parameter that identifies the kind of object to create. All objects the factory method creates will share the Product interface. In the Document example, Application might support different kinds of Documents. You pass CreateDocument an extra parameter to specify the kind of document to create.

The Unidraw graphical editing framework [VL90] uses this approach for reconstructing objects saved on disk. Unidraw defines a `Creator` class with a factory method `Create` that takes a class identifier as an argument. The class identifier specifies the class to instantiate. When Unidraw saves an object to disk, it writes out the class identifier first and then its instance variables. When it reconstructs the object from disk, it reads the class identifier first.

Once the class identifier is read, the framework calls `Create`, passing the identifier as the parameter. `Create` looks up the constructor for the corresponding class and uses it to instantiate the object. Last, `Create` calls the object's `Read` operation, which reads the remaining information on the disk and initializes the object's instance variables.

A parameterized factory method has the following general form, where `MyProduct` and `YourProduct` are subclasses of `Product`:

```
class Creator {
public:
    virtual Product* Create(ProductId id);
};

Product* Creator::Create (ProductId id) {
    if (id == MINE)  return new MyProduct;
    if (id == YOURS) return new YourProduct;
    // repeat for remaining products...

    return 0;
}
```

Overriding a parameterized factory method lets you easily and selectively extend or change the products that a Creator produces. You can introduce new identifiers for new kinds of products, or you can associate existing identifiers with different products.

For example, a subclass `MyCreator` could swap MyProduct and YourProduct and support a new `TheirProduct` subclass:

```
Product* MyCreator::Create (ProductId id) {
    if (id == YOURS)  return new MyProduct;
    if (id == MINE)   return new YourProduct;
        // N.B.: switched YOURS and MINE

    if (id == THEIRS) return new TheirProduct;

    return Creator::Create(id); // called if all others fail
}
```

Notice that the last thing this operation does is call `Create` on the parent class. That's because `MyCreator::Create` handles only `YOURS`, `MINE`, and