SHRI RAMDEOBABA COLLEGE OF ENGINEERING AND MANAGEMENT, NAGPUR - 440013

DESIGN PATTERNS
(CST355-4)
V SEMESTER SECTION A, B

COURSE COORDINATOR: RINA DAMDOO

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Abstract Factory Design Patterns

Intent

- Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

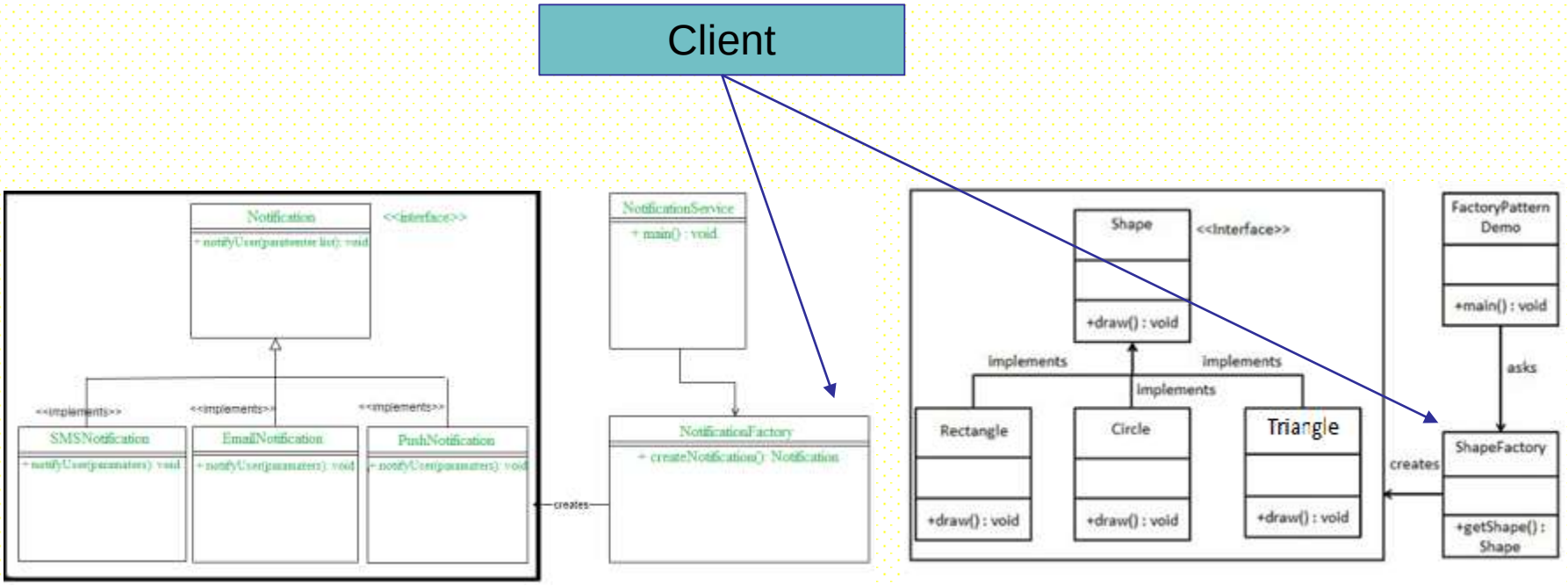
Also Known As

- Kit

ABSTRACT FACTORY DESIGN PATTERNS

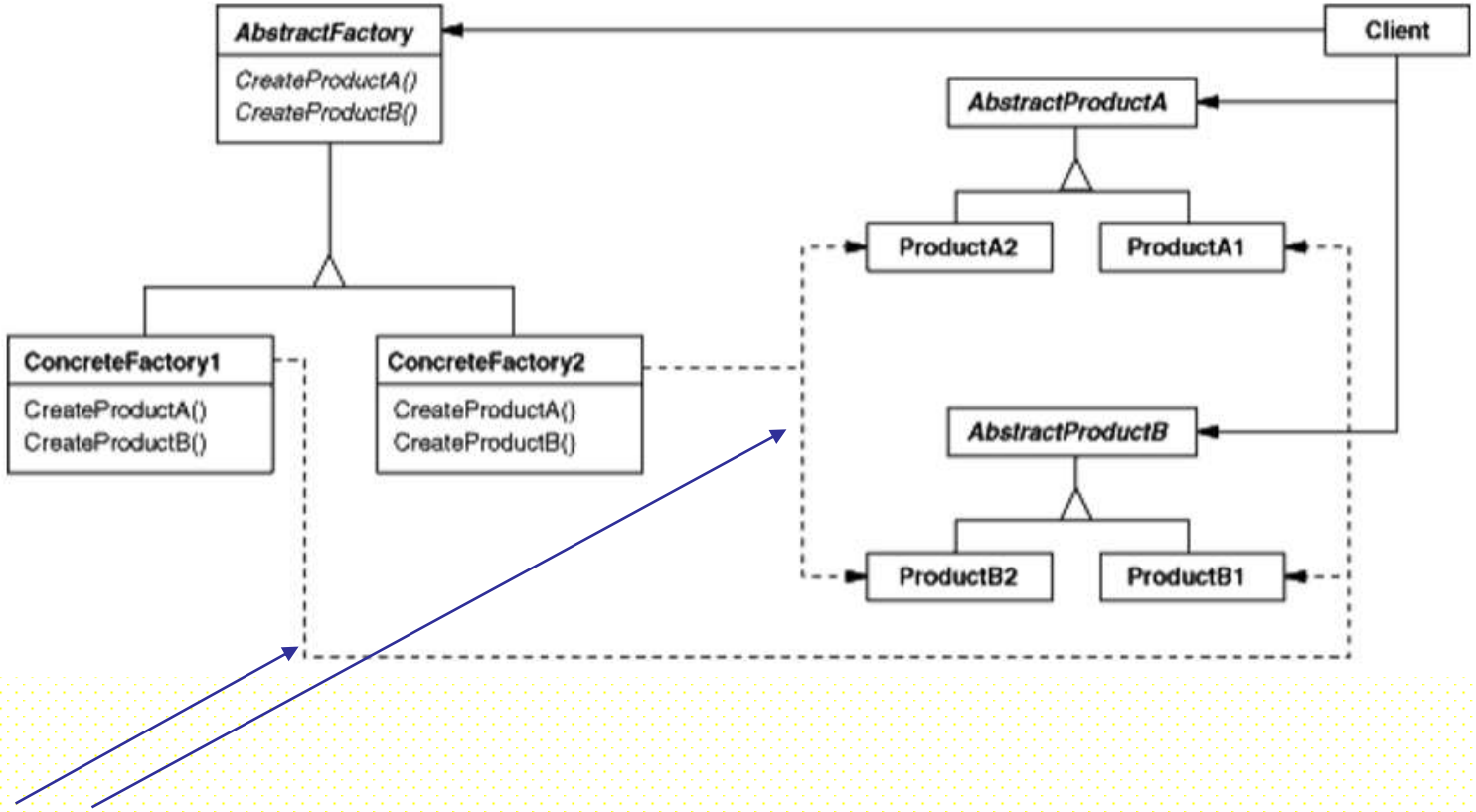
Problem Statement :

Implement a notification service through email, SMS, and push notification to be printed in shapes circle, square, rectangle respectively. Implement this with the help of Abstract Factory design pattern.

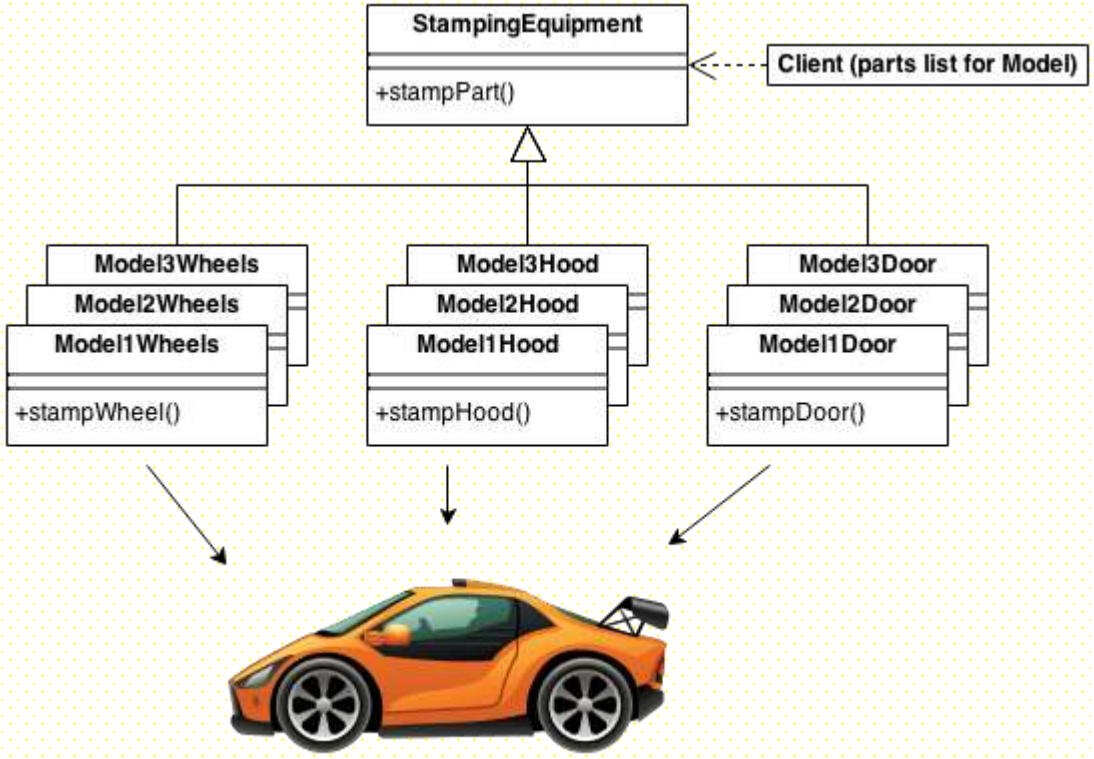


Abstract Factory Design Patterns

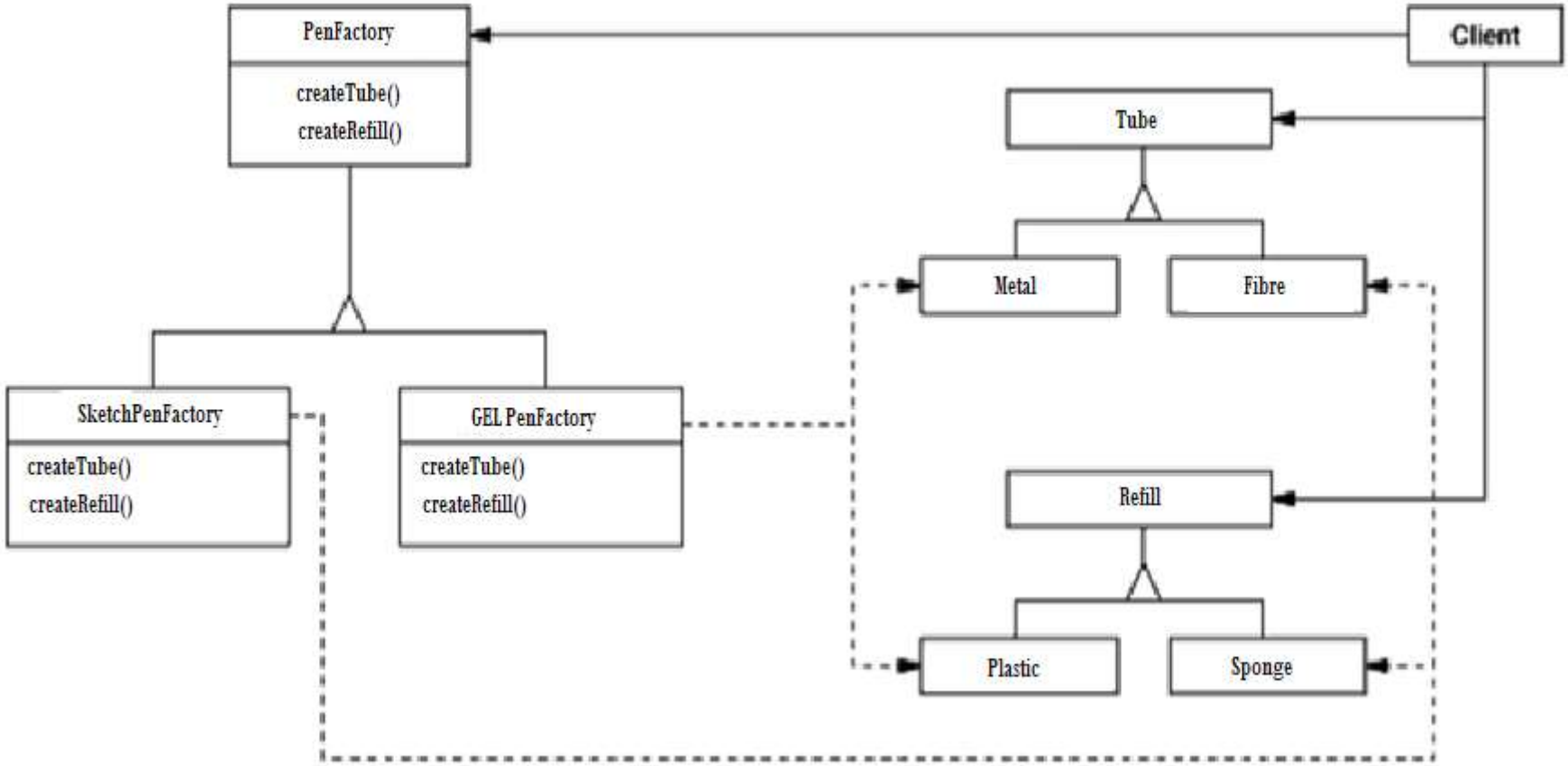
Structure



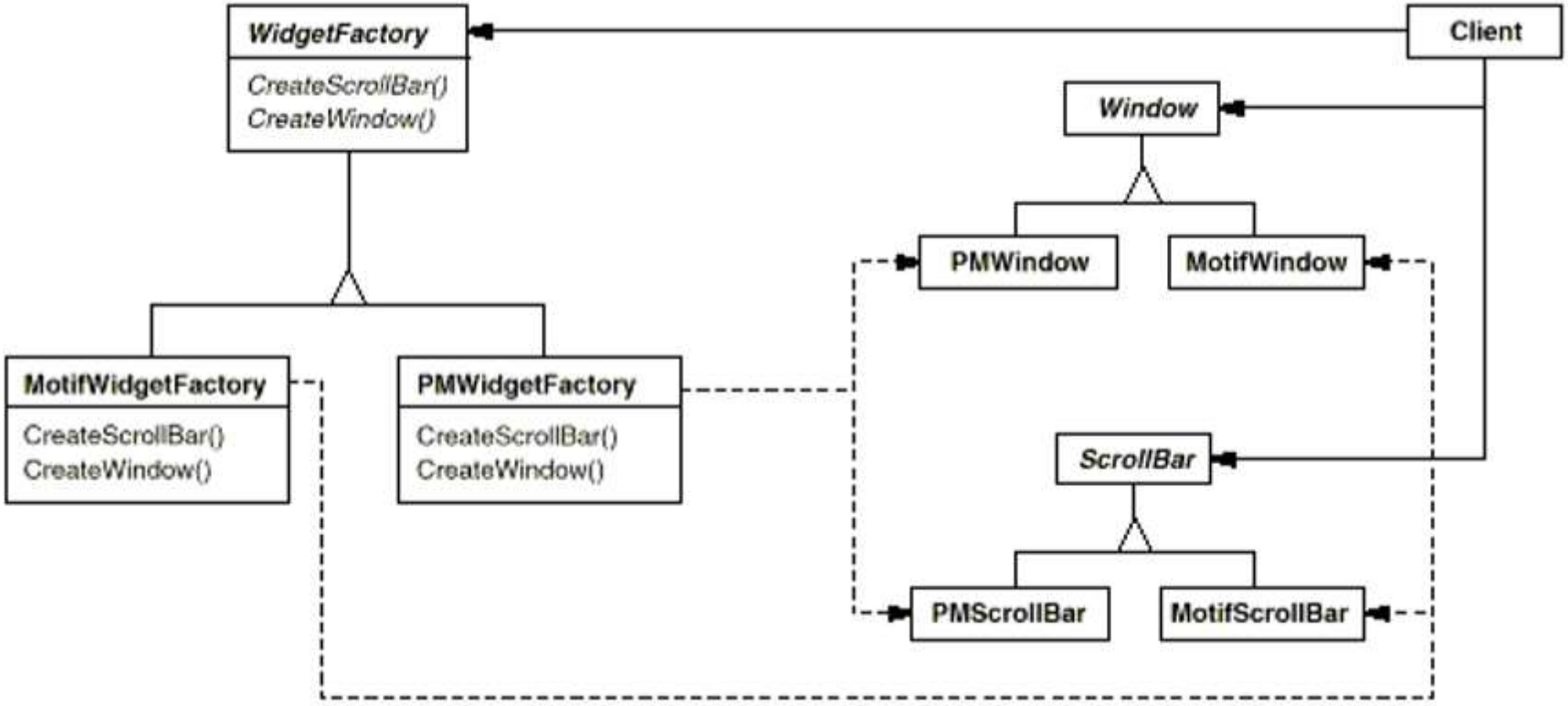
Abstract Factory Example



ABSTRACT FACTORY EXAMPLE



MOTIVATION


WidgetFactory
CreateScrollBar()
CreateWindow()
Client
Window
PMWindow
MotifWindow
MotifWidgetFactory
CreateScrollBar()
CreateWindow()
PMWidgetFactory
CreateScrollBar()
CreateWindow()
ScrollBar
PMScrollBar
MotifScrollBar

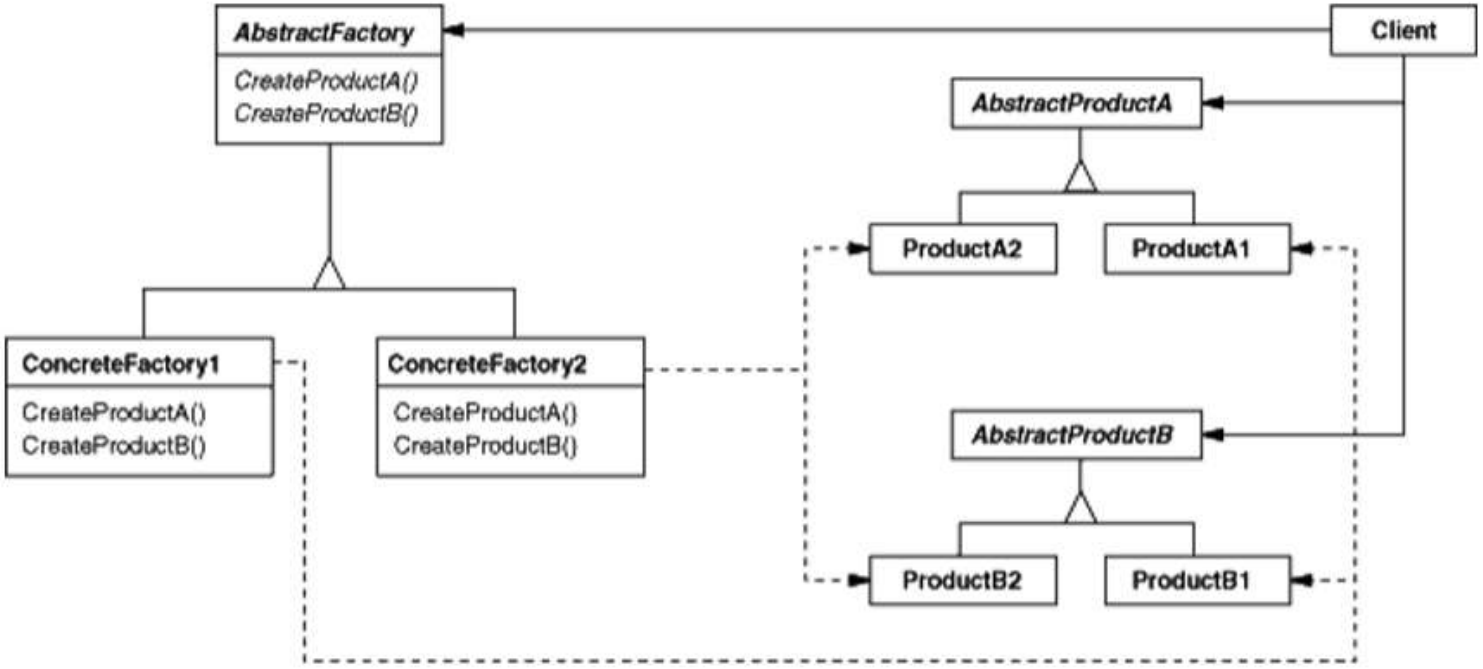
APPLICABILITY

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.

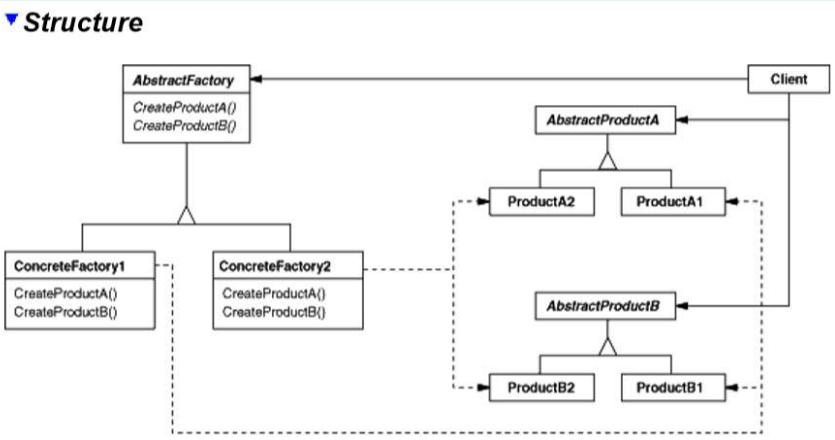
Abstract Factory Design Patterns

Structure



PARTICIPANTS

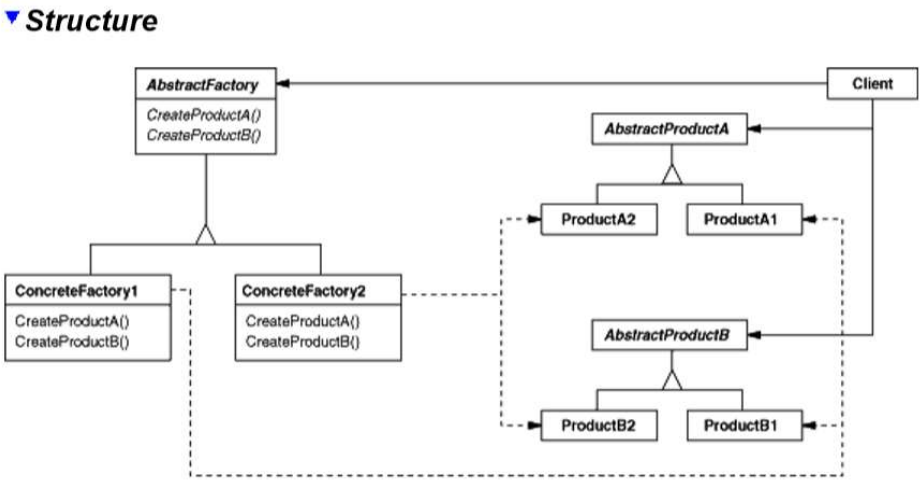
Participant	Responsibility
AbstractFactory	Declares an interface for operations that create abstract product objects.
ConcreteFactory	Implements the operations to create concrete product objects.
AbstractProduct	Declares an interface for a type of product object.
ConcreteProduct	• Defines a product object to be created by the corresponding concrete factory. • Implements the AbstractProduct interface.
Client	Uses only interfaces declared by AbstractFactory and AbstractProduct classes.



COLLABORATIONS

• Normally a single instance of a **ConcreteFactory** class is created at run-time. This concrete factory creates product objects having a particular implementation. To create different product objects, clients should use a different concrete factory.

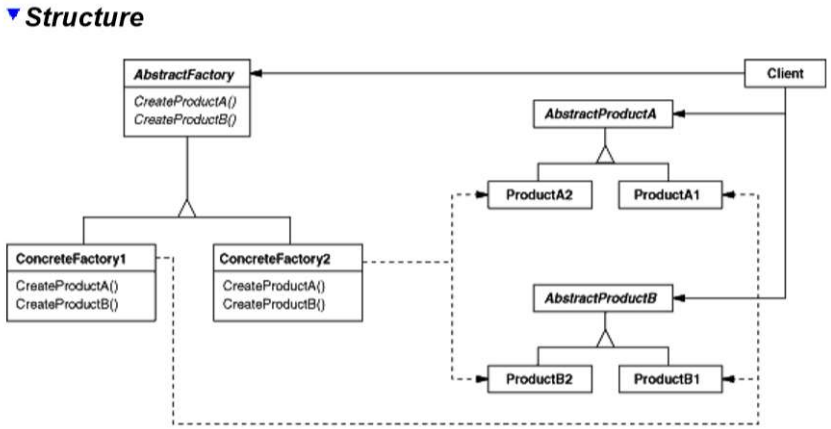
• **AbstractFactory** defers creation of product objects to its **ConcreteFactory** subclass.



CONSEQUENCES

1. **It isolates concrete classes.** The Abstract Factory pattern helps you control the classes of objects that an application creates. Product class names are isolated in the implementation of the concrete factory; they do not appear in client code.

2. **It makes exchanging product families easy.** The class of a concrete factory appears only once in an application—that is, where it's instantiated. This makes it easy to change the concrete factory an application uses. It can use different product configurations simply by changing the concrete factory.

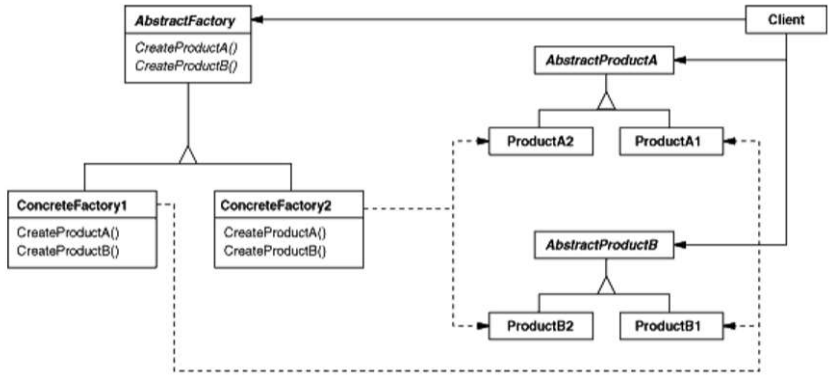


CONSEQUENCES

3. It promotes consistency among products. When product objects in a family are designed to work together, it's important that an application use objects from only one family at a time. **AbstractFactory** makes this easy to enforce.

4. Supporting new kinds of products is difficult. Extending abstract factories to produce new kinds of Products isn't easy. That's because the **AbstractFactory** interface fixes the set of products that can be created. Supporting new kinds of products requires extending the factory interface, which involves changing the **AbstractFactory** class and all of its subclasses.

Structure



IMPLEMENTATION

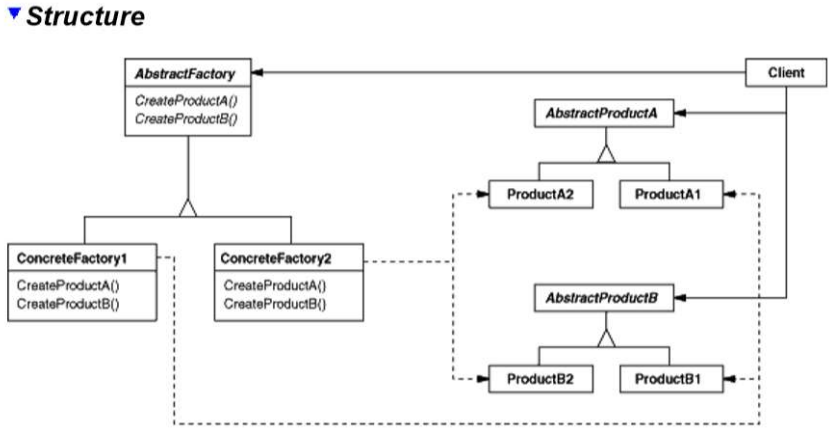
Implementation Consider the following issues when applying the Abstract Factory pattern:

1. **Factories as singletons**

   An application typically needs only one instance of a ConcreteFactory per product family. So it's usually best implemented as a Singleton.

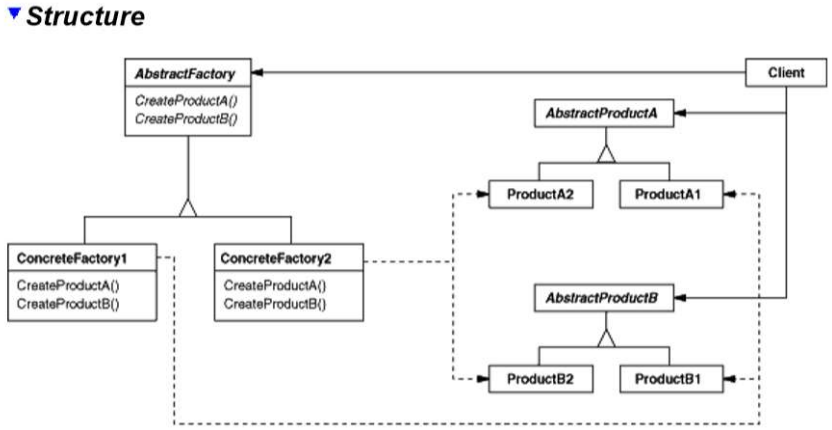
2. **Creating the products**

   AbstractFactory only declares an interface for creating products. The most common way to do this is to define a **factory method** for each product. While this implementation is simple, it requires a new concrete factory subclass for each product family, even if the product families differ only slightly.



IMPLEMENTATION

3. Defining extensible factories. AbstractFactory usually defines a different **operation** for each kind of **product** it can produce. The kinds of products are encoded in the operation signatures. Adding a new kind of product requires changing the AbstractFactory **interface** and all the classes that depend on it.



KNOWN USES & RELATED PATTERNS

- WidgetKit and DialogKit for generating look-and-feel-specific user interface objects

- **Factory method DP**
- **Prototype DP**
- **Singleton**