Abstract Factory Pattern

Jim Fawcett

CSE776 – Design Patterns

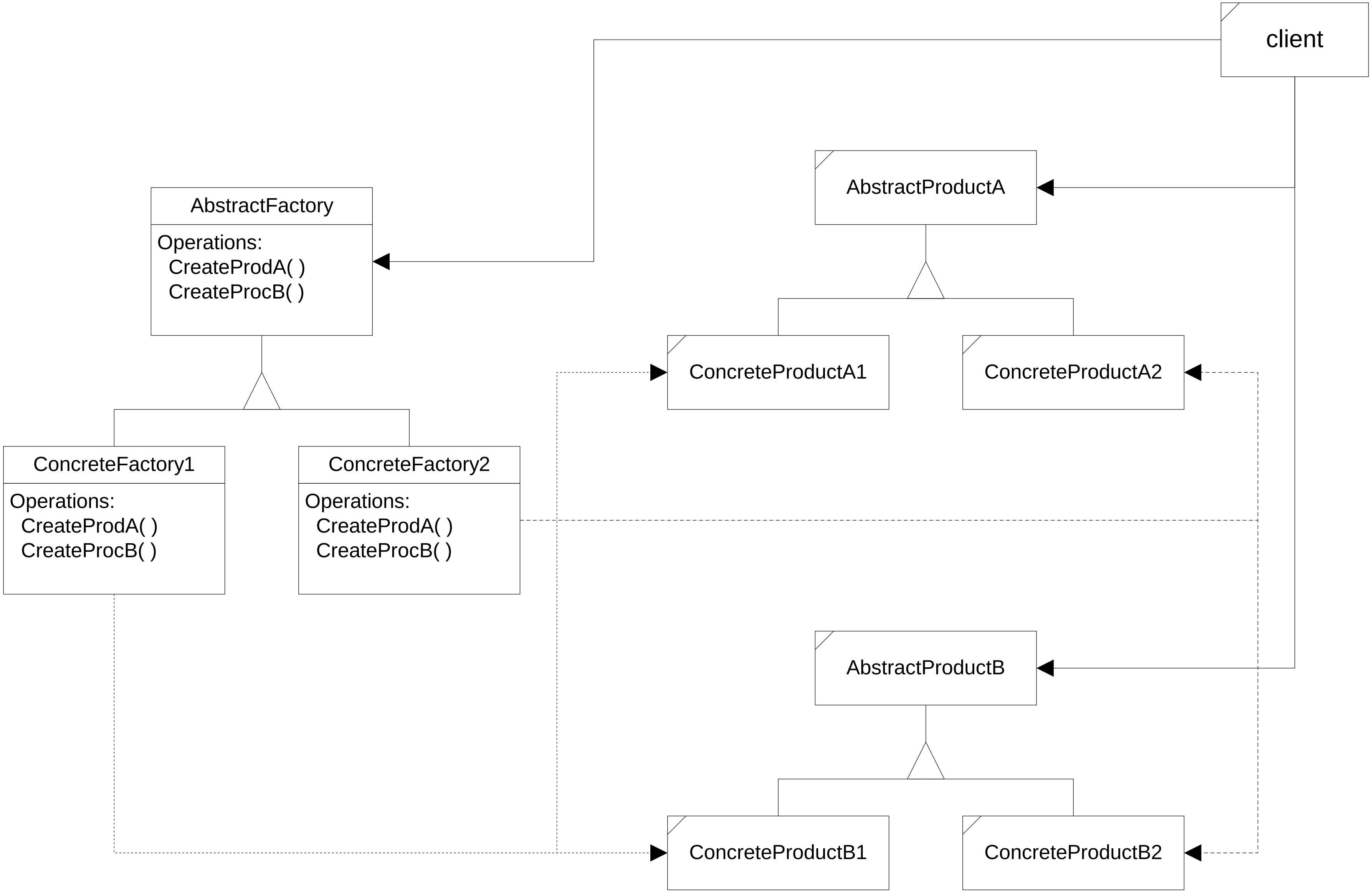
Fall 2017

Intent

- “Provide an interface for creating families of related or dependent objects without specifying their concrete classes.”
  - provide a simple creational interface for a complex family of classes
    - Client does not have to know any of those details.
  - avoid naming concrete classes
    - Clients use abstract creational interfaces and abstract product interfaces. Concrete classes can be changed without affecting clients.
    - Clients can stay blissfully unaware of implementation details
    - This is critically important!

Why is this so important?

Abstract Factory Structure



Motivating Examples

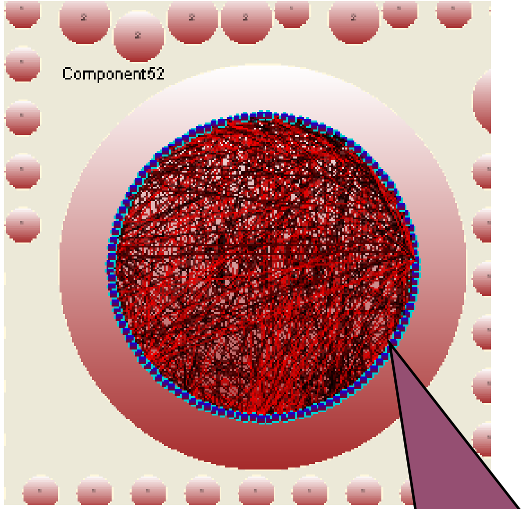
- Target multiple platforms by creating component instances for selected platform.

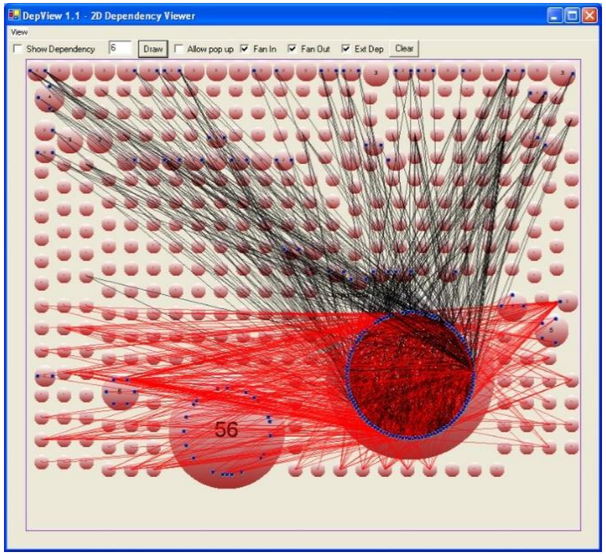
- Design neural network layers to contain the technology for their own construction, allowing networks to focus on learning strategies.

Forces

- The deep copy object model, used by the C++ language, binds a class to all the concrete classes it uses.
    - The class's header file holds information about supporting classes to allow them to be created and used.
    - If one of the serving classes changes, the using class must also change, and every class that uses this class must also change.
- To prevent this "top-to-bottom" binding, a class must bind to interface abstractions instead. It must also use factories as proxies to create the instances it uses.
- In a large system we will be likely to use interfaces and factories between subsystems. A factory will then need to manage creation of many of the objects within a subsystem.

Dependencies – Mozilla, version 1.4.1





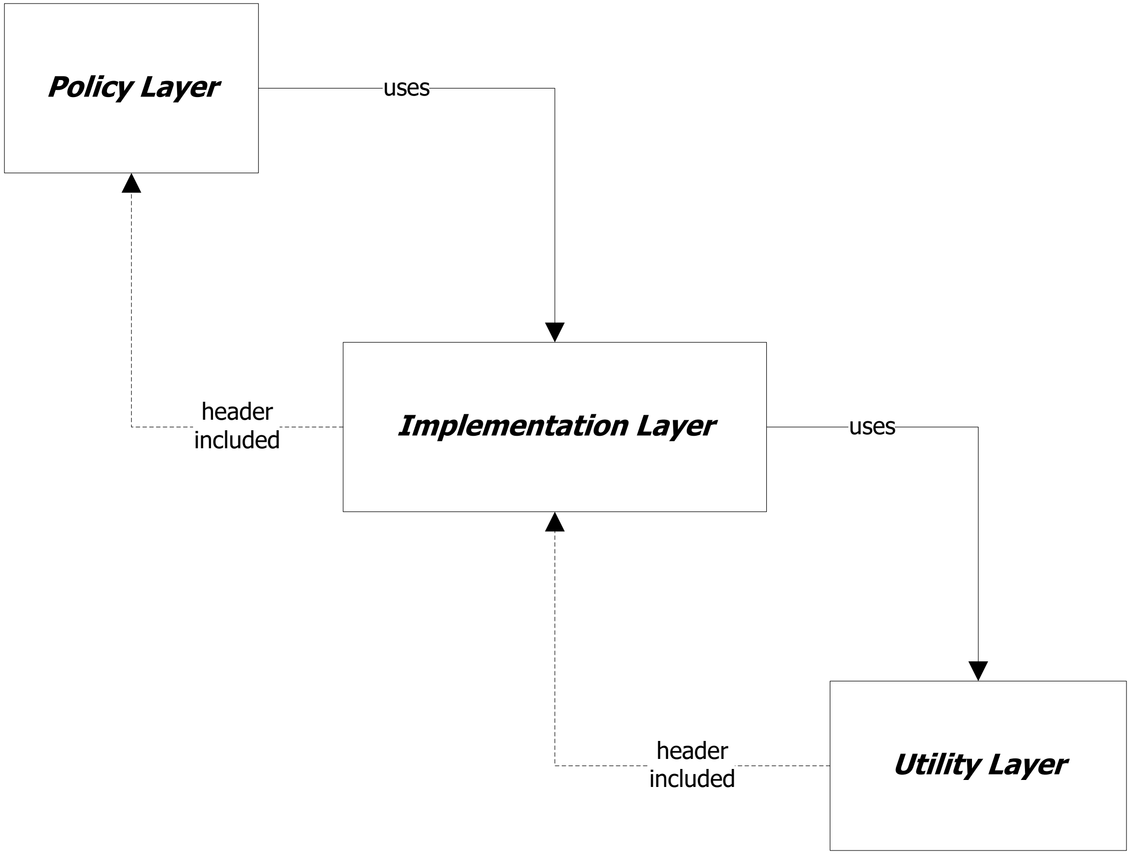
☐ Show Dependency 6 ☐ Allow pop up ☑ Fan In ☑ Fan Out ☑ Ext Dep

Strong component of dependency graph, e.g., set of mutually dependent files

Forces

- Languages like C# and Java have less need for factories due to their shallow reference object model.
    - The physical layout of code for a given class does not depend on the sizes of the objects it uses, unlike C++.
    - The new operator extracts information about how to build an instance from its class's metadata, not from the classes that use it, unlike C++.
- However, even for these languages the Abstract Factory Pattern is still useful!
    - Interfaces support the exchange of implementing types, even at run-time, which is often very useful.
    - Factories support binding of specific objects to these interfaces, without requiring clients to have knowledge of the specific types.

Logical System Layering



Policy Layer

uses

Implementation Layer

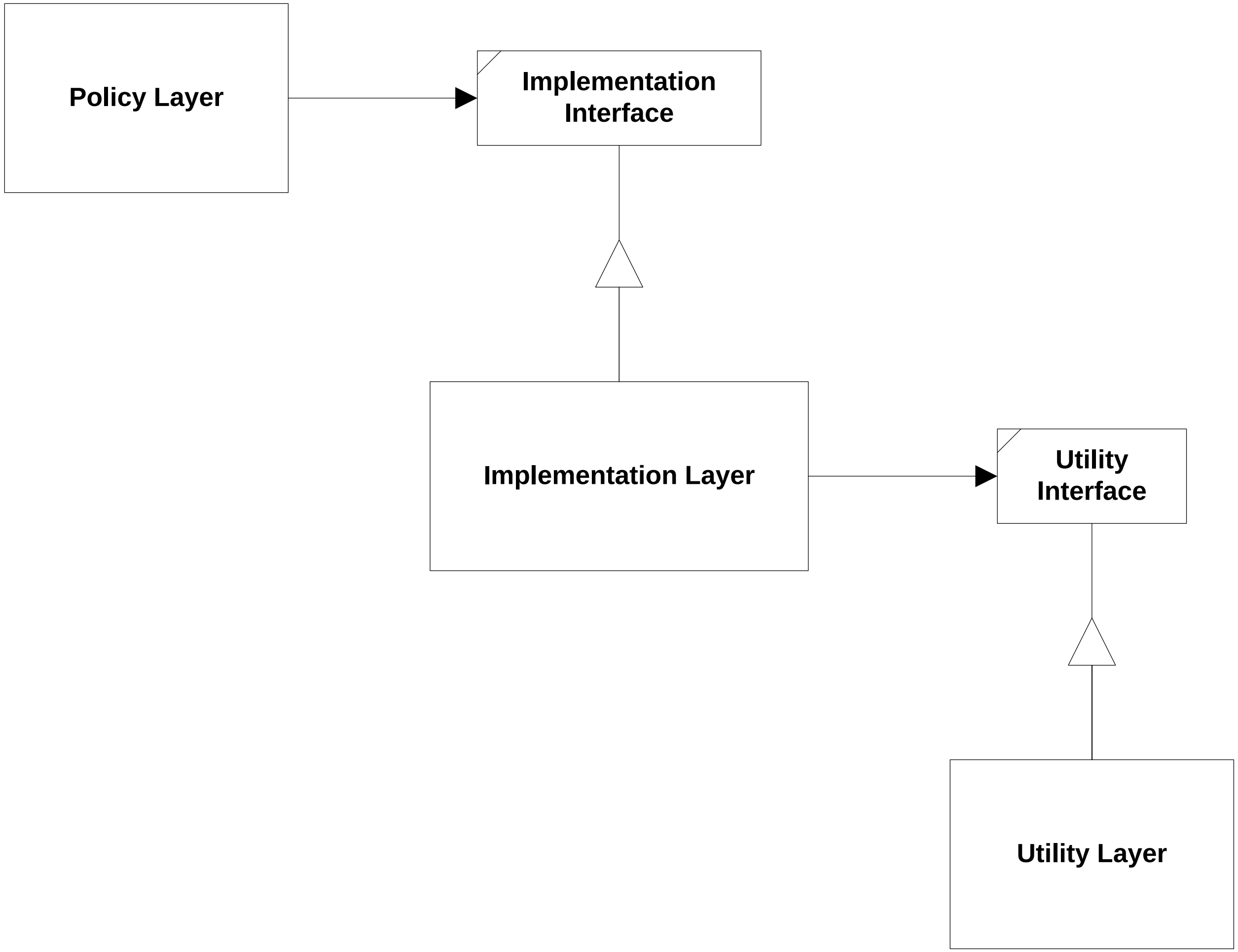
header included

uses

Utility Layer

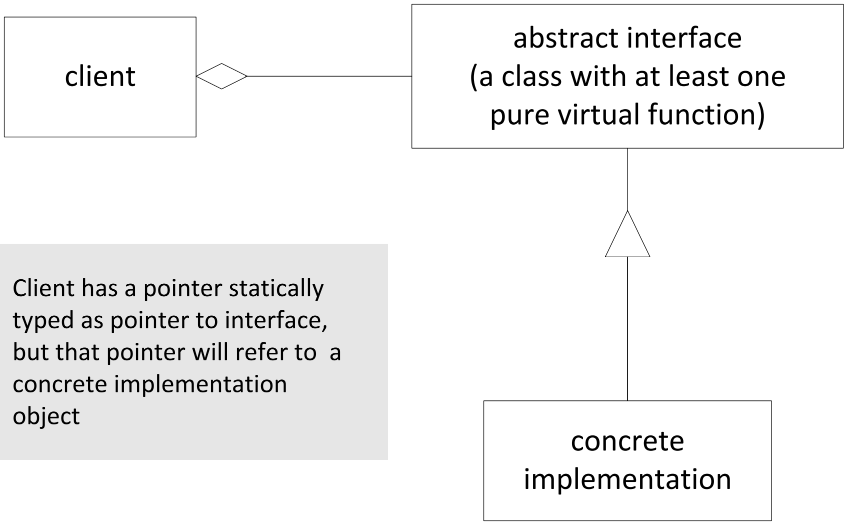
header included

Dependency Inversion Principle



Abstract Interface

Fact:
This client will be compile-time independent of the concrete implementation if, and only if, it does not directly create an instance of the concrete class



client

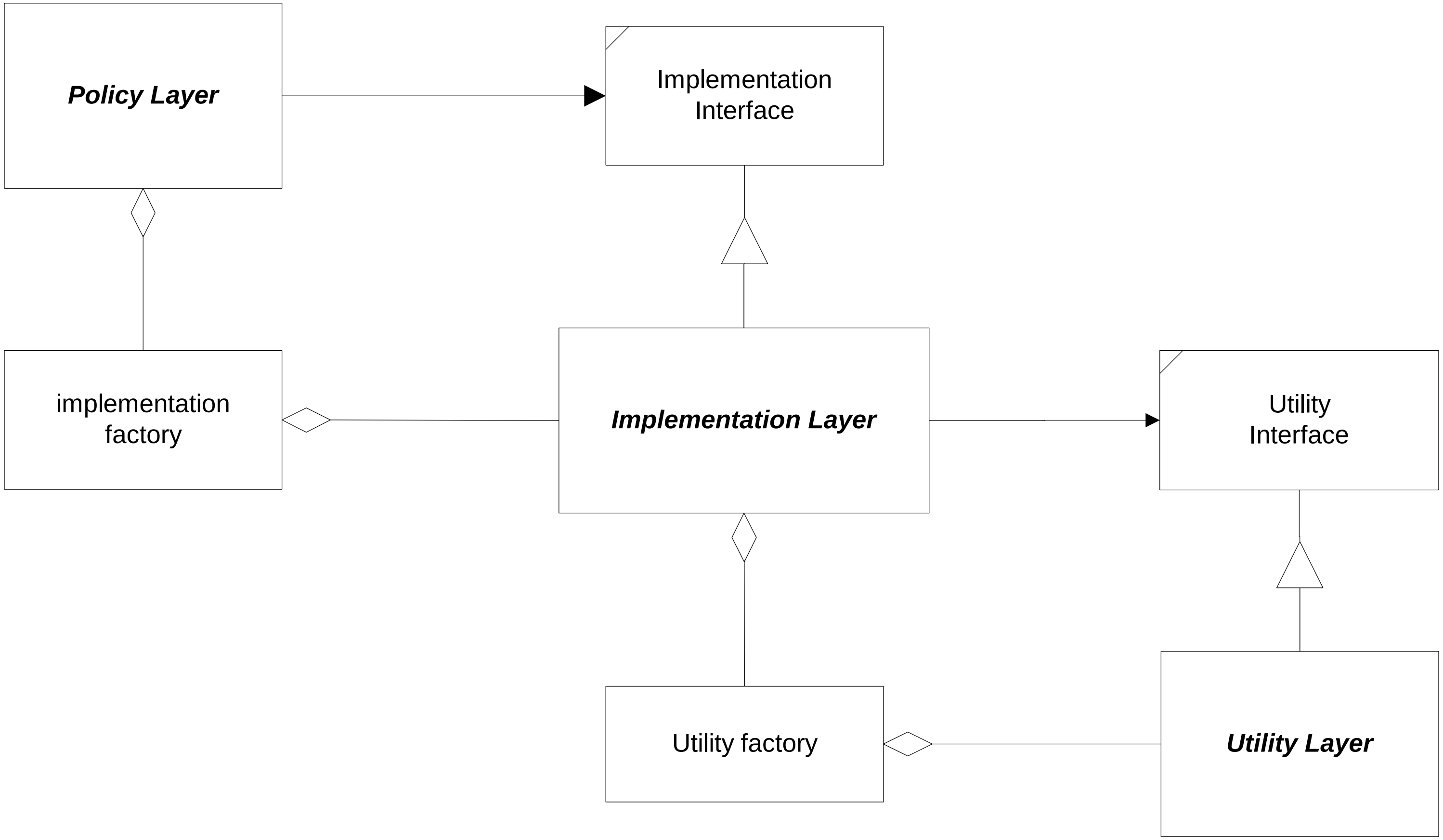
abstract interface
(a class with at least one pure virtual function)

concrete implementation

The purpose of an abstract interface is to provide a protocol for clients to use to request service from concrete objects without coupling to their implementations

Client has a pointer statically typed as pointer to interface, but that pointer will refer to a concrete implementation object

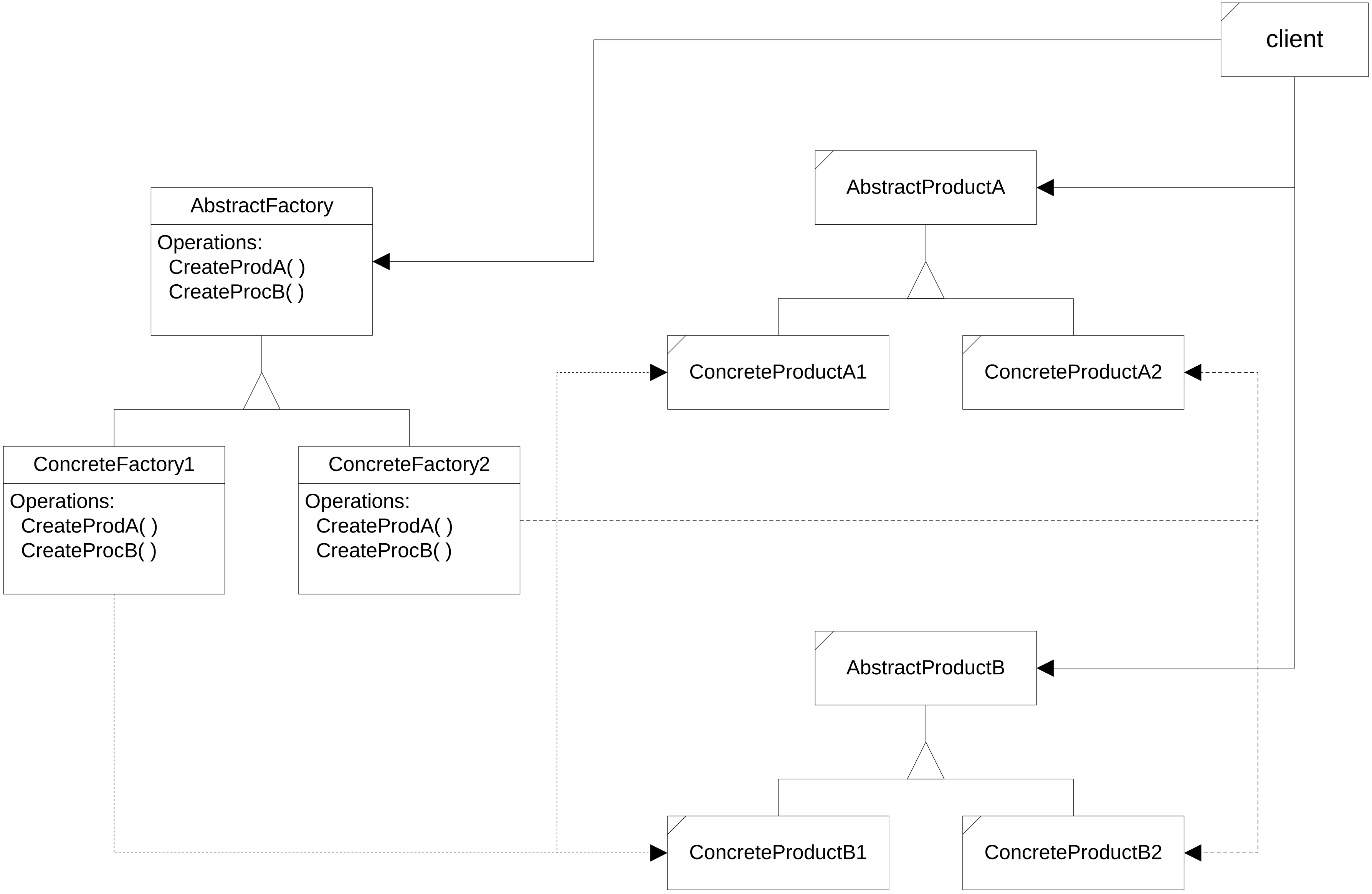
Layering with Abstract Interfaces



Applicability

- Use the Abstract Factory Pattern if:
  - clients need to be ignorant of how servers are created, composed, and represented.
  - clients need to operate with one of several families of products
  - a family of products must be used together, not mixed with products of other families.
  - you provide a library and want to show just the interface, not implementation of the library components.
    - Giving customers your product header files may disclose some of your proprietary value.

Abstract Factory Structure



Participants

Why is there a parallel hierarchy of factories and products?

- AbstractFactory
  - provide an interface for building product objects
- ConcreteFactory
  - implements the creation functionality for a specific product family
- AbstractProduct
  - provides an interface for using product objects
- ConcreteProduct
  - created by a ConcreteFactory, implements the AbstractProduct interface for a specific product family
- Client
  - uses only abstract interfaces so is independent of the implemen-tation.

Collaborators

- Usually only one ConcreteFactory instance is used for an activation, matched to a specific application context. It builds a specific product family for client use -- the client doesn’t care which family is used -- it simply needs the services appropriate for the current context.

- The client may use the AbstractFactory interface to initiate creation, or some other agent may use the AbstractFactory on the client’s behalf.

- The factory returns, to its clients, specific product instances bound to the product interface. This is what clients use for all access to the instances.

Presentation Remark

- Here, we often use a sequence diagram (event-trace) to show the dynamic interactions between participants.

- For the Abstract Factory Pattern, the dynamic interaction is simple, and a sequence diagram would not add much new information.

Consequences

- The Abstract Factory Pattern has the following benefits:
  - It isolates concrete classes from the client.
    - You use the Abstract Factory to control the classes of objects the client creates.
    - Product names are isolated in the implementation of the ConcreteFactory, clients use the instances through their abstract interfaces.
  - Exchanging product families is easy.
    - None of the client code breaks because the abstract interfaces don’t change.
    - Because the abstract factory creates a complete family of products, the whole product family changes when the concrete factory is changed.
  - It promotes consistency among products.
    - It is the concrete factory’s job to make sure that the right products are used together.

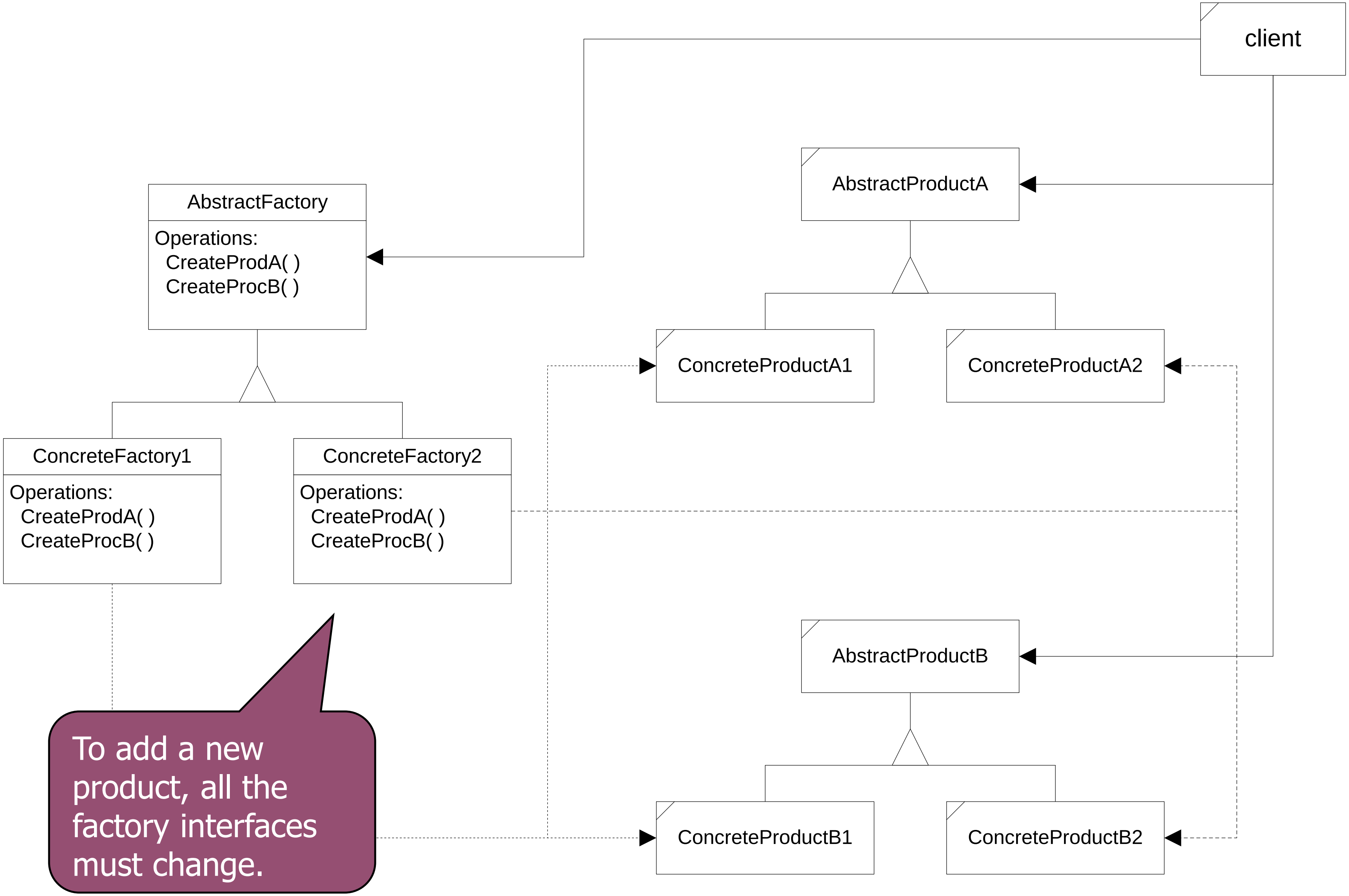
Consequences

- More benefits of the Abstract Factory Pattern
  - It supports the imposition of constraints on product families, e.g., always use A1 and B1 together, otherwise use A2 and B2 together.

Consequences

- The Abstract Factory pattern has the following liability:
    - Adding new kinds of products to existing factory is difficult.
        - Adding a new product requires extending the abstract interface which implies that all of its derived concrete classes also must change.
        - Essentially everything must change to support and use the new product family
            - abstract factory interface is extended
            - derived concrete factories must implement the extensions
            - a new abstract product class is added
            - a new product implementation is added
            - client has to be extended to use the new product

Abstract Factory Structure



To add a new product, all the factory interfaces must change.

Implementation

- Concrete factories are often implemented as singletons.

- Creating the products
  - Concrete factory usually use the factory method.
    - simple
    - new concrete factory is required for each product family

  - alternately concrete factory can be implemented using prototype.
    - only one is needed for all families of products
    - product classes now have special requirements - they participate in the creation

Implementation

- Defining extensible factories by using create function with an argument
  - only one virtual create function is needed for the AbstractFactory interface
  - all products created by a factory must have the same base class or be able to be safely coerced to a given type
  - it is difficult to implement subclass specific operations

Know Uses

- Interviews
  - used to generate “look and feel” for specific user interface objects
  - uses the Kit suffix to denote AbstractFactory classes, e.g., WidgetKit and DialogKit.
  - also includes a layoutKit that generates different composite objects depending on the needs of the current context

- ET++
  - another windowing library that uses the AbstractFactory to achieve portability across different window systems (X Windows and SunView).

- COM – Microsoft’s Component Object Model technology
  - Each COM component provides a concrete factory bound to the IClassFactory interface and provides clients specific instances bound to the server’s product interface.

Related Patterns

- Factory Method -- a “virtual” constructor
- Prototype -- asks products to clone themselves
- Singleton -- allows creation of only a single instance

Code Examples

- Skeleton Example
  - Abstract Factory Structure
  - Skeleton Code

- Neural Net Example
  - Neural Net Physical Structure
  - Neural Net Logical Structure
  - Simulated Neural Net Example

End of Presentation