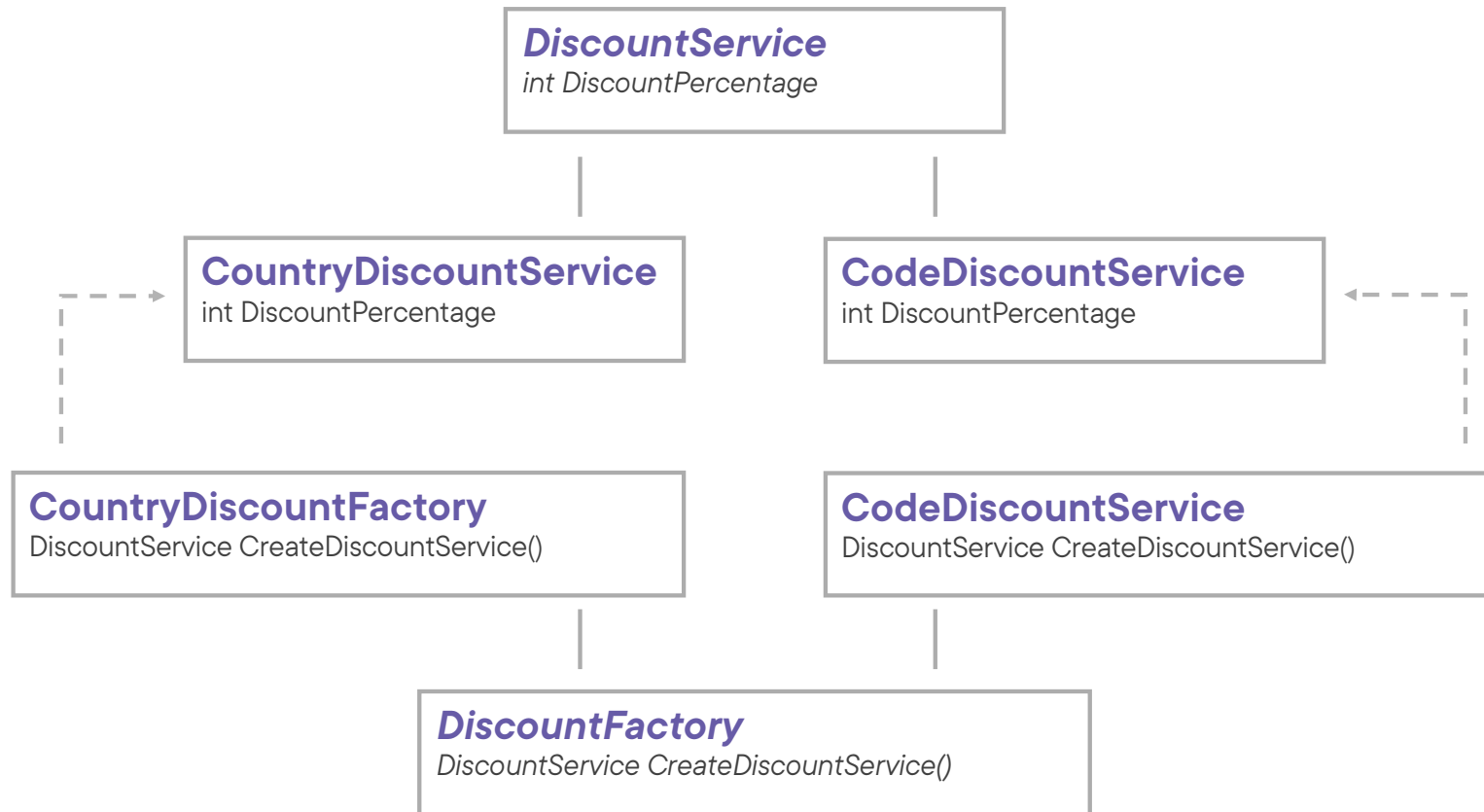
# Describing the Factory Method Pattern



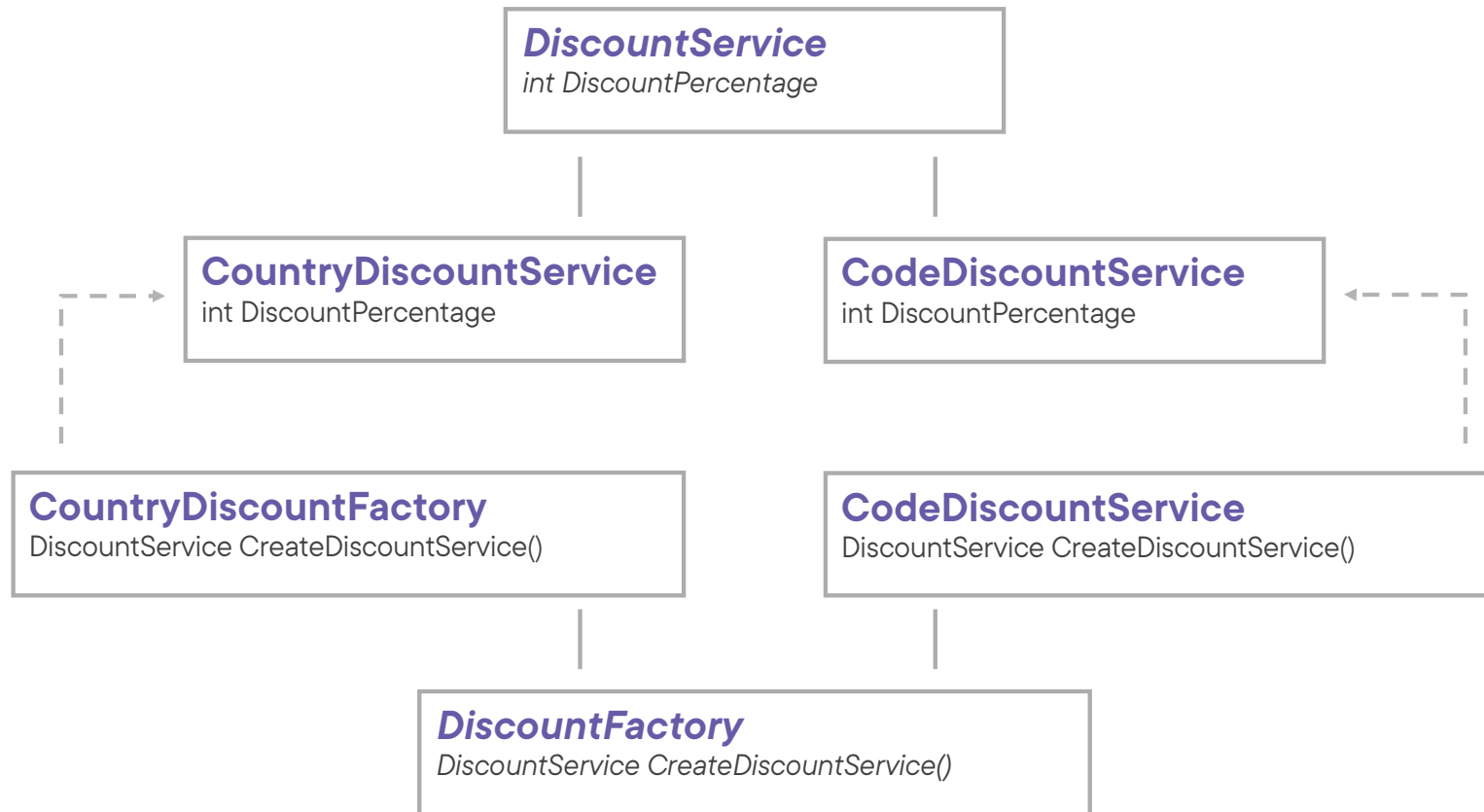
# Describing the Factory Method Pattern



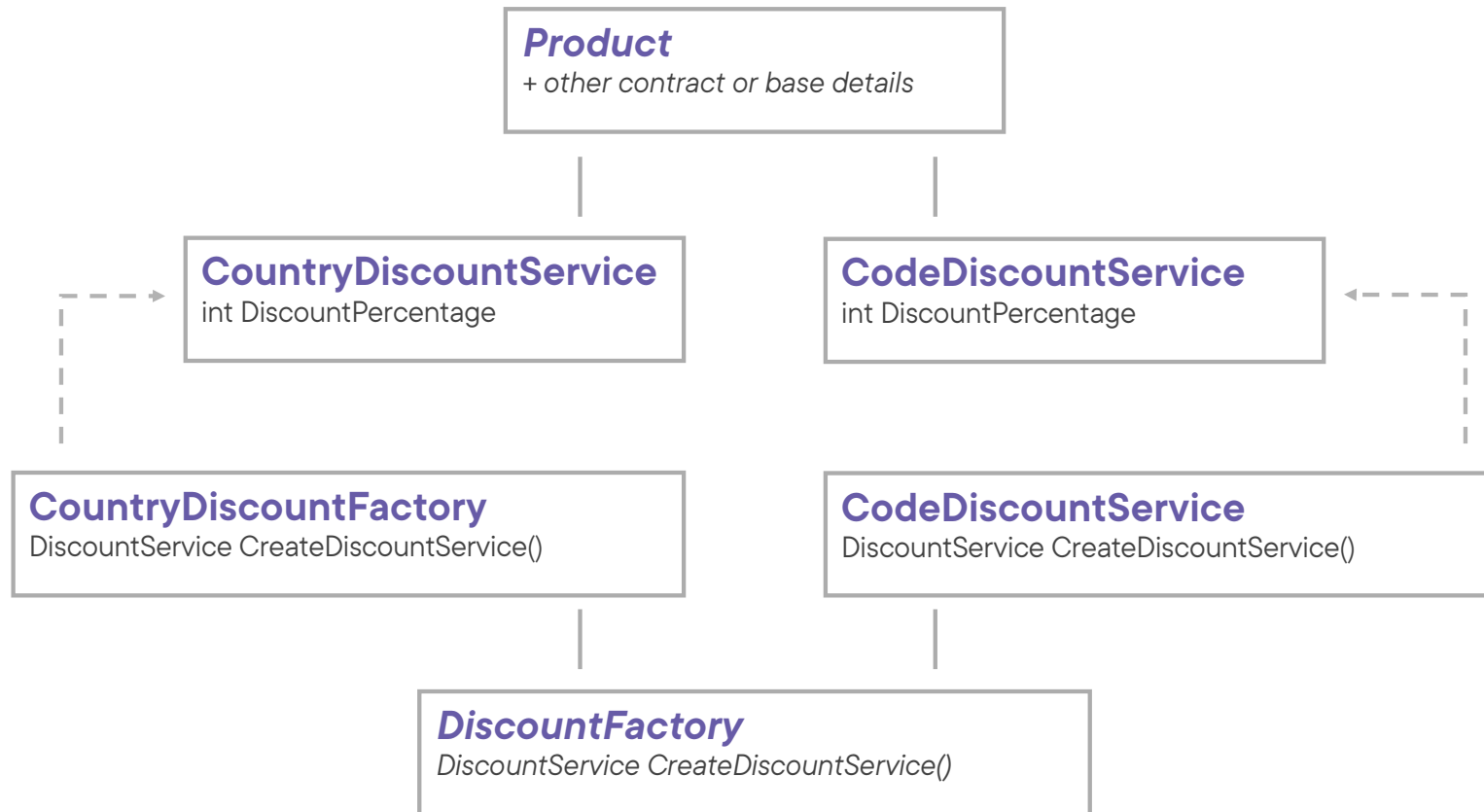
# Describing the Factory Method Pattern



# Factory Method Pattern Structure



# Factory Method Pattern Structure



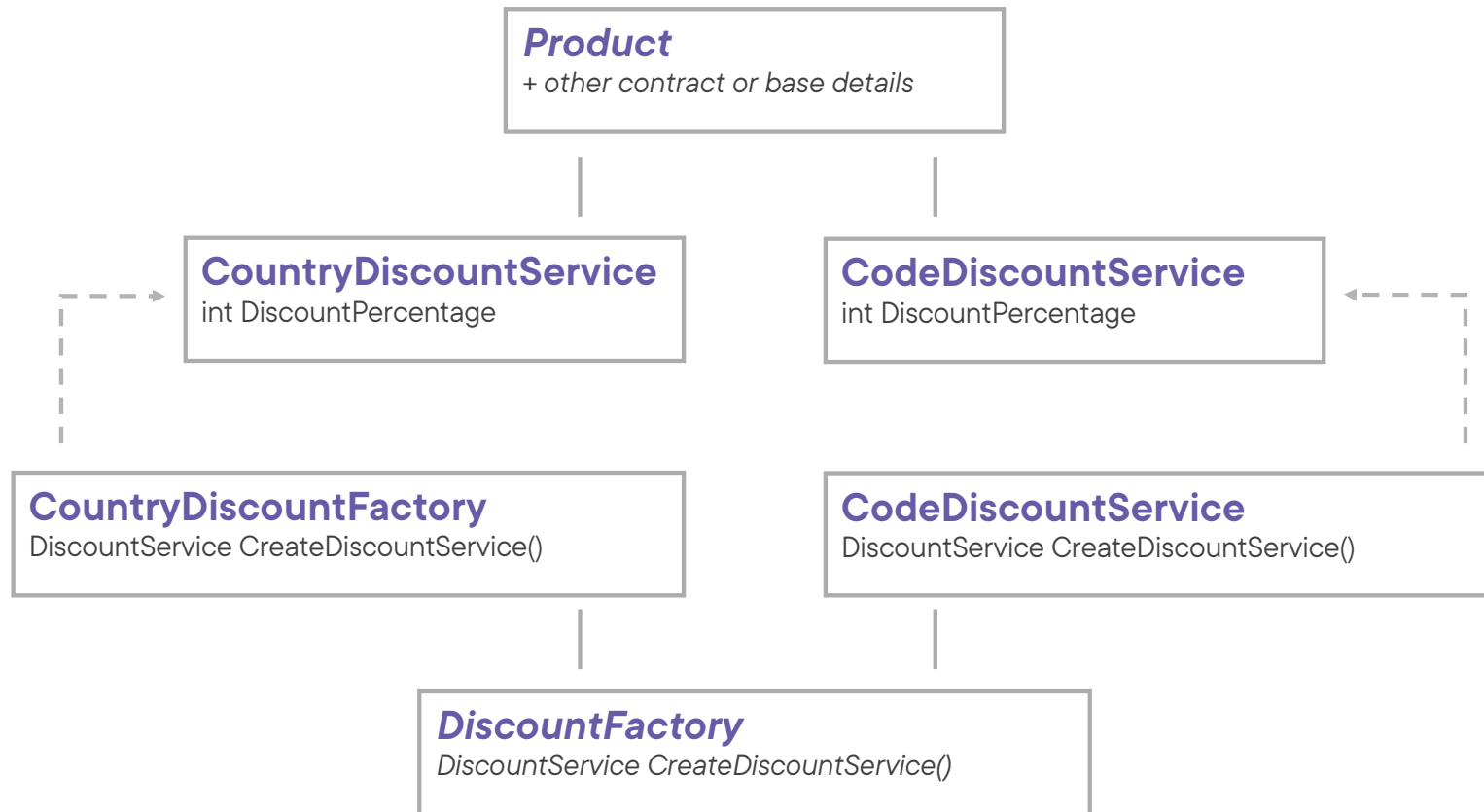


**Product** defines the interface of objects that the factory method creates





# Factory Method Pattern Structure

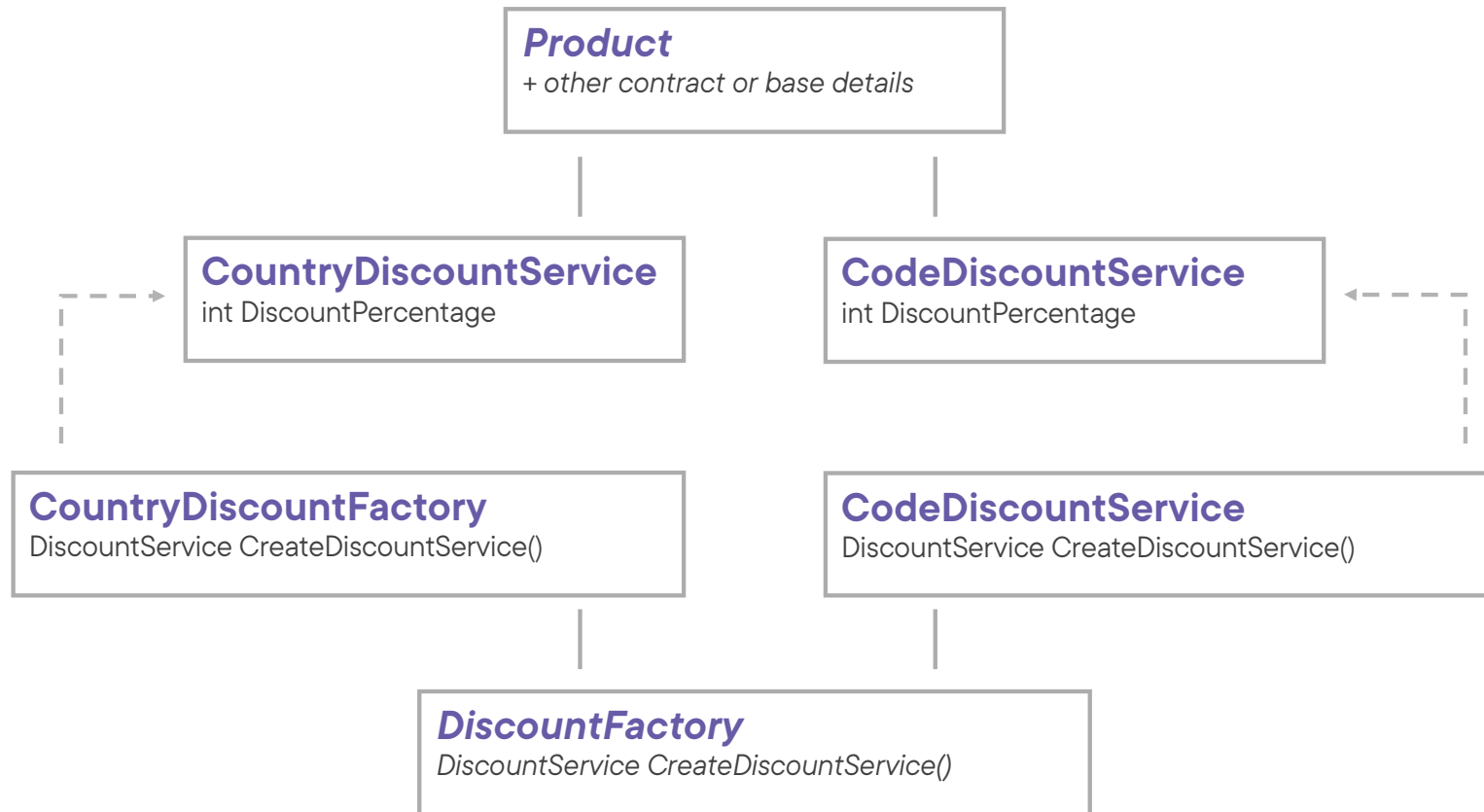




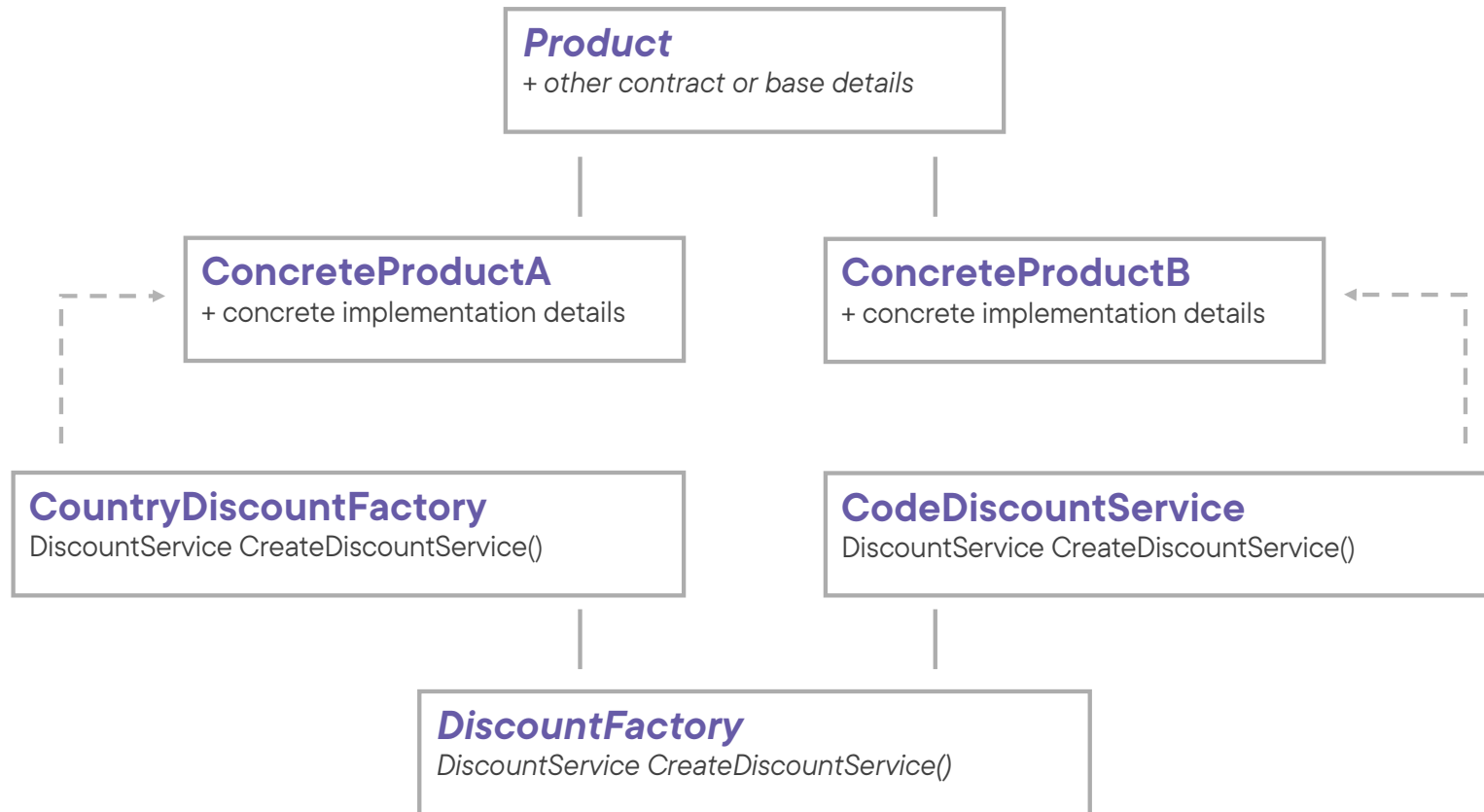
**ConcreteProduct** implements  
the **Product** interface



# Factory Method Pattern Structure



# Factory Method Pattern Structure

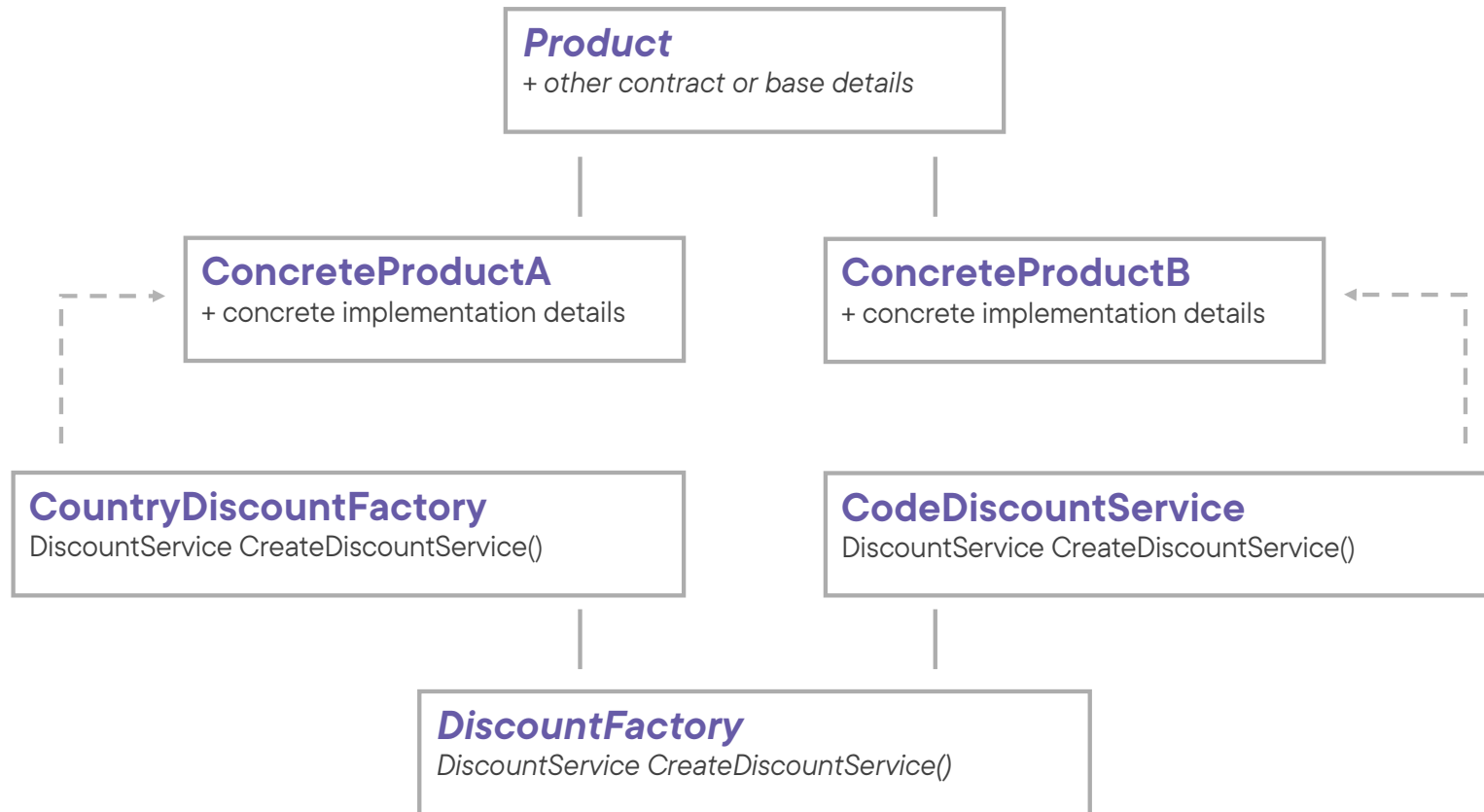




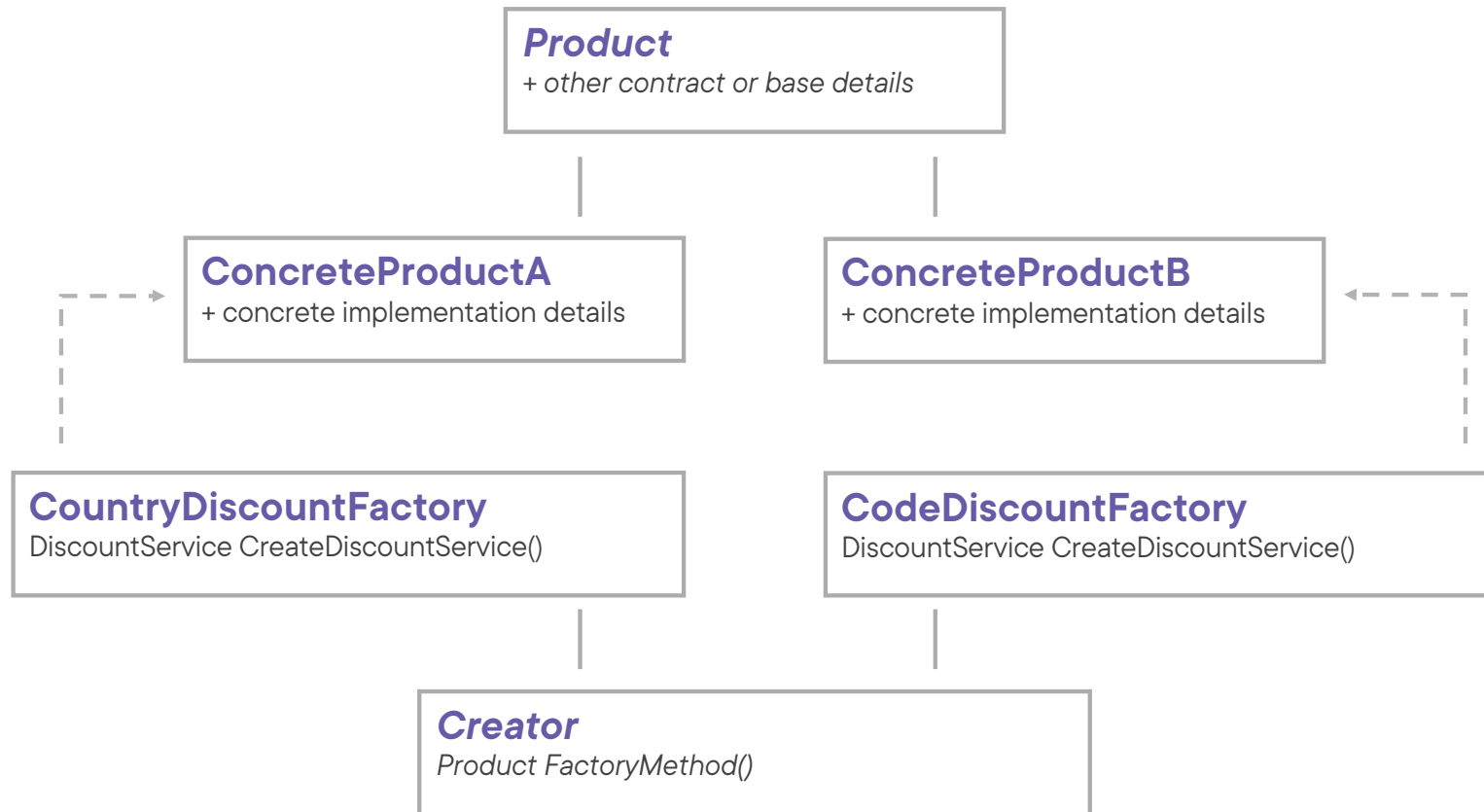
The **Creator** declares the factory method, which must return a **Product**



# Factory Method Pattern Structure



# Factory Method Pattern Structure



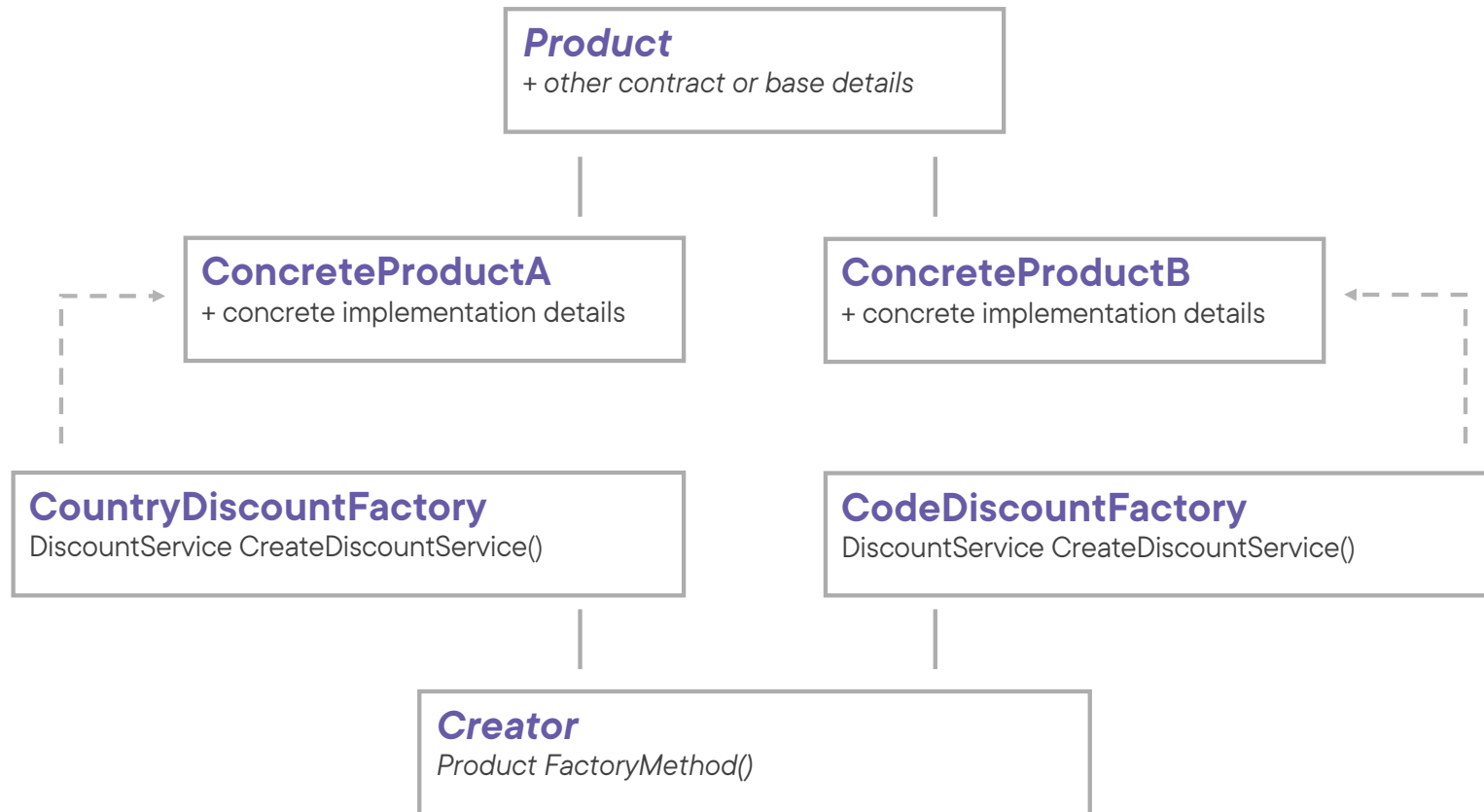


The **ConcreteCreator**  
implements **Creator** and overrides  
the factory method to return an  
instance of **ConcreteProduct**

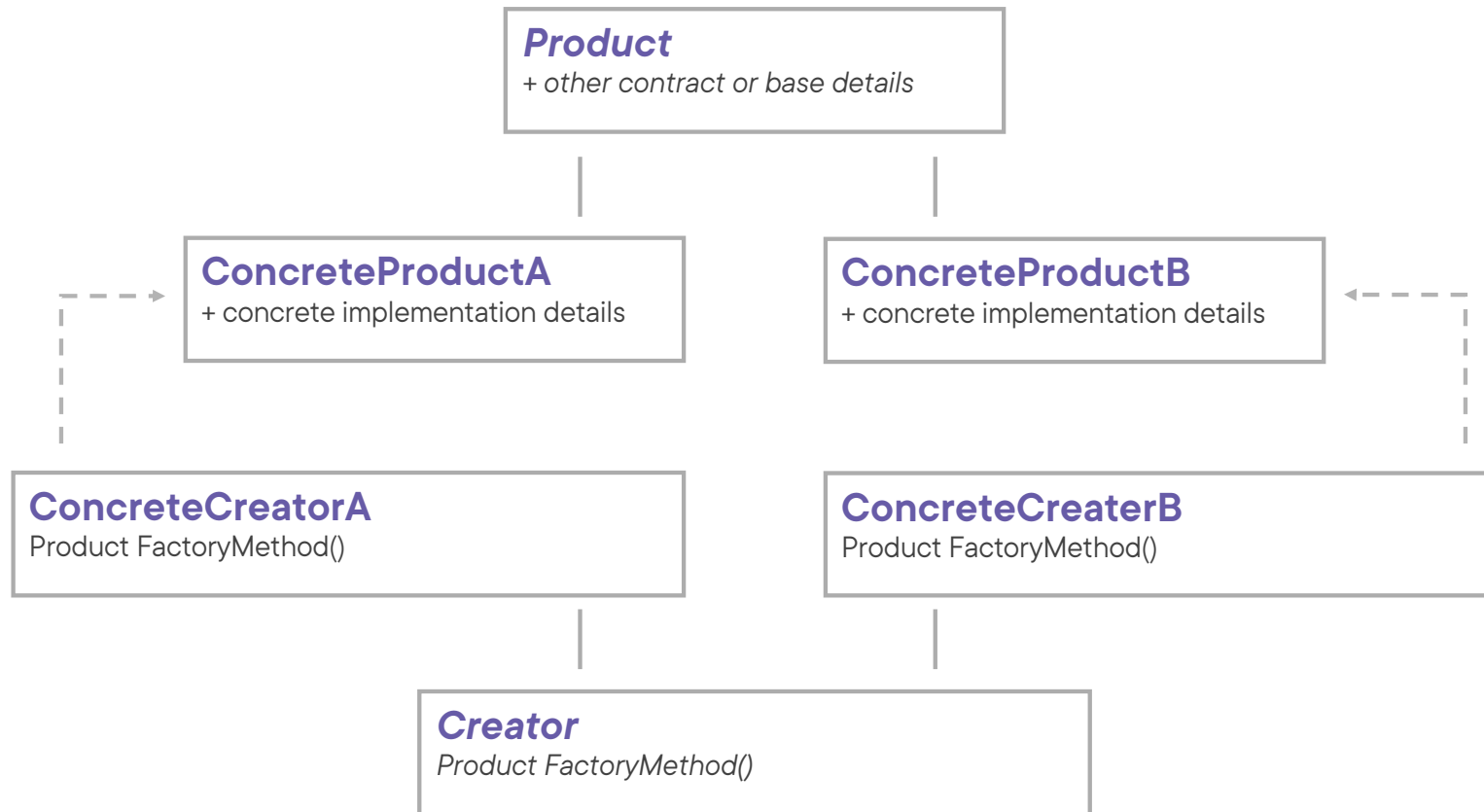




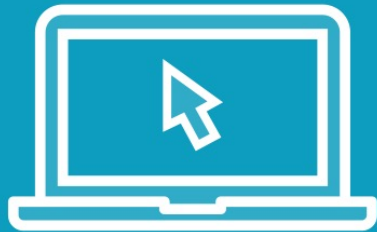
# Factory Method Pattern Structure



# Factory Method Pattern Structure



Demo



**Implementing the factory method pattern**



# Use Cases for the Factory Method Pattern



**When a class can't anticipate the class of objects it must create**



**When a class wants its subclasses to specify the objects it creates**



**When classes delegate responsibility to one of several helper subclasses, and you want to localize the knowledge of which helper subclass is the delegate**



**As a way to enable reusing of existing objects**



# Pattern Consequences



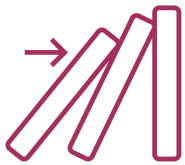
Factory methods eliminate the need to bind application-specific classes to your code



New types of products can be added without breaking client code:  
**open/closed principle**



Creating products is moved to one specific place in your code, the creator: **single responsibility principle**



Clients might need to create subclasses of the creator class just to create a particular ConcreteProduct object

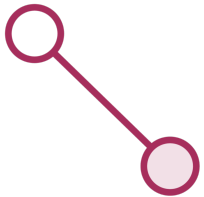


# Related Patterns



## **Abstract Factory**

Often implemented with factory methods



## **Prototype**

No subclassing is needed (not based on inheritance), but an initialize action on Product is often required



## **Template**

Factory methods are often called from within template methods



## Summary



### **Intent of the factory method pattern:**

- To define an interface for creating an object, but to let subclasses decide which class to instantiate

**Eliminates the need to bind application-specific classes to your code**



## Summary



### Implementation:

- Product and Creator can be implemented as interface or (abstract) base class
- ConcreteCreator must implement a method to create a ConcreteProduct





Up Next:

Creational Pattern: Abstract Factory

---

