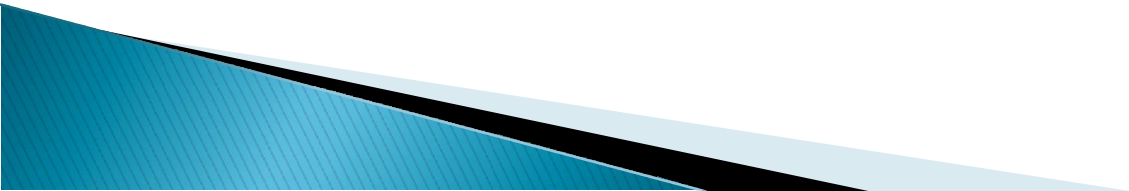
# 创建型设计模式
# Creational patterns

吴映波
wyb@cqu.edu.cn
虎溪Office: 虎溪学院楼410
A区Office：九教205
Tel：13594686661

# creational patterns

- Factory Method(virtual constructor)
- Abstract Factory
- Builder
- Prototype
- Singleton
- Finder

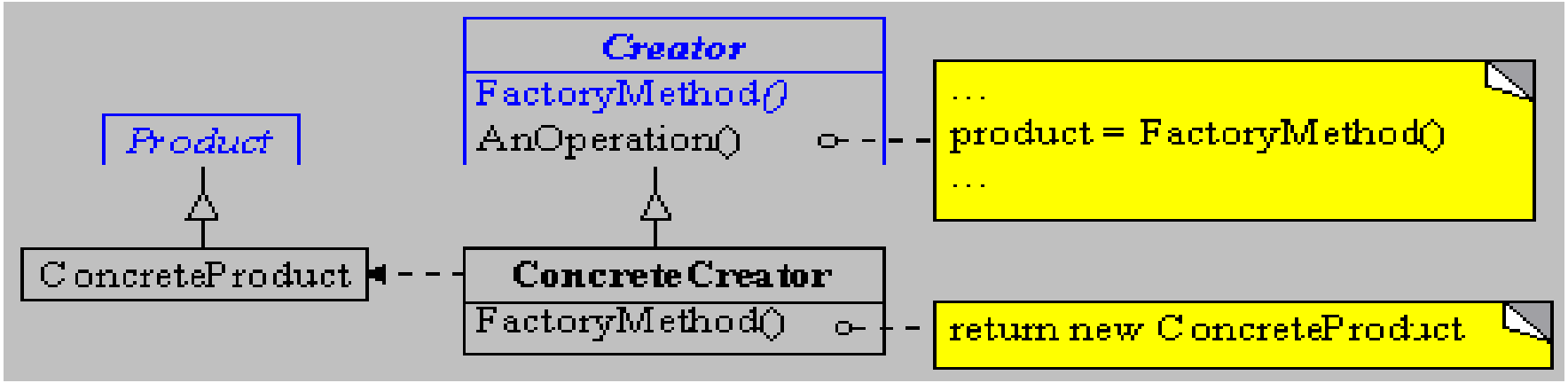


# 模式 1：Factory Method (一)

- Aliases：virtual constructor
- Problem
  - **Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.**

# Factory Method模式(二)

- Struct.




- Participants
  - **Product、ConcreteProduct、Creator、ConcreteCreator**

# Factory Method模式(三)

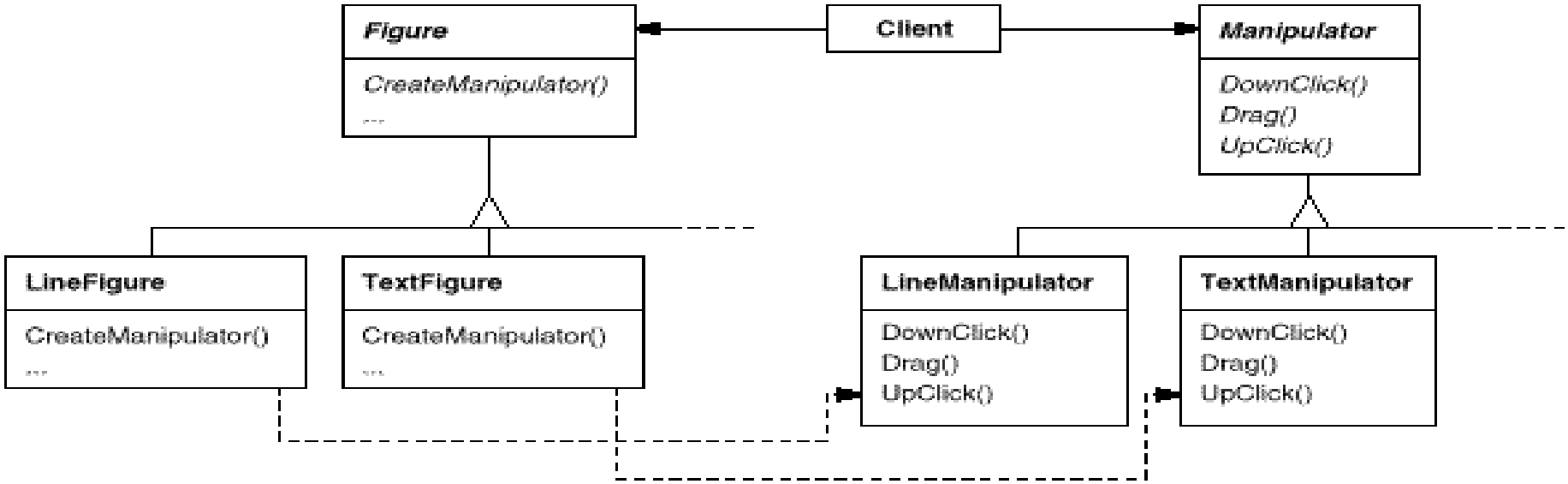
- Context：Use the Factory Method pattern when
  - **a class can't anticipate the class of objects it must create.**
  - **a class wants its subclasses to specify the objects it creates.**
  - **classes delegate responsibility to one of several helper subclasses, and you want to localize the knowledge of which helper subclass is the delegate.**

# Factory Method模式(四)

- Consequence
  - 缺点：需要Creator和相应的子类作为factory method的载体，如果应用模型确实需要creator和子类存在，则很好；否则的话，需要增加一个类层次
  - 优点：
    - **(1) Provides hooks for subclasses**。基类为factory method提供缺省实现，子类可以重写新的实现，也可以继承父类的实现。
      体现了：加一层间接性，增加了灵活性
    - **(2) Connects parallel class hierarchies**

# Factory Method模式(五)

- Connects parallel class hierarchies

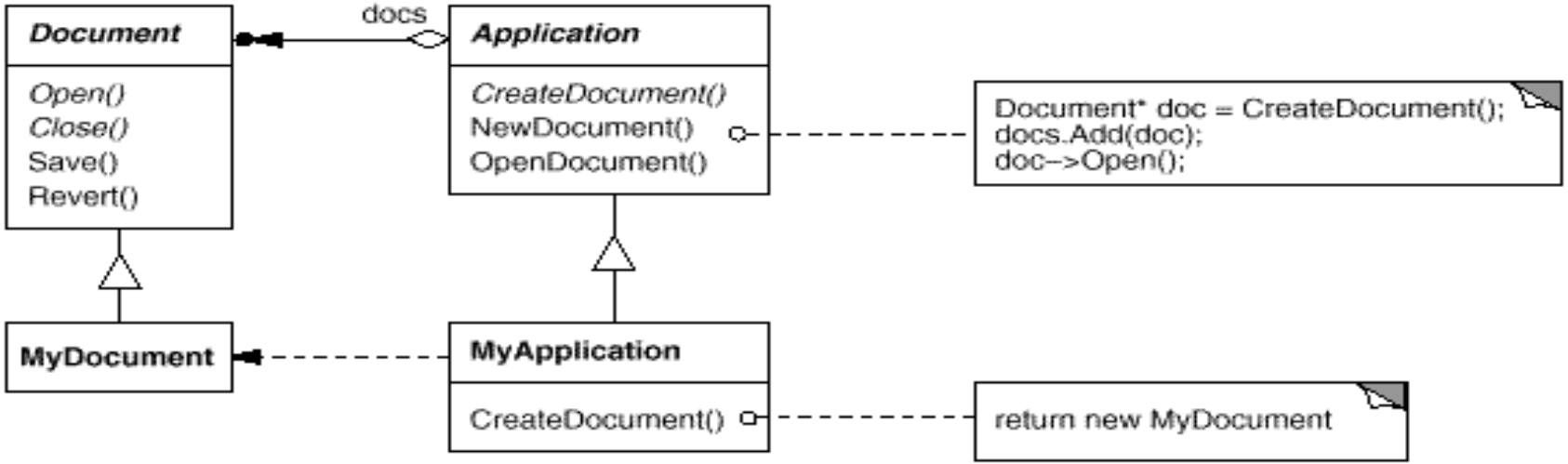


# Factory Method模式(六)

- Implementation
  - (1) 父类是否提供缺省的实现
  - (2) factory method的参数
  - (3) Language-specific variants and issues
    - SmallTalk，使用类型
    - C++，使用lazy initialization技术
  - (4) Using templates to avoid subclassing

# Factory Method模式(七)

- Related Patterns
  - **Abstract factory**
  - **Prototype**
- Examples



# 模式 2 ： Abstract Factory(一)

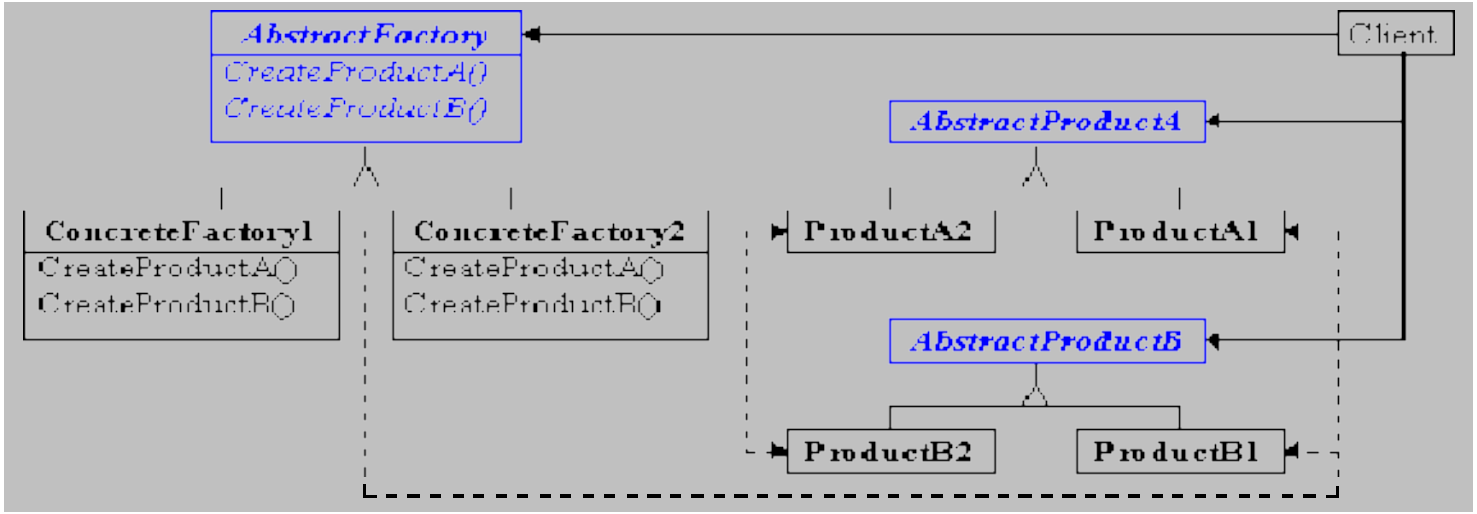
- Aliases：Kit
- Problem
  - **Provide an interface for creating families of related or dependent objects without specifying their concrete classes**
  - 解决一族相关或者相依对象的创建工作，专门定义一个用于创建这些对象的接口(基类)。客户只需与这个基接口打交道，不必考虑实体类的类型。

# Abstract Factory(二)

- Context：Use the Abstract Factory pattern when
  - **a system should be independent of how its products are created, composed, and represented.**
  - **a system should be configured with one of multiple families of products.**
  - **a family of related product objects is designed to be used together, and you need to enforce this constraint.**
  - **you want to provide a class library of products, and you want to reveal just their interfaces, not their implementations.**

# Abstract Factory(三)

- Struct.

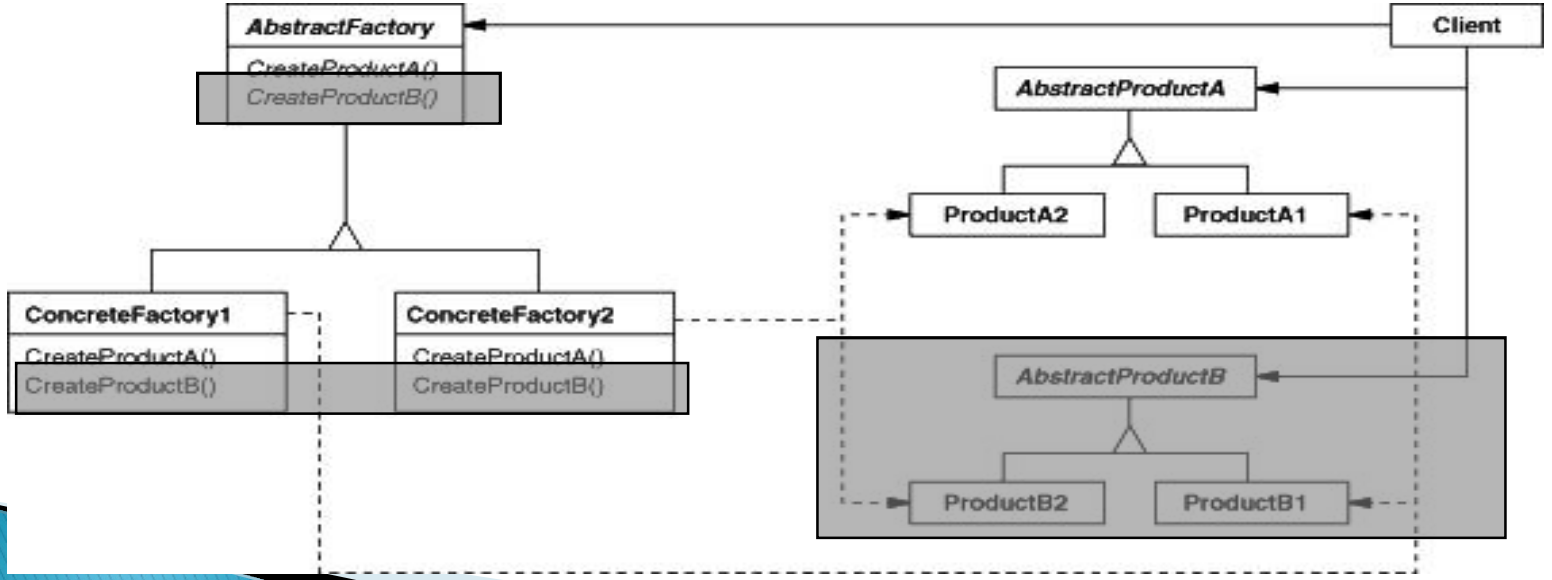



- Participants：
  - Client、AbstractFactory、ConcreteFactory、AbstractProduct、ConcreteProduct

# Abstract Factory(四)

- Consequence
  - 与**factory method**的关系

    **多个factory method合在一起，factory method一定是virtual的**



# Abstract Factory(五)

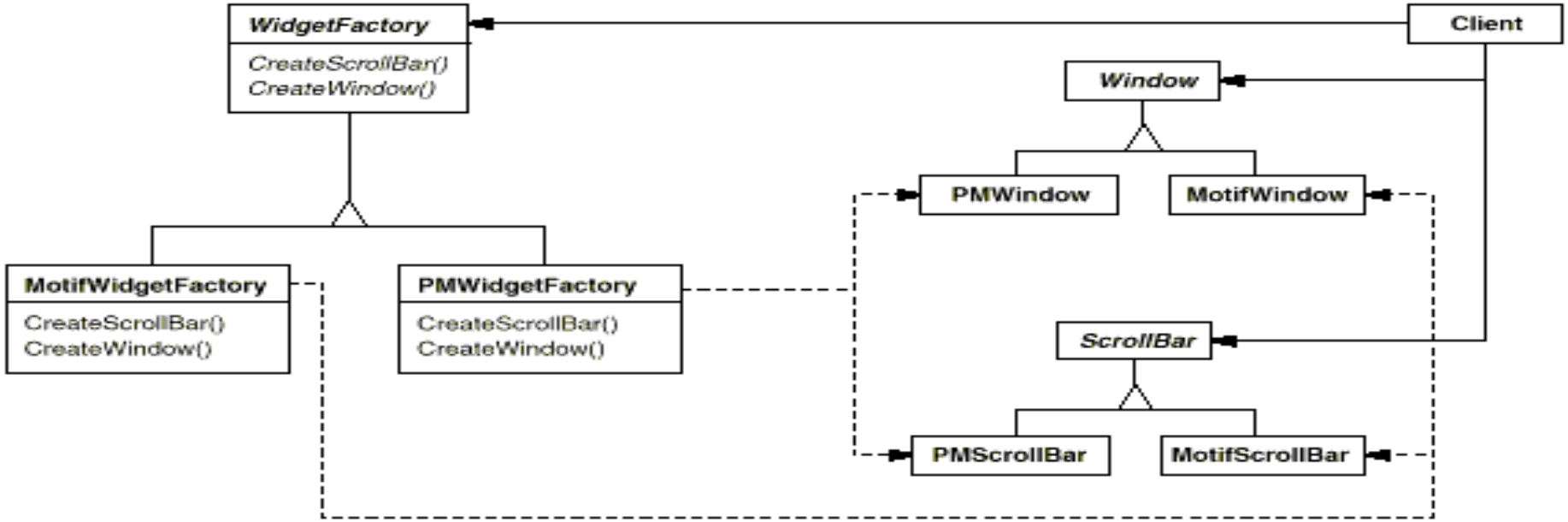
- Evaluation(续)
  - 优点：
    - factory把product的类型封装起来，分离了具体的类
    - 易于变换product族
    - 保证不同族之间的product相互不会碰撞，即保证products的一致性
  - 缺点：
    - factory对象的工厂方法数目对应product种类数目，增加新的product种类比较困难，要影响到factory的基类，进而影响到所有的子类

# Abstract Factory(六)

- Implementation
  - Factories as singletons:每个product族往往只需要一个factory对象就可以了
  - Creating the products:对于product族比较多的情况，可以使用prototype模式来实现这些factories，而不必对于每一个具有细微差别的product族都使用一个concrete factory class
  - Defining extensible factories:针对Evaluation中提到的缺点，通过参数化技术提高factory的适应能力和扩展性
    - 问题在于，返回给客户什么样的类型？

# Abstract Factory(七)

- Related Patterns
  - **Factory Method、Prototype 、Singleton**
- Examples: WidgetFactory

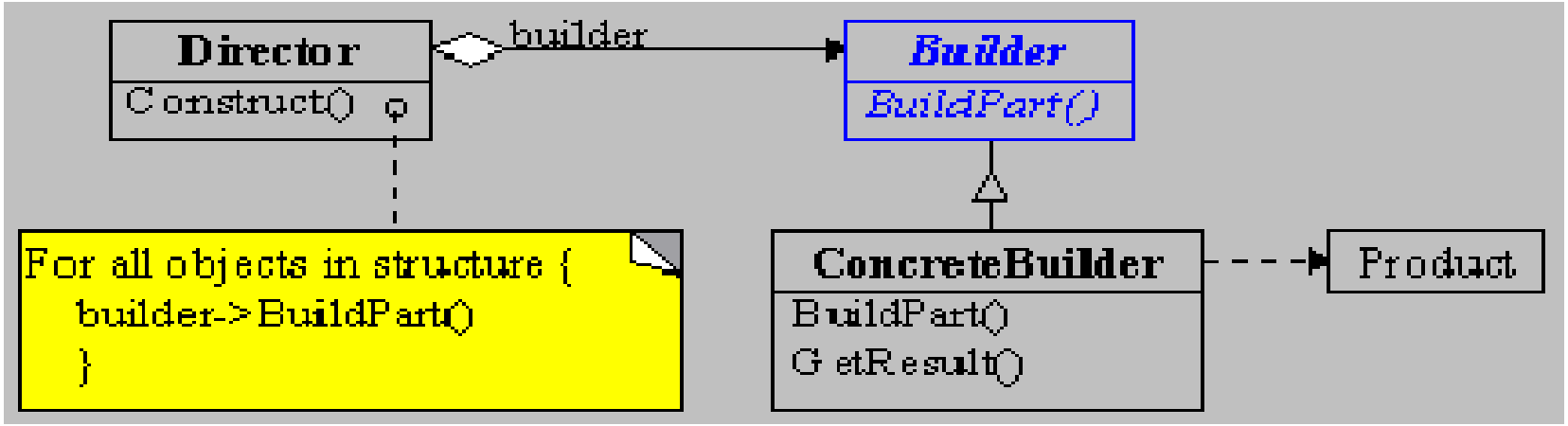


# 模式三：Builder (一)

- Intent
  - **Separate the construction of a complex object from its representation so that the same construction process can create different representations**
- Motivation
  - 在复杂对象的构造过程中，允许同样的构造过程能够加入新的被构造元素
  - “结构化构造过程”
- Applicability, Use the Builder pattern when
  - **the algorithm for creating a complex object should be independent of the parts that make up the object and how they're assembled.**
  - **the construction process must allow different representations for the object that's constructed.**

# Builder (二)

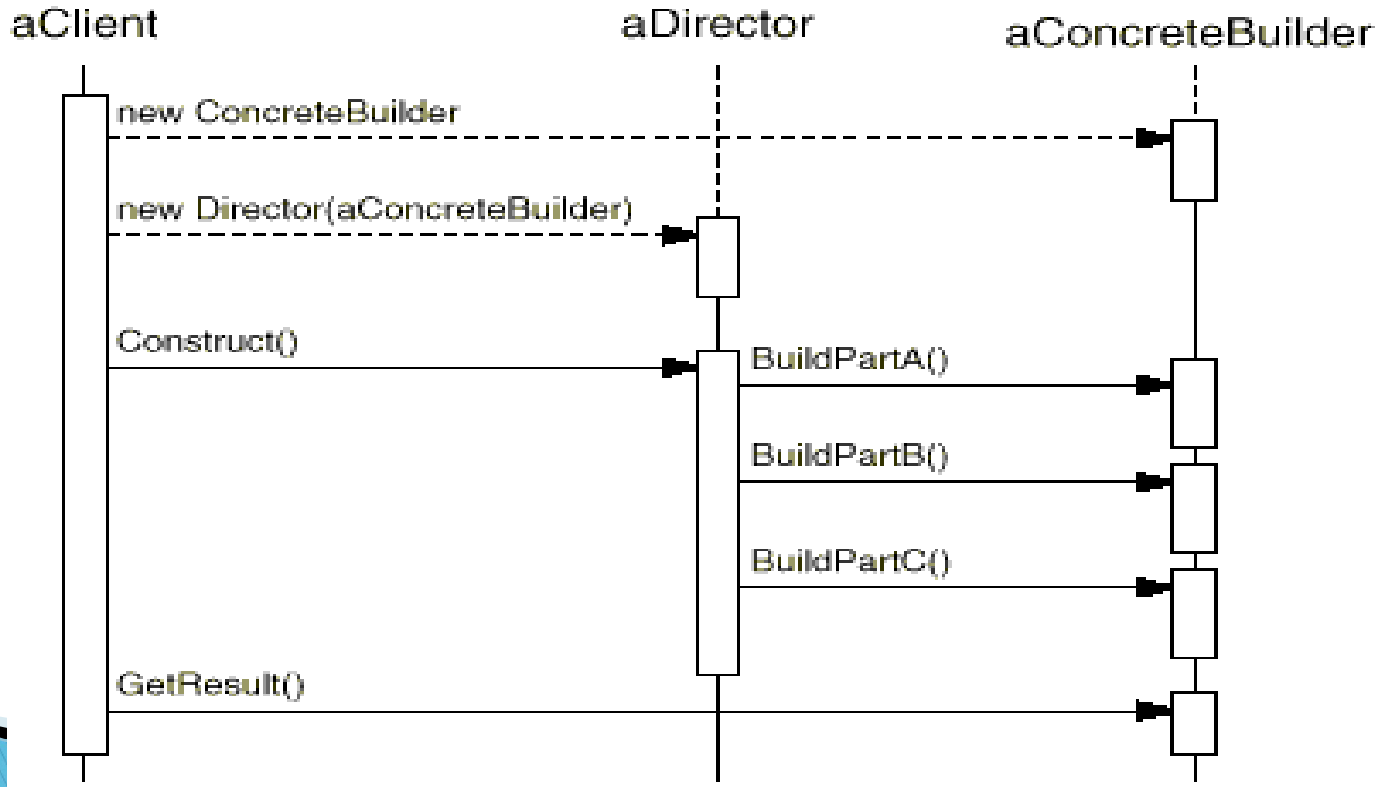
- Struct.




- Participants
  - Director、 Builder、ConcreteBuilder、 Product

# Builder (三)

- Collaborations

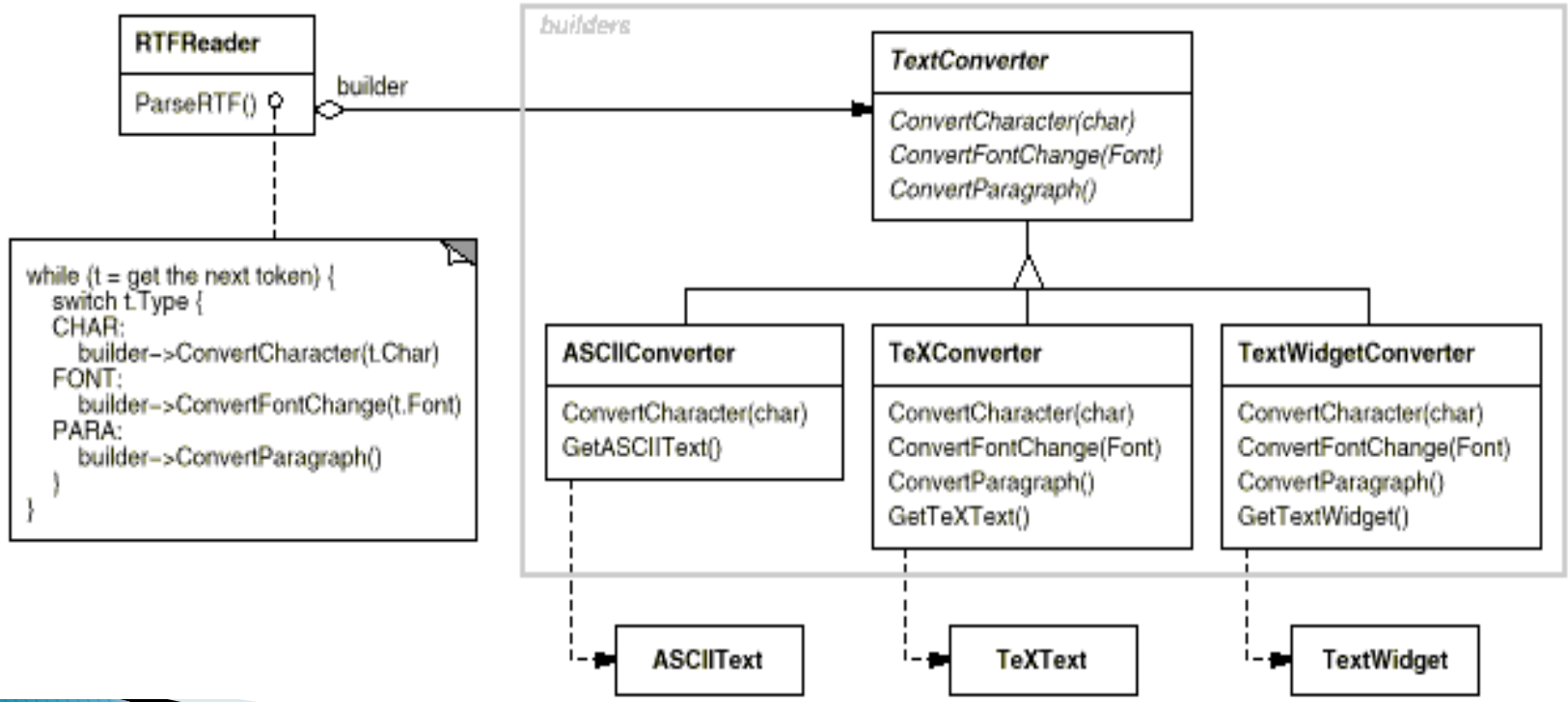


# Builder (四)

- Evaluation
  - **It lets you vary a product's internal representation**
  - **It isolates code for construction and representation**
  - **It gives you finer control over the construction process**
- Implementation
  - **Builder interface(Assembly and construction)**
  - **Why no abstract class for products?**
  - **Empty methods as default in Builder.**
- Related patterns
  - **Abstract Factory**
    - 区别：(1) builder重在构造过程，最后一步返回结果;
      (2) builder构造许多复杂对象

# Builder (五)

- Examples

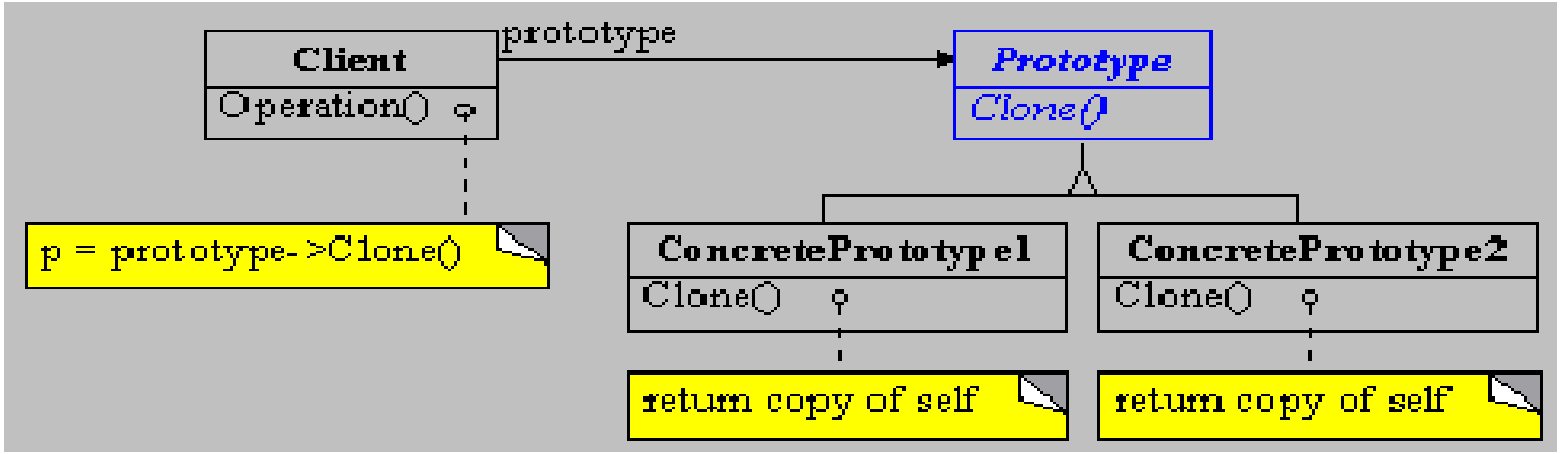


# 模式四：Prototype(一)

- Intent
  - **Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.**
- Motivation
  - 以一个已有的对象作为原型，通过它来创建新的对象。在增加新的对象的时候，新对象的细节创建工作由自己来负责，从而使新对象的创建过程与框架隔离开来
- Applicability
  - 当产品的创建过程要独立于系统时
  - 当产品的类型是在**runtime**时被指定的情况下
  - 避免创建一个与**product**层次平行的**factory**层次时
  - 当产品类的实例只能是几种确定的不同实例状态中的一种时

# Prototype(二)

- Struct.



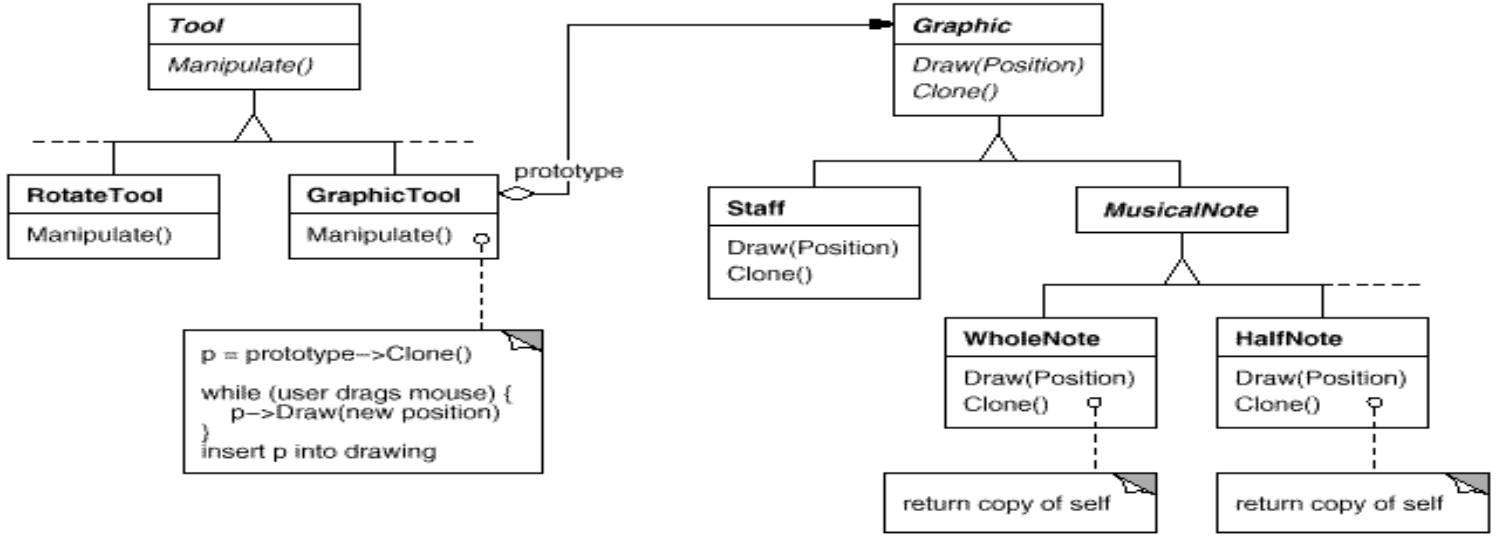

- Participants
  - Prototype、ConcretePrototype、Client
- Collaborations

# Prototype(三)

- Evaluation
  - Adding and removing products at run-time
  - Specifying new objects by varying values，降低系统中类的数目
  - Configuring an application with classes dynamically
  - 要求：每一个product类都必须实现Clone操作
  - 对于C++语言特别有意义：C++的class不是first-class objects
- Implementation
  - Using a prototype manager
  - Implementing the Clone operation
    - shallow copy versus deep copy
    - Save (Persistence) & Load
  - Initializing clones
    - 两阶段构造（静态构造+运行态构造）

# Prototype(四)

- Related patterns
  - **Prototype与Abstract Factory往往是相互竞争的**
  - **factory method**
- Examples
  - **DrawClip，music editor**

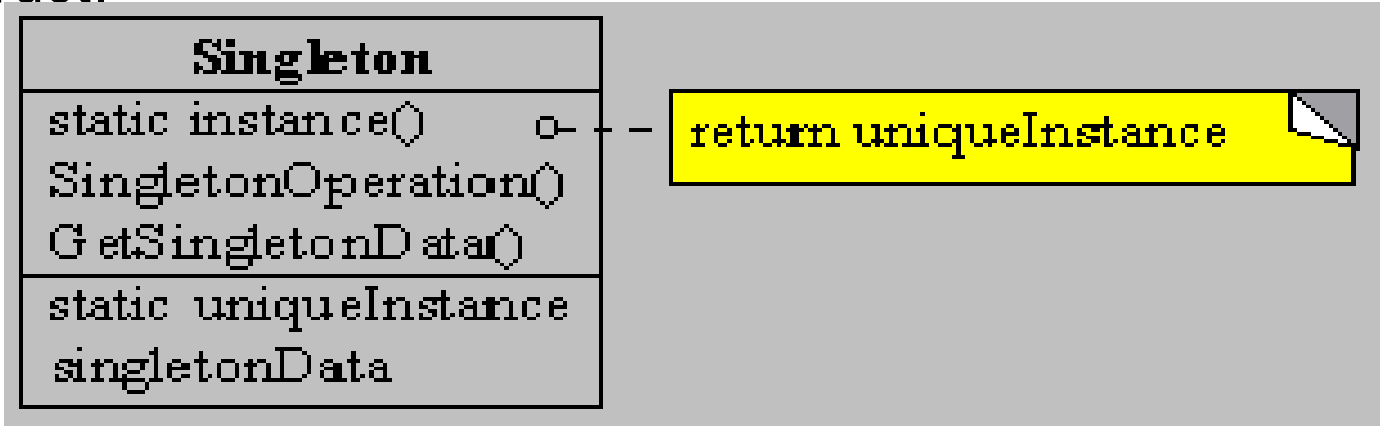


# 模式五：Singleton(一)

- Intent
  - Ensure a class only has one instance, and provide a global point of access to it.
- Motivation
  - It's important for some classes to have exactly one instance.
  - Instance-controlled class
- Applicability, Use the Singleton pattern when
  - there must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.
  - when the sole instance should be extensible by subclassing, and clients should be able to use an extended instance without modifying their code.

# Singleton(二)

- Struct.




- Participants
  - Singleton
- Collaborations
  - Clients access a Singleton instance solely through Singleton's Instance operation.

# Singleton(三)

- Evaluation
  - Controlled access to sole instance
  - Reduced name space
  - Permits refinement of operations and representation，允许子类化
  - Permits a variable number of instances
  - More flexible than class operations (static member functions in C++ )
  - 这种思想比较适用于Object-Based中的许多情形
- Implementation
  - Ensuring a unique instance
    - 考虑使用lazy initialize
    - 使用global/static object的缺点
  - Subclassing the Singleton class

# Singleton(四)

- Related patterns
  - Singleton与其他创建型模式并不矛盾，可以用singleton来实现其他模式中的对象。包括Abstract Factory、Builder、Prototype等。
    多个实例对于构造过程往往并无意义，所以在许多情况下singleton模式比较符合应用背景

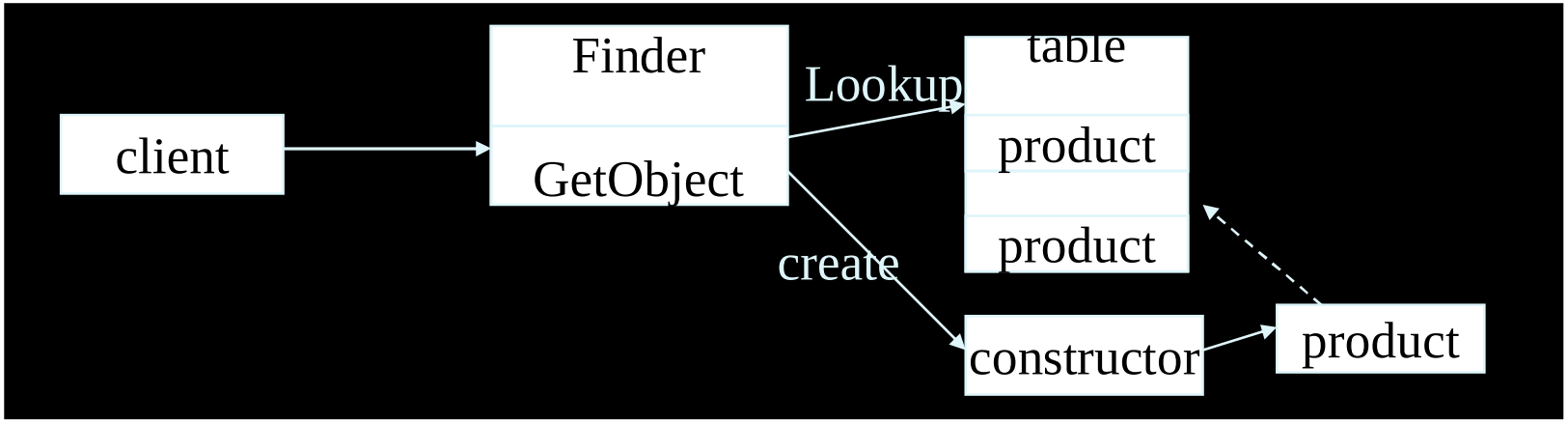
- Examples
  - MFC中的CWinApp派生类实例theApp
  - ……

# 增加模式六：Finder(一)

- Intent
  - 利用环境信息，根据客户的请求，找到已有的、符合要求的对象，返回给客户
- Alias
  - Object-retriever
- Motivation
  - 在有些情况下，客户希望连接到一个对象上，它提供一些状态标识信息，由Finder返回已经被创建的对象，或者重新创建新的对象(如果当前不存在满足条件的对象)
- Applicability, Use the Finder pattern when
  - 当需要在软件不同部分之间建立Client/Object连接时
  - 把获取对象的过程隐藏起来
  - view finder：The application demands user customizability of the actions taken when a particular file format is encountered in the browser.

# Finder(二)

- Struct.




- Participants
  - client、finder、product-table、constructor、product
- Collaborations

# Finder(三)

- Evaluation
  - 避免同样的对象被实例化两次，从而提高资源利用率，避免发生资源竞争
  - 把连接对象的过程与客户隔离开
  - 带来的问题：多个客户共享同样的资源，如何有效管理对象的所有权？

- Implementation
  - 实现product table以及相应的管理设施，保证查找过程的有效性
  - 每一个product的生命周期管理
  - 如何标识product的类型，客户如何多态地提供状态标识信息
  - Finder对象本身可以是一个singleton

# Finder(四)

- Related patterns
  - 与Singleton的区别：singleton是一个类的单个实例；而Finder是避免相同的对象(通常是类型和状态信息都相同)被创建两次。
  - 与Prototype的区别
  - 在创建product子步骤中，需要与其他创建型模式结合使用
- Examples
  - moniker in COM
  - 在Netscape浏览器中，根据MIME类型，找到插件，然后创建view

# creational patterns小结

- Factory Method
  - 本质：通过一个一致化的factory method完成产品对象的创建
- Abstract Factory
  - 基于多个factory method实现多个product族对象的创建
- Prototype
  - 通过product原型来clone创建product对象
- Builder
  - 完成一个包含多个子对象的复合对象的构造
- Singleton
  - Product类的单对象实例创建
- Finder
  - 把对象的获取过程与客户隔离开

这些patterns都很常见，有时需要结合两种或者多种模式完成系统中对象的构造过程。