Patterns of Patterns

CSCI 4448/5448: Object-Oriented Analysis & Design

Lecture 26 — 04/14/2011

Goals of the Lecture

- Wrap Up Textbook
- Cover Patterns of Patterns
  - Model-View-Controller
  - Multiple Patterns being used in one system
- Executive Summaries
  - Share some of the executive summaries of the graduate student presentations (more next week)

End of the Textbook (I)

- The textbook ends with three chapters
  - Chapter 24 summarizes the lessons associated with creational, or factory, patterns
  - Chapter 25 summarizes the major lessons of the book
  - Chapter 26 is a set of recommendations for follow-on reading; the suggestions are excellent (be sure to take a look)

End of the Textbook (II)

- At the end of Chapter 25, the authors make one final point about Design Patterns…
  - … by sharing how Christopher Alexander ends his book on architectural patterns
    - That book is 549 pages long; on page 545, Alexander says
      - "At this final stage, the patterns are no longer important…" (!)

End of the Textbook (III)

- Alexander continues
  - "The patterns have taught you to be receptive to what is real."
- That is, patterns provide you with example after example of the types of techniques that help you tackle software design problems

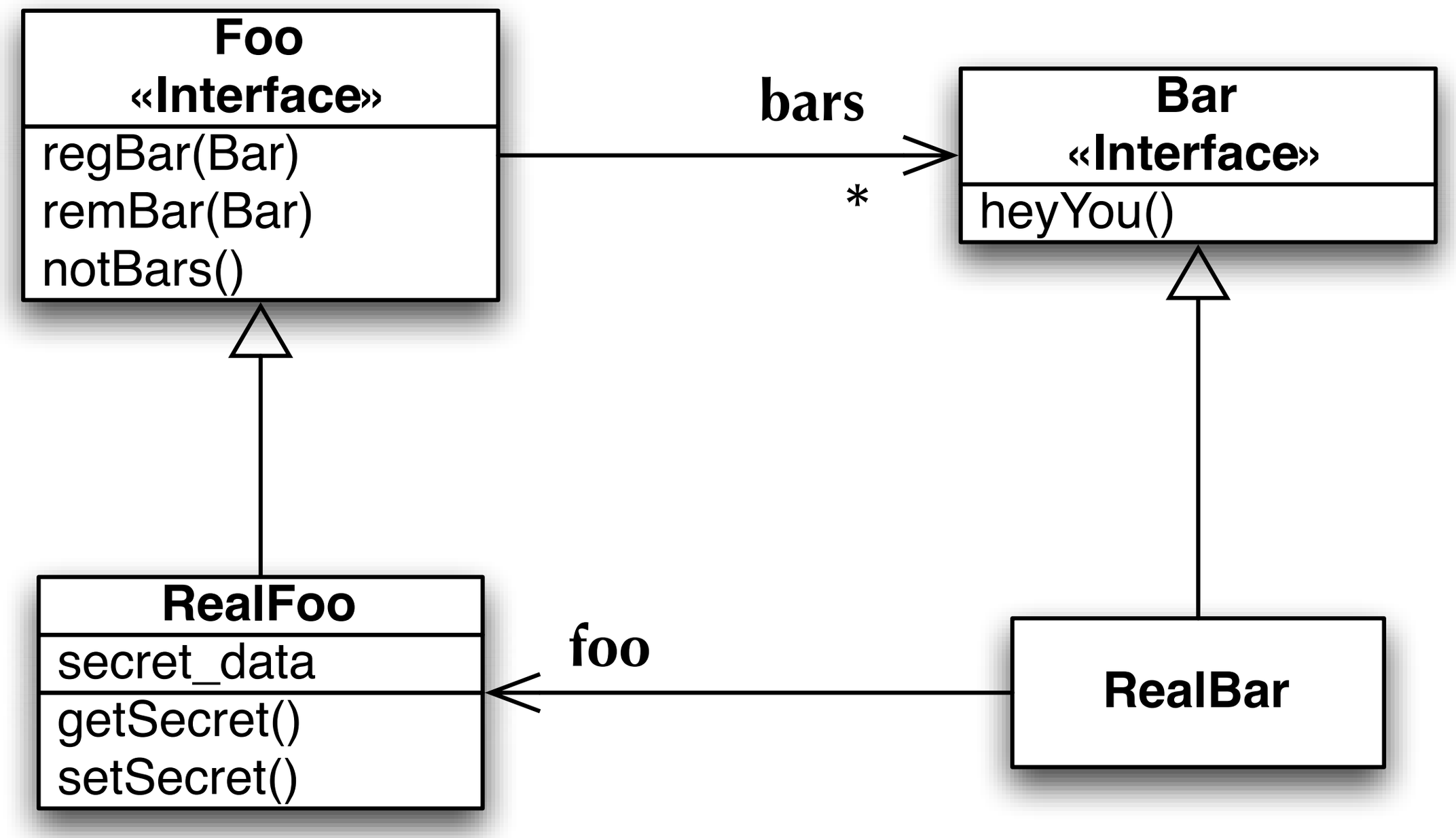
End of the Textbook (IV)

- If you learn those lessons, you can apply them to your own designs independent on any particular pattern
- If you then combine those lessons with the good object-oriented principles and heuristics we've seen throughout the semester
  - you will be well on your way to creating solid, flexible, extensible OO designs

Patterns of Patterns

- Patterns can be
  - used together in a single system (we've seen this several times)
  - can be combined to create, in essence, a new pattern
- Two examples
  - DuckSimulator Revisited: An example that uses six patterns at once
  - Model View Controller: A pattern that makes use of multiple patterns

But first… what pattern is this?



Remember that the names of classes participating in a pattern is unimportant; Its the structure (of the relationships and methods) that's important!

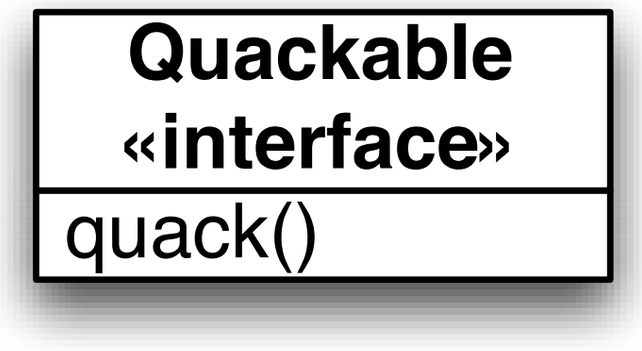
Duck Simulator Revisited

- We've been asked to build a new Duck Simulator by a Park Ranger interested in tracking various types of water fowl, ducks in particular.
- New Requirements
  - Ducks are the focus, but other water fowl (e.g. Geese) can join in too
  - Need to keep track of how many times duck's quack
  - Control duck creation such that all other requirements are met
  - Allow ducks to band together into flocks and subflocks
  - Generate a notification when a duck quacks
- Note: to avoid coding to an implementation, replace all instances of the word "duck" above with the word "Quackable"

Opportunities for Patterns

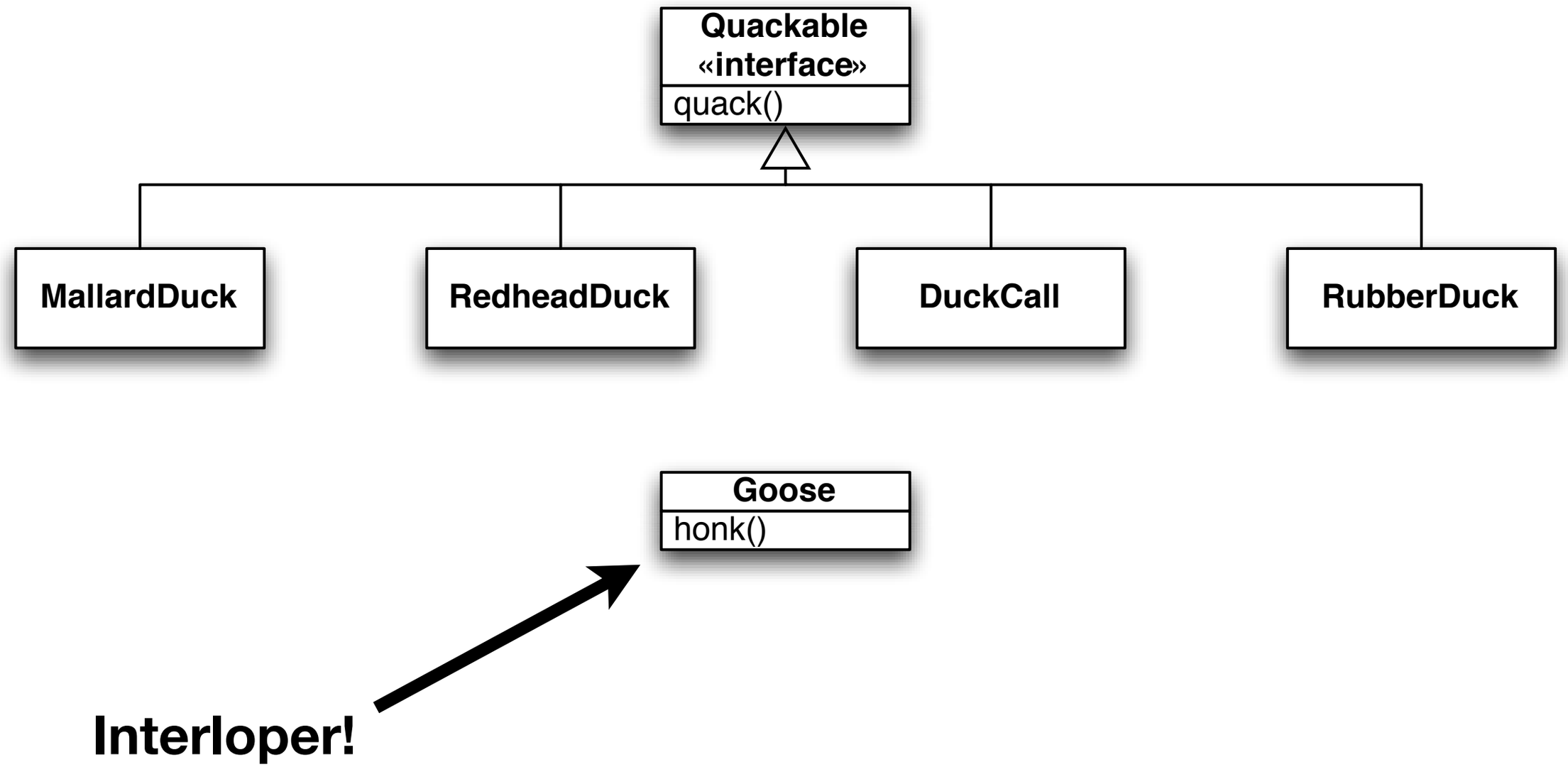
- There are several opportunities for adding patterns to this program
- New Requirements
  - Ducks are the focus, but other water fowl (e.g. Geese) can join in too **(ADAPTER)**
  - Need to keep track of how many times duck's quack **(DECORATOR)**
  - Control duck creation such that all other requirements are met **(FACTORY)**
  - Allow ducks to band together into flocks and subflocks **(COMPOSITE and ITERATOR)**
  - Generate a notification when a duck quacks **(OBSERVER)**
- Lets take a look at this example via a class diagram perspective

Step 1: Need an Interface

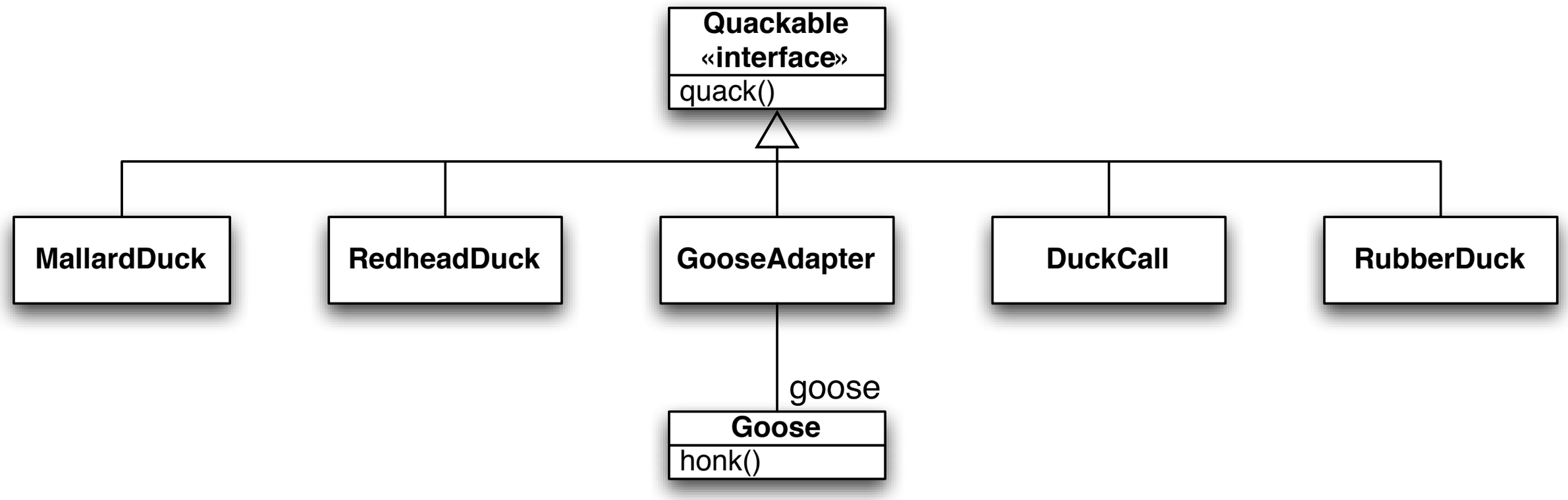


All simulator participants will implement this interface

Step 2: Need Participants

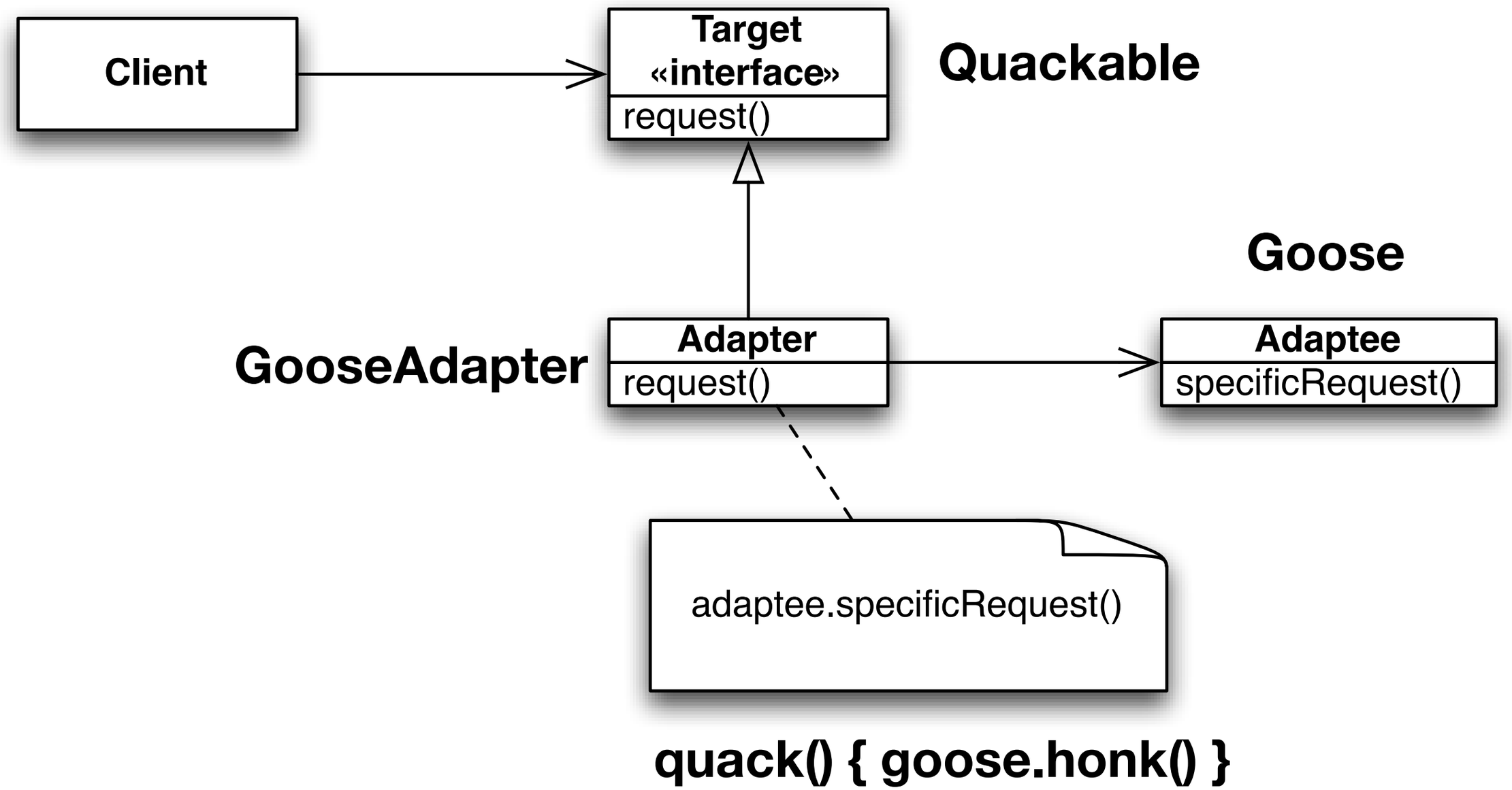


Step 3: Need Adapter

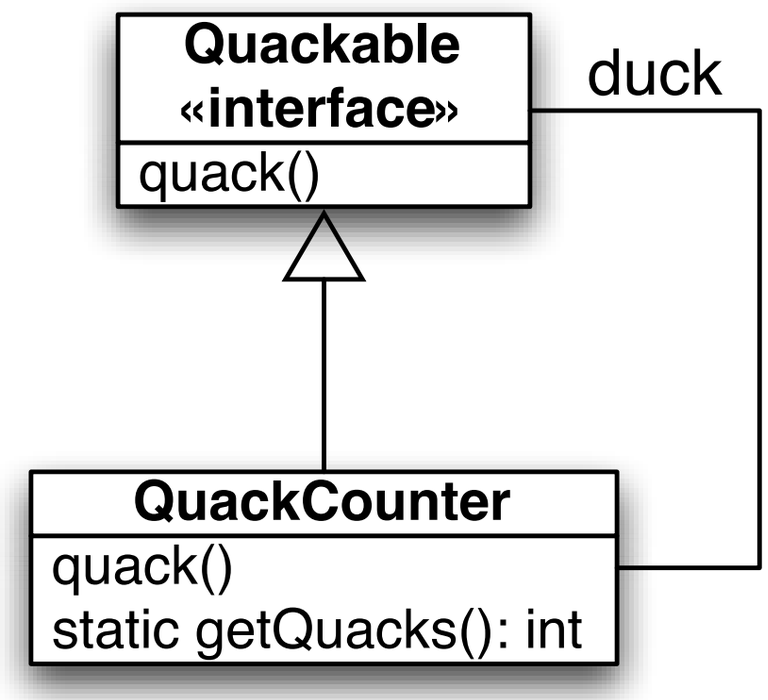


All participants are now Quackables,
allowing us to treat them uniformly

Review: (Object) Adapter Structure



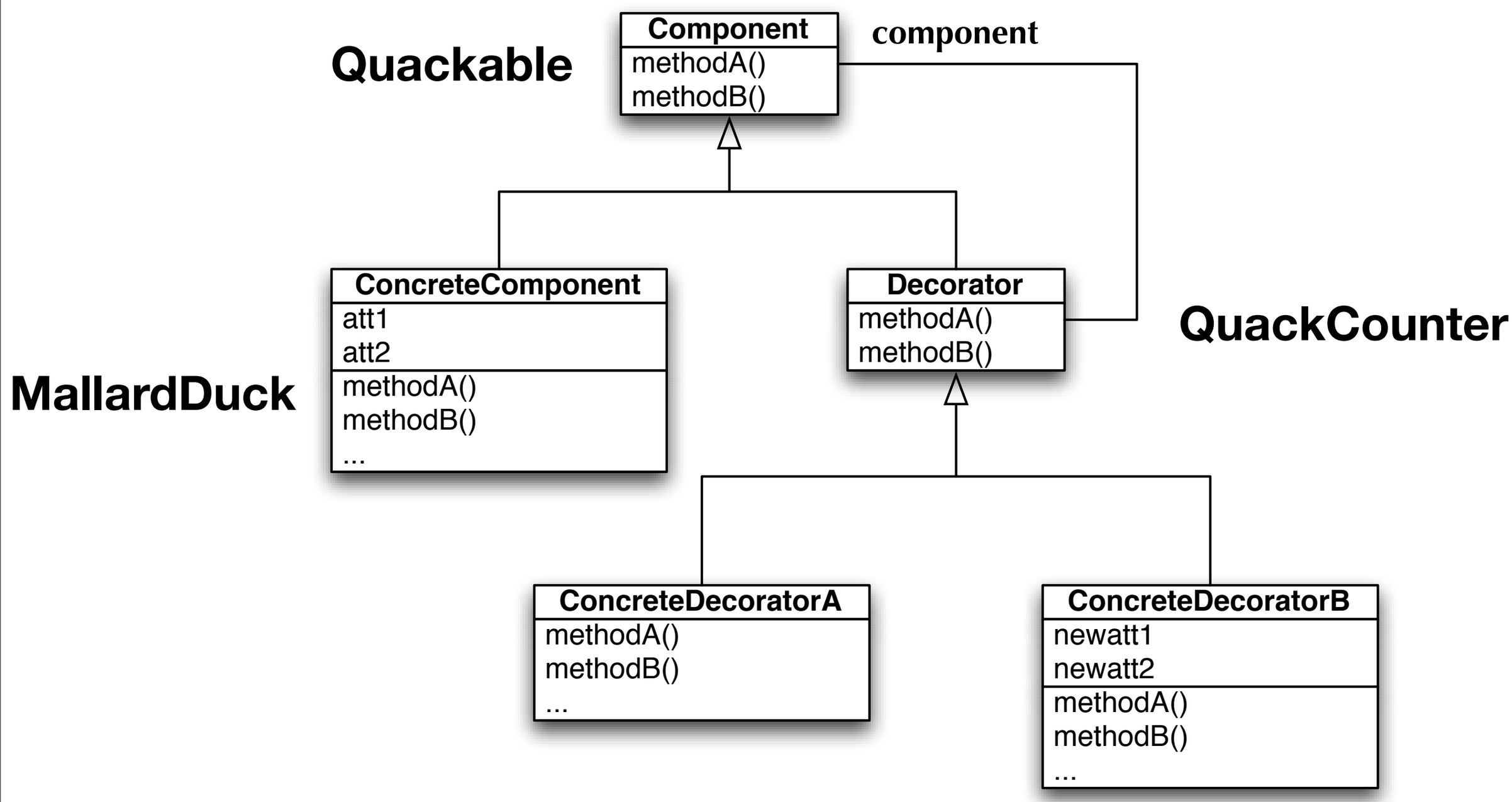
Step 4: Use Decorator to Add Quack Counting



Note: two relationships between QuackCounter and Quackable
What do they mean?

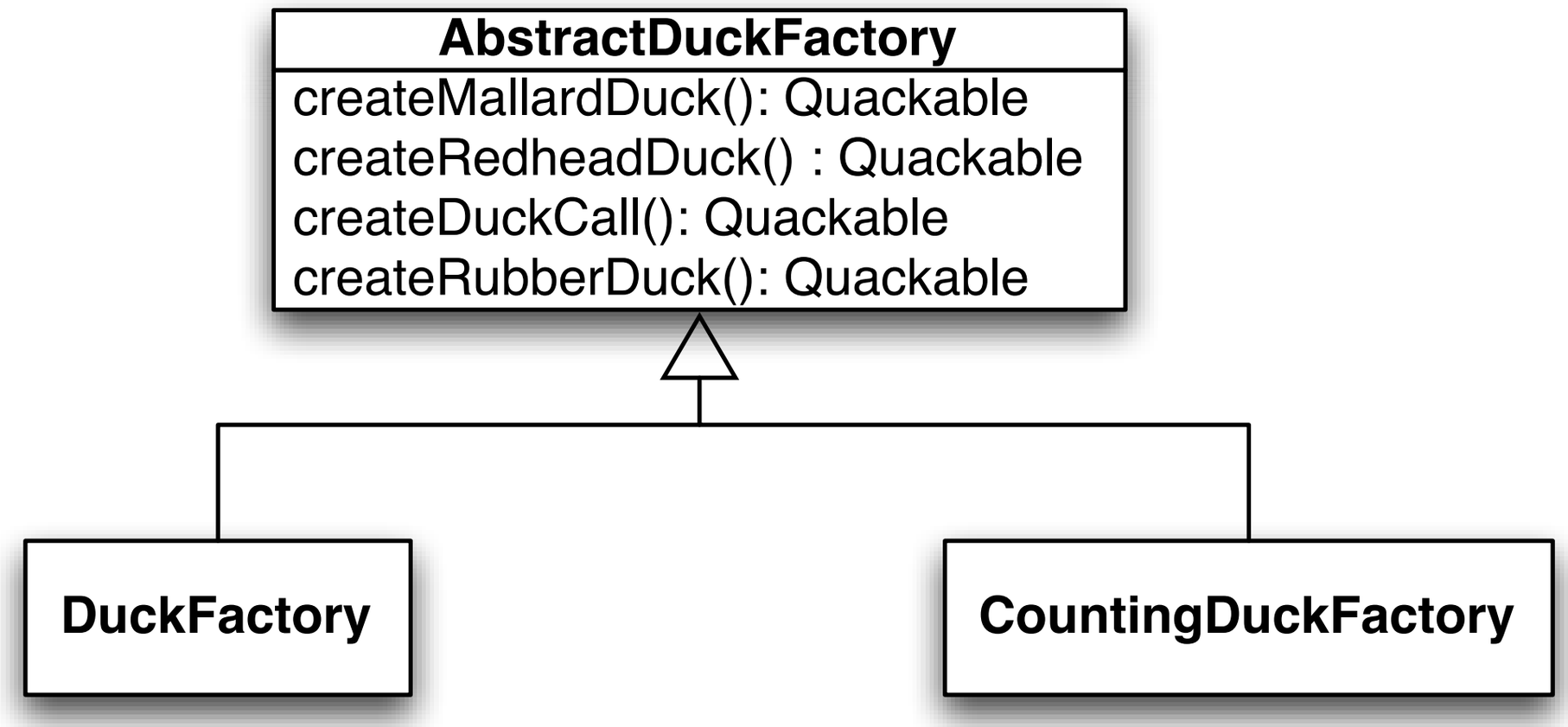
Previous classes/relationships are all still there… just elided for clarity

Review: Decorator Structure



No need for abstract Decorator interface in this situation; note that QuackCounter follows ConcreteDecorators, as it adds state and methods on top of the original interface.

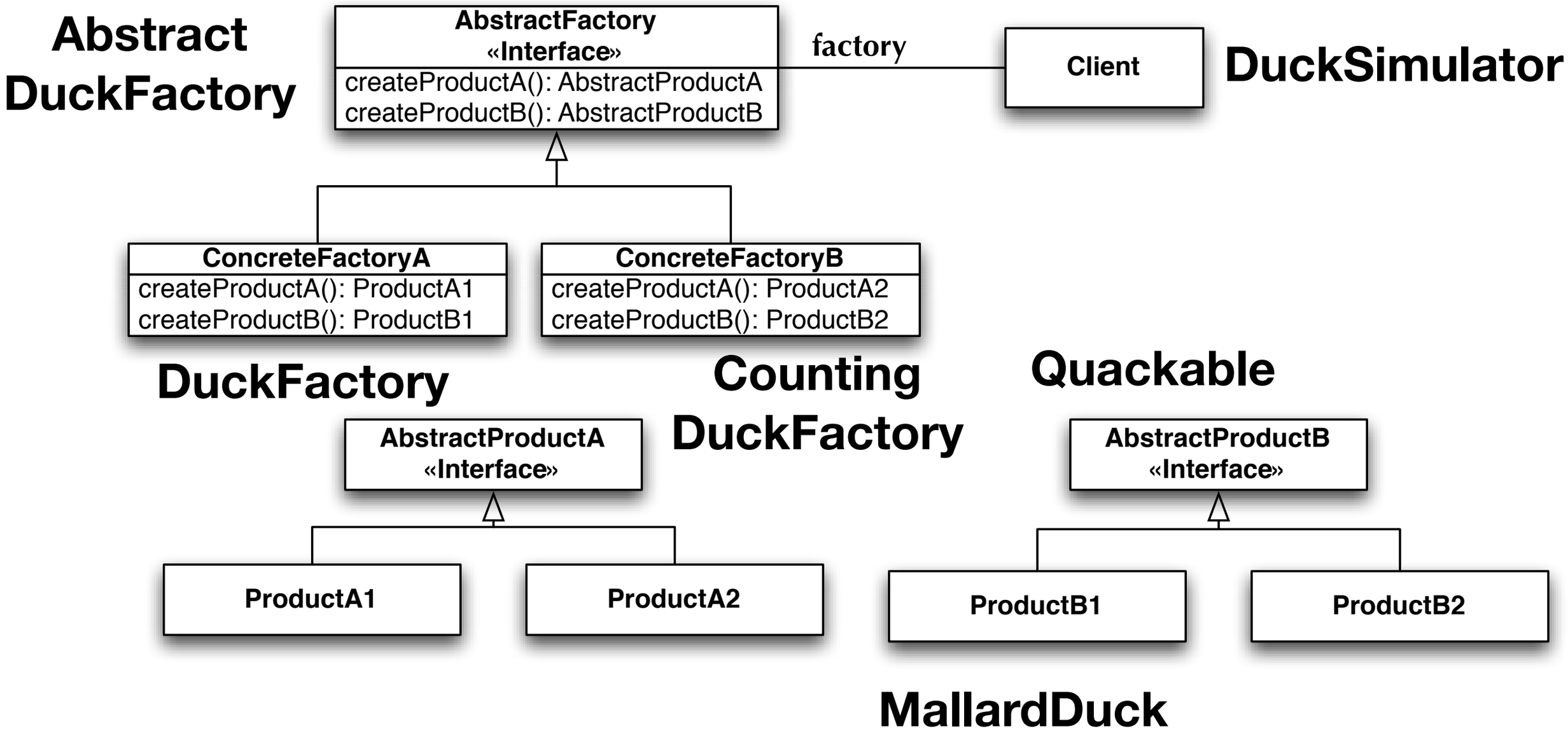
Step 5: Add Factory to Control Duck Creation



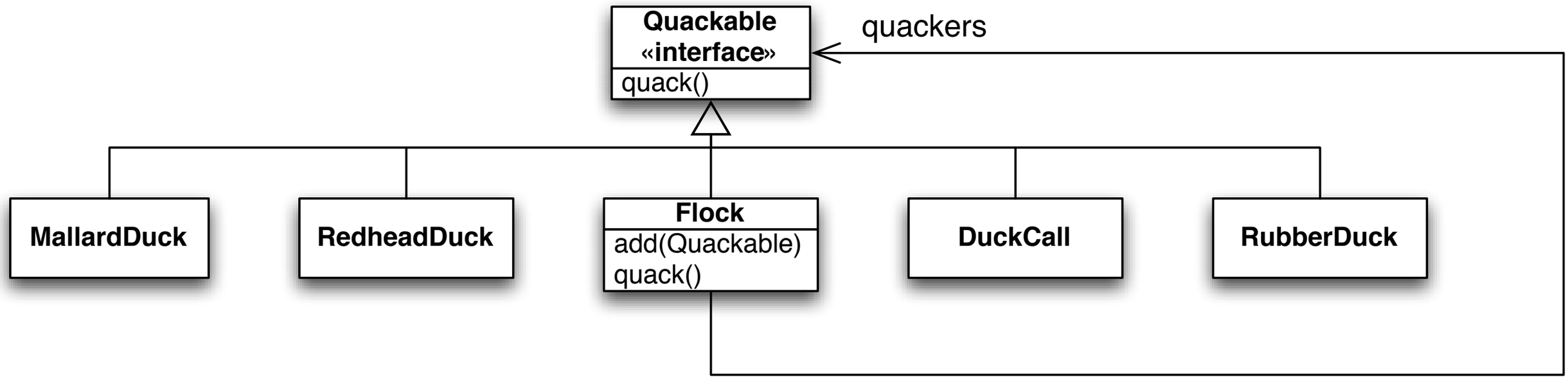
CountingDuckFactory returns ducks that are automatically wrapped by the QuackCounter developed in Step 4

This code is used by a method in DuckSimulator that accepts an instance of AbstractDuckFactory as a parameter. **Demonstration**.

Review: Abstract Factory Structure



Step 6: Add support for Flocks with Composite



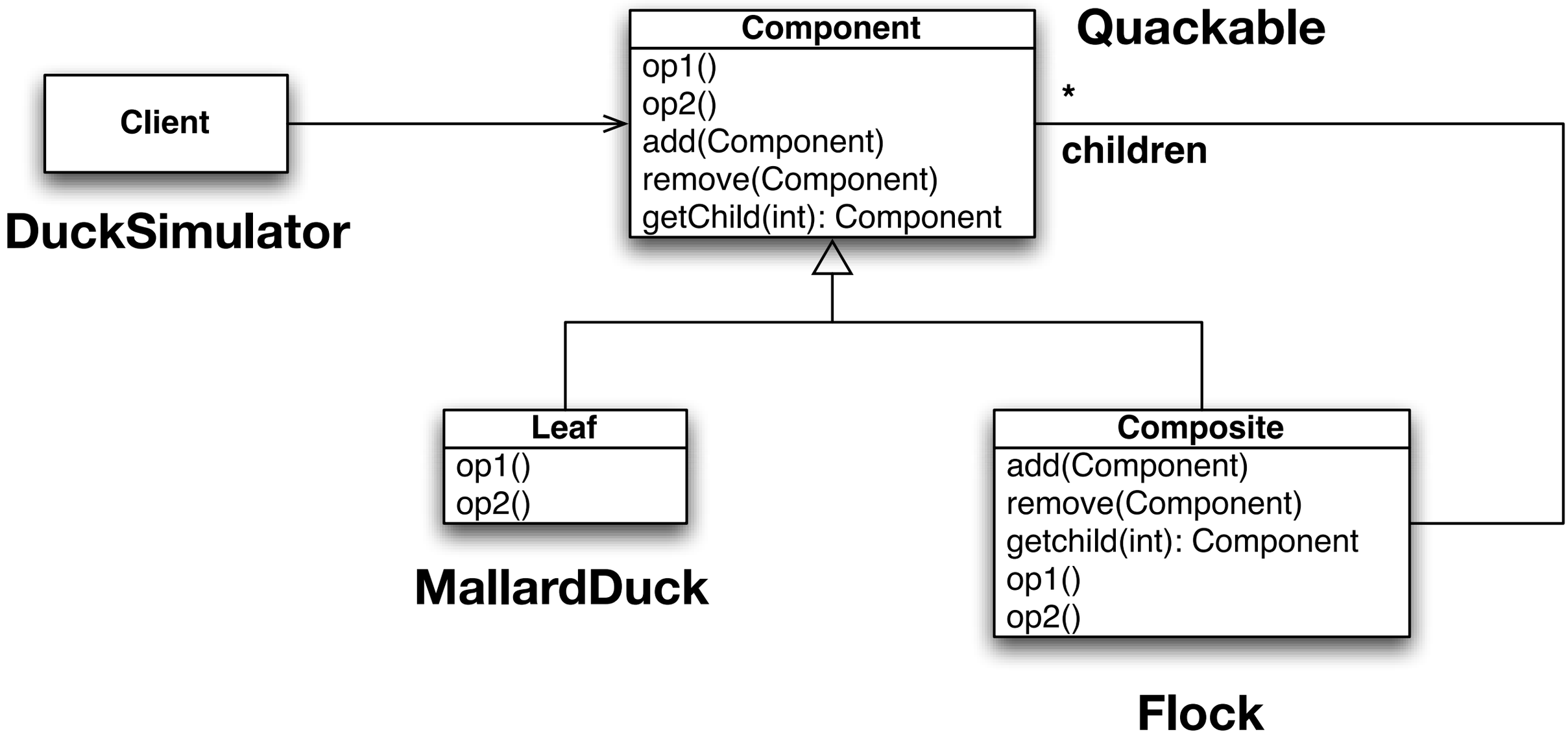
Note: Iterator pattern is hiding inside of Flock.quack(); **Demonstration**

Note: This is a variation on Composite, in which the Leaf and Composite classes have different interfaces;

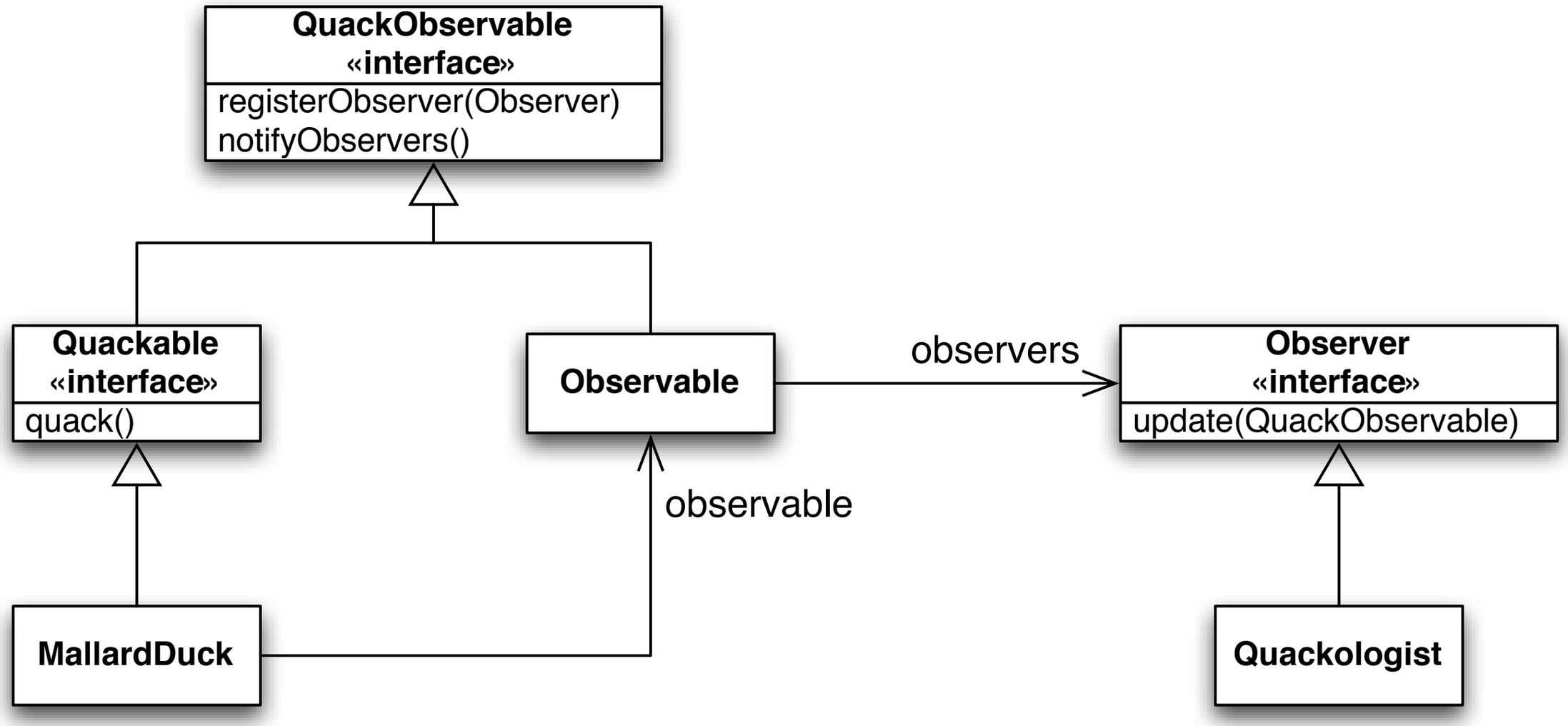
Only Flock has the “add(Quackable)” method.

Client code has to distinguish between Flocks and Quackables as a result. Resulting code is “safer” but less transparent.

Review: Composite Structure

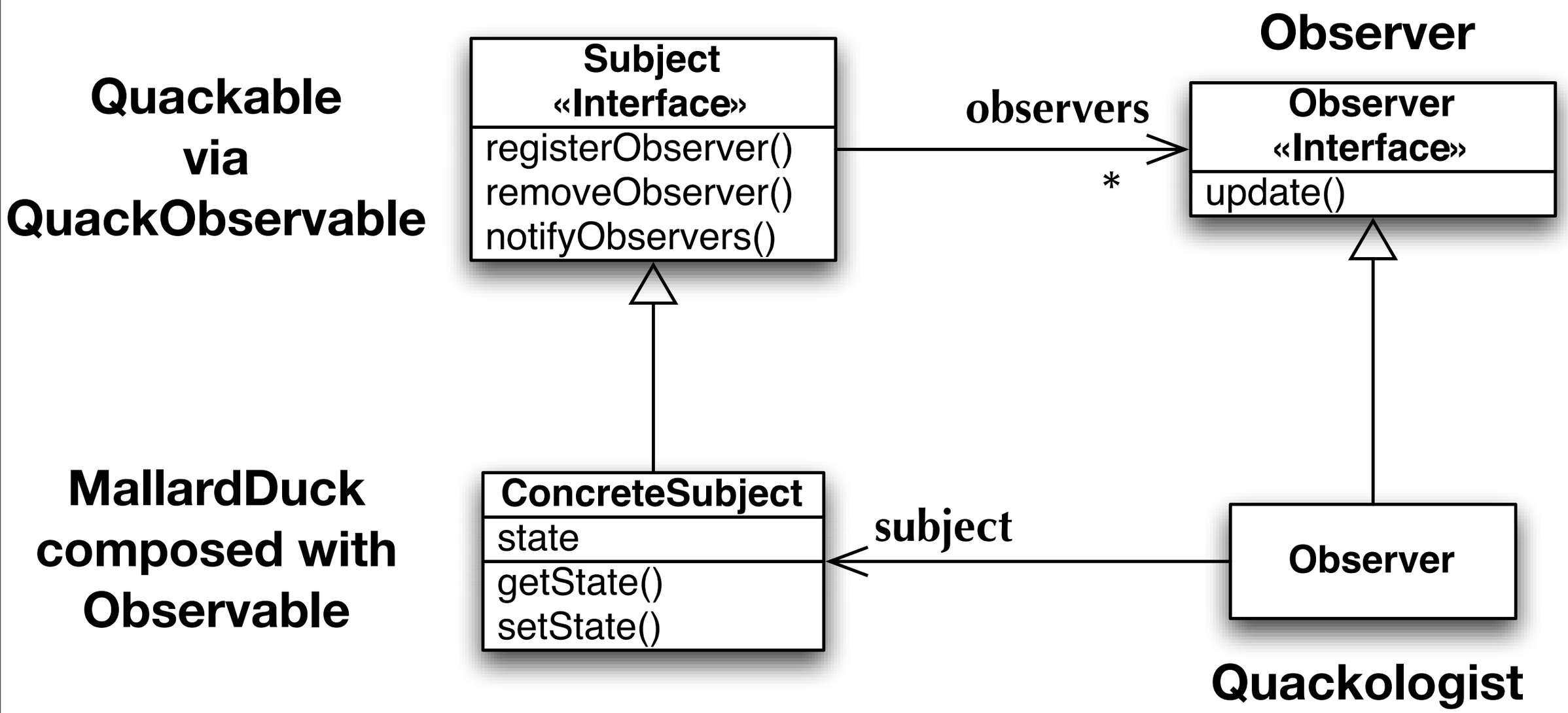


Step 7: Add Quack Notification via Observer



Cool implementation of the Observer pattern. All Quackables are made Subjects by having Quackable inherit from QuackObserver. To avoid duplication of code, an Observable helper class is implemented and composed with each ConcreteQuackable class. Flock does not make use of the Observable helper class directly; instead it delegates those calls down to its leaf nodes. **Demonstration**.

Review: Observer Structure



Counting Roles

- As you can see, a single class will play multiple roles in a design
  - Quackable defines the shared interface for five of the patterns
  - Each Quackable implementation has four roles to play: Leaf, ConcreteSubject, ConcreteComponent, ConcreteProduct
- You should now see why names do not matter in patterns
  - Imagine giving MallardDuck the following name:
    - MallardDuckLeafConcreteSubjectComponentProduct
  - !!!
- Instead, its the structure of the relationships between classes and the behaviors implemented in their methods that make a pattern REAL
  - And when these patterns live in your code, they provide multiple extension points throughout your design. Need a new product, no problem. Need a new observer, no problem. Need a new dynamic behavior, no problem.

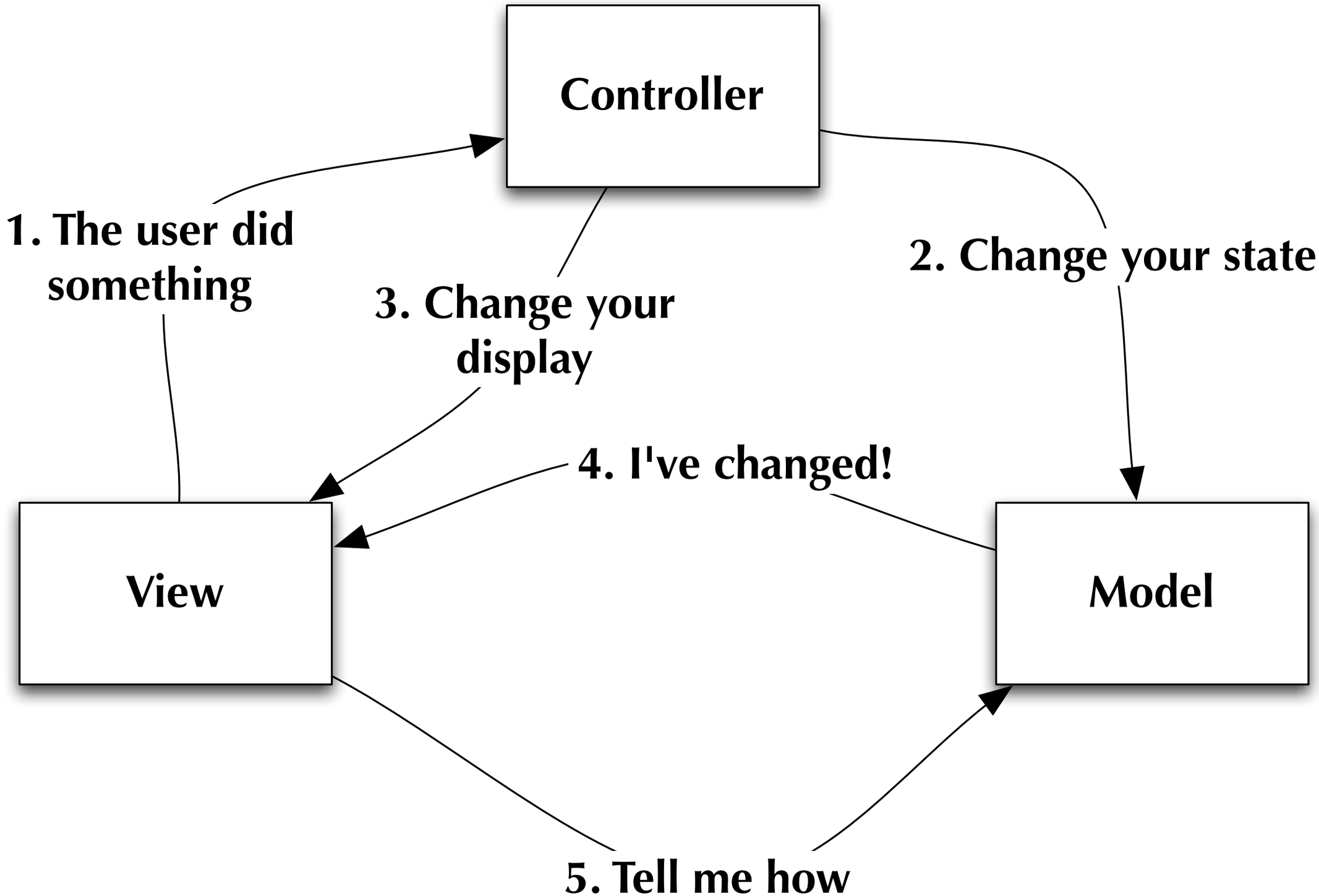
Model-View-Controller: A Pattern of Patterns

- Model-View-Controller (MVC) is a ubiquitous pattern that allows information (stored in models) to be viewed in a number of different ways (views), with each view aided by code that handles user input or notifies the view of updates to its associated models (controllers)
  - Speaking broadly
    - tools/frameworks for creating views are ubiquitous
      - the widgets of any GUI toolkit, templates in Web frameworks, etc.
    - data storage frameworks abound for handling models
      - generic data structures + persistence mechanisms (files, RDBMs, …)
    - controllers are almost ALWAYS written by hand
      - lone exception (that I know of) is Apple's Cocoa Bindings
        - ability to specify a binding between a value maintained by a widget and a similar value in your application's model

MVC Roles

- As mentioned, MVC is a pattern for manipulating information that may be displayed in more than one view
  - Model: data structure(s) being manipulated
    - may be capable of notifying observers of state changes
  - View: a visualization of the data structure
    - having more than one view is fine
    - MVC keeps all views in sync as the model changes
  - Controller: handle user input on views
    - make changes to model as appropriate
    - more than one controller means more than one “interaction style” is available

MVC: Structure



MVC: Hidden Patterns

- Observer pattern used on models
  - Views keep track of changes on models via the observer pattern
    - A variation is to have controllers observe the models and notify views as appropriate
- View and Controller make use of the Strategy pattern
  - When an event occurs on the view, it delegates to its current controller
    - Want to switch from direct manipulation to audio only? Switch controllers
- Views (typically) implement the composite pattern
  - In GUI frameworks, tell the root level of a view to update and all of its sub-components (panels, buttons, scroll bars, etc.) update as well
- Others: Events are often handled via the Command pattern, views can be dynamically augmented with the decorator pattern, etc.

Very Flexible Pattern

- One Model, One Controller, Multiple Views
  - Consider multiple open windows in MacOS X Finder
- One Model, Multiple Controllers, Multiple Views
  - Same example but with Finder windows in multiple modes (icon, list, column)
  - Another example: Spreadsheet, rows and columns view and chart view
- And in certain cases, almost all of the view and controller can be automated
  - Example: Cocoa Bindings

MVC Examples

- DJView
  - Example of one model, one controller, two views
  - Allows you to set a value called “beats per minute” and watch a progress bar “pulse” to that particular value
  - In book, referenced a bunch of midi-related code that did not work on my machine: was supposed to play a “beat track” that matched the specified tempo
    - I ripped that code out and substituted a thread that emits “beats” at the specified rate
- Heart Controller
  - Shows how previous behavior can be altered by changing the model and controller classes… now progress bar “pulse” mimics a human heart

Wrapping Up

- We've shown two ways in which "patterns of patterns" can appear in code
  - The first is when you use multiple patterns in a single design
    - Each individual pattern focuses on one thing, but the combined effect is to provide you with a design that has multiple extension points
  - The second is when two or more patterns are combined into a solution that solves a recurring or general problem
    - MVC is such a pattern (also known as a Compound pattern) that makes use of Observer, Strategy, and Composite to provide a generic solution to the problem of visualizing and interacting with the information stored in an application's model classes

Executive Summaries

- C# Threads by Khalid Al-Enazi
- CakePHP by Khalid Al-Harbi
- Getting to Know Javascript by Peter Alston
- Python Multiprocessing by Ali Alzabarah
- Java @Annotations by Matt Beldyk
- Java Concurrency Framework by Aditya Bhave
- Object Relational Mapping by Alex Boughton

C# Threads – Presentation Summary

- This presentation starts by discussing the basic threading operations and the classes in ***System.Threading*** namespace that supports them, namely, **Thread Class**, **Monitor Class** in addition to the **Lock Statement**
- This serves as a good foundation for someone who's new to the topic.
- Afterwards, the presentation covers on selective basis more classes serving more complex requirements (e.g. Client\Server Architecture with concurrent users). The classes are **Mutex**, **Semaphore**, **ReaderWriterLock** and **ThreadPool**
- Viewing my presentation is beneficial for a student aiming to develop non trivial C# applications. A non-trivial application should have threading logic to be responsive to its users

By: Khaled Alanezi

CakePHP

- The rapid development PHP framework that simplifies the development of robust web applications.
- This presentation helps you understand the Cake's MVC pattern, and shows you why to choose CakePHP and how to use it.
- You'll walk through the process from installing and running it, through building a simple application and a database application.
- Core CakePHP concepts are shown and advanced techniques are explained.

Khalid Alharbi

"Getting to know JavaScript"

by Pete Alston

The aim of this presentation is to give you an overview of the JavaScript language and look at how it functions. It will also discuss the nature of Client Side Scripting and why there is a need for it in modern day Web sites.

Examples of code are presented, along with an explanation of how JavaScript is effectively used today (incorporating AJAX, through Libraries and Frameworks) and an indication of the direction the language is taking in the future.

If you have any questions, email me: peter.alston@colorado.edu

Python Multiprocessing

- My presentation covers the following points :
  - **Quick Python OO concepts introduction with example.**
  - **Quick Python concurrency introduction with example.**
  - **Introduction to Multiprocessing package with link to my previous presentation.**
  - **Detailed analysis of Pool/Forking modules**
    - Class diagram for each module
    - Design patterns principles used.
    - Design patterns violated and how.
    - What is good / bad about them
    - Suggested design patterns for each module and why.
    - Suggested an addition to python concurrency.
    - Thoughts on why those modules were poorly designed.
    - Tips on how to analyze python module.
- If you are interested in any of the above points, this presentation is for you !

Java @Annotations
Presented by Matt Beldyk

Java annotations are a powerful way to add metadata to your classes that can be accessed via reflection. They can enable much cleaner maintainable code and add error checking at compile time. Annotations are a recent addition to Java that enable some behaviors and additions of extra information to classes in an easy manner that is normally reserved to dynamic languages.

Bonus points for including an example that references the Lord of the Rings!

Executive Summary

- The presentation talks about the following things:
  - General concurrency: what it means, how is it important
  - Concurrency in C: with some code thrown in, we explore how it is to write concurrent programs in C, and how it is complicated.
  - Concurrency in Java: Based off previous, point describe the need for abstracting all the concurrency related constructs from application developers, thus a need for Java framework.
  - Threads: How to create and start threads in efficient and scalable manner
  - Synchronization:

    Explore the need for Thread synchronization and different methods to do so like synchronized methods, synchronized statements, atomic variables etc.
  - Explore abstractions for thread creation and starting like Executor, ExecutorService and ScheduledExecutorService
  - Introduction to Thread Pools and usage
  - Usage of ExecutorService and Thread Pools together explained using code examples
  - Explanation of Future interface with code
  - More on Synchronizers like Semaphores and CyclicBarriers with code examples
  - Explanation of AbstractQueueSynchronizer class and the abstract framework it provides based on which various Synchronizers mentioned before are implemented, including code example
  - Introduction to BlockingQueueInterfaces

Object Relational Mapping

- Overview of database management systems
- What is ORDM
- Comparison of ORDM with other DBMSs
- Motivation for ORDM
  - Quick Example
- How does ORDM work
  - Attributes of ORDM
  - Discussion of ORMLite framework
    - Main Components
    - Conceptual Diagrams and Comparisons
- In depth look at ORMLite and using its support for Android OS
  - More Motivation
  - Comparison using ORDM and RDBMS in an Android application
    - Trade offs specific to ORMLite and Android Applications

By Alex Boughton

Wrapping Up

- Individual patterns are not important
  - the important thing is the lessons they provide about good design
- Patterns of Patterns
  - The use of multiple patterns within a system provides a significant amount of flexibility and extensibility in a software system
    - Classes will typically play multiple, well-defined roles in such systems
  - Will sometimes manifest as compound patterns, such as MVC

Coming Up Next

- Homework 8 Due on Friday
- Homework 9 Assigned on Friday
  - Final Homework of the Class
- Lecture 27 and 28: More patterns and other OO topics
  - Refactoring, Dependency Injection, Object-Relational Mapping