THEIRS differently than the parent class. It isn't interested in other classes. Hence MyCreator *extends* the kinds of products created, and it defers responsibility for creating all but a few products to its parent.

3. *Language-specific variants and issues.* Different languages lend themselves to other interesting variations and caveats.

   Smalltalk programs often use a method that returns the class of the object to be instantiated. A Creator factory method can use this value to create a product, and a ConcreteCreator may store or even compute this value. The result is an even later binding for the type of ConcreteProduct to be instantiated.

   A Smalltalk version of the Document example can define a documentClass method on Application. The documentClass method returns the proper Document class for instantiating documents. The implementation of documentClass in MyApplication returns the MyDocument class. Thus in class Application we have

```
clientMethod
    document := self documentClass new.

documentClass
    self subclassResponsibility
```

   In class MyApplication we have

```
documentClass
    ^MyDocument
```

   which returns the class MyDocument to be instantiated to Application.

   An even more flexible approach akin to parameterized factory methods is to store the class to be created as a class variable of Application. That way you don't have to subclass Application to vary the product.

   Factory methods in C++ are always virtual functions and are often pure virtual. Just be careful not to call factory methods in the Creator's constructor—the factory method in the ConcreteCreator won't be available yet.

   You can avoid this by being careful to access products solely through accessor operations that create the product on demand. Instead of creating the concrete product in the constructor, the constructor merely initializes it to 0. The accessor returns the product. But first it checks to make sure the product exists, and if it doesn't, the accessor creates it. This technique is sometimes called **lazy initialization**. The following code shows a typical implementation: