- **ConcreteCreator** (MyApplication)
  - overrides the factory method to return an instance of a ConcreteProduct.

Collaborations

- Creator relies on its subclasses to define the factory method so that it returns an instance of the appropriate ConcreteProduct.

Consequences

Factory methods eliminate the need to bind application-specific classes into your code. The code only deals with the Product interface; therefore it can work with any user-defined ConcreteProduct classes.

A potential disadvantage of factory methods is that clients might have to subclass the Creator class just to create a particular ConcreteProduct object. Subclassing is fine when the client has to subclass the Creator class anyway, but otherwise the client now must deal with another point of evolution.

Here are two additional consequences of the Factory Method pattern:

1. *Provides hooks for subclasses.* Creating objects inside a class with a factory method is always more flexible than creating an object directly. Factory Method gives subclasses a hook for providing an extended version of an object.

   In the Document example, the Document class could define a factory method called CreateFileDialog that creates a default file dialog object for opening an existing document. A Document subclass can define an application-specific file dialog by overriding this factory method. In this case the factory method is not abstract but provides a reasonable default implementation.

2. *Connects parallel class hierarchies.* In the examples we've considered so far, the factory method is only called by Creators. But this doesn't have to be the case; clients can find factory methods useful, especially in the case of parallel class hierarchies.

   Parallel class hierarchies result when a class delegates some of its responsibilities to a separate class. Consider graphical figures that can be manipulated interactively; that is, they can be stretched, moved, or rotated using the mouse. Implementing such interactions isn't always easy. It often requires storing and updating information that records the state of the manipulation at a given time. This state is needed only during manipulation; therefore it needn't be kept in the figure object. Moreover, different figures behave differently when the user manipulates them. For example, stretching a line figure might have the effect of moving an endpoint, whereas stretching a text figure may change its line spacing.

   With these constraints, it's better to use a separate Manipulator object that implements the interaction and keeps track of any manipulation-specific state