Design Patterns

Your friendly assistant

(<name>@inf.ethz.ch)

Chair of Programming Methodology

The slides in this section are based on
“.NET Design Patterns in C#”

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

1. **What are Design Patterns?**
2. Why use Design Patterns?
3. Design Pattern Catalogue
4. Creational Patterns
5. Structural Patterns
6. Behavioral Patterns
7. Quiz

DESIGN PATTERNS

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

What are Design Patterns?

- Solutions to common design problems
  - Abstract recurring design structures
  - Comprise object interaction and structure
  - Distil design experience

1. What are Design Patterns?
2. **Why use Design Patterns?**
3. Design Pattern Catalogue
4. Creational Patterns
5. Structural Patterns
6. Behavioral Patterns
7. Quiz

DESIGN PATTERNS

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Why use Design Patterns?

- Abstraction
  - Explicit design information
- Flexibility
  - Refactoring
- Modularity
- Elegance

1. What are Design Patterns?
2. Why use Design Patterns?
3. **Design Pattern Catalogue**
4. Creational Patterns
5. Structural Patterns
6. Behavioral Patterns
7. Quiz

DESIGN PATTERNS

Design Pattern Catalogue

Design Patterns:

Elements of Reusable Object-Oriented Software

by

Erich Gamma, Richard Helm,

Ralph Johnson and John Vlissides

(Gang of Four or GoF)

Design Pattern Catalogue

Creational Patterns	Structural Patterns	Behavioral Patterns
Abstract Factory	Adapter	Chain of Responsibility
Builder	Bridge	Command
Factory Method	Composite	Interpreter
Prototype	Decorator	Iterator
Singleton	Façade	Mediator
	Flyweight	Memento
	Proxy	Observer
		State
		Strategy
		Template Method
		Visitor

1. What are Design Patterns?
2. Why use Design Patterns?
3. Design Pattern Catalogue
4. **Creational Patterns**
5. Structural Patterns
6. Behavioral Patterns
7. Quiz

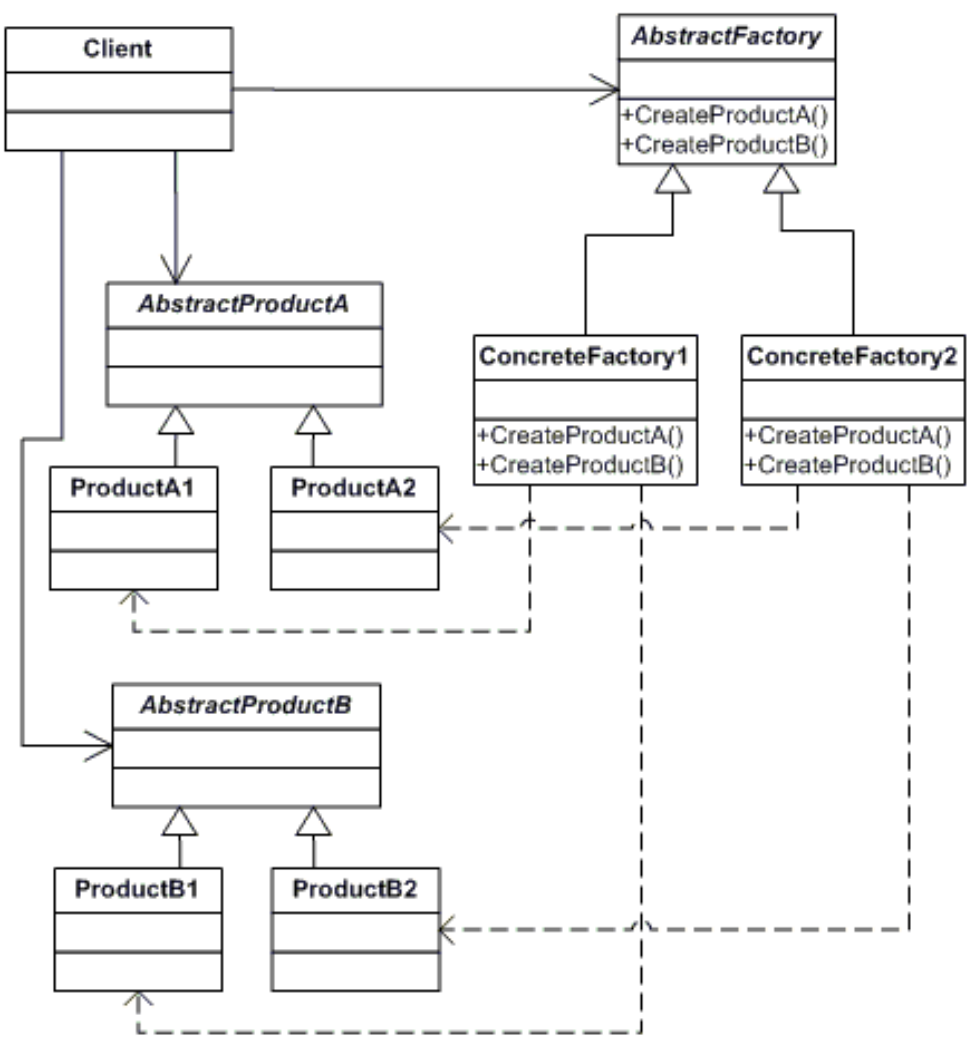
DESIGN PATTERNS

Creational Pattern: Abstract Factory

Definition:

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

Creational Pattern: Abstract Factory



Creational Pattern: Abstract Factory

Example:

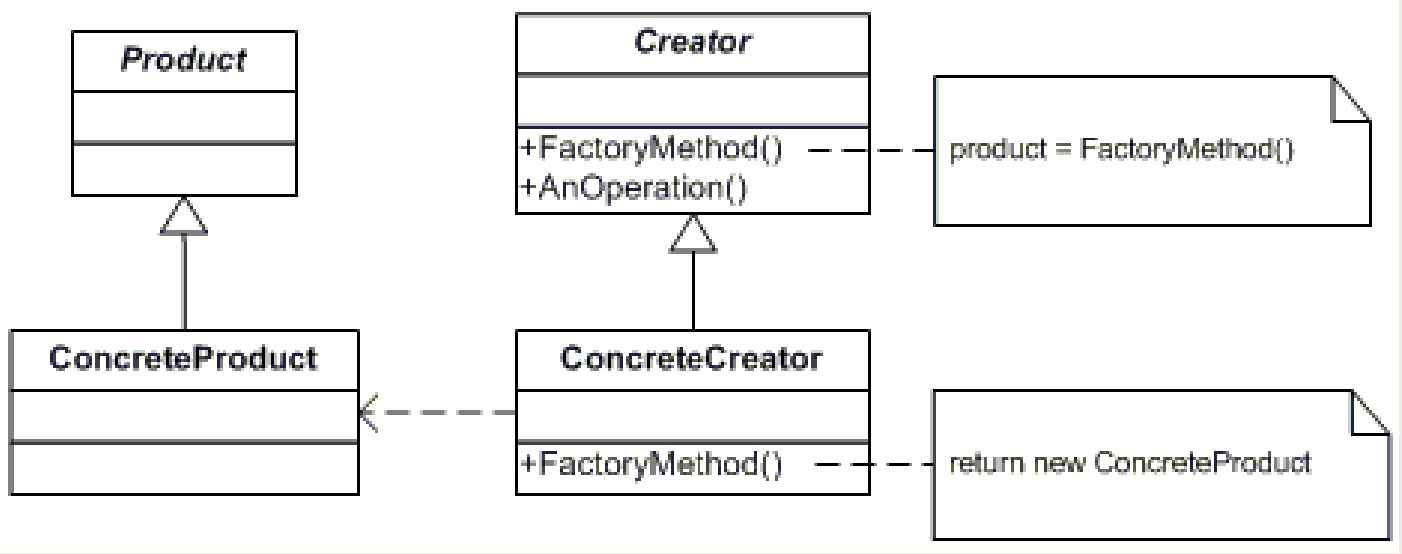
Creating different animal worlds for a computer game using different factories. Although the animals created by the Continent factories are different, the interactions among the animals remain the same.

Creational Pattern: Factory Method

Definition:

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

Creational Pattern: Factory Method



Creational Pattern: Factory Method

Example:

Offering flexibility in creating different documents. The derived Document classes Report and Resume instantiate extended versions of the Document class.

Creational Pattern: Singleton

Definition:

Ensures a class has only one instance and provides a global point of access to it.

Creational Pattern: Singleton

Creational Pattern: Singleton

Example:

A LoadBalancing object. Only a single instance (the singleton) of the class can be created because servers may dynamically come on- or off-line and every request must go through the one object that has knowledge about the global state.

1. What are Design Patterns?
2. Why use Design Patterns?
3. Design Pattern Catalogue
4. Creational Patterns
5. **Structural Patterns**
6. Behavioral Patterns
7. Quiz

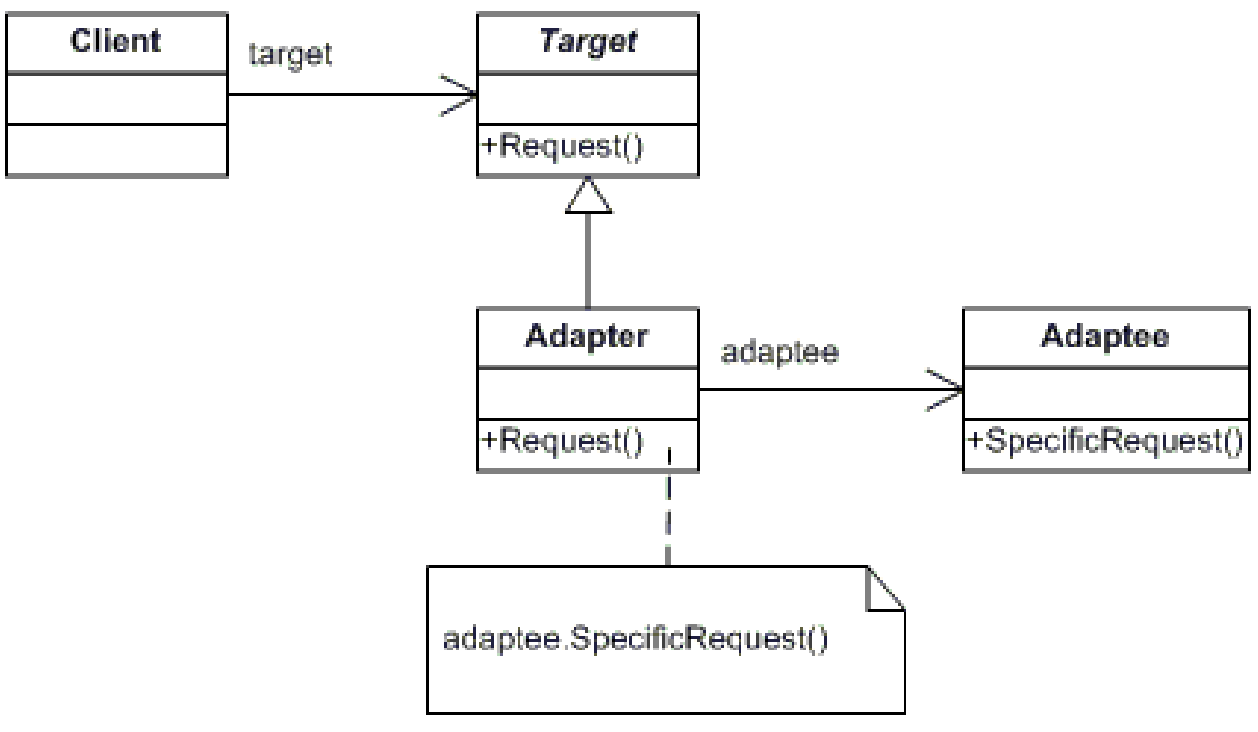
DESIGN PATTERNS

Structural Pattern: Adapter

Definition:

Converts the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

Structural Pattern: Adapter



Structural Pattern: Adapter

Example:

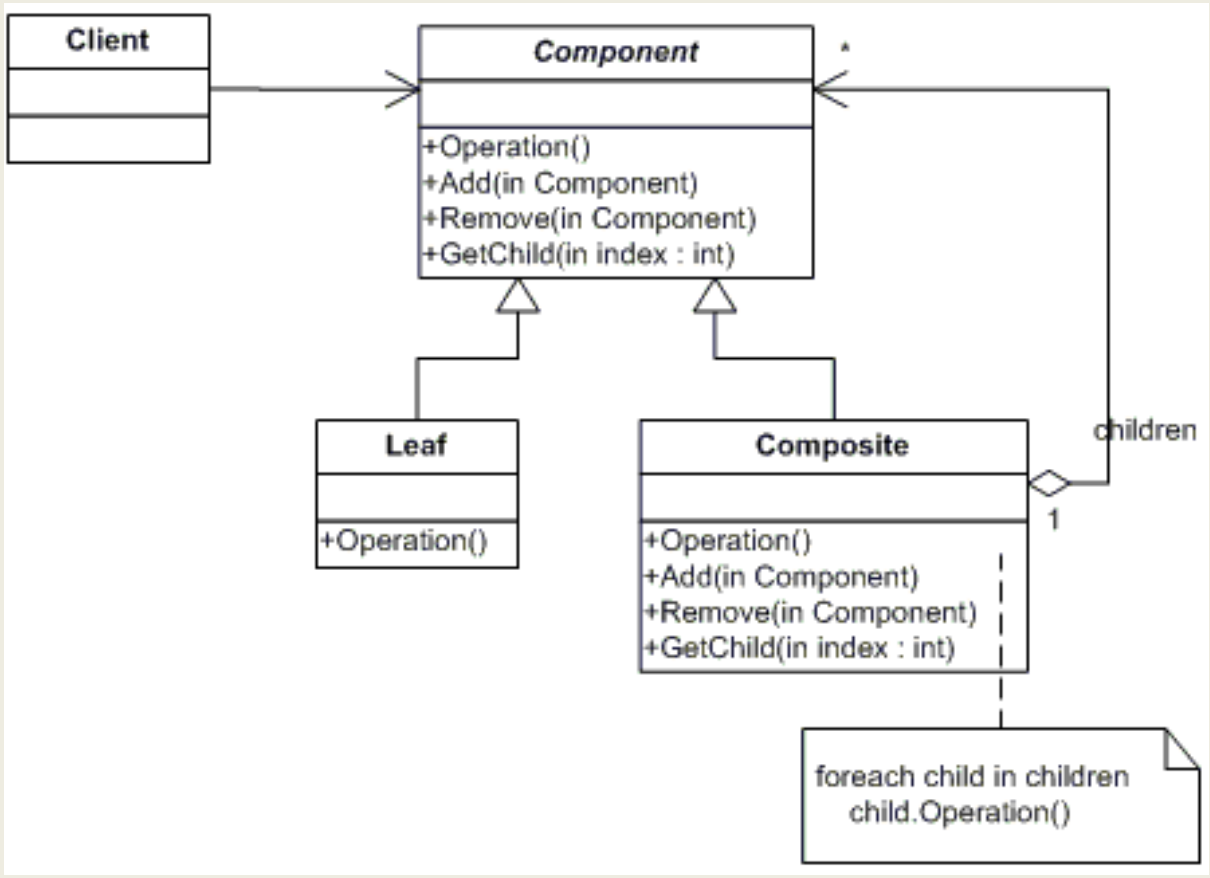
A legacy chemical databank. Chemical compound objects access the databank through an Adapter interface.

Structural Pattern: Composite

Definition:

Composes objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

Structural Pattern: Composite



Structural Pattern: Composite

Example:

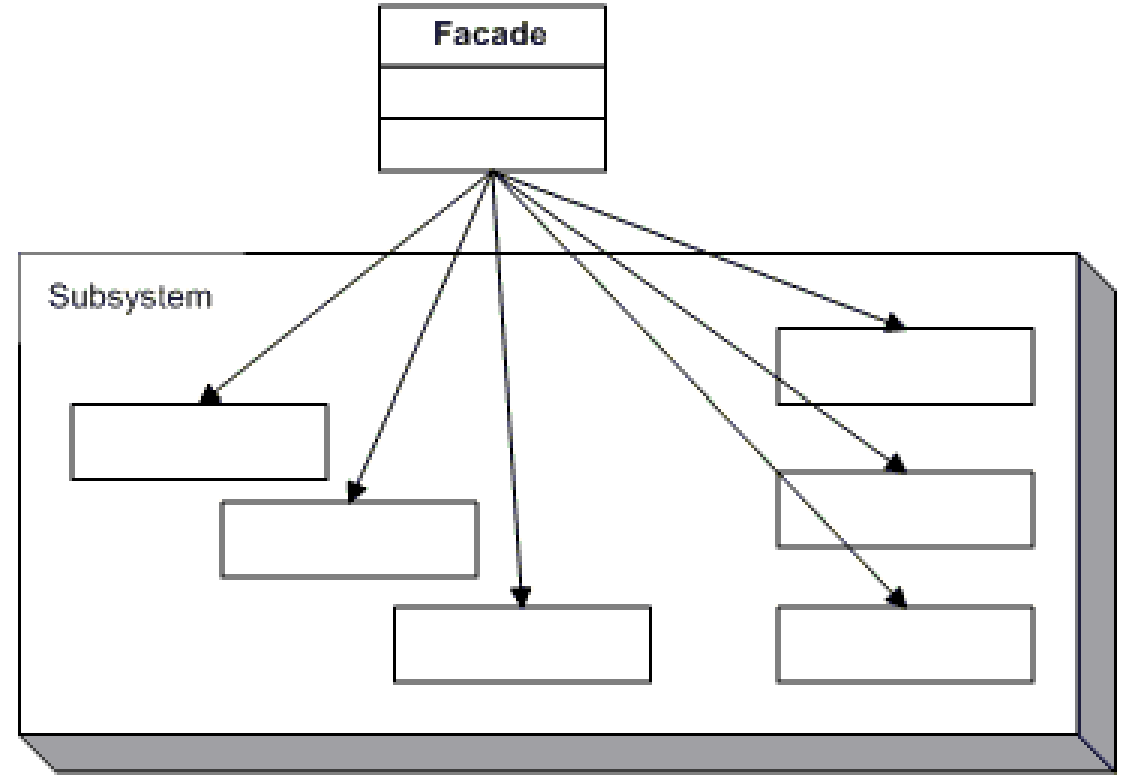
Building a graphical tree structure made up of primitive nodes (lines, circles, etc.) and composite nodes (groups of drawing elements that make up more complex elements).

Structural Pattern: Façade

Definition:

Provides a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use.

Structural Pattern: Façade



Structural Pattern: Façade

Example:

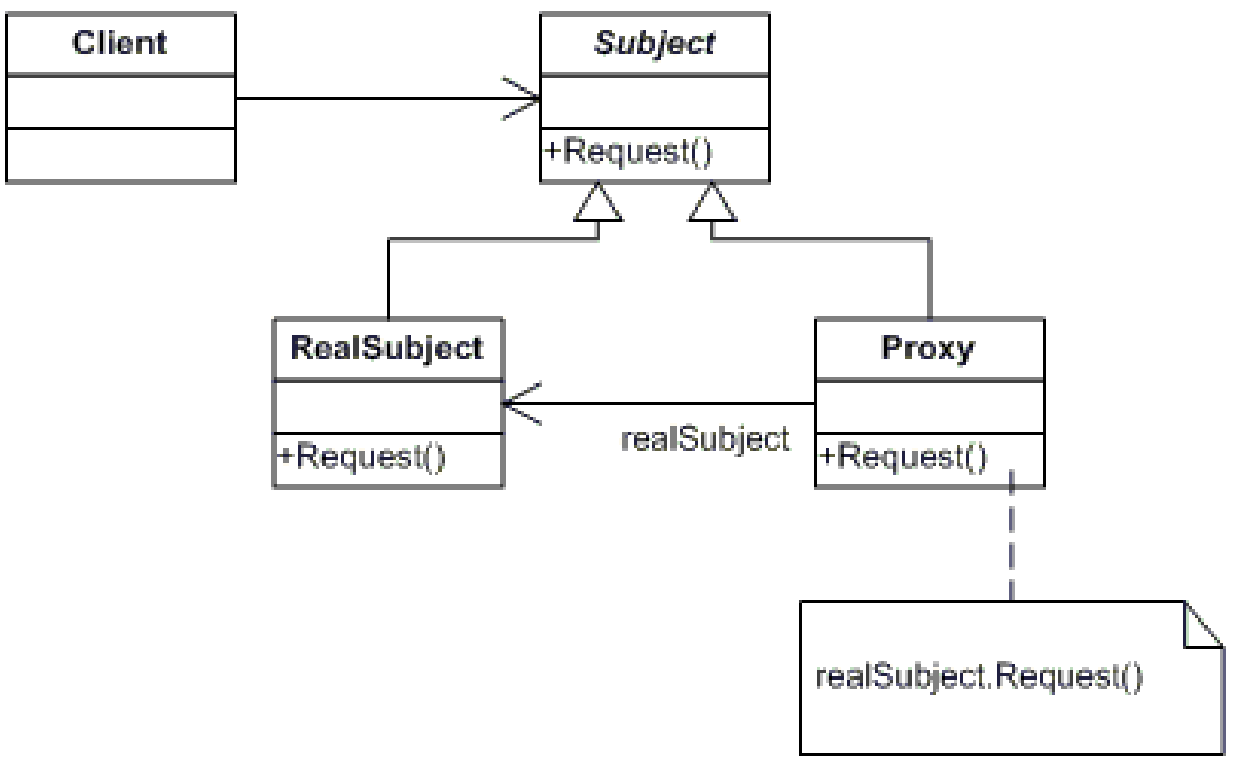
A MortgageApplication object which provides a simplified interface to a large subsystem of classes measuring the creditworthiness of an applicant.

Structural Pattern: Proxy

Definition:

Provides a surrogate or placeholder for another object to control access to it.

Structural Pattern: Proxy



Structural Pattern: Proxy

Example:

A Math object represented by a MathProxy object.

1. What are Design Patterns?
2. Why use Design Patterns?
3. Design Pattern Catalogue
4. Creational Patterns
5. Structural Patterns
6. **Behavioral Patterns**
7. Quiz

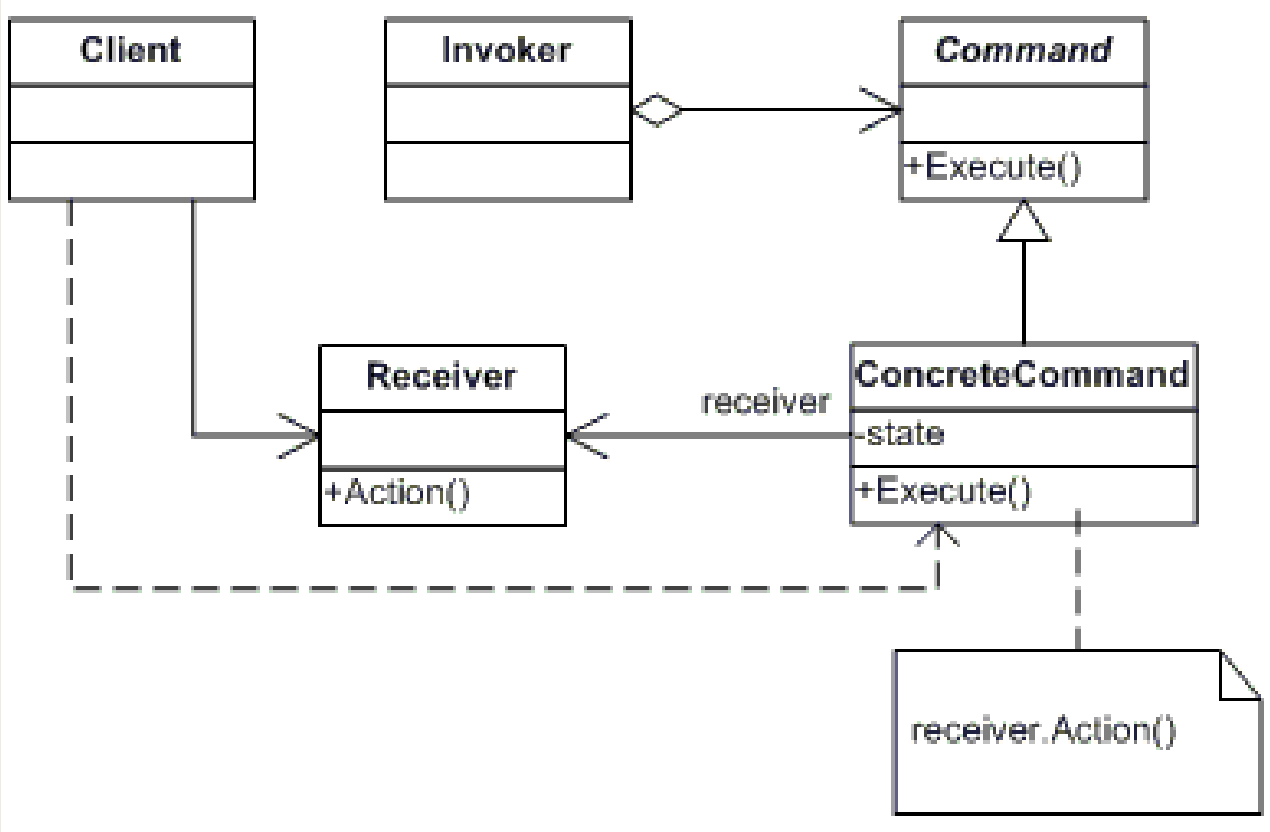
DESIGN PATTERNS

Behavioral Pattern: Command

Definition:

Encapsulates a request as an object, thereby letting you parameterize clients with different requests and supports undoable operations.

Behavioral Pattern: Command



Behavioral Pattern: Command

Example:

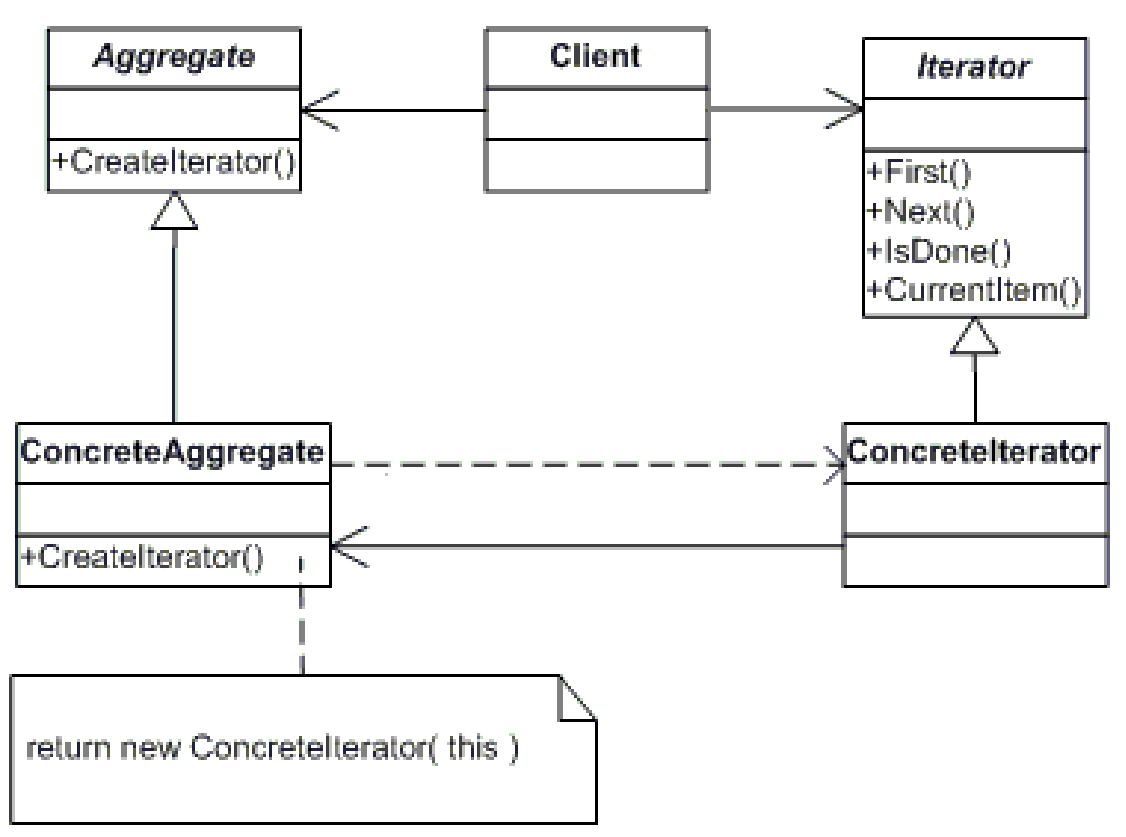
A simple calculator with unlimited number of undo's and redo's.

Behavioral Pattern: Iterator

Definition:

Provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Behavioral Pattern: Iterator



Behavioral Pattern: Iterator

Example:

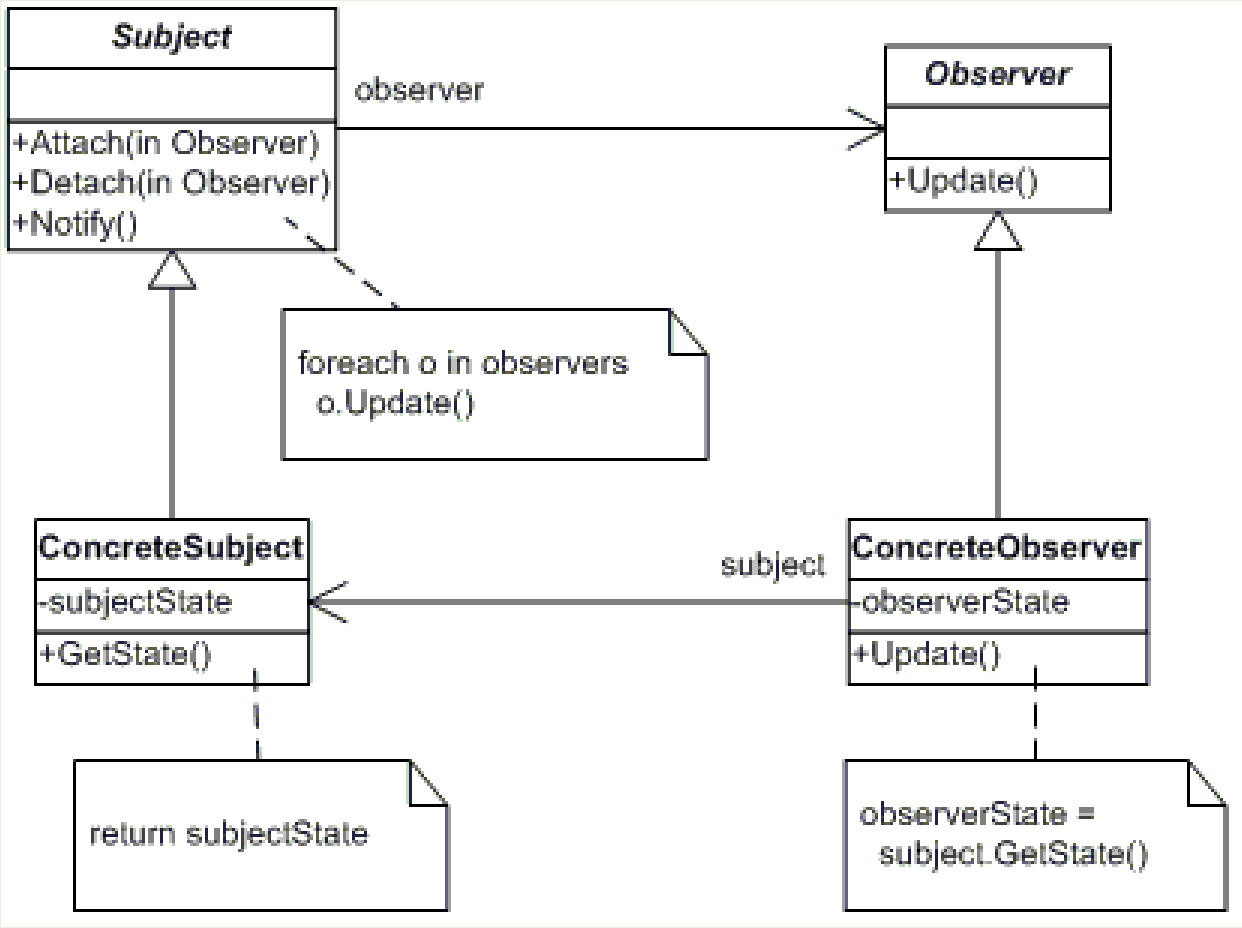
Iterating over a collection of items and skipping a specific number of items in each iteration.

Behavioral Pattern: Observer

Definition:

Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

Behavioral Pattern: Observer



Behavioral Pattern: Observer

Example:

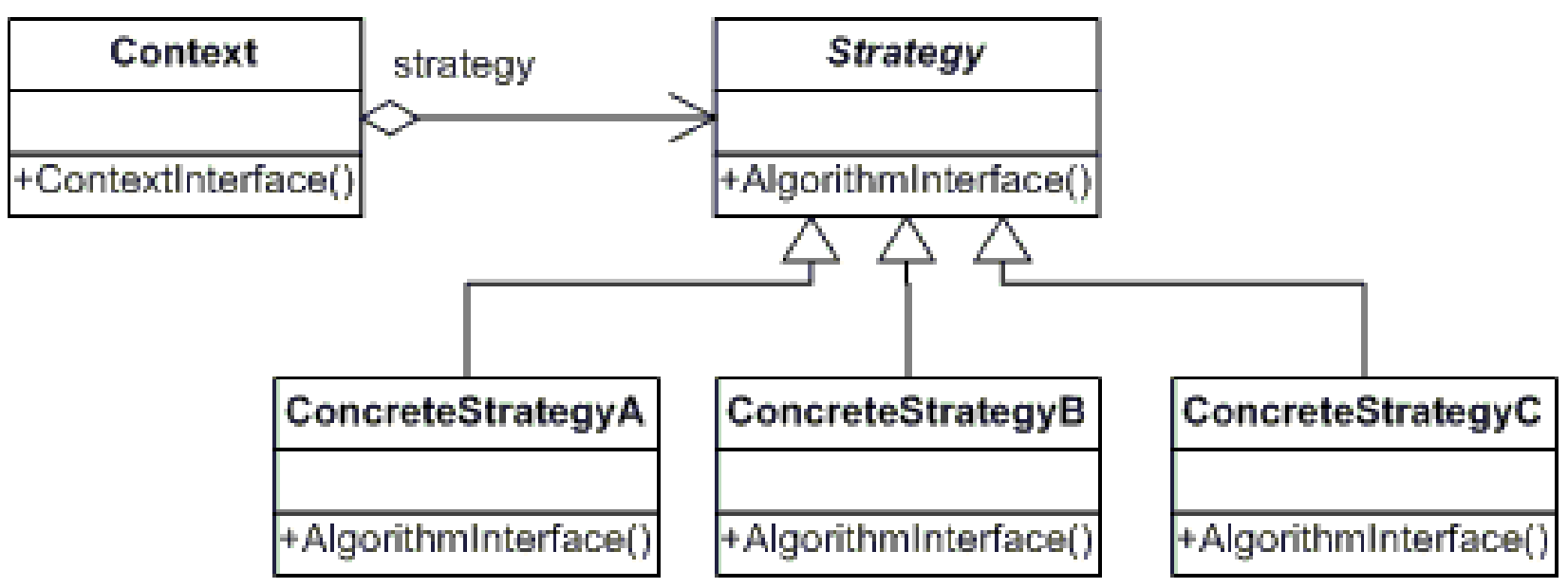
Notifying registered investors every time a stock changes value.

Behavioral Pattern: Strategy

Definition:

Defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Behavioral Pattern: Strategy



Behavioral Pattern: Strategy

Example:

Encapsulating sorting algorithms in the form of sorting objects. This allows clients to dynamically change sorting strategies including Quicksort, Shellsort, and Mergesort.

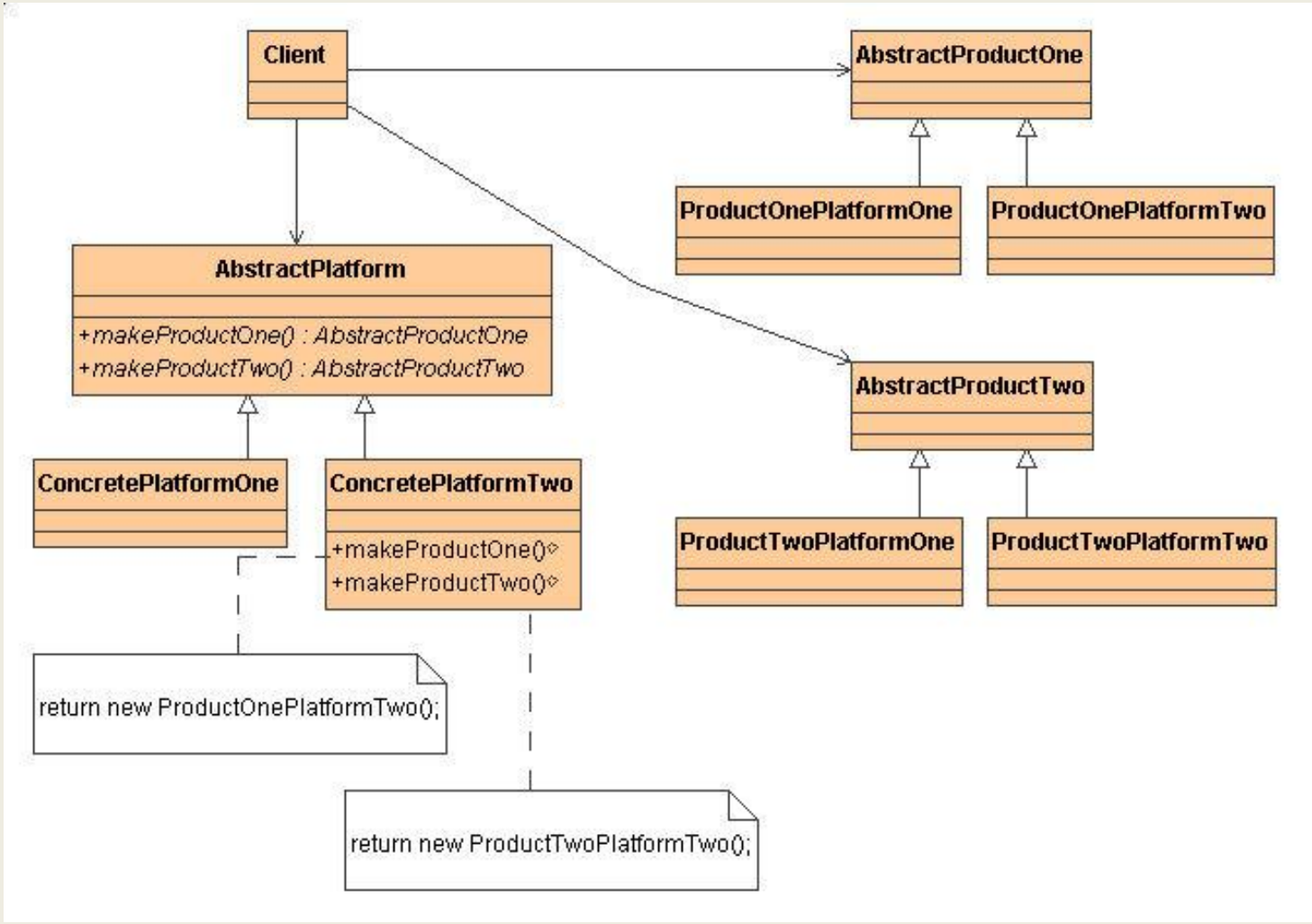
1. What are Design Patterns?
2. Why use Design Patterns?
3. Design Pattern Catalogue
4. Creational Patterns
5. Structural Patterns
6. Behavioral Patterns
7. **Quiz**

DESIGN PATTERNS

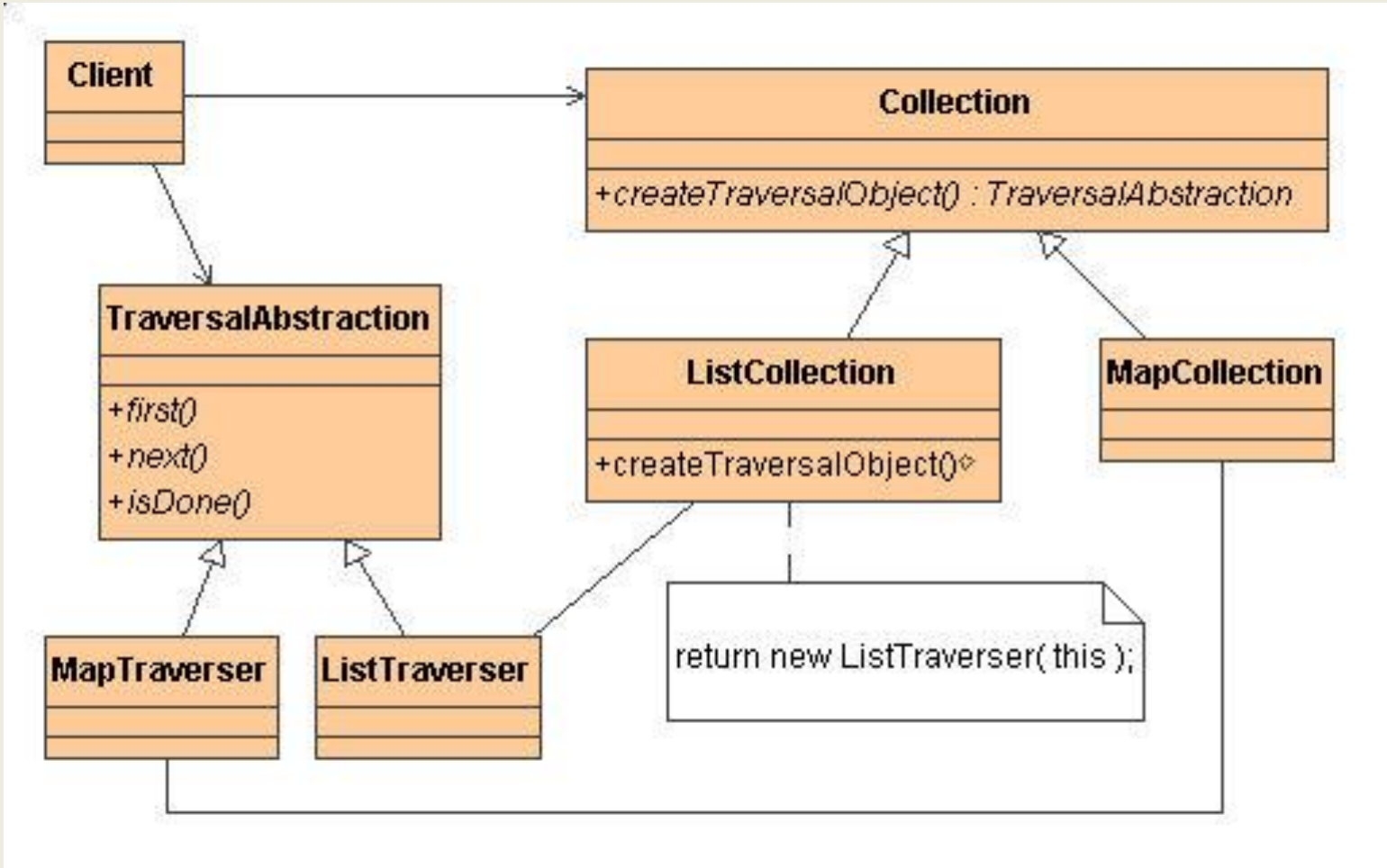
Quiz


GlobalResource
-theInstance : GlobalResource
+getInstance() : GlobalResource

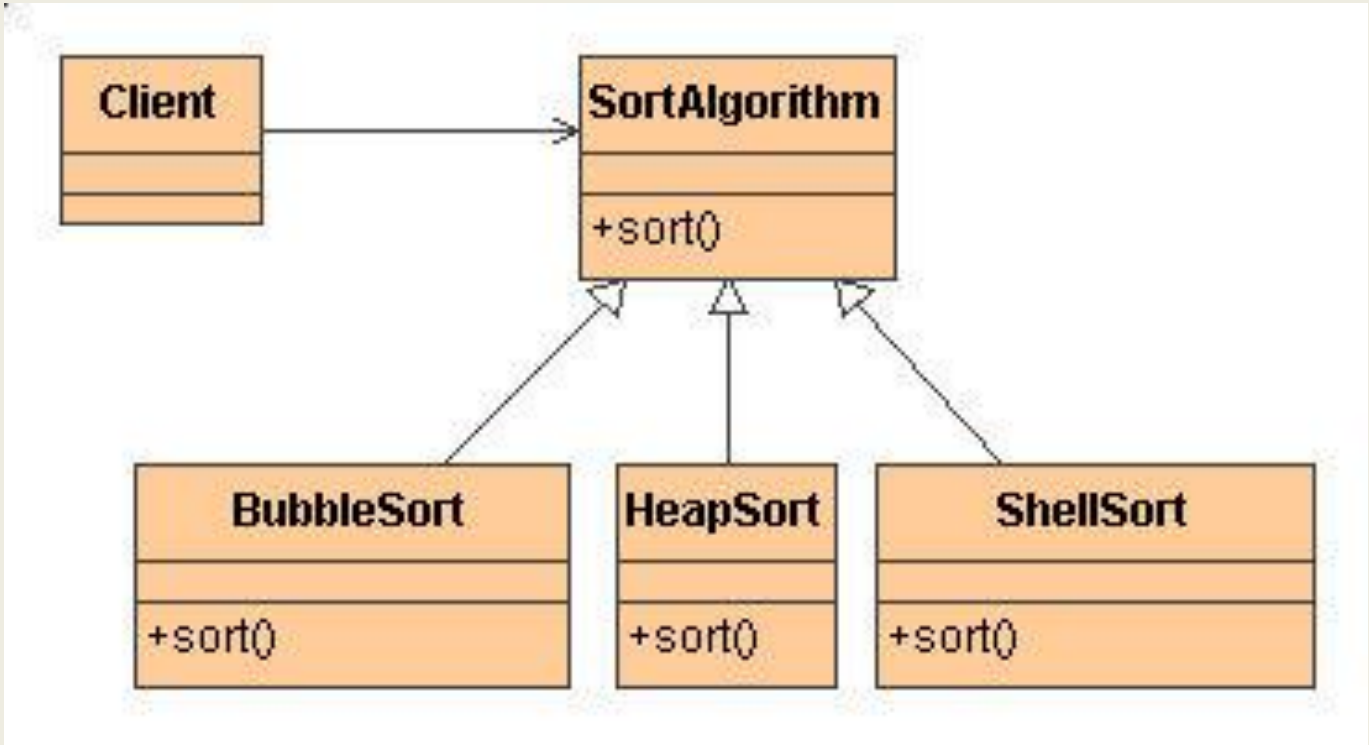
Quiz


Client
AbstractProductOne
ProductOnePlatformOne
ProductOnePlatformTwo
AbstractPlatform
+makeProductOne() : AbstractProductOne
+makeProductTwo() : AbstractProductTwo
AbstractProductTwo
ConcretePlatformOne
ConcretePlatformTwo
+makeProductOne()
+makeProductTwo()
ProductTwoPlatformOne
ProductTwoPlatformTwo
return new ProductOnePlatformTwo();
return new ProductTwoPlatformTwo();

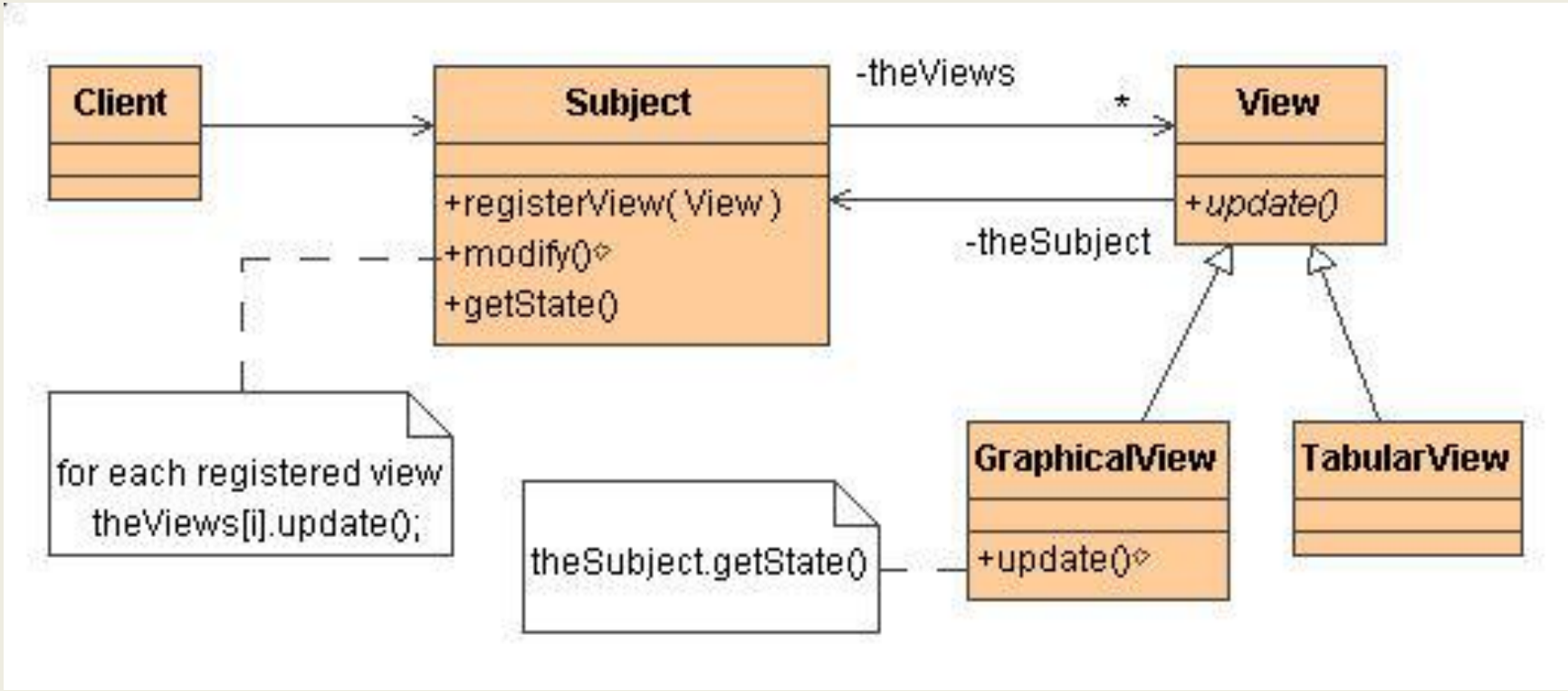
Quiz


Client
Collection
+createTraversalObject() : TraversalAbstraction
TraversalAbstraction
+first()
+next()
+isDone()
ListCollection
+createTraversalObject()
MapCollection
MapTraverser
ListTraverser
return new ListTraverser(this);

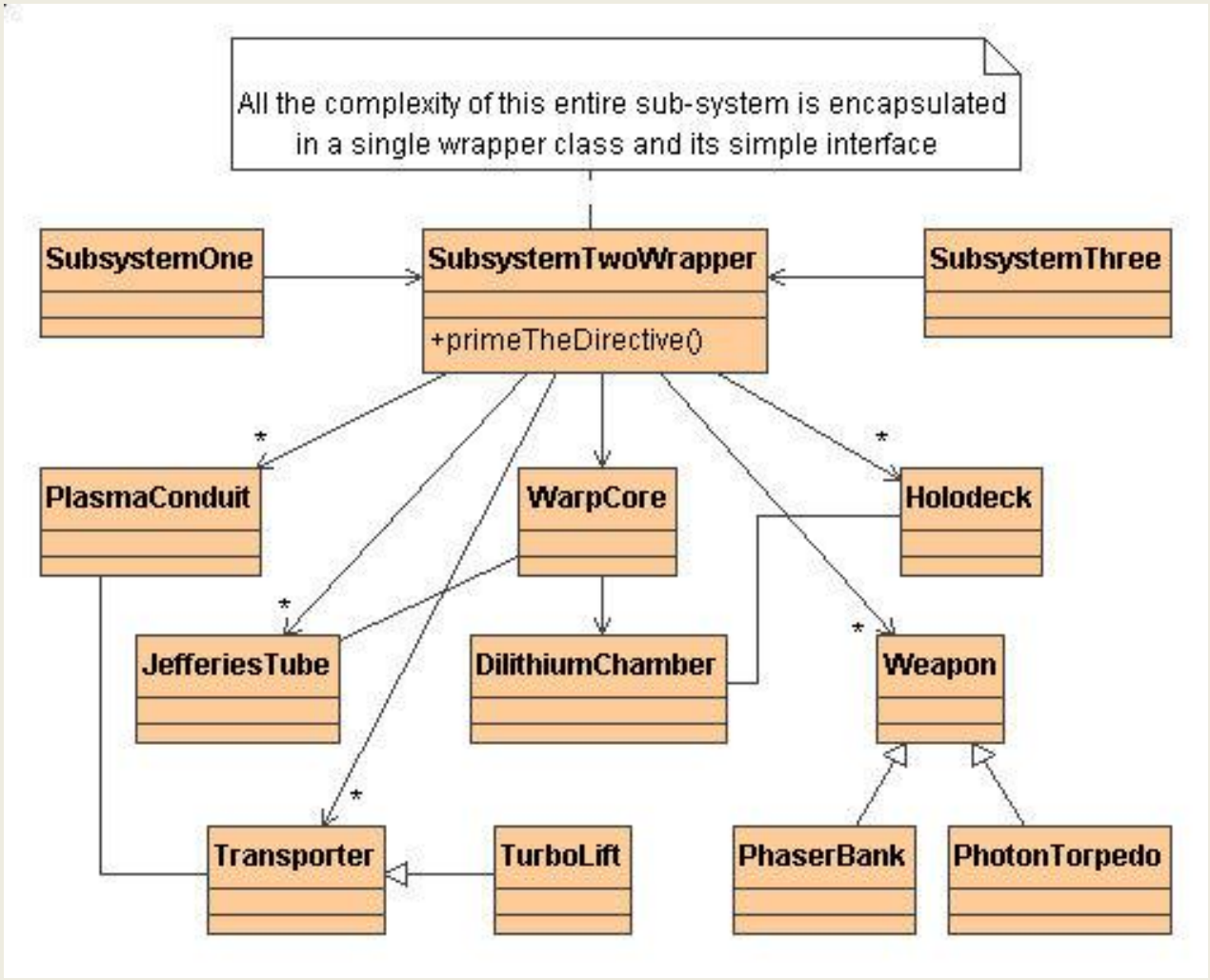
Quiz


Client
SortAlgorithm
+sort()
BubbleSort
+sort()
HeapSort
+sort()
ShellSort
+sort()

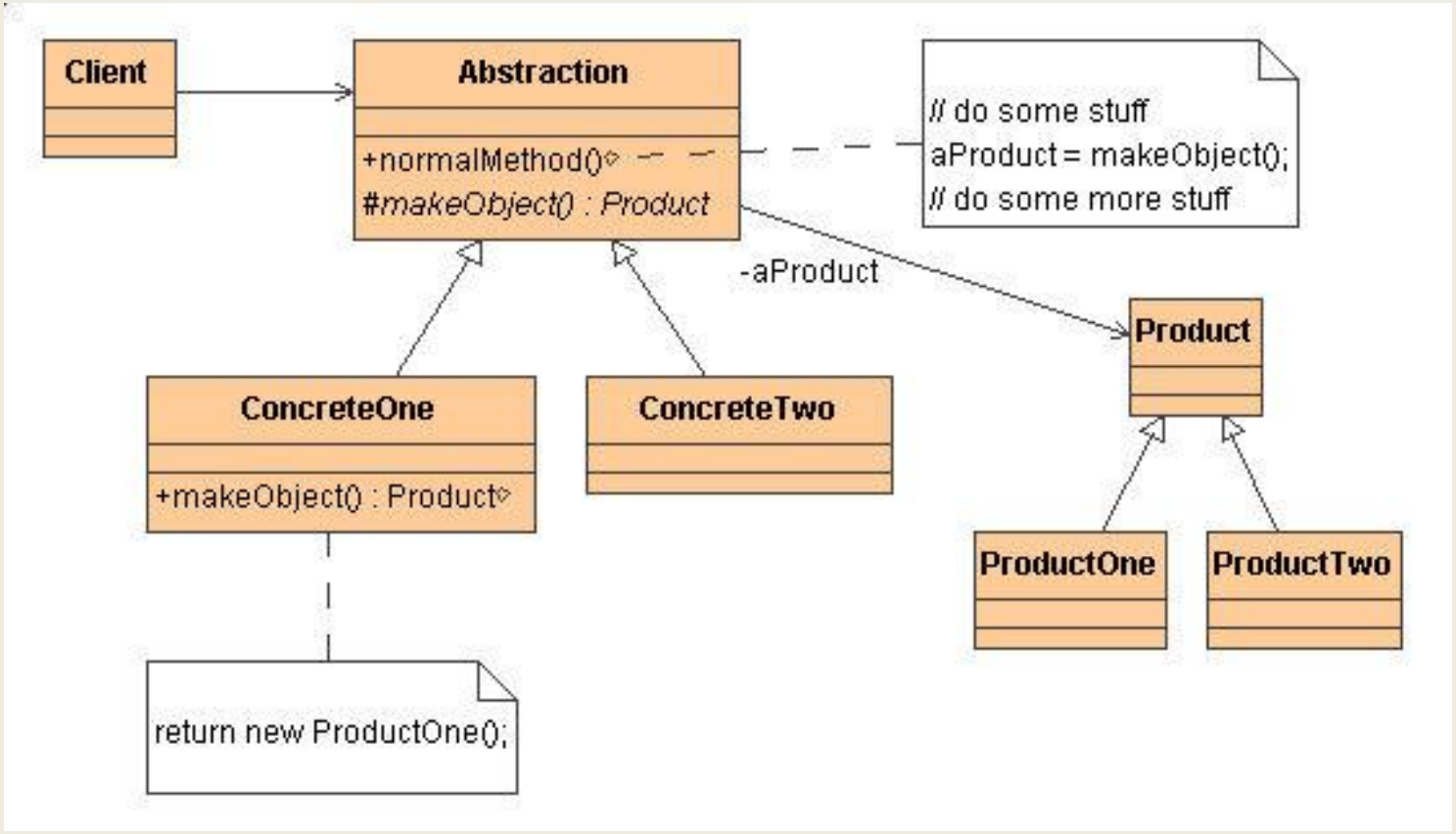
Quiz


Client
Subject
-theViews
*
View
+registerView(View)
+modify()
+getState()
+update()
-theSubject
for each registered view
theViews[i].update();
theSubject.getState()
GraphicalView
+update()
TabularView

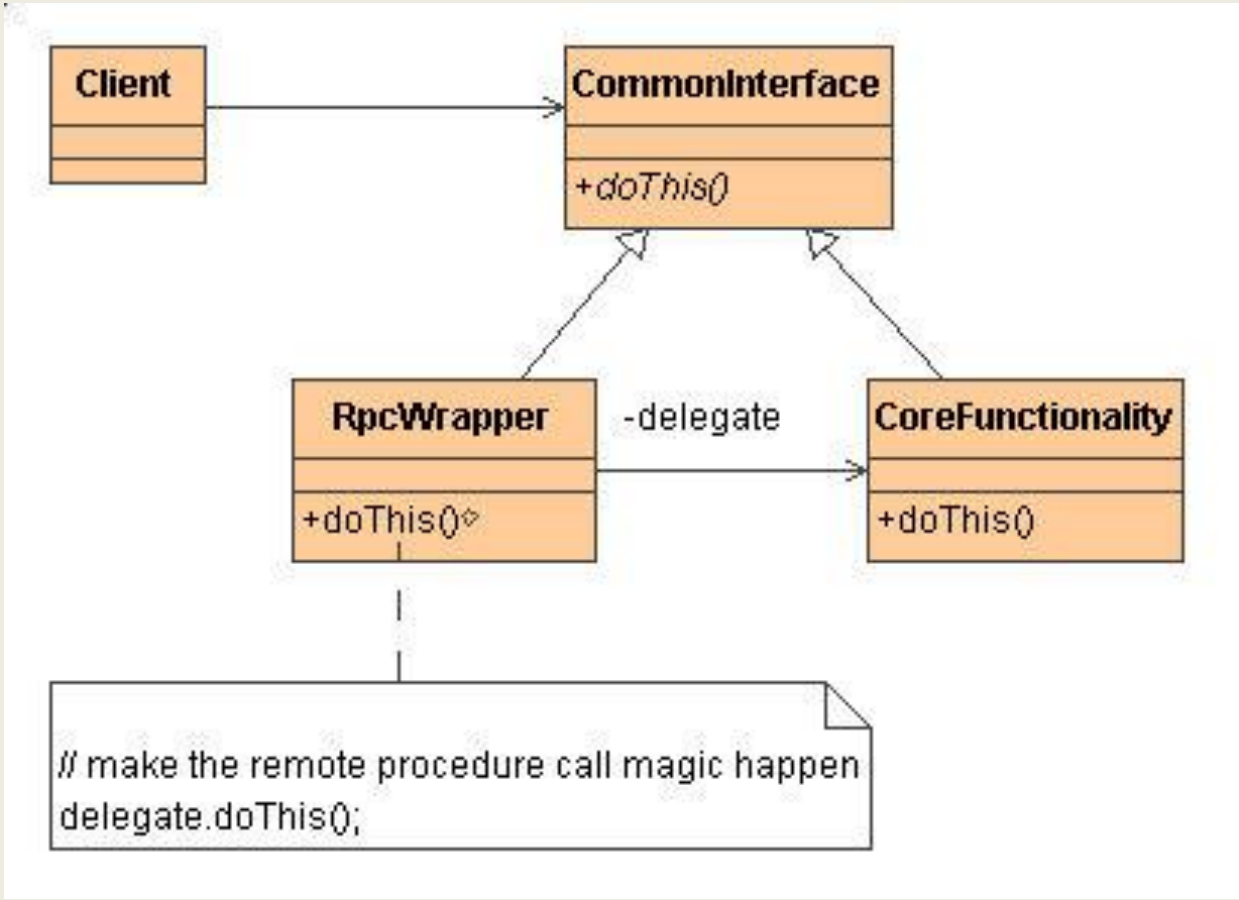
Quiz


All the complexity of this entire sub-system is encapsulated in a single wrapper class and its simple interface
SubsystemOne
SubsystemTwoWrapper
+primeTheDirective()
SubsystemThree
PlasmaConduit
WarpCore
Holodeck
JefferiesTube
DilithiumChamber
Weapon
Transporter
TurboLift
PhaserBank
PhotonTorpedo
*

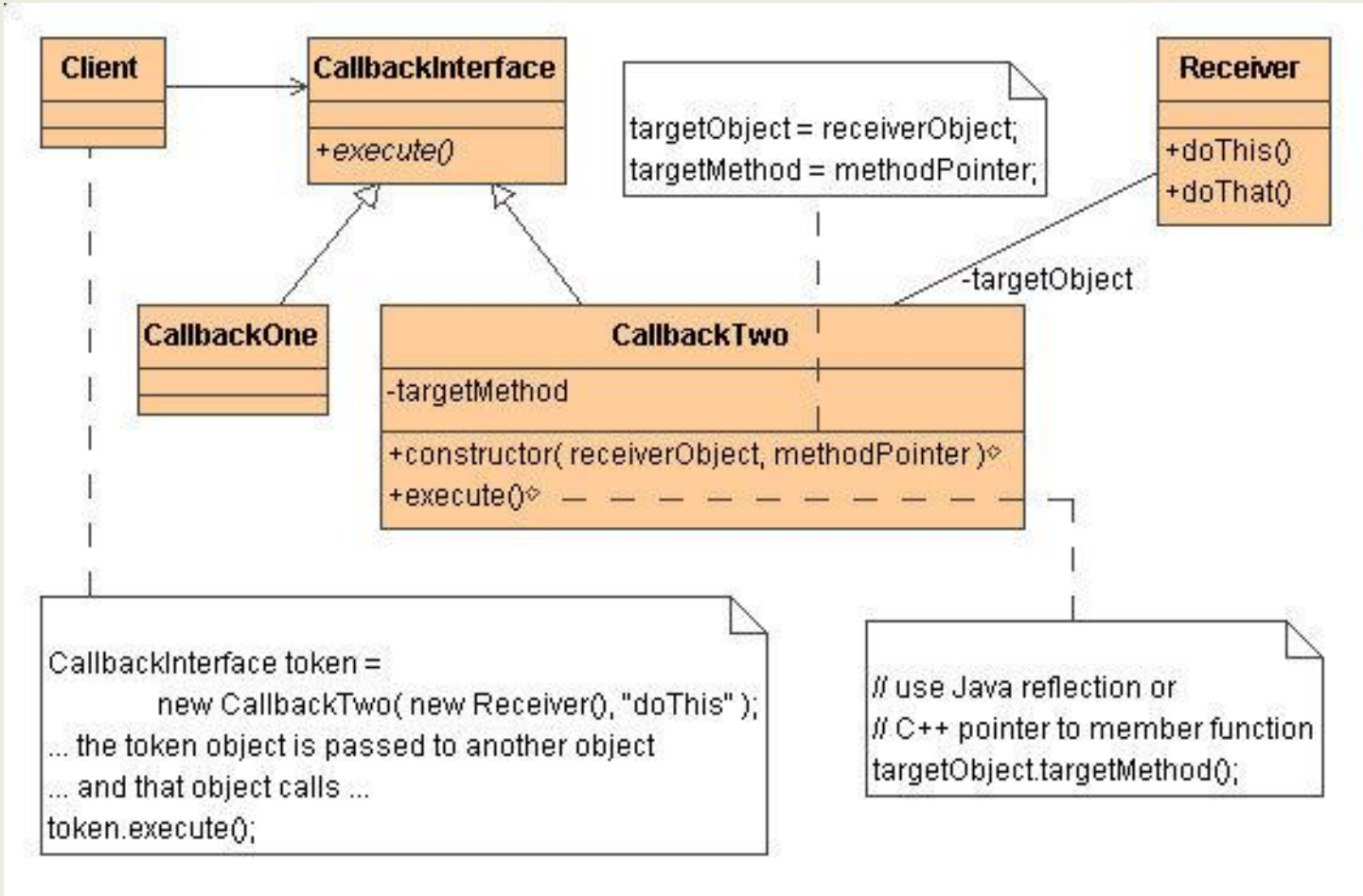
Quiz


Client
Abstraction
+normalMethod()
#makeObject() : Product
// do some stuff
aProduct = makeObject();
// do some more stuff
-aProduct
Product
ConcreteOne
+makeObject() : Product
ConcreteTwo
ProductOne
ProductTwo
return new ProductOne();

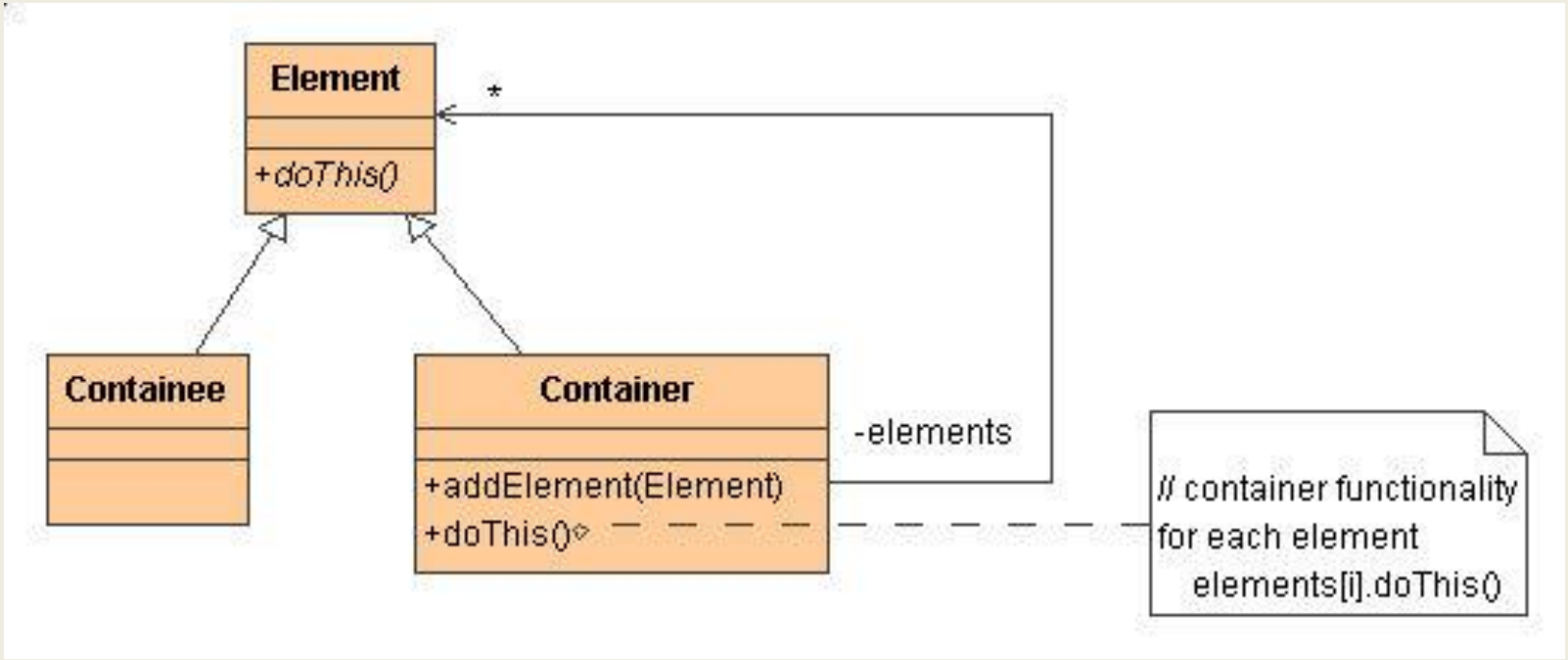
Quiz


Client
CommonInterface
+doThis()
RpcWrapper
-delegate
CoreFunctionality
+doThis()
+doThis()
// make the remote procedure call magic happen
delegate.doThis();

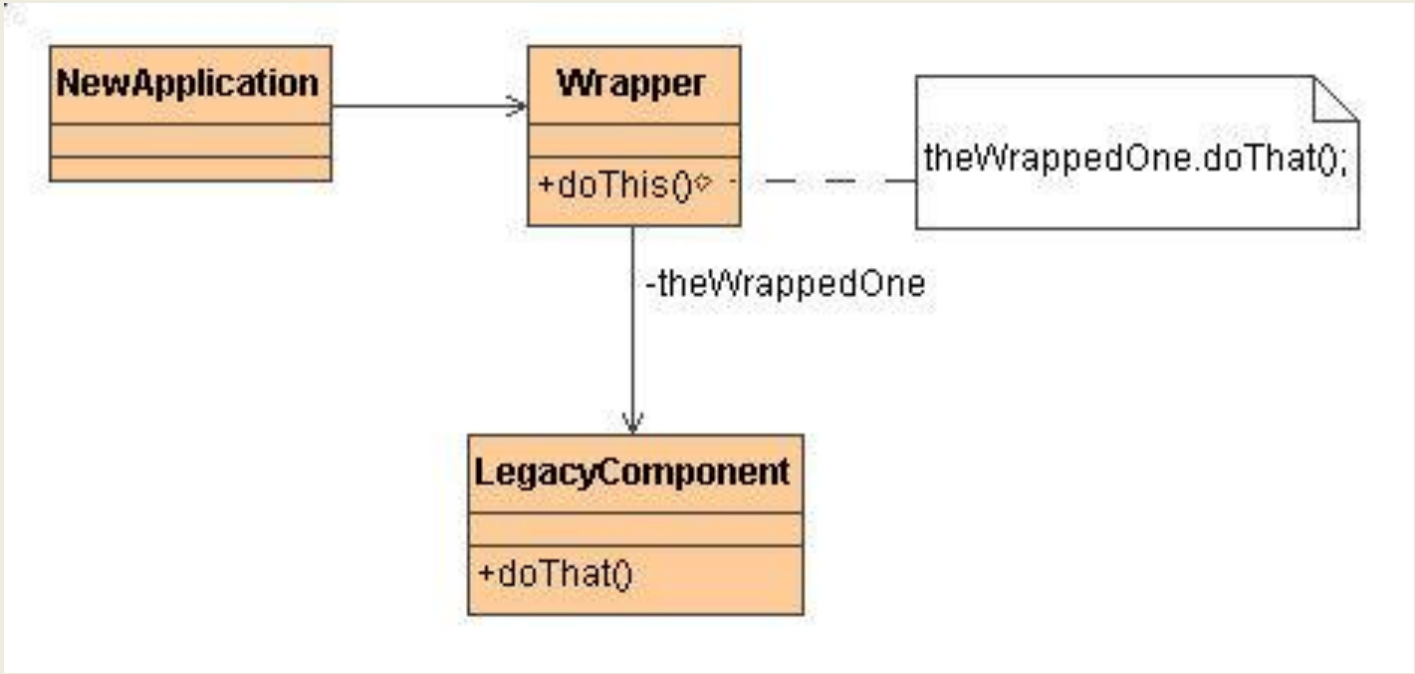
Quiz


Client
CallbackInterface
+execute()
targetObject = receiverObject;
targetMethod = methodPointer;
Receiver
+doThis()
+doThat()
-targetObject
CallbackOne
CallbackTwo
-targetMethod
+constructor(receiverObject, methodPointer)
+execute()
CallbackInterface token =
new CallbackTwo(new Receiver(), "doThis");
... the token object is passed to another object
... and that object calls ...
token.execute();
// use Java reflection or
// C++ pointer to member function
targetObject.targetMethod();

Quiz


Element
+doThis()
*
Containee
Container
-elements
+addElement(Element)
+doThis()
// container functionality
for each element
elements[i].doThis()

Quiz


NewApplication
Wrapper
+doThis()
theWrappedOne.doThat();
-theWrappedOne
LegacyComponent
+doThat()

Quiz

http://www.vincehuston.org/dp/patterns_quiz.html