Design Pattern

- Christopher Alexander (1977, buildings / towns):
  - "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way".
- At high-level design: Architectural Styles
- At low-level design: Design Patterns
- Encourages design reuse (intended for OO)
- Used as a means for transferring knowledge from experienced designers to novice designers

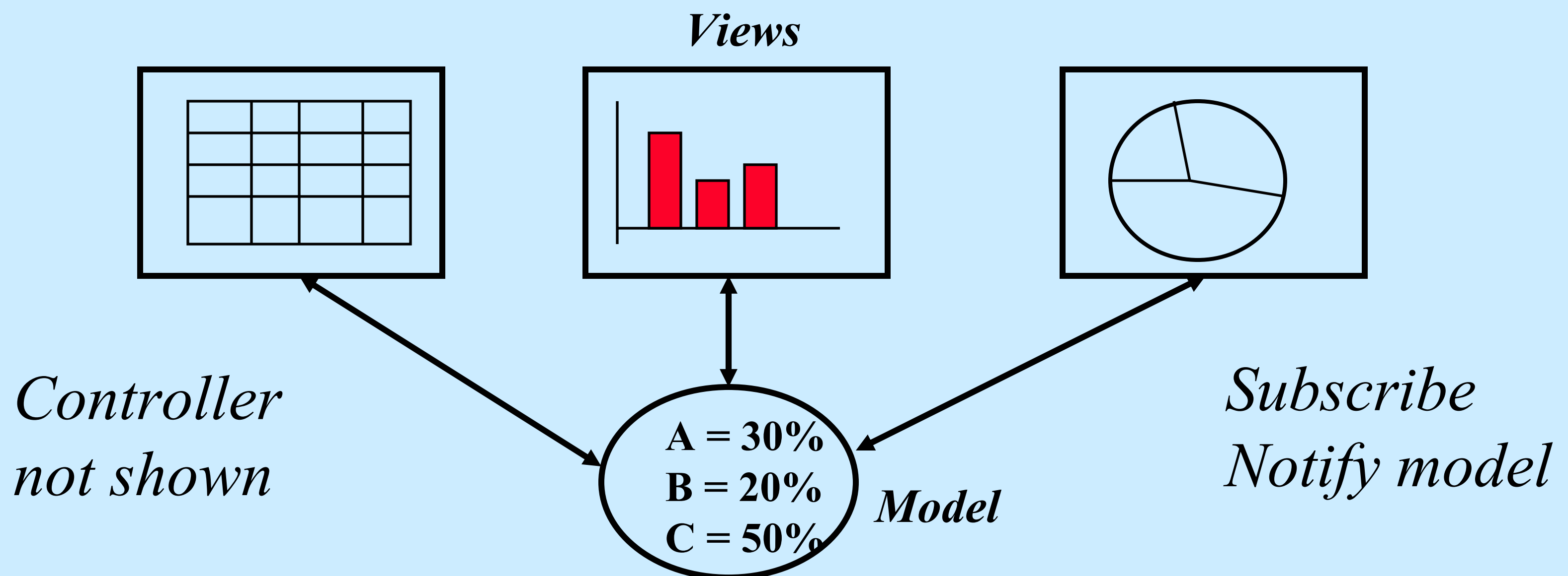
Design pattern definition

- Describe the objects and classes (can be modules) that communicate with each other and are customized to solve a general design problem in a particular context.
- A pattern has four essential elements:
  - Name: a handle we can use to describe a design problem, its solution, and consequences in a word or two.
  - Problem: describes when to apply the pattern; can be a specific design problem or a set of conditions to exist.
  - Solution: describes the elements that make up the design, their relationships, responsibilities, and collaborations.
  - Consequences: are the results and trade-offs applying the pattern.

MVC Pattern in Smalltalk

MVC (Model / View / Controller) consists of:

- Model: the application program (object)
- View: its screen presentation
- Controller: defines the way the user interface reacts to user input.



MVC

- Model notifies the views when its value changes
- Views communicate with model to access new values
- The problem that MVC addresses:
  - Decoupling objects so that changes to one can affect any number of others without requiring the changed object to know details of the others. Observer design pattern.
- MVC: a control panel of buttons, ie, nested view:
  - A design pattern that let us treat a composite view just like we treat one of its components. Composite design pattern.
- MVC: possible to change a view's controller at run-time
  - Lets you change the way a view responds to user input without changing its visual presentation. Strategy design pattern

Describing Design Patterns

- Pattern name and classification
  - Conveys the essence of the pattern (Creatinal, Structural, Behavioral)
- Intent: short statement to answer:
  - What particular design issue it addresses? What is its rationale?
- Also known as:
  - other name for the pattern
- Motivation:
  - A scenario that illustrates a design problem
- Applicability:
  - In what situations the pattern applies?
- Structure:
  - Graphical representation of the classes in OMT notation

Describing Design Patterns ….

- Participants:
  - Classes and their responsibilities
- Collaborations:
  - Collaboration among participants
- Consequences:
  - How does the pattern support its objectives?
- Implementation:
  - Pitfalls, hints, techniques to be aware of when implementing
- Sample code:
  - Code in C++ , Smalltalk, Java to implement pattern
- Known uses:
  - Examples of the pattern found in real systems in other domains.
- Related patterns

The Catalog of Design Pattern

- Abstract Factory
  - Provide an interface for creating families of related or dependent objects without specifying their concrete classes
  - Example: a user-interface toolkit that supports multiple look-and-feel standards such as Motif and Open-window.
- Adapter
  - Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces
  - Example: component programming
- Bridge
  - Decouple an abstraction from its implementation so that two can vary independently.
  - Example: implementation of a portable window abstraction in a user interface toolkit. The abstraction should enable us to write applications that work on both the X window system and Presentation Manager.

The Catalog of Design Pattern …

- Decorator
  - Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub-classing for extending functionality.
  - Example: a graphical user interface toolkit should let you add properties like borders or behaviors like scrolling to any user interface component.
- Façade
  - Provide a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use.
  - Example: consider a compiler component having scanner, parser, … Some specialized applications may need to access these classes directly. But most clients of compiler only want to compile the program and don't care about the details of compiler.

The Catalog of Design Pattern …

- Iterator
  - Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation
  - Example: an aggregate object such as a list should provide a way to access its elements without exposing its internal structure. You should be able to traverse the list in either direction.
- Mediator
  - Define an object that encapsulates how a set of object interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

The Catalog of Design Pattern …

- Strategy
  - Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.
  - Example: many algorithms exist for breaking a stream of text into lines. Hard coding all such algorithms into the classes that require them is not desirable.
- Visitor
  - Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

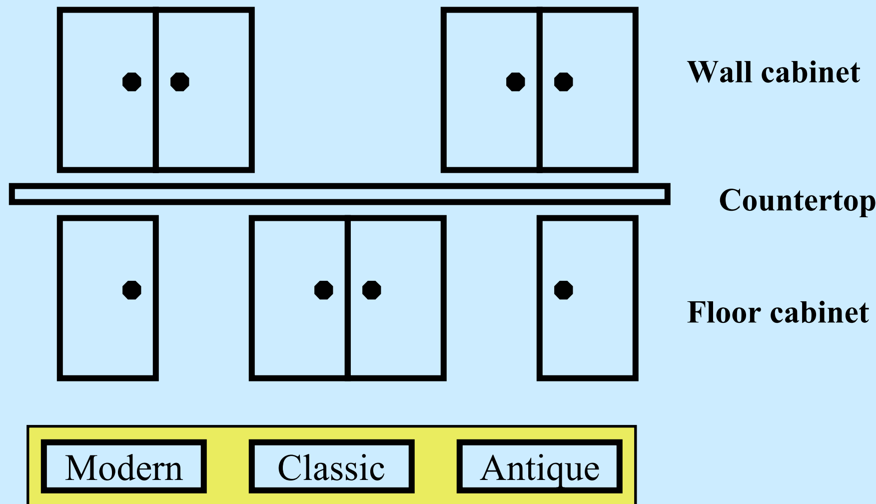
Organizing the Catalog

		Purpose		
		Creational	*Structural*	*Behavioral*
Scope	*Class*	Factory Method	Adapter	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Façade Flyweight Proxy	Chain Command Iterator Mediator Memento Observer State Strategy Visitor

Abstract Factory Design Pattern

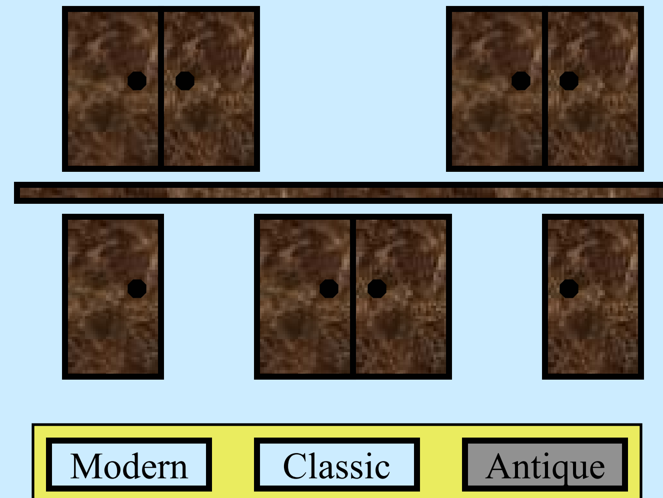
- Intent:
  - Provide an interface for creating families of related or dependent objects without specifying their concrete classes
- Motivation:
  - A user-interface toolkit that supports multiple look-and-feel standards such as Motif and Open-window.
  - A Kitchen-viewer software that allows the home owners to choose between two styles modern an antique for their wall and floor kitchen-cabinets and then view the kitchen.

Abstract Factory Pattern (Kitchen View Example)

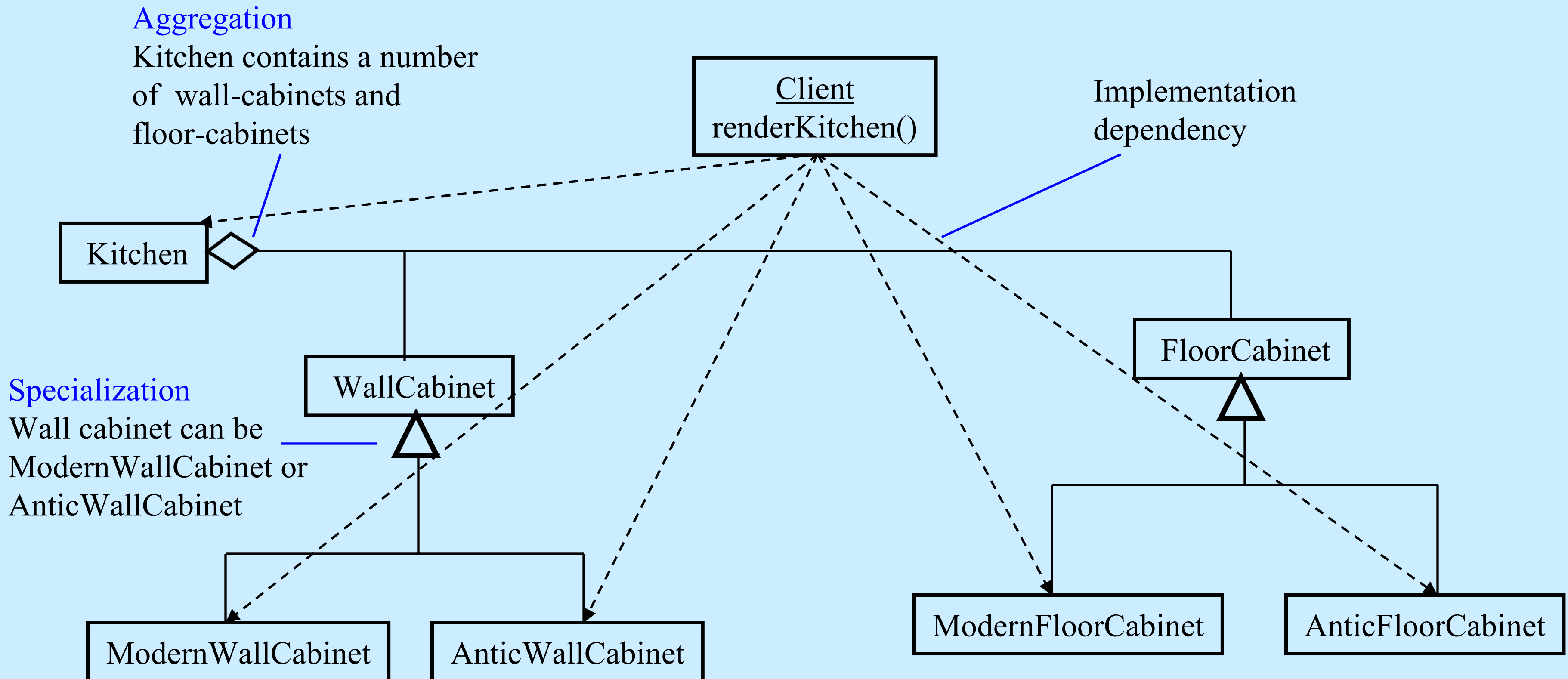


Provide an interface for creating families of related or dependent objects without specifying their concrete classes

Example: a user-interface toolkit that supports multiple look-and-feel standards such as Motif and Open-window.

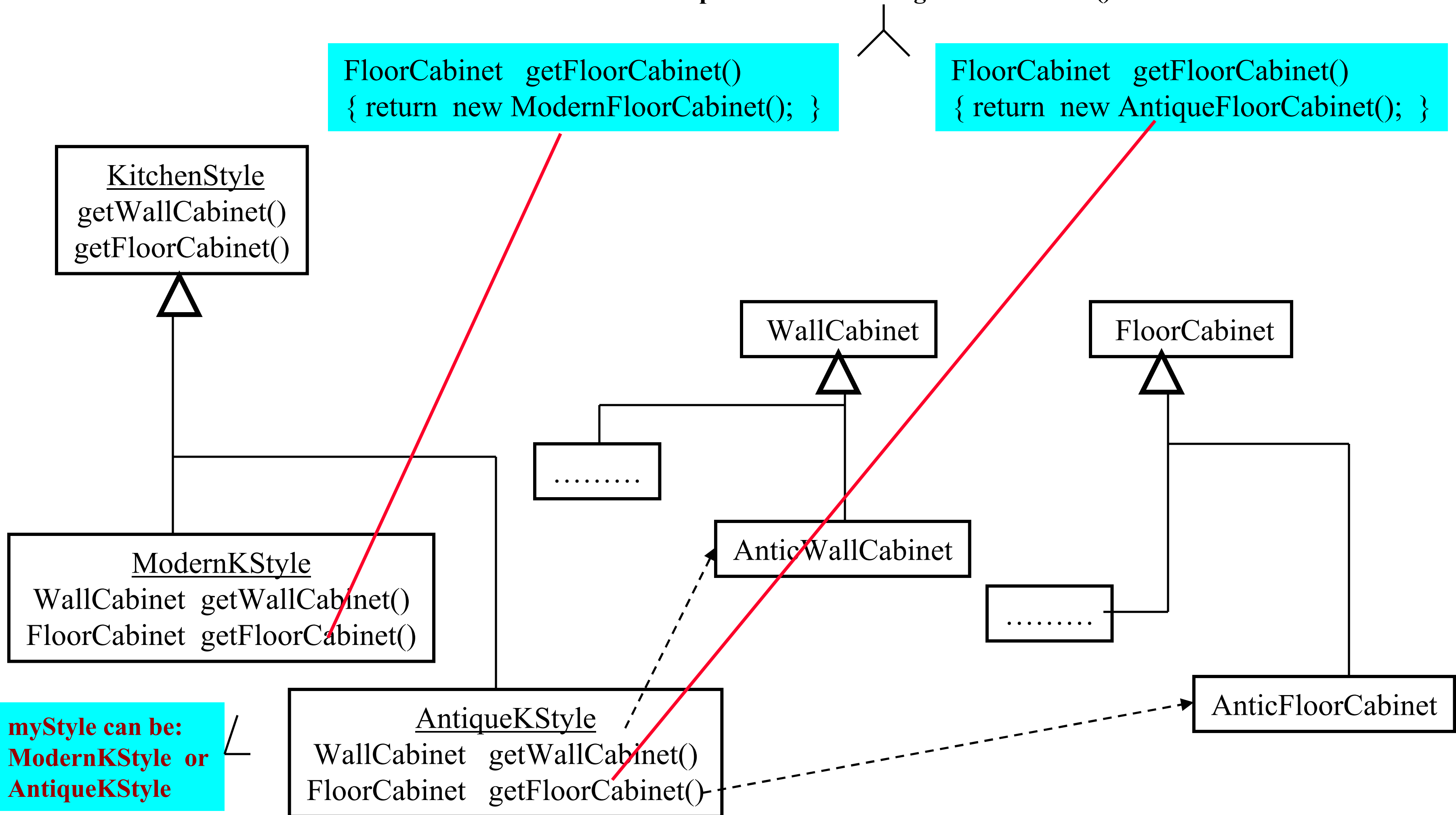


Kitchen viewer without Design Pattern

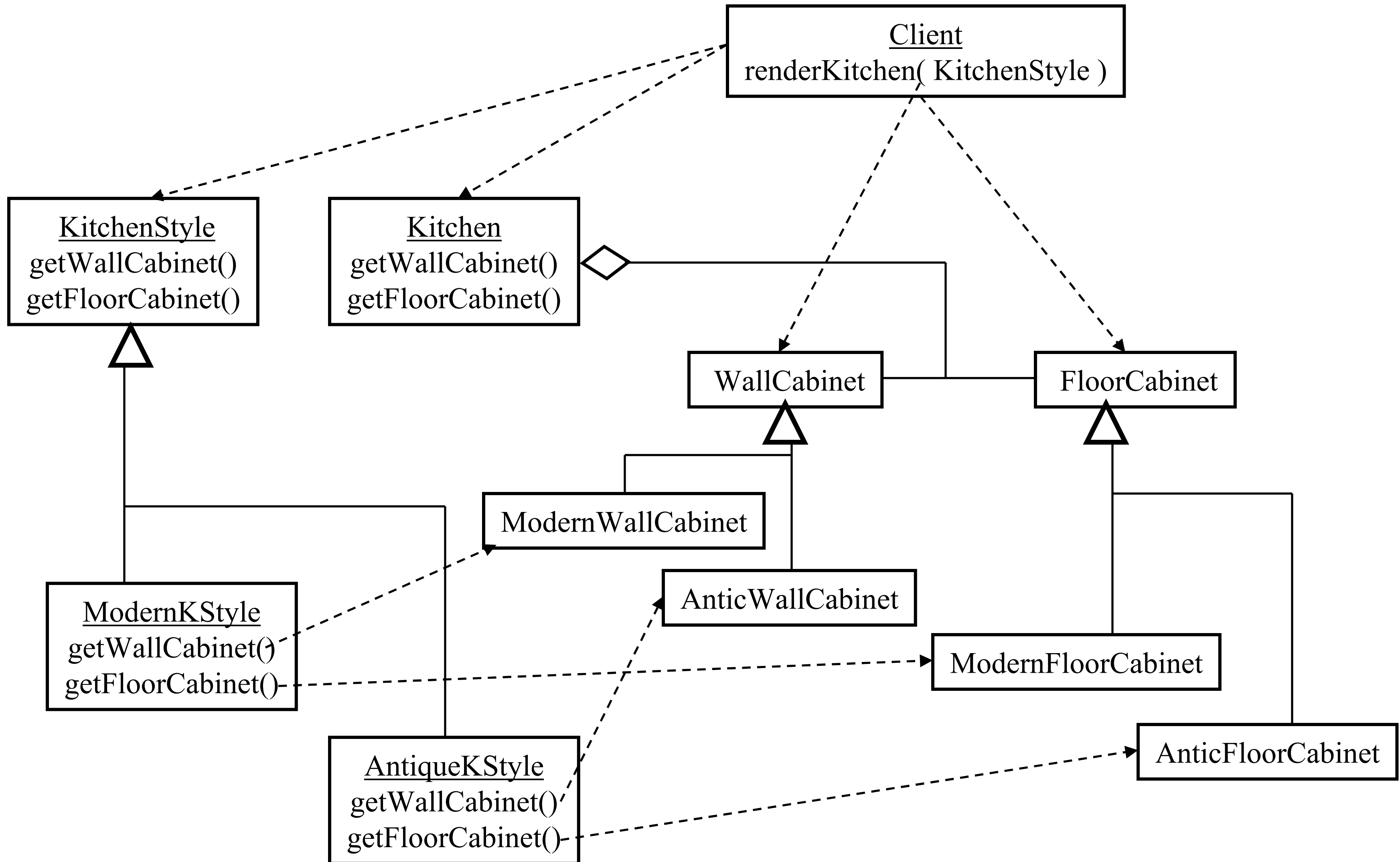


Abstract Factory Design Pattern idea

Applied to KitchenView



Abstract Factory Design Pattern Applied to KitchenView

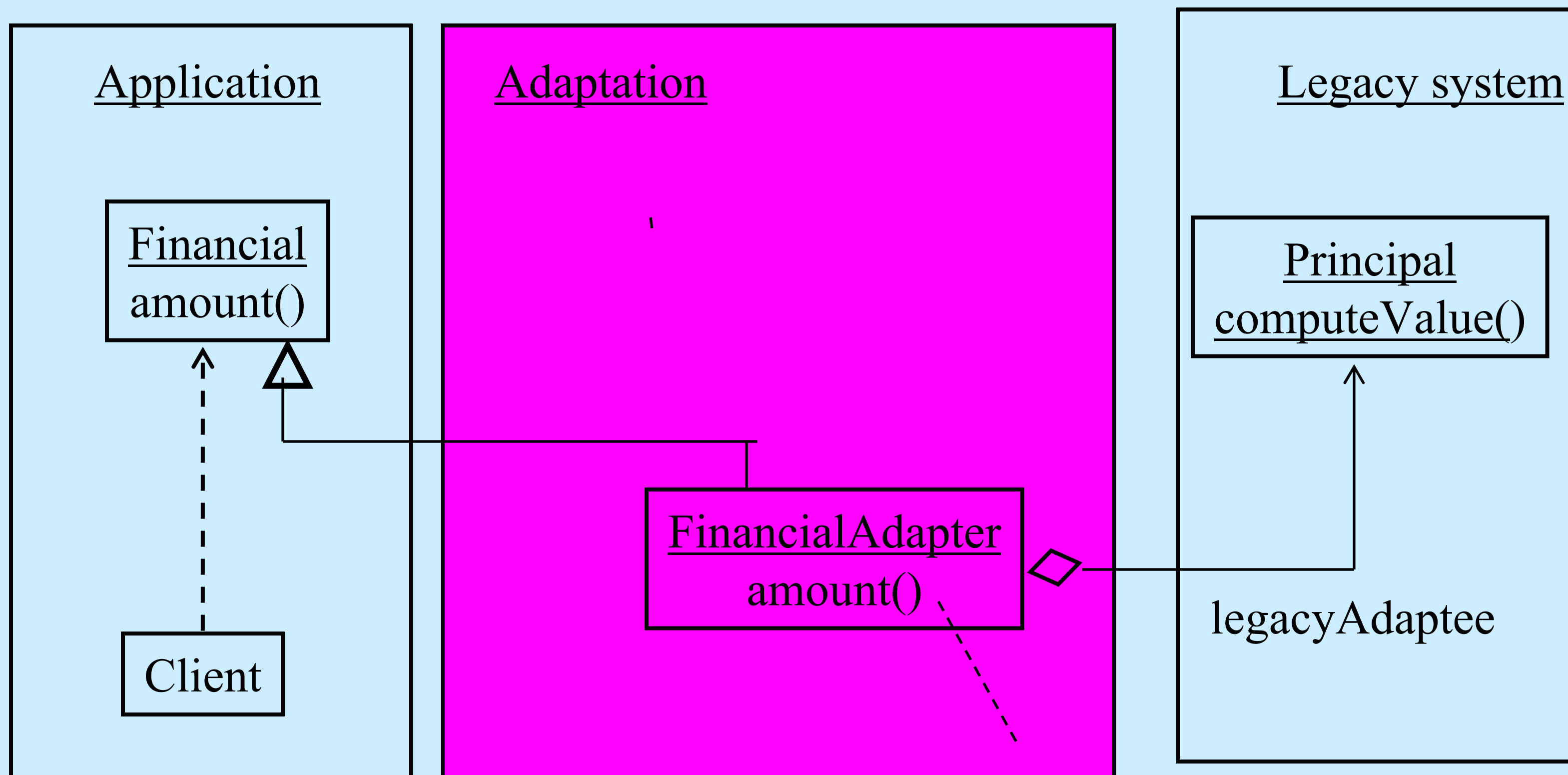


Adapter Design Pattern

- Intent:
  - Convert the interface of a class into another interface that the clients expect.
- Motivation:
  - We want to use the functionality provided by an existing application.
  - We want to modify our application as little as possible
  - We want to be able to easily switch to alternative implementations.
  - Example: an existing financial application computes the principal money obtained from investing a given amount of money for a given number of years in a special type of investment

Example of Adapter: Financial application

Legacy application …..: computeValue (float years, float interest, float amount)
New application …….: amount (float originalAmount, float numYears, float intRate)



{ adaptee.requiredMethod (); }