Software Design and Architecture

Pattern Focus Design: A Case Study

Duck Simulator

We are to build a new Duck Simulator with the requirements

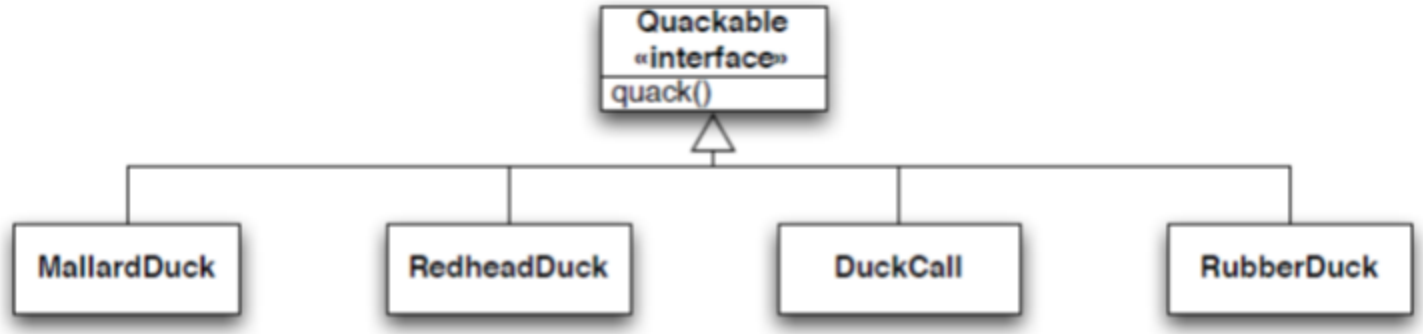
- Ducks are the focus, but other water fowl (e.g. Geese) can join in too
- Need to keep track of how many times duck's quack
- Control duck creation such that all other requirements are met
- Allow ducks to band together into flocks and subflocks
- Generate a notification when a duck quacks

Duck Simulator

To avoid coding to an implementation, create a “Quackable” Interface and replace all instances of “duck” above with the word “Quackable”.

Step 1: Need an Interface

All simulator participants will implement this interface



Duck Simulator

We are to build a new Duck Simulator with the 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

Opportunities for Patterns

Requirements:

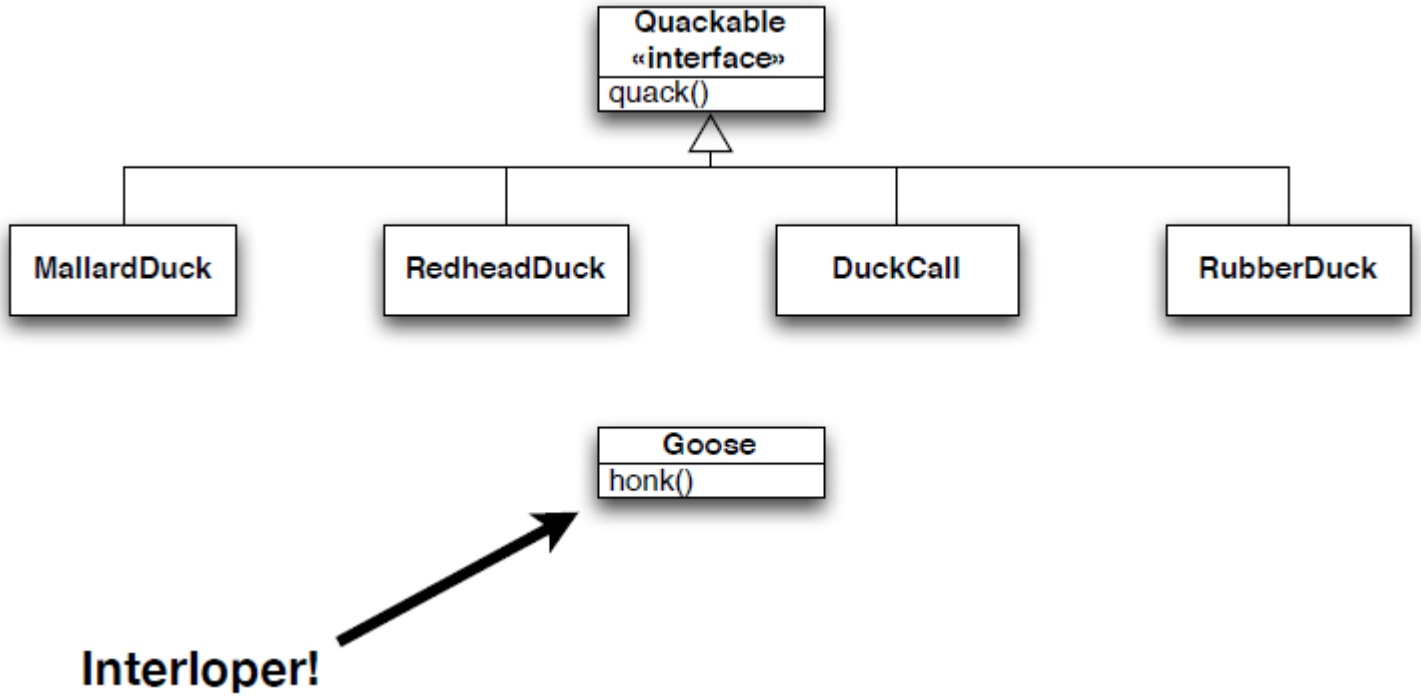
- Ducks are the focus, but other water fowl (e.g. Geese) can join in too
- Need to keep track of how many times duck's quack
- Control duck creation such that all other requirements are met
- Allow ducks to band together into flocks and subflocks
- Generate a notification when a duck quacks

Opportunities for Patterns

Requirements:

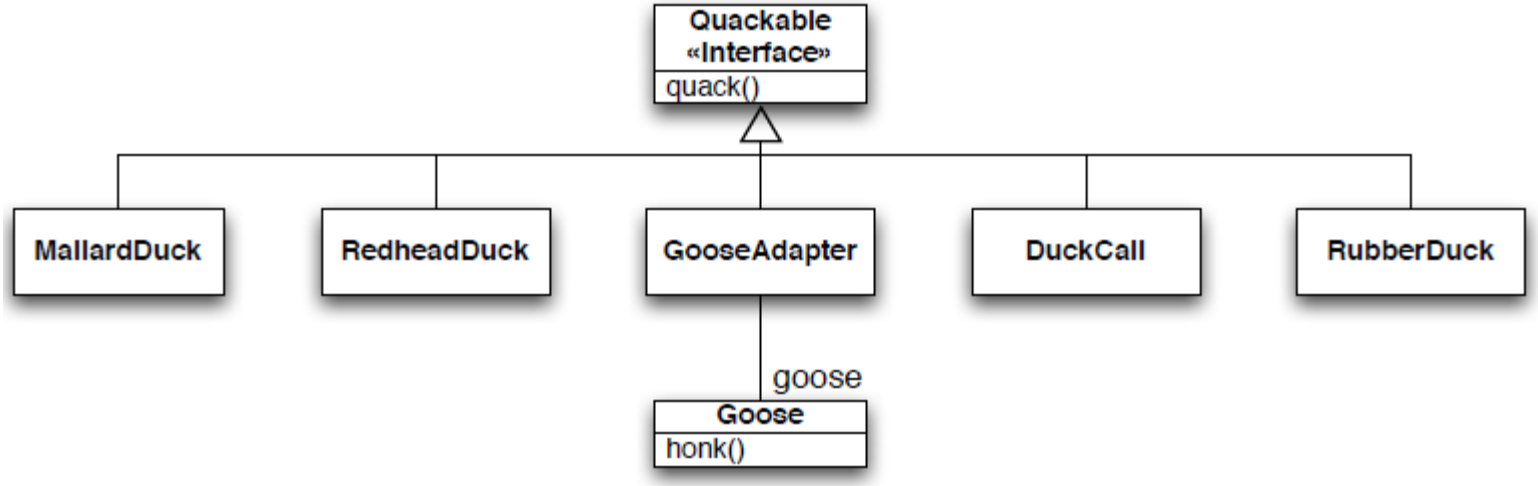
- Ducks are the focus, but other water fowl (e.g. Geese) can join in too **(ADAPTER)**
- Need to keep track of how many times duck's quack
- Control duck creation such that all other requirements are met
- Allow ducks to band together into flocks and subflocks
- Generate a notification when a duck quacks

Step 2: Need Participants

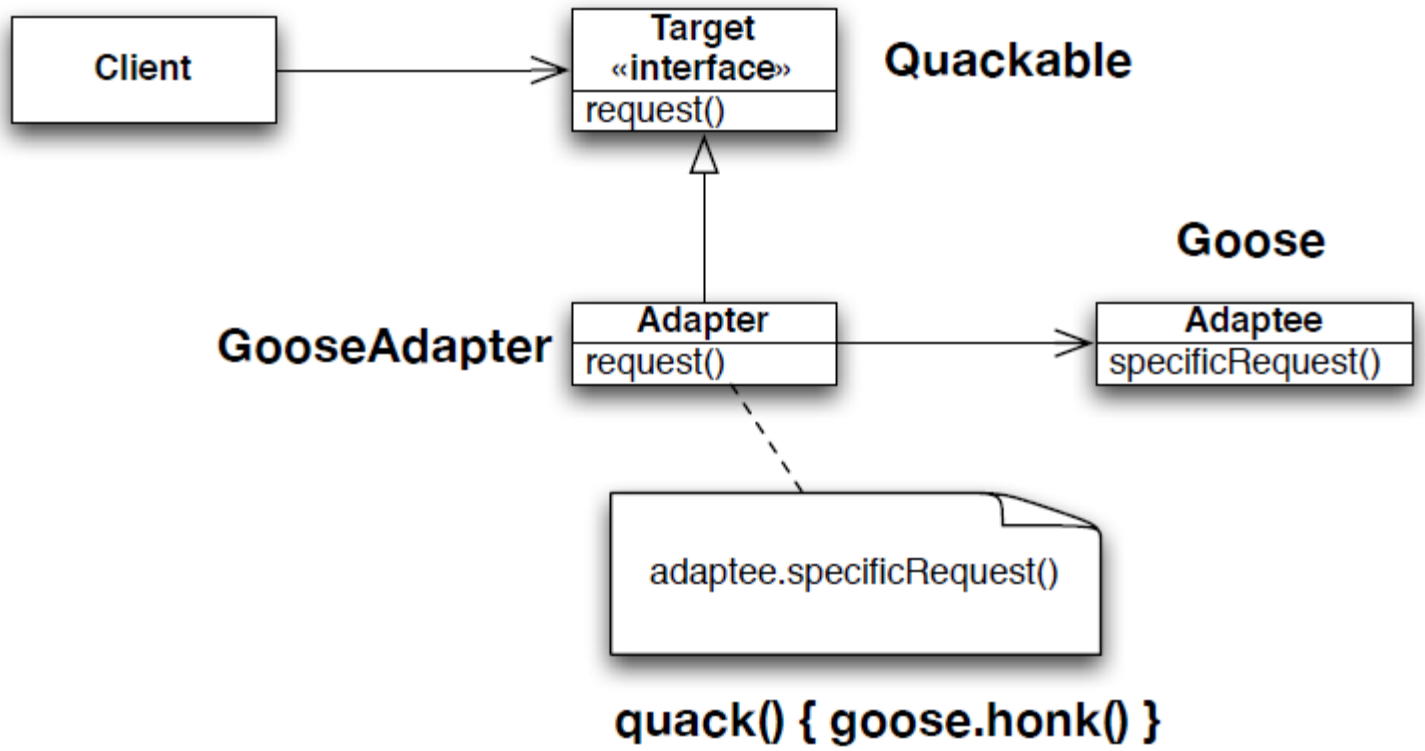


Step 3: Need Adapter

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



Review: (Object) Adapter Structure



Opportunities for Patterns

Requirements:

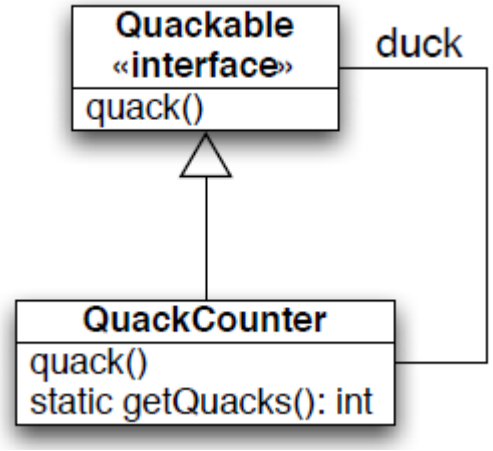
- Ducks are the focus, but other water fowl (e.g. Geese) can join in too **(ADAPTER)**
- Need to keep track of how many times duck's quack
- Control duck creation such that all other requirements are met
- Allow ducks to band together into flocks and subflocks
- Generate a notification when a duck quacks

Opportunities for Patterns

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
- Allow ducks to band together into flocks and subflocks
- Generate a notification when a duck quacks

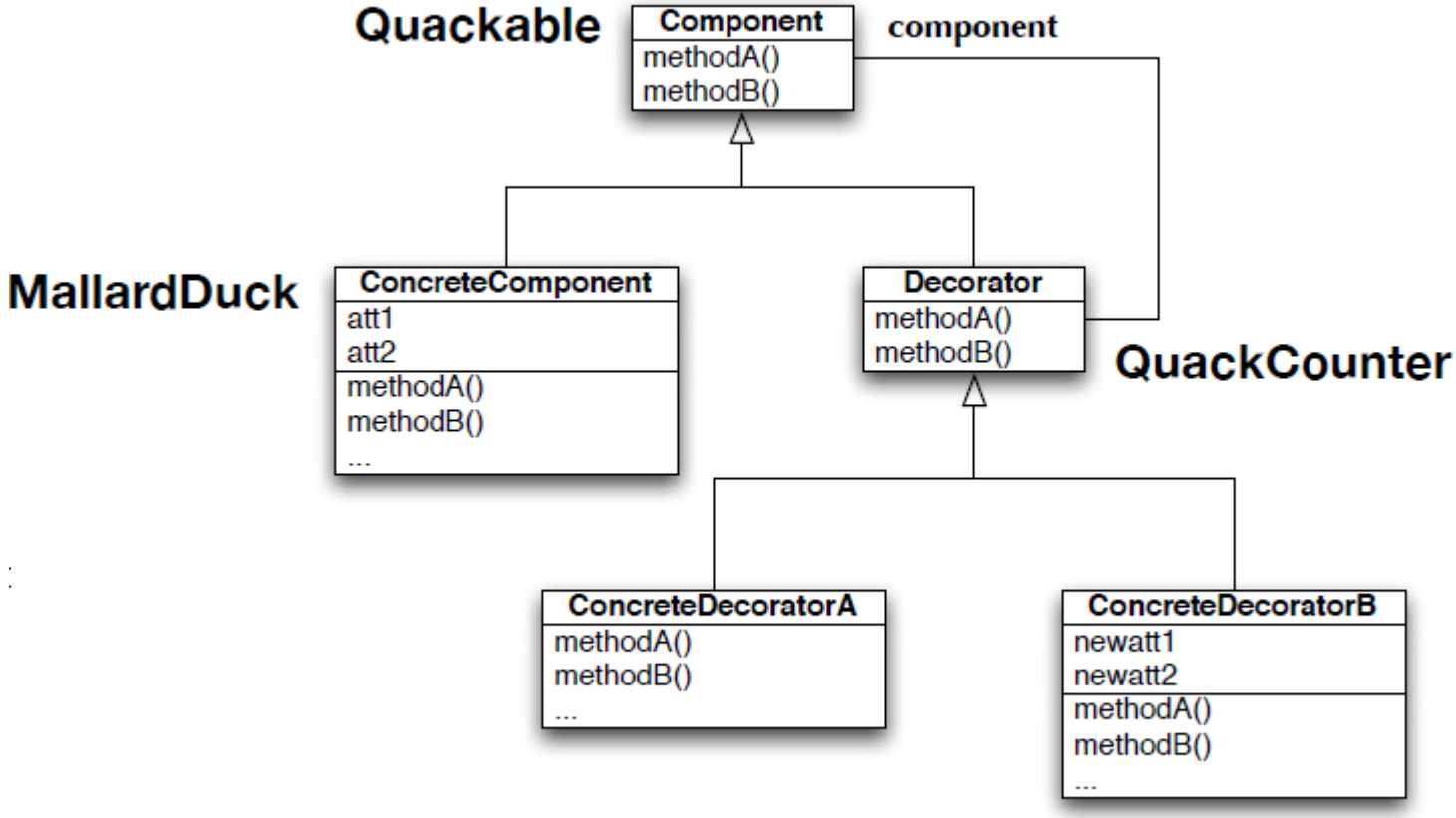
Step 4: Use Decorator to Add Quack Counting



Note: two relationships between QuackCounter and Quackable

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

Review: Decorator Structure



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

Opportunities for Patterns

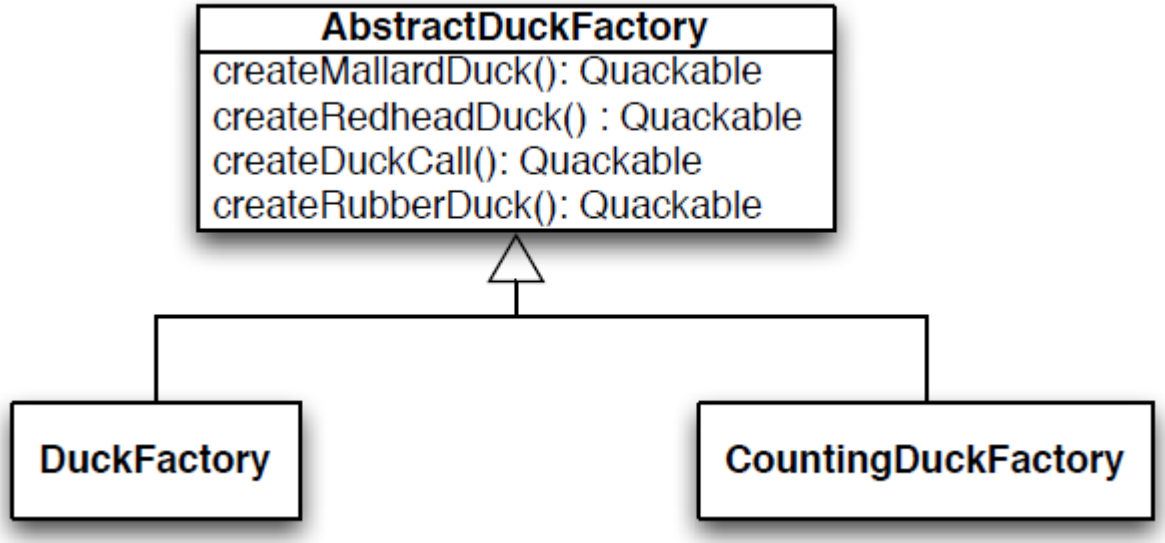
Requirements:

- 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
- Generate a notification when a duck quacks

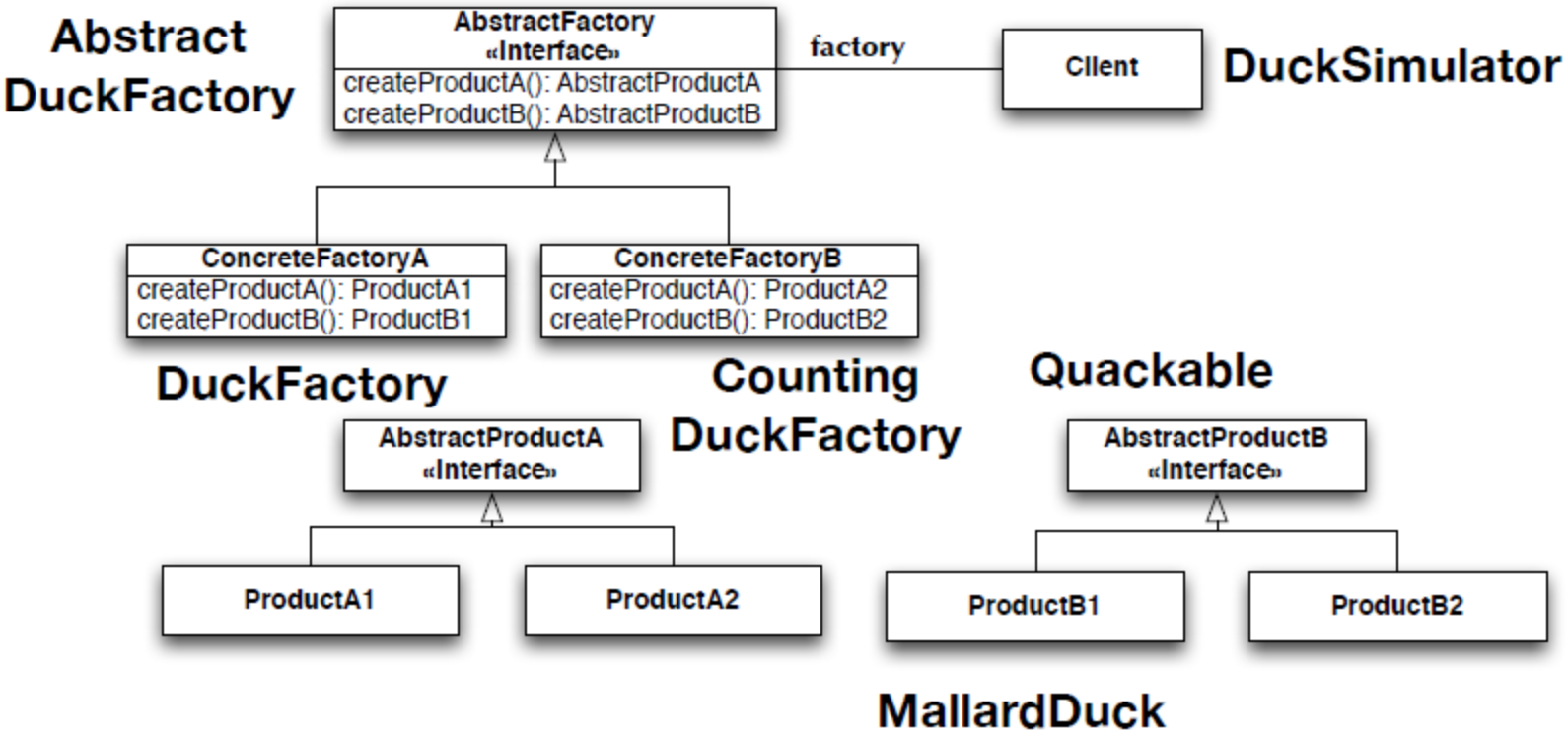
Step 5: Add Factory to Control Duck Creation

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.



Review: Abstract Factory Structure



Opportunities for Patterns

Requirements:

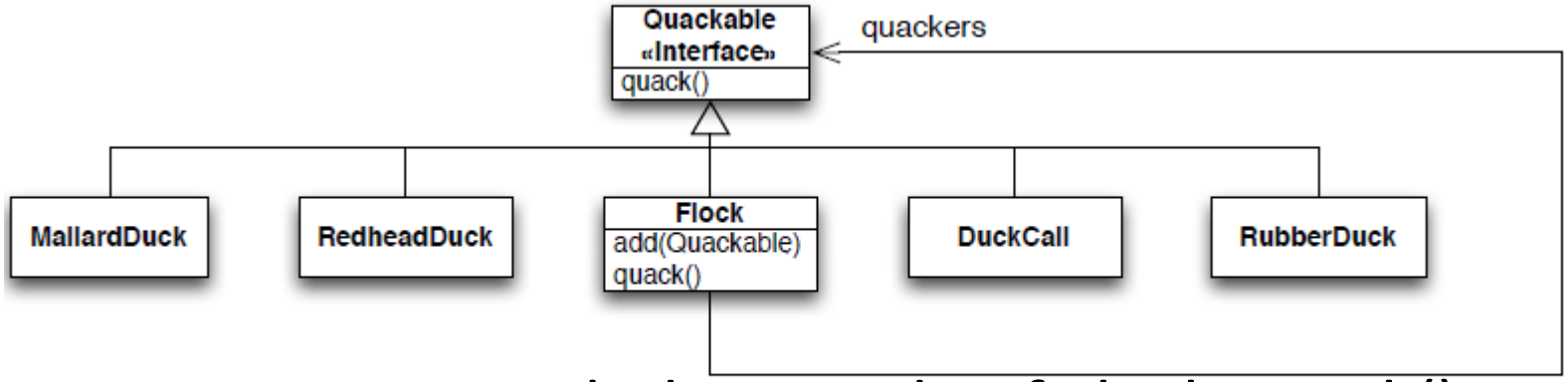
- Ducks are the focus, but other water fowl (e.g. Geese) can join in too **(ADAPTER)**
- Need to keep track of how many times duck's quack **(DECORATOR)**
- Control duck creation such that all other requirements are met **(FACTORY)**
- Allow ducks to band together into flocks and subflocks
- Generate a notification when a duck quacks

Opportunities for Patterns

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

Step 6: Add support for Flocks with Composite

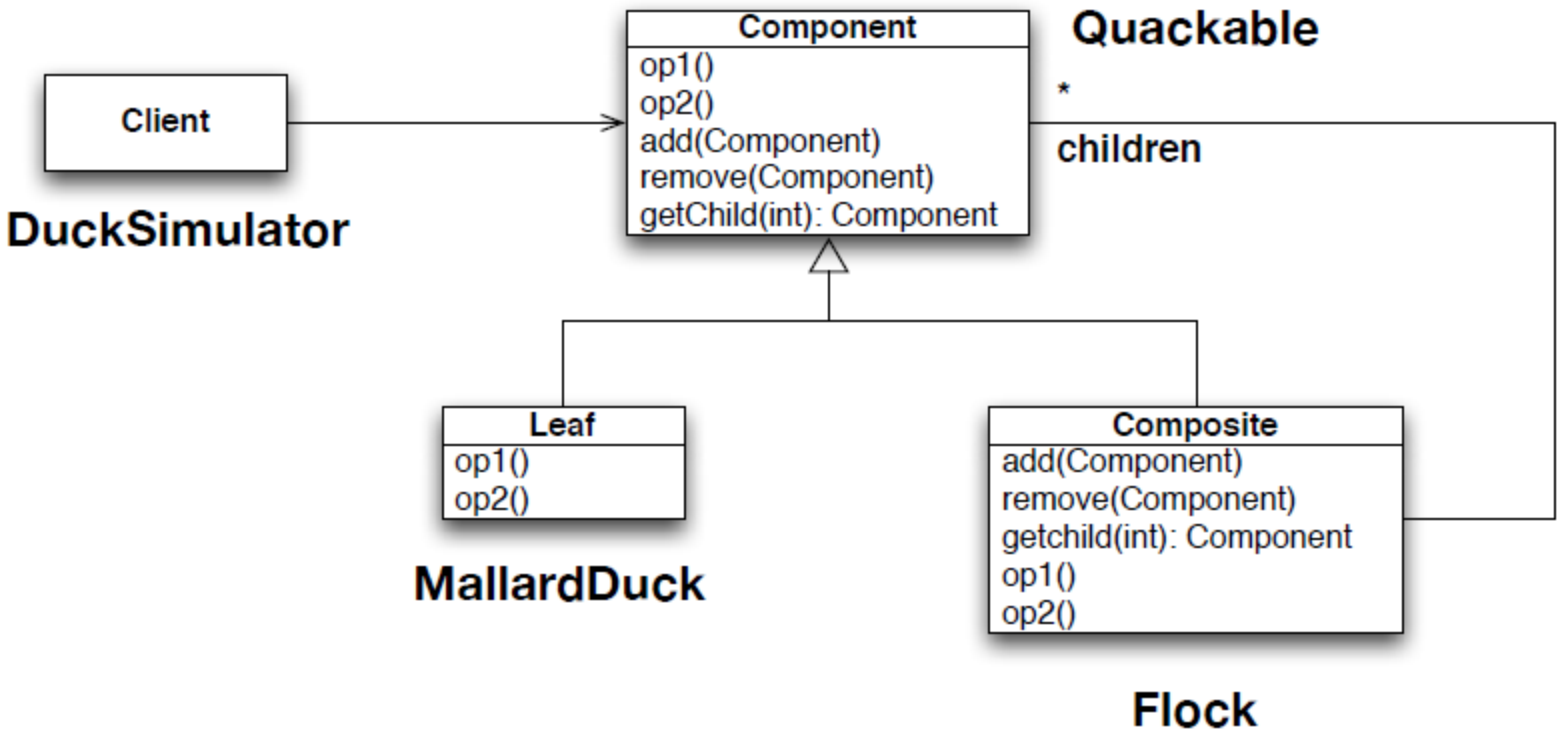


Note: Iterator pattern is hiding inside of Flock.quack();

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



Opportunities for Patterns

Requirements:

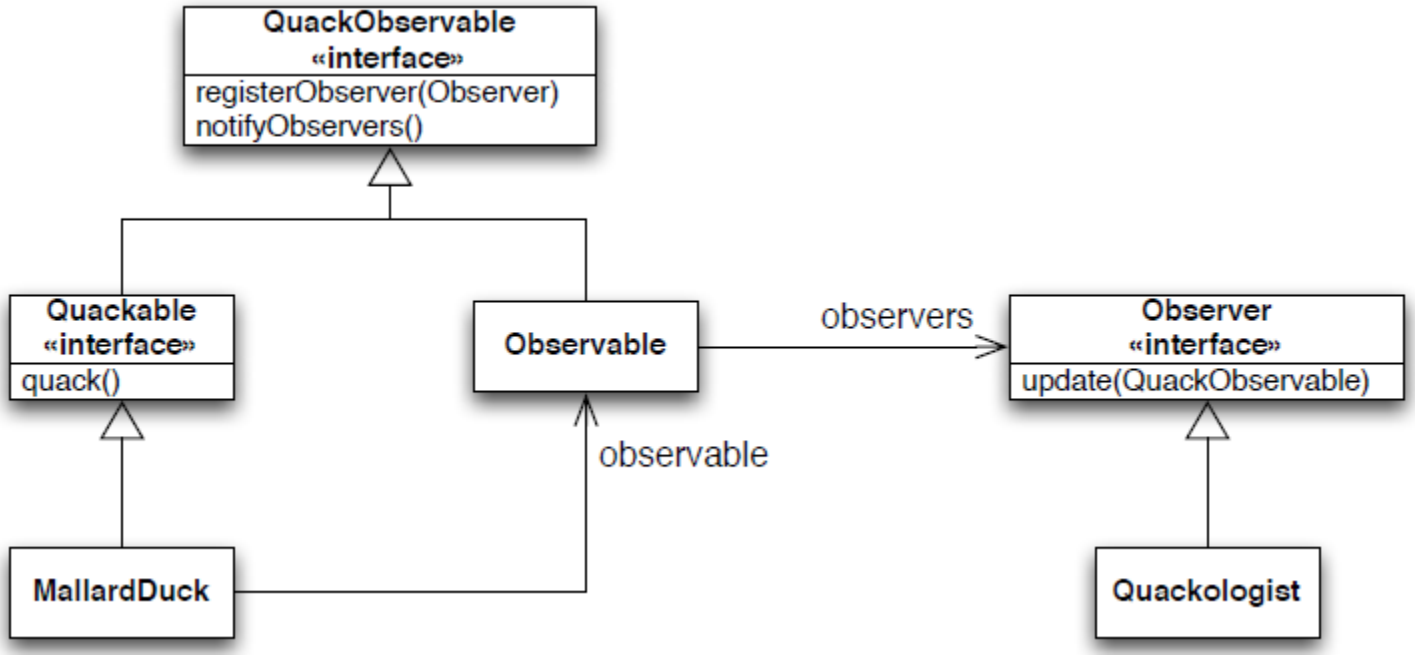
- Ducks are the focus, but other water fowl (e.g. Geese) can join in too **(ADAPTER)**
- Need to keep track of how many times duck's quack **(DECORATOR)**
- Control duck creation such that all other requirements are met **(FACTORY)**
- Allow ducks to band together into flocks and subflocks **(COMPOSITE and ITERATOR)**
- Generate a notification when a duck quacks

Opportunities for Patterns

Requirements:

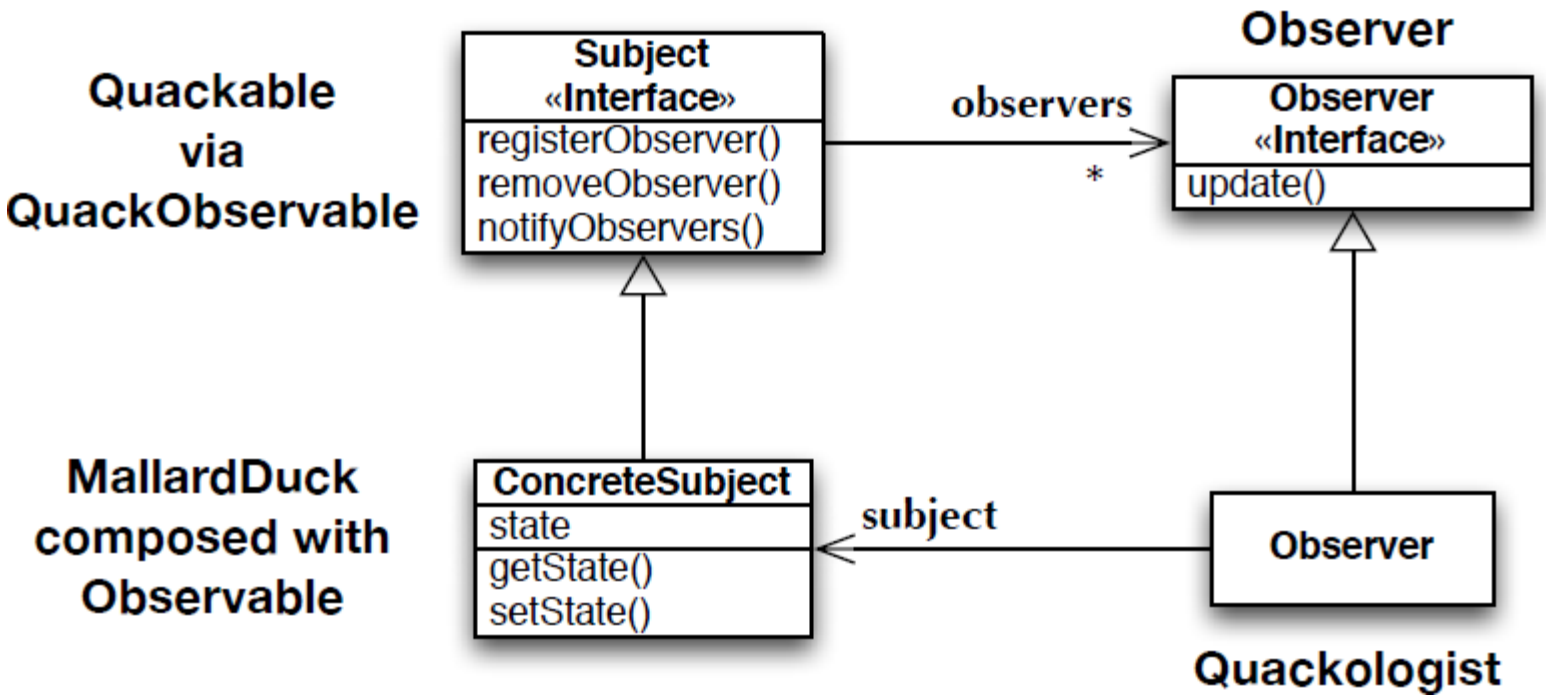
- Ducks are the focus, but other water fowl (e.g. Geese) can join in too **(ADAPTER)**
- Need to keep track of how many times duck's quack **(DECORATOR)**
- Control duck creation such that all other requirements are met **(FACTORY)**
- Allow ducks to band together into flocks and subflocks **(COMPOSITE and ITERATOR)**
- Generate a notification when a duck quacks **(Observer)**

Step 7: Add Quack Notification via Observer



Cool implementation of the Observer pattern. All Quackables are made Subjects by having Quackable inherit from QuackObservable. 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.

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

Counting Roles

You should now see why names do not matter in patterns

- Imagine giving MallardDuck the following name: MallardDuckLeafConcreteSubjectComponentProduct

Instead, it's 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.