- Factory Method ?
- Simple Factory ?
- Abstract Factory ?

Intent (意图)

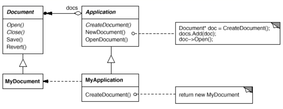
Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

Also Known As (别名)

Virtual Constructor

Motivation (动机)

Consider a framework for applications that can present multiple documents to the user.
- Two key abstractions in this framework are the classes Application and Document.
- The Application classes only knows **when** a new document should be created, not **what** what kind of Document to create.

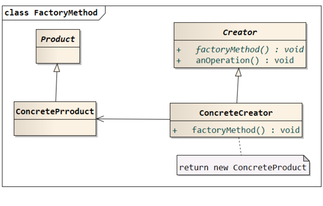


- We call CreateDocument a factory method because it's responsible for "manufacturing" an object.

Applicability (适用性)

- a class can't anticipate the class of objects it must create.
- a class wants its subclasses to specify the objects it creates.
- classes delegate responsibility to one of several helper subclasses, and you want to localise the knowledge of which helper subclass is the delegate.

Structure (结构)



Participants (参与者)

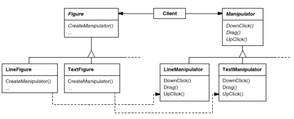
- Product (Document)
  - defines the interface of objects the factory method creates.
- ConcreteProduct (MyDocument)
  - implements the Product Interface
- Creator (Application)
  - declares the factory method, which returns an object of type Product. Creator may also define a default implementation of the factory method that returns a default ConcreteProduct object.
  - may call the factory method to create a Product object.
- ConcreteCreator (MyApplication)
  - overrides the factory method to return an instance of a ConcreteProduct.

Collaborations (协作)

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

Consequences (结果)

- Provides hooks for subclasses.
- Connects 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, the can be stretched, moved, or rotated using the mouse.
  - It often requires storing and updating information that records the state of the manipulation.
  - This state is needed only during manipulation; therefore it needn't be kept in the figure object. (like Action class in Cocos)



A potential disadvantage:
- clients might have to subclass the Creator class just to create a particular ConcreteProduct object.

Implementation (实现)

- Two major varieties.

1. the case when the Creator class is an abstract class and does not provide an implementation for the factory method it declares.
2. the case when Creator is a concrete class and provides a default implementation for the factory method.

- Parameterized factory methods.

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

Product *Creator::create(ProductId id)
{
if (id == MINE) return new MyProduct;
if (id == YOURS) return new YourProduct;

return 0;
}

class MyCreator : public Creator
{
public:
virtual Product *create(ProductId);
};

Product *MyCreator::create(ProductId id)
{
//switched YOURS and MINE
if (id == YOURS) return new MyProduct;
if (id == MINE) return new YourProduct;

if (id == THEIRS) return new TheirProduct;

return Creator::create(id);
}
```

- Language-specific variants and issues.

- Smalltalk ...
- Factory methods in C++ are always virtual functions and are often pur 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.
- Lazy initialization.

```
class Creator
{
public:
Product *getProduct();
protected:
virtual Product *createProduct();
private:
Product *_product;
};

Product *Creator::getProduct()
{
//lazy initialization
if (_product == 0) {
_product = createProduct();
}

return _product;
}
```

- Using templates to avoid subclassing.

- get around that Factory Methods might force you to subclass just to create the appropriate Product objects.

```
class Creator
{
public:
virtual Product *createProduct() = 0;
};

template <class TheProduct>
class StandardCreator : public Creator
{
public:
virtual Product *createProduct();
};

template <class TheProduct>
Product *StandardCreator<TheProduct>::createProduct()
{
return new TheProduct;
}

//usage:
StandardCreator<MyProduct> myCreator;
```

- Naming conventions.

for example:
- createXX()
- doMakeClass() in the MacApp Macintosh application framework.

Sample Code (代码示例)

CreateMaze Simple (old codes see Creational Patterns section)

```
class MazeGame
{
public:
Maze *createMaze();

//factory method:
virtual Maze *makeMaze() const;
virtual Room *makeRoom(int n) const;
virtual Wall *makeWall() const;
virtual Door *makeDoor(Room * r1, Room *r2);
};

Maze*MazeGame::createMaze()
{
Maze *aMaze = makeMaze();

Room *r1 = makeRoom(1);
Room *r2 = makeRoom(2);

Door *theDoor = makeDoor(r1, r2);

aMaze->addRoom(r1);
aMaze->addRoom(r2);

r1->setSide(North, makeWall());
r1->setSide(East, theDoor);
r1->setSide(South, makeWall());
r1->setSide(West, makeWall());

r2->setSide(North, makeWall());
r2->setSide(East, makeWall());
r2->setSide(South, makeWall());
r2->setSide(West, theDoor);

return aMaze;
}

class BombedMazeGame : public MazeGame
{
};

class EnchantedMazeGame : public MazeGame
{
};
```

Known Uses (已知应用)

...

Related Patterns (相关模式)

- **Abstract Factory** classes are often implemented with **Factory Methods.**
- **Factory methods** are usually called within **Template Methods.**
- **Prototypes** often require an Initialize operation on the Product class. **Factory Method** doesn't require such an operation.

Summary

定义了一个创建对象的接口，但由子类决定要实例化的类是哪一个。工厂方法让类把实例化推迟到子类。

"简单工厂"，其实不是一个设计模式，反而比较像是一种编程习惯。但是经常被使用。

"简单工厂"和"工厂方法"的区别

简单工厂：把全部的事情，在一个地方都处理完了

工厂方法：是创建一个框架，让子类决定要如何实现