Factory Method Design Pattern

Intent

- Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
- Defining a "virtual" constructor.
- The `new` operator considered harmful.

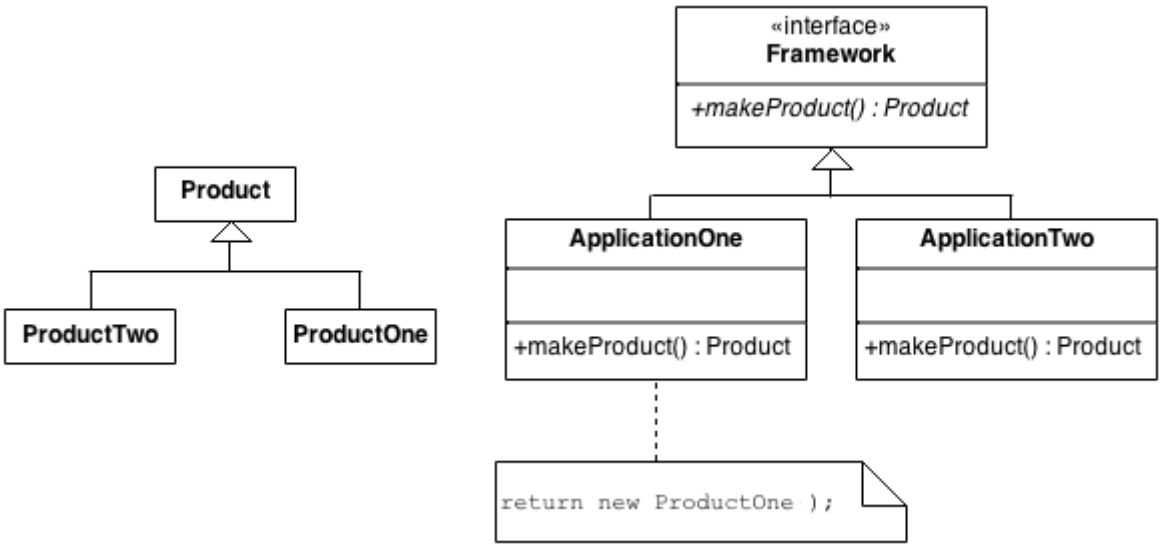
Problem

A framework needs to standardize the architectural model for a range of applications, but allow for individual applications to define their own domain objects and provide for their instantiation.

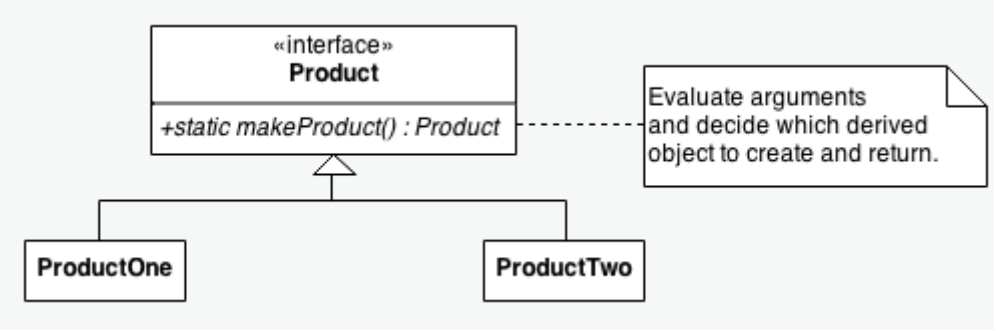
Discussion

Factory Method is to creating objects as Template Method is to implementing an algorithm.
A superclass specifies all standard and generic behavior (using pure virtual "placeholders" for creation steps), and then delegates the creation details to subclasses that are supplied by the client.
Factory Method makes a design more customizable and only a little more complicated. Other design patterns require new classes, whereas Factory Method only requires a new operation.
People often use Factory Method as the standard way to create objects; but it isn't necessary if: the class that's instantiated never changes, or instantiation takes place in an operation that subclasses can easily override (such as an initialization operation).

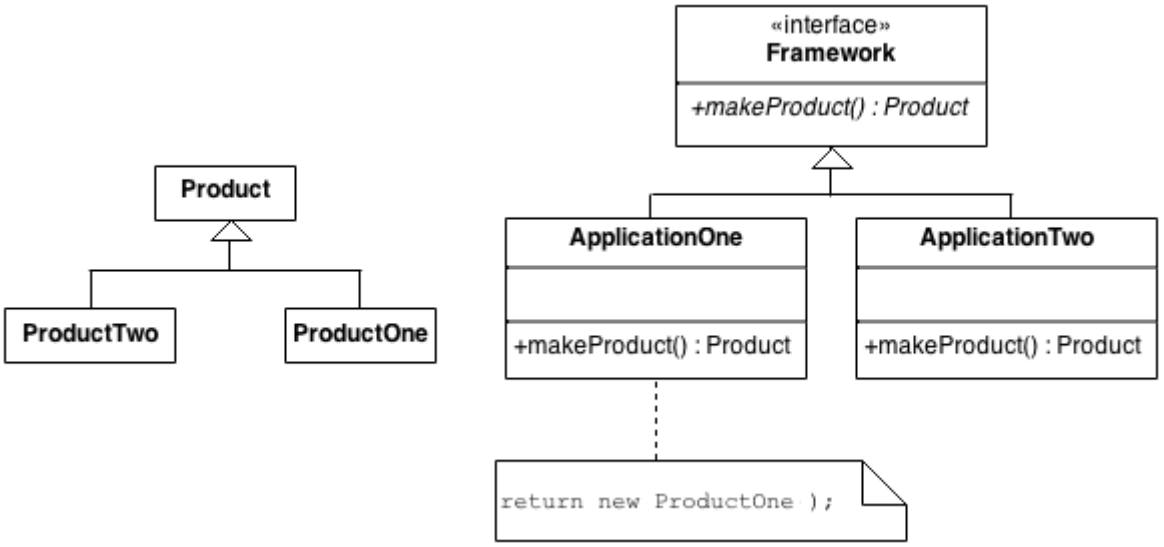
Structure



An increasingly popular definition of factory method is: a `static` method of a class that returns an object of that class' type.
But unlike a constructor, the actual object it returns might be an instance of a subclass. Unlike a constructor, an existing object might be reused, instead of a new object created.
Unlike a constructor, factory methods can have different and more descriptive names (e.g. `Color.make_RGB_color(float red, float green, float blue)` and`Color.make_HSB_color(float hue, float saturation, float brightness)`



The client is totally decoupled from the implementation details of derived classes.
Polymorphic creation is now possible.



Pros

- You avoid tight coupling between the creator and the concrete products.
- *Single Responsibility Principle*. You can move the product creation code into one place in the program, making the code easier to support.
- *Open/Closed Principle*. You can introduce new types of products into the program without breaking existing client code.

Cons

- The code may become more complicated since you need to introduce a lot of new subclasses to implement the pattern. The best case scenario is when you're introducing the pattern into an existing hierarchy of creator classes.