Using Design Patterns in Java Application Development

ExxonMobil Research & Engineering Co.
Clinton, New Jersey

Michael P. Redlich
(908) 730-3416
michael.p.redlich@exxonmobil.com

About Myself

- **Degree**
  - **B.S. in Computer Science**
  - **Rutgers University (go Scarlet Knights!)**
- **ExxonMobil Research & Engineering**
  - **Senior Research Technician (1988-1998, 2004-present)**
  - **Systems Analyst (1998-2002)**
- **Ai-Logix, Inc.**
  - **Technical Support Engineer (2003-2004)**
- **ACGNJ**
  - **Java Users Group Leader**
- **Publications**
  - **James: The Java Apache Mail Enterprise Server**
    - **co-authored with Barry Burd**
    - **Java Boutique**

Gang of Four (GoF)

- **Erich Gamma**
- **Richard Helm**
- **Ralph Johnson**
- **John Vlissides**
- **Design Patterns – Elements of Reusable Object-Oriented Software**
  - **Erich Gamma, et. al**
  - **ISBN 0-201-63361-2**
  - **1995**

What are Design Patterns?

- **Recurring solutions to software design problems that are repeatedly found in real-world application development**
- **All about the design and interaction of objects**
- **Four essential elements:**
  - **The pattern name**
  - **The problem**
  - **The solution**
  - **The consequences**

How Design Patterns Solve Design Problems

- **Find appropriate objects**
  - **Helps identify less obvious abstractions**
- **Program to an interface, not an implementation**
  - **Clients should only know about abstract classes that define an interface**
  - **Reduces implementation dependencies**
- **Design for change**
  - **Avoid creating objects directly**
  - **Avoid dependencies on specific operations**
  - **Avoid algorithmic dependencies**
  - **Avoid tight coupling**

Design Pattern Categories

- **Creational**
  - **Abstracts the instantiation process**
  - **Dynamically create objects so that they don't have to be instantiated directly**
- **Structural**
  - **Composes groups of objects into larger structures**
- **Behavioral**
  - **Defines communication among objects in a given system**
  - **Provides better control of flow in a complex application**

Creational Patterns

- **Abstract Factory**
  - **Provides an interface for creating related objects without specifying their concrete classes**
- **Builder**
  - **Reuses the construction process of a complex object**
- **Factory Method**
  - **Lets subclasses decide which class to instantiate from a defined interface**
- **Prototype**
  - **Creates new objects by copying a prototype**
- **Singleton**
  - **Ensures a class has only one instance with a global point of access to it**

Structural Patterns

- **Adapter**
  - **Converts the interface of one class to an interface of another**
- **Bridge**
  - **Decouples an abstraction from its implementation**
- **Composite**
  - **Composes objects into tree structures to represent hierarchies**
- **Decorator**
  - **Attaches responsibilities to an object dynamically**
- **Façade**
  - **Provides a unified interface to a set of interfaces**

Structural Patters (continued)

- **Flyweight**
  - **Supports large numbers of fine-grained objects by sharing**
- **Proxy**
  - **Provides a surrogate for another object to control access to it**

Behavioral Patterns

- **Chain of Responsibility**
  - **Passes a request along a chain of objects until the appropriate one handles it**
- **Command**
  - **Encapsulates a request as an object**
- **Interpreter**
  - **Defines a representation and an interpreter for a language grammar**
- **Iterator**
  - **Provides a way to access elements of an object sequentially without exposing its implementation**
- **Mediator**
  - **Defines an object that encapsulates how a set of objects interact**

Behavioral Patterns (continued)

- **Memento**
  - **Captures an object's internal state so that it can be later restored to that state if necessary**
- **Observer**
  - **Defines a one-to-many dependency among objects**
- **State**
  - **Allows an object to alter its behavior when its internal state changes**
- **Strategy**
  - **Encapsulates a set of algorithms individually and makes them interchangeable**
- **Template Method**
  - **Lets subclasses redefine certain steps of an algortithm**
- **Visitor**
  - **Defines a new operation without changing the classes on which it operates**

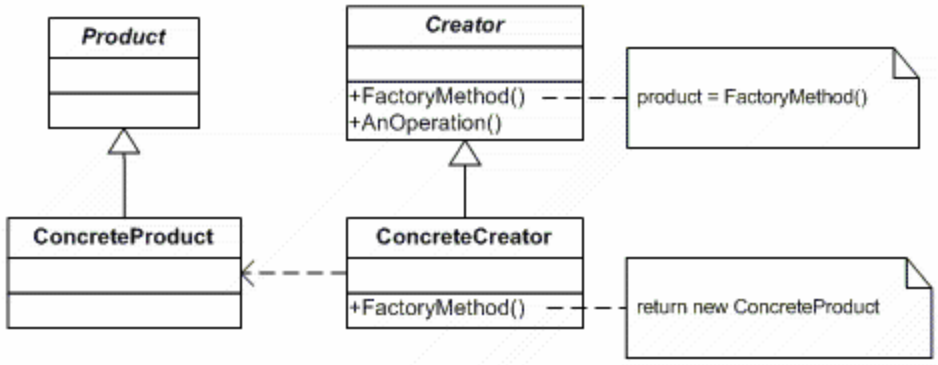
Factory Method

- **Intent**
  - **Defines an interface for creating an object, but lets subclasses decide which class to instantiate**
  - **Lets a class defer instantiation to subclasses**
- **Also known as**
  - **Virtual Constructor**
- **Motivation**
  - **To solve the problem of one class knowing *when* to create a class of another type, but not knowing *what kind* of class to create**
- **Design Principle**
  - **Depend upon abstractions; do not depend upon concrete classes**

Factory Method

- **Use this pattern when:**
  - **A class can't anticipate the class of objects is must create**
  - **A class would prefer for its subclasses to specify the objects it creates**
  - **There is a need for a class to localize one of several helper classes that can be delegated a responsibility**

Factory Method



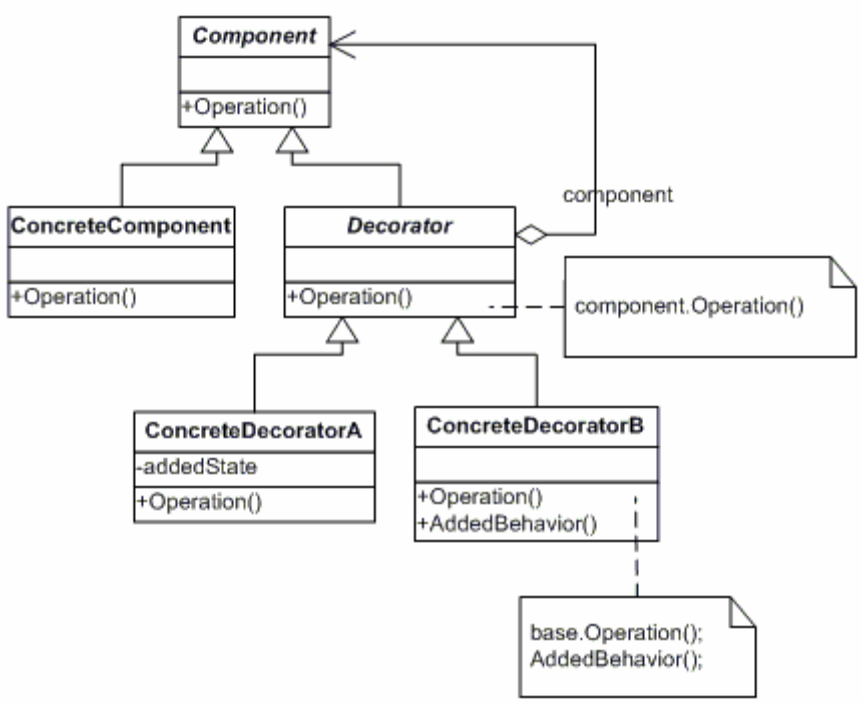
Decorator

- **Intent**
  - **Attaches additional responsibilities to an object dynamically**
  - **Provides a flexible alternative to subclassing for extending functionality**
- **Also known as**
  - **Wrapper**
- **Motivation**
  - **Allows classes to be easily extended to incorporate new behavior without modifying existing code**
- **Design Principle**
  - **Classes should be open for extension, but closed for modification**

Decorator

- **Use this pattern:**
  - **To add responsibilities to individual objects dynamically and transparently without affecting other objects**
  - **For responsibilities that can be withdrawn**
  - **When extension by subclassing is impractical**

Decorator


Component
+Operation()
component
ConcreteComponent
+Operation()
Decorator
+Operation()
component.Operation()
ConcreteDecoratorA
-addedState
+Operation()
ConcreteDecoratorB
+Operation()
+AddedBehavior()
base.Operation();
AddedBehavior();

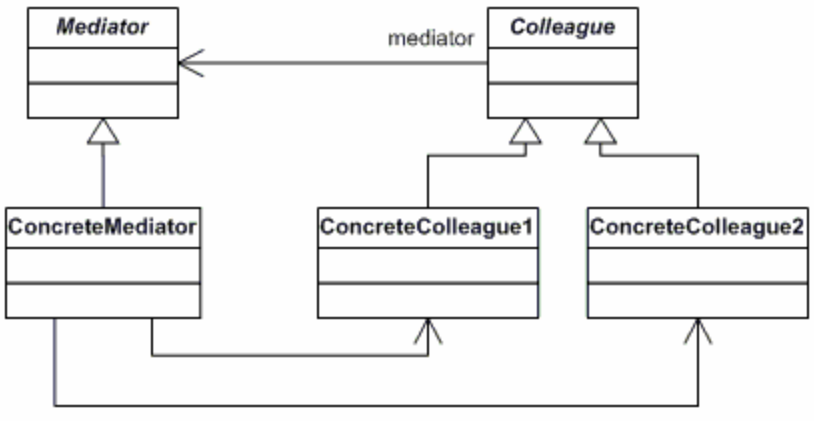
Mediator

- **Intent**
  - **Defines simplified communication among classes**
  - **Defines an object that encapsulates how a set of objects interact**
  - **Promotes loose coupling by keeping objects from referring to each other explicitly, and allow the developer to vary their interaction independently**
- **Motivation**
  - **To avoid the many interconnections among objects that can lead to a maintenance headache**

Mediator

- **Use this pattern when:**
  - **A set of objects communicate in well-defined but complex ways**
  - **Reusing an object is difficult because it refers to and communicates with many other objects**
  - **A behavior that is distributed among several classes should be customizable without a lot of subclassing**

Mediator


Mediator
mediator
Colleague
ConcreteMediator
ConcreteColleague1
ConcreteColleague2

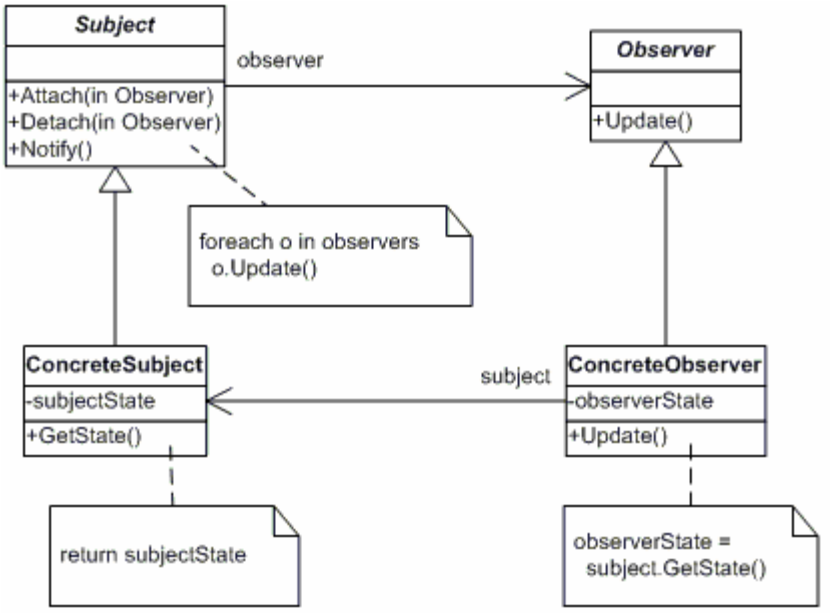
Observer

- **Intent**
  - **Defines a one-to-many dependency among objects so that when one object changes state, all its dependents are notified and updated automatically**
  - **A way of notifying change to a number of classes**
- **Also known as**
  - **Dependents**
  - **Publish-Subscribe**
- **Motivation**
  - **To avoid making classes tightly coupled that would reduce their reusability**
- **Design Principle**
  - **Strive for loosely coupled designs among objects that interact**

Observer

- **Use this pattern when:**
  - **A change to one object requires changing others, and the number of objects to be changed is unknown**
  - **An object should be able to notify other objects without making assumptions about who these objects are**
    - **Avoids having these objects tightly coupled**

Observer


Subject
observer
Observer
+Attach(in Observer)
+Detach(in Observer)
+Notify()
+Update()
foreach o in observers
o.Update()
ConcreteSubject
-subjectState
+GetState()
subject
ConcreteObserver
-observerState
+Update()
return subjectState
observerState =
subject.GetState()

Resources

- **Design Patterns – Elements of Reusable Object-Oriented Software**
  - **Erich Gamma, et. al**
  - **ISBN 0-201-63361-2**
- **Java Design Patterns**
  - **James W. Cooper**
  - **ISBN 0-201-48539-7**
- **UML Distilled**
  - **Martin Fowler (with Kendall Scott)**
  - **ISBN 0-201-32563-2**
- **Head First Design Patterns**
  - **Eric & Elisabeth Freeman (with Kathy Sierra & Bert Bates)**
  - **ISBN 0-596-00712-4**
- **Data & Object Factory**
  - **http://www.dofactory.com/**