CSCI E-97
Lecture 7

October 16, 2014

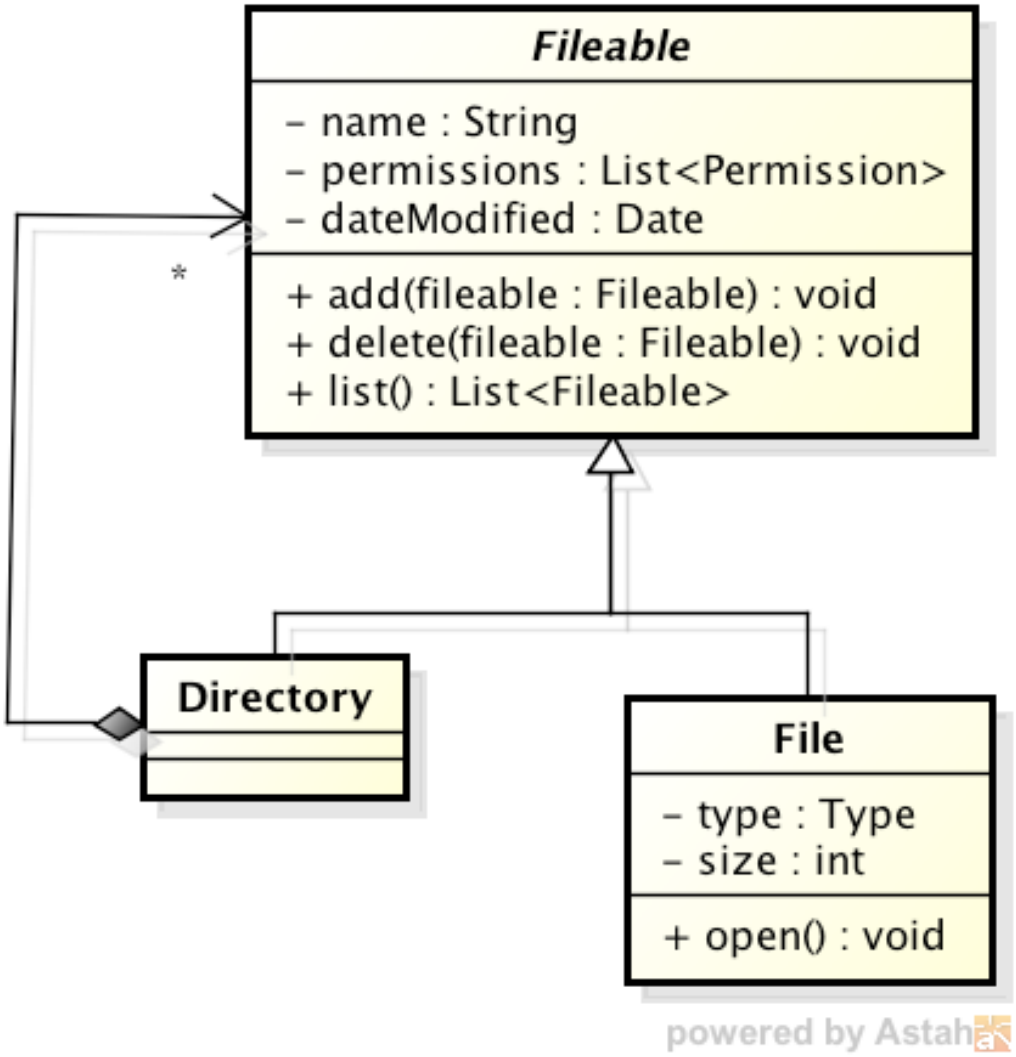
Outline

- A Brief Look at Two Design Patterns
  - Composite (structural)
  - Visitor (behavioral)
- Comments on Assignment 2
- Suggestions for Completing Assignment 3
- Context for Assignment 4

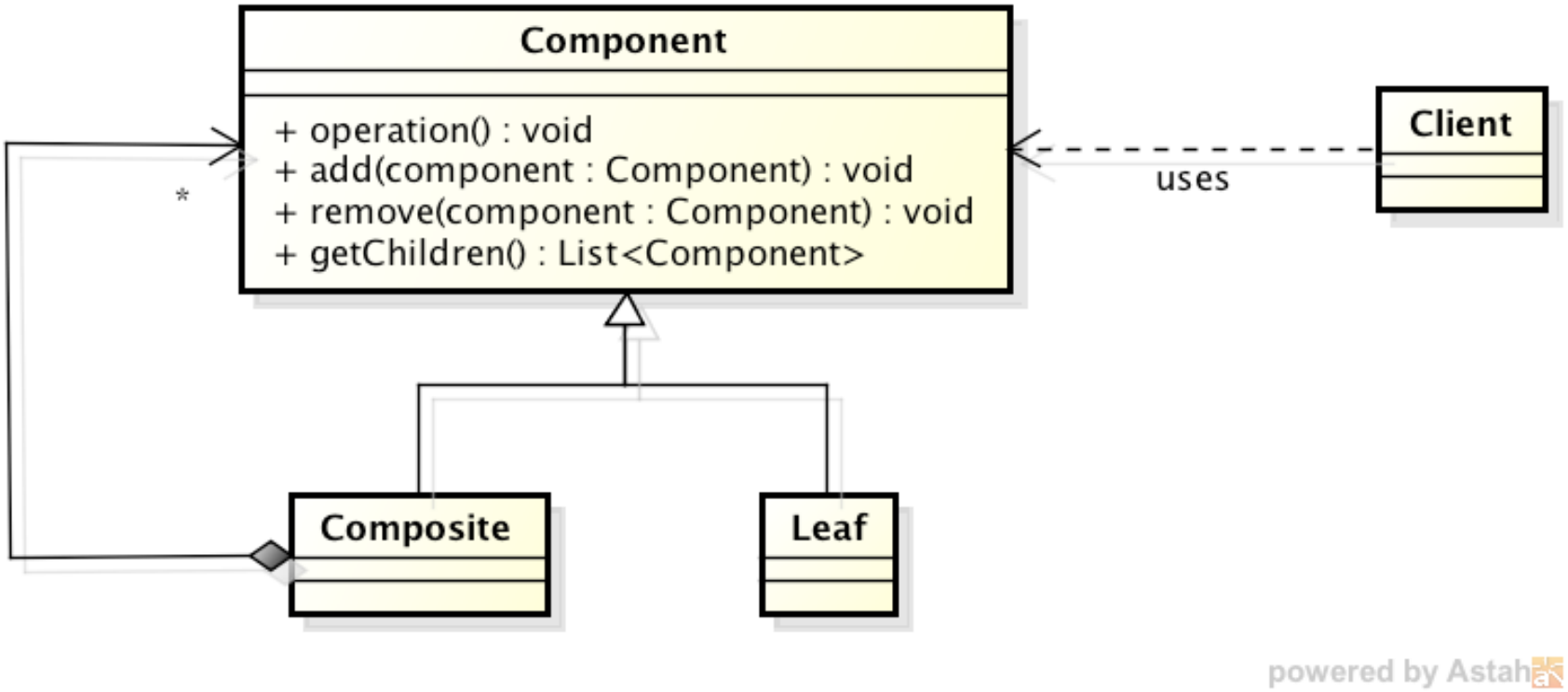
The Composite Pattern

- The Problem: We want to compose objects into tree structures representing whole-part hierarchies so that clients can treat individual objects and containers of objects uniformly

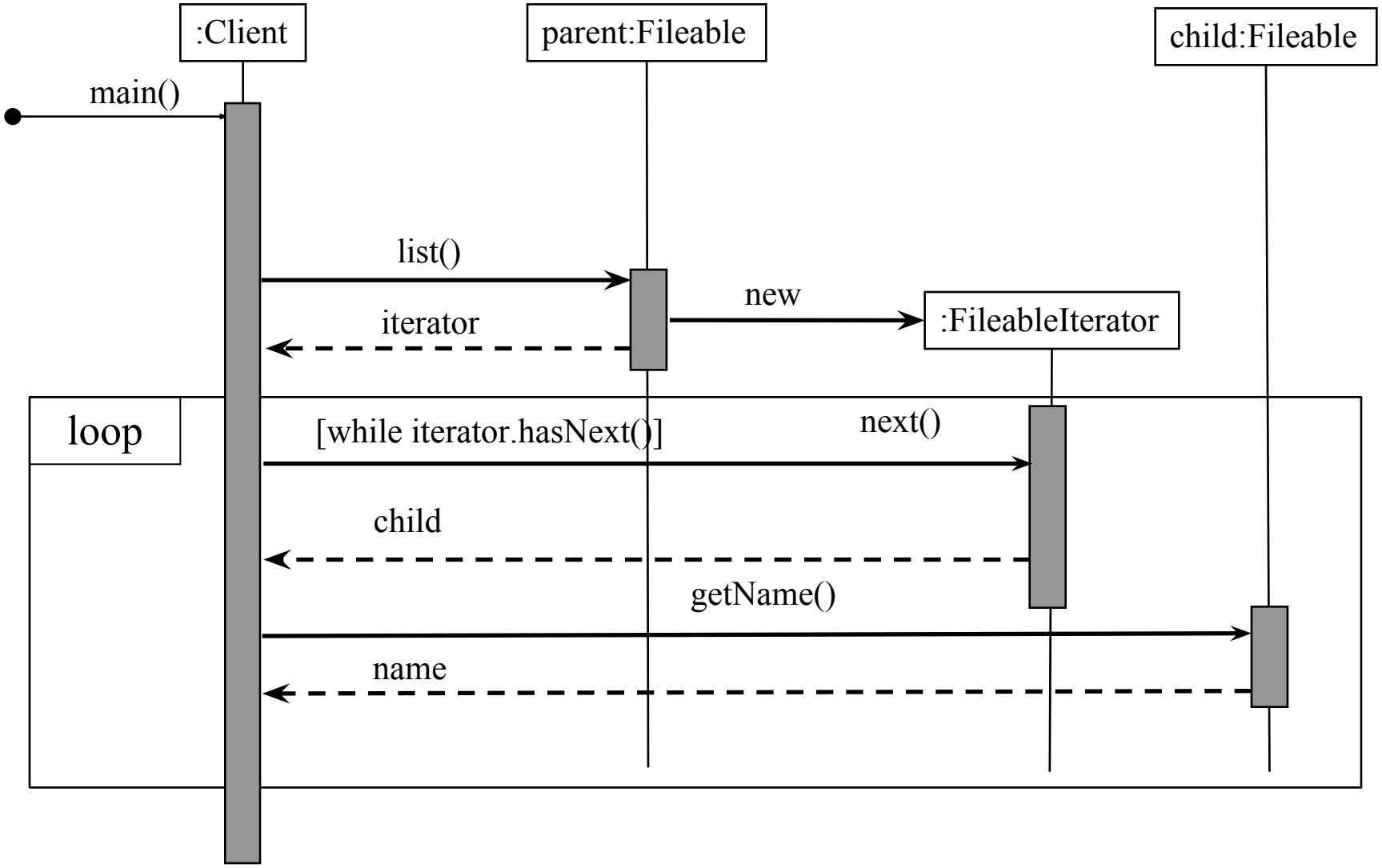
Example – The Filing System is a Composite



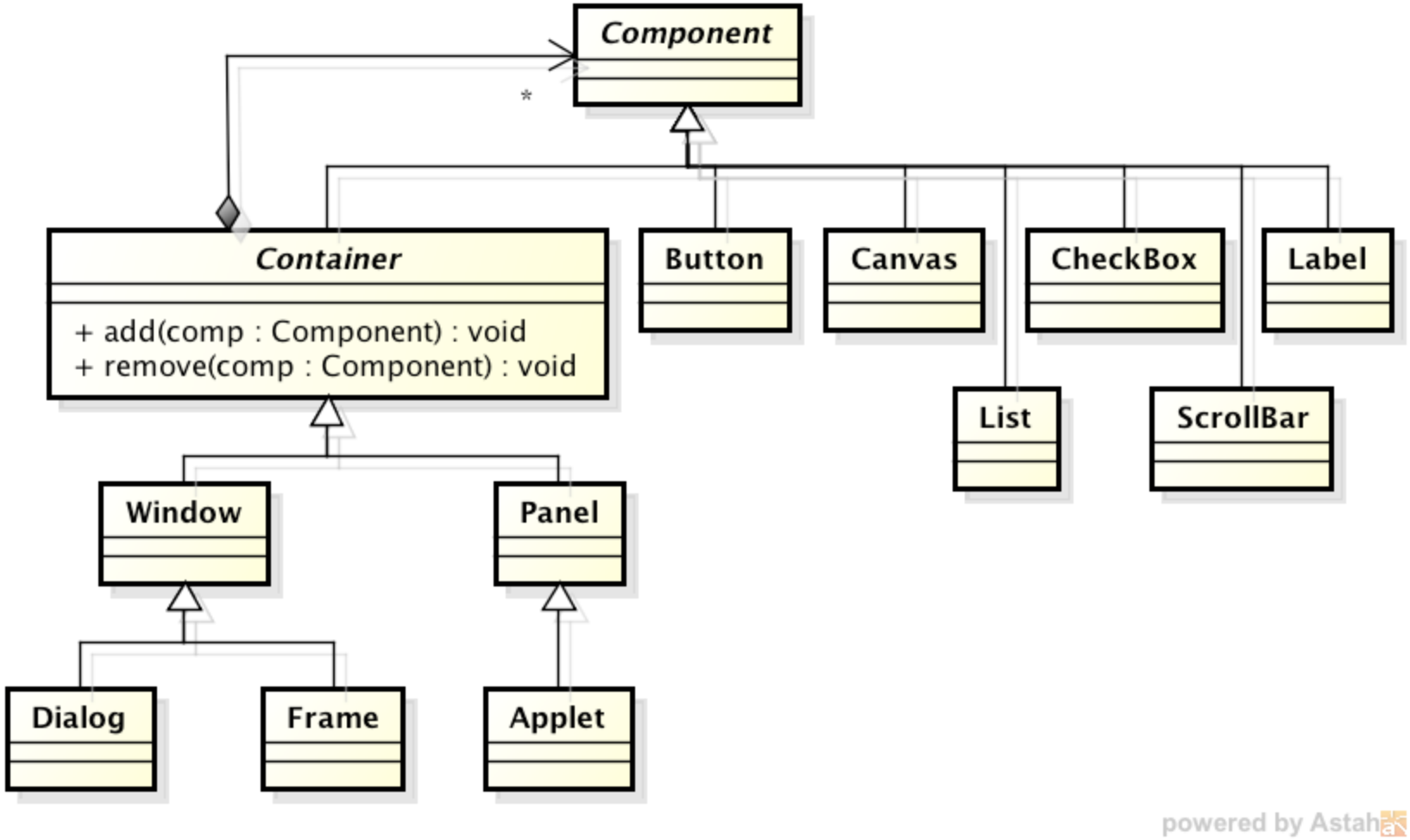
The Generic Model for the Composite Pattern



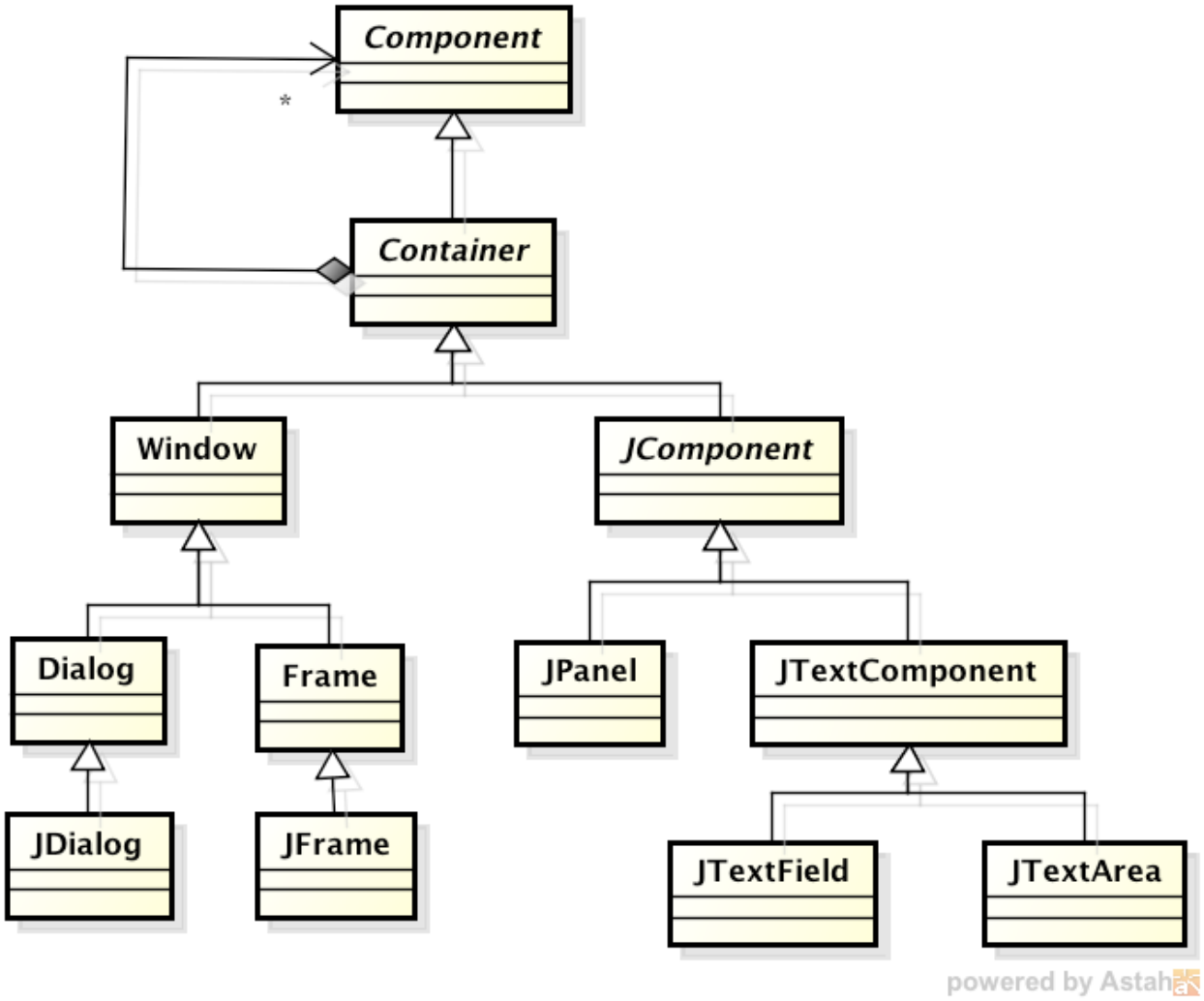
Sequence Diagram for Traversing the File System



Example – Java AWT uses the Composite Pattern



Example – Java Swing Extends AWT



When to Use the Composite Pattern

- Use the Composite Pattern when you have a whole-part hierarchy that has significant similarities among the things that are containers and the things which are not

Tradeoffs for the Composite Pattern

- If the number of exceptional cases for adding one `Component` to a `Composite` is too large, implementing the structure as a composite could be more trouble than it's worth

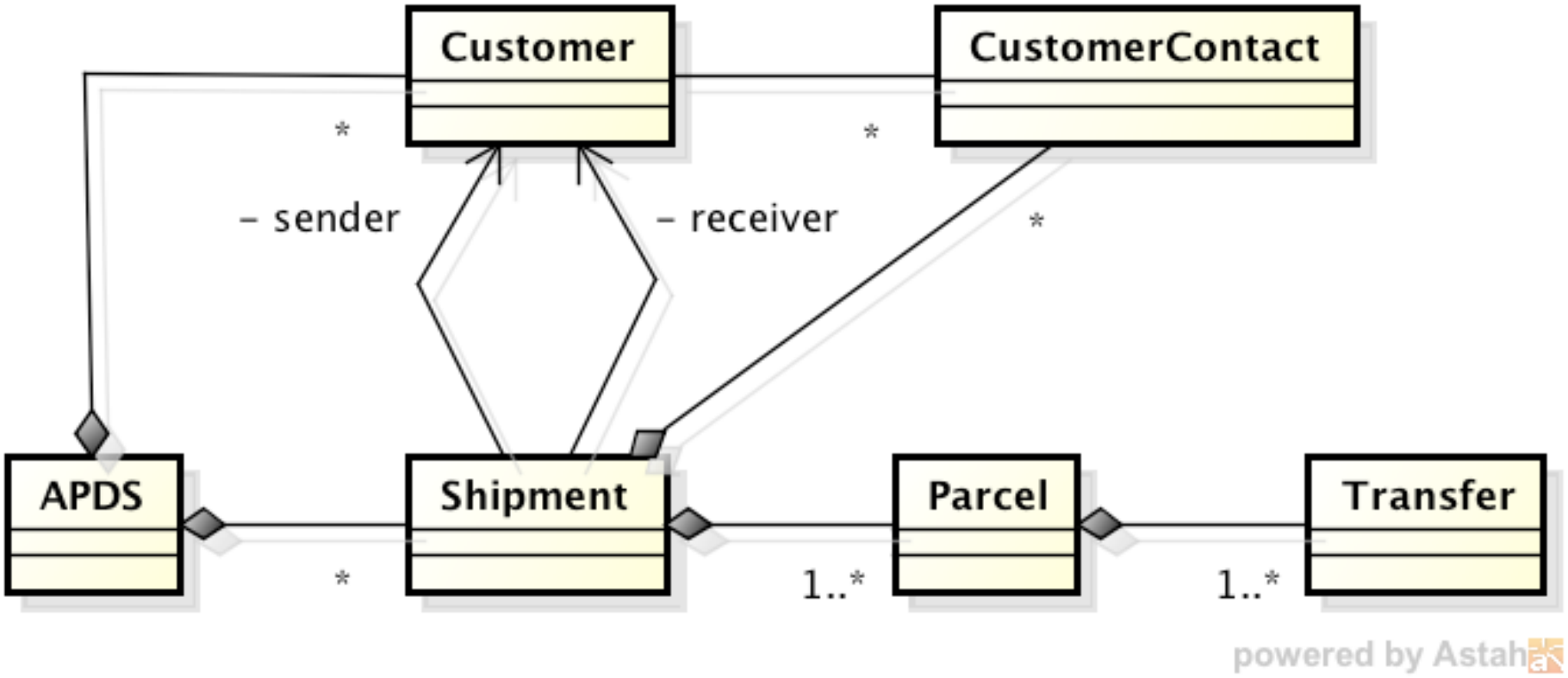
The Visitor Pattern

- The Problem: We have a set of algorithms that are to be applied to all the nodes in a tree or network of objects – we'd like to do this without having to recode the implementation classes every time we add a new algorithm

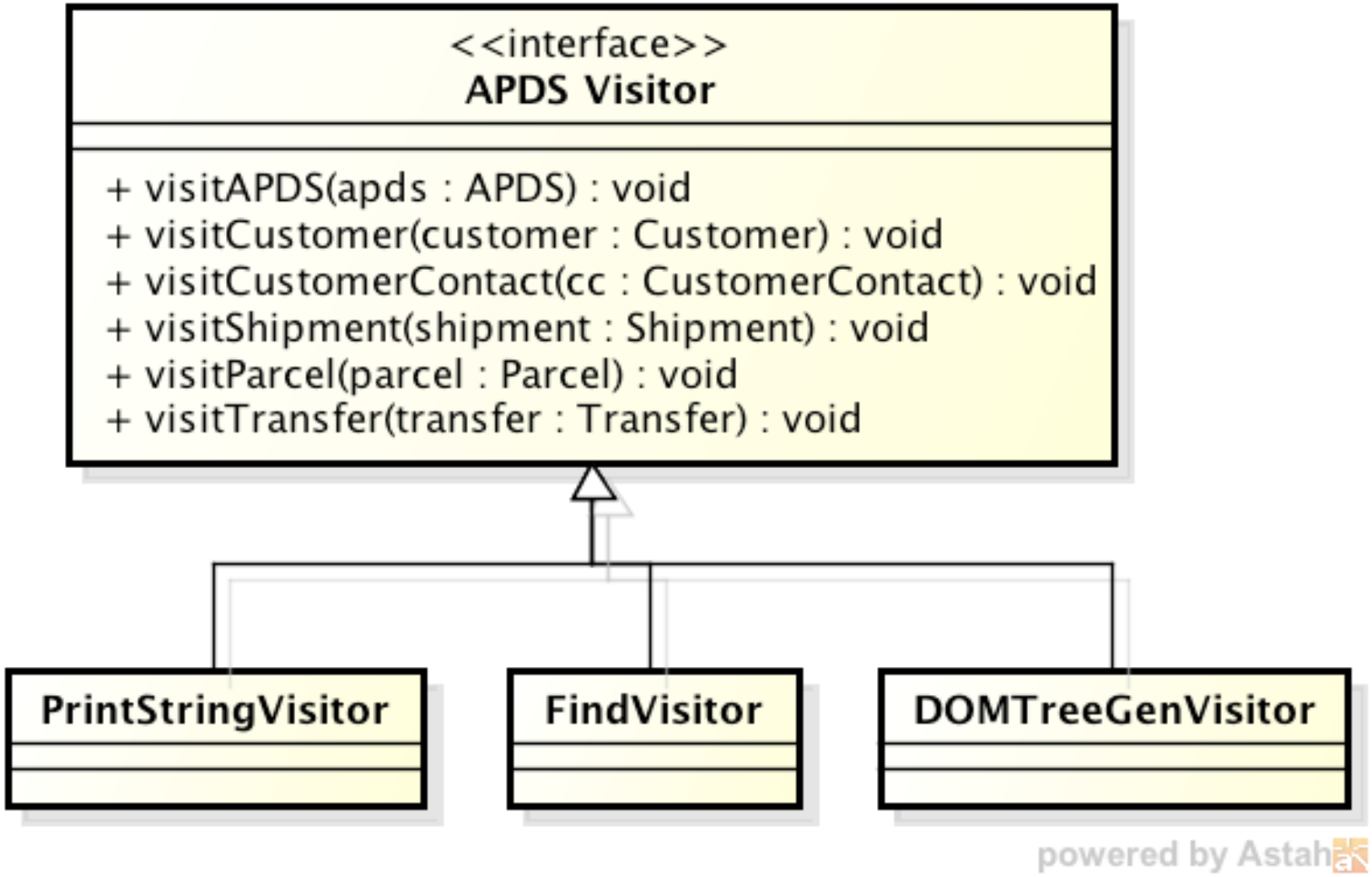
Example – A Package Delivery System

- APDS is a simple package delivery system. We might need to make multiple passes over the set of objects in the system for different purposes:
  - get a list of Customers for a mailing
  - get a list of the actions (transfers) taken on a given shipment
  - add up the weight of the parcels in a shipment
  - to generate an XML description of a shipment, or even of all active shipments (do not try this at home)

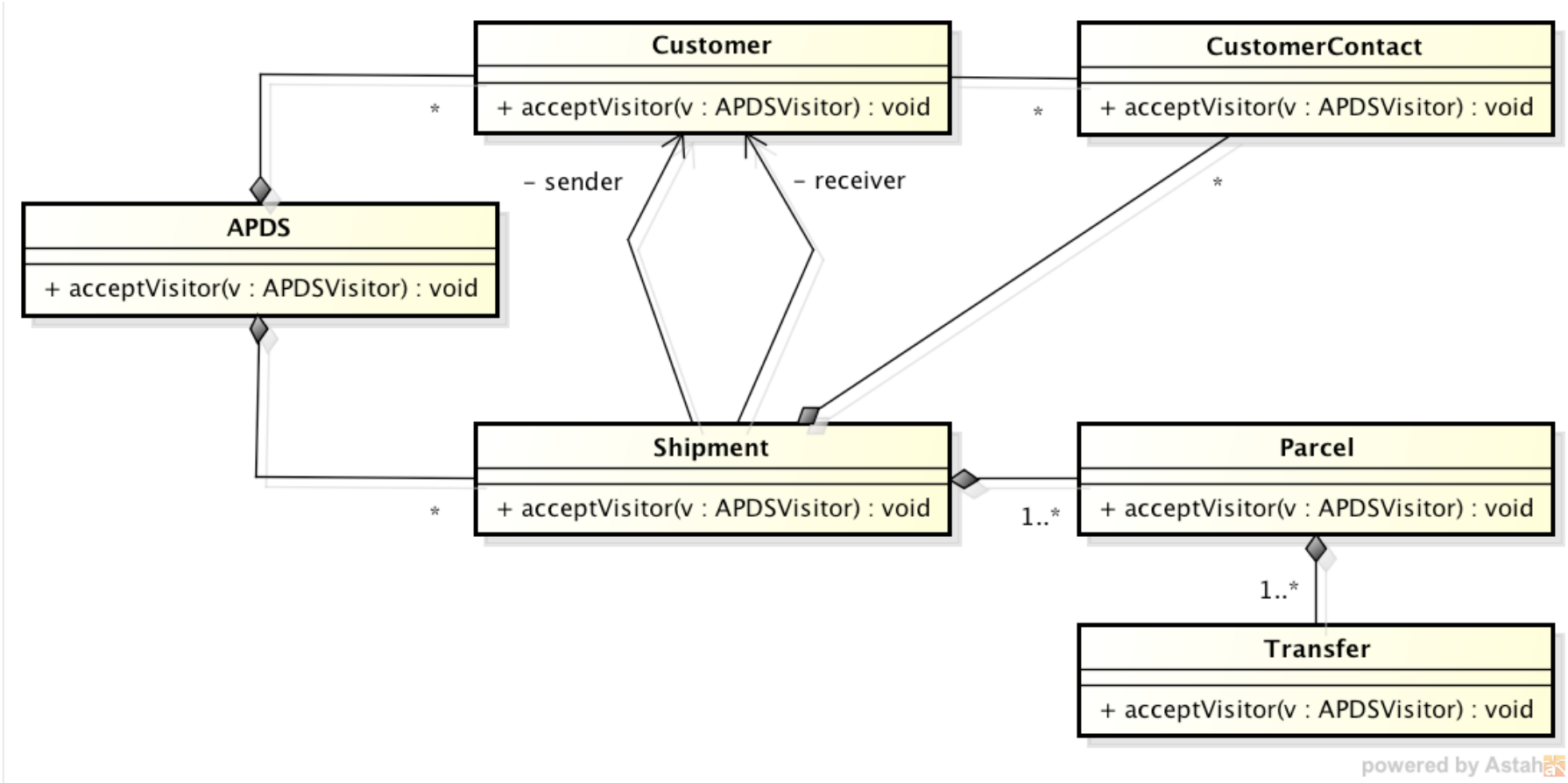
A Model for APDS



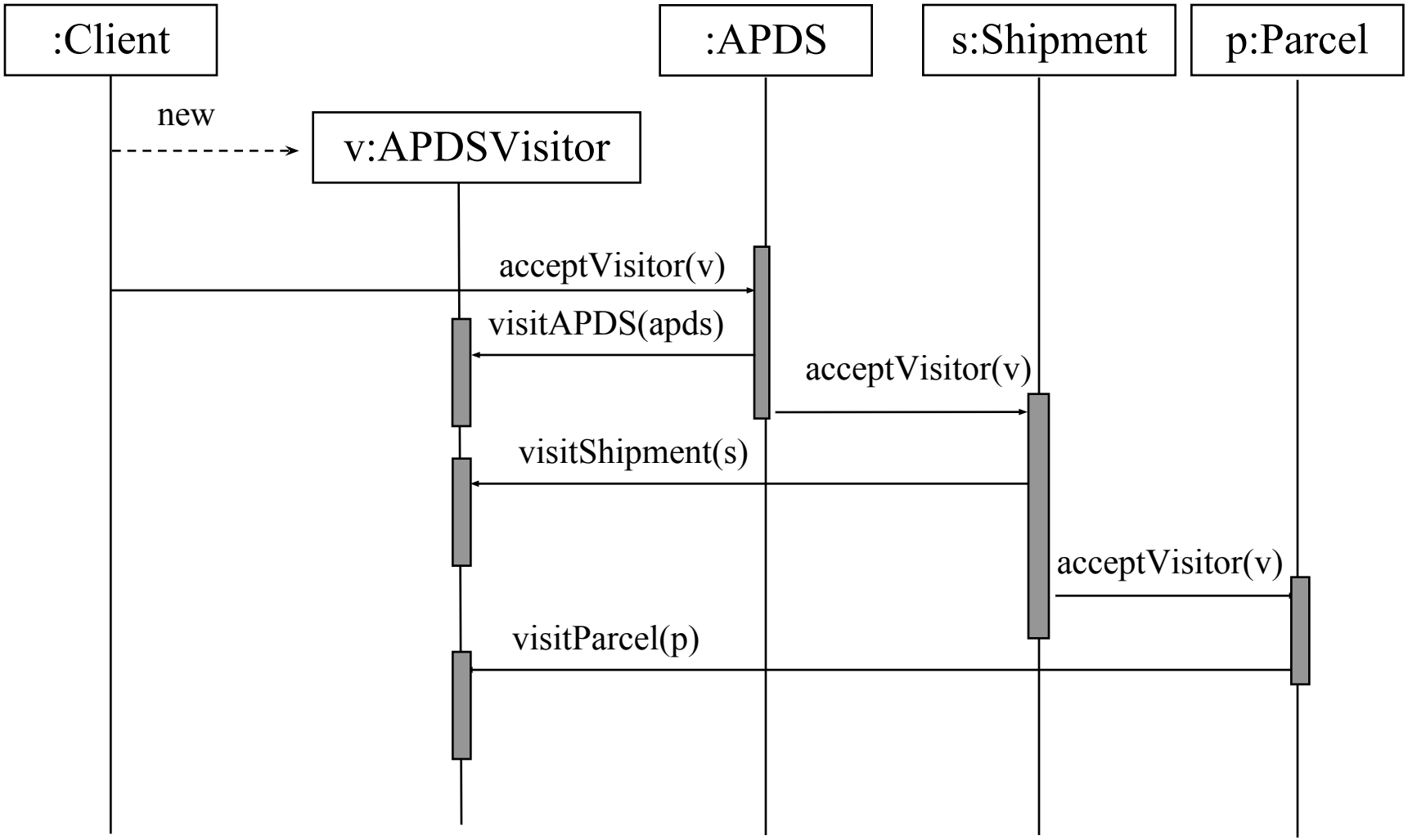
The `APDSVisitor` Classes



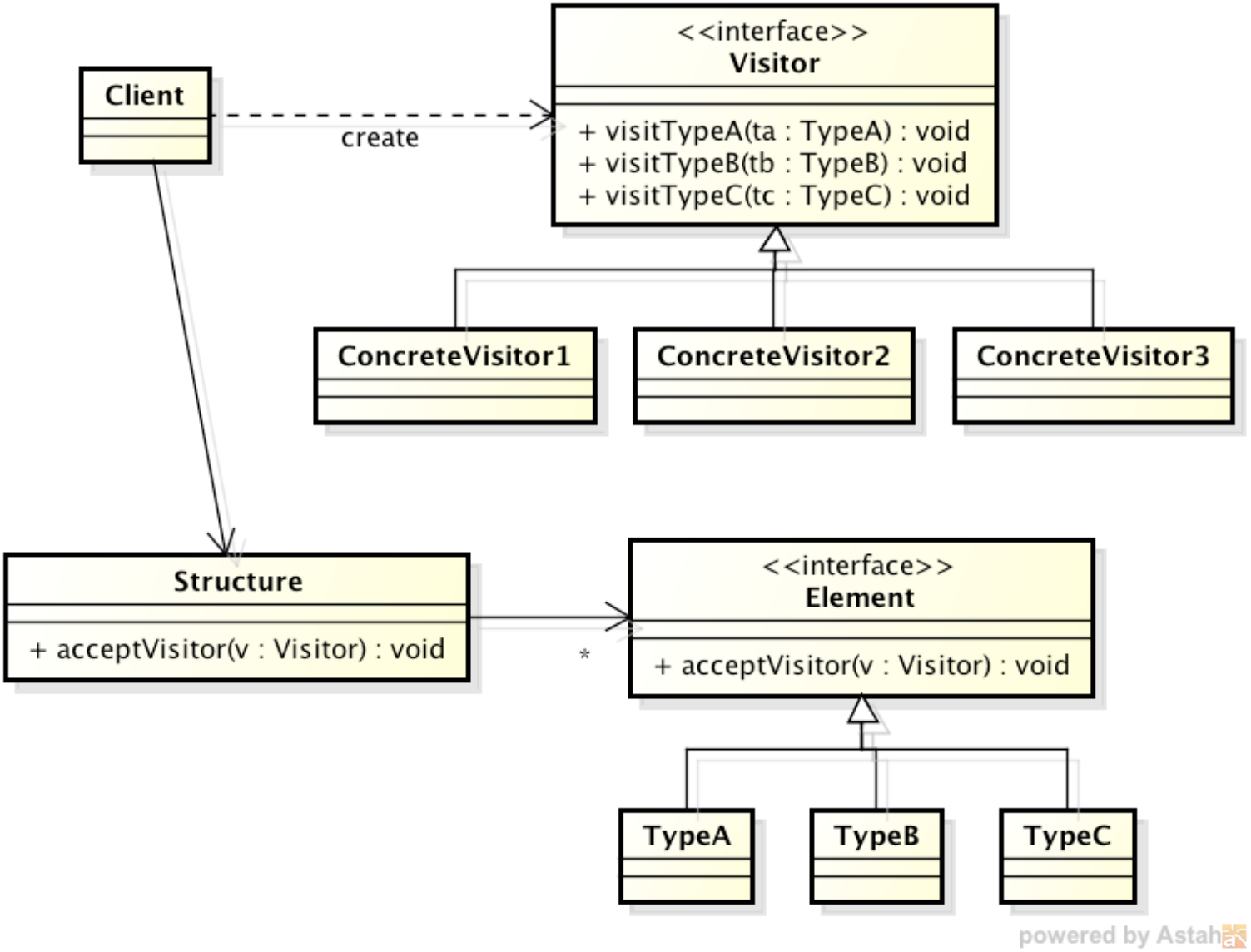
Accepting a Visitor



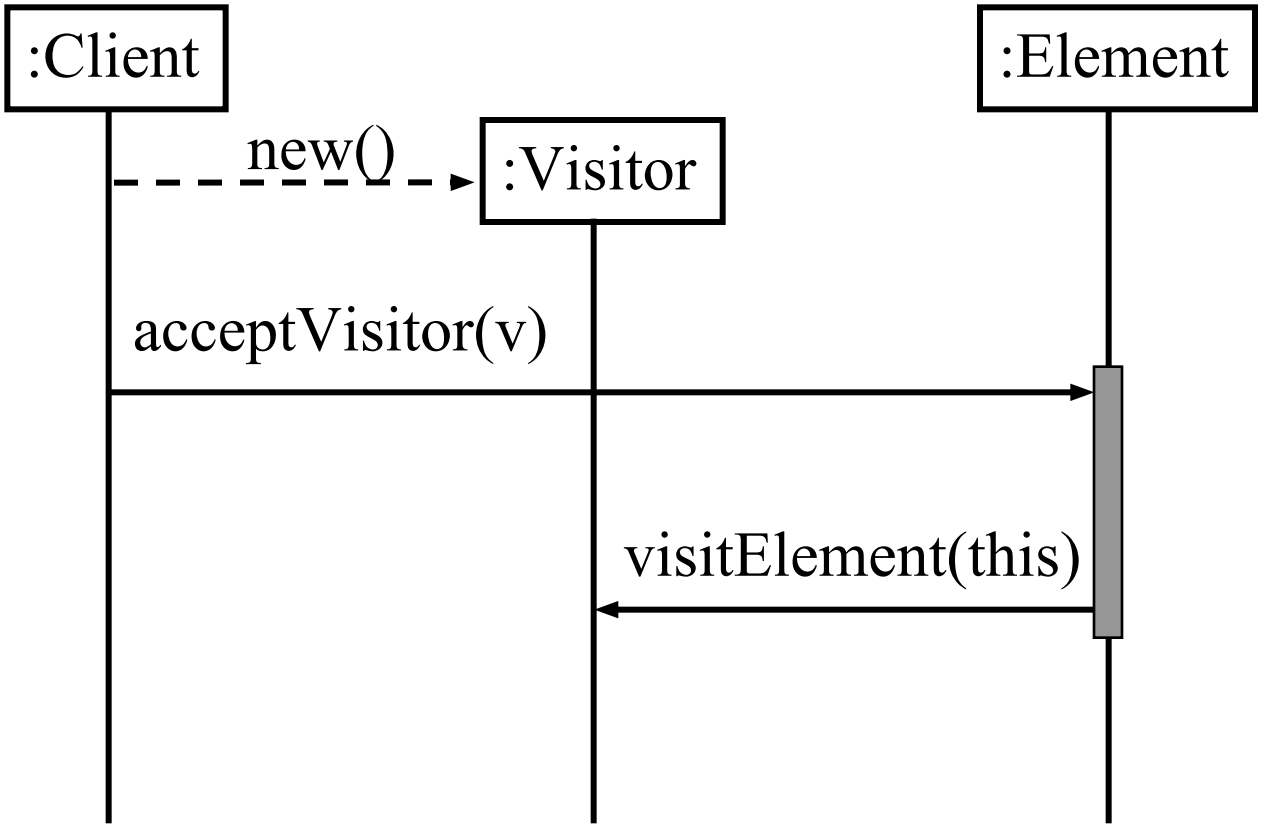
A Sequence Diagram for the APDSVisitor



The Generic Visitor Pattern



The Generic `Visitor` Interaction



Consequences of Using the Visitor Pattern

- The Visitor Pattern allows you to apply new algorithms to existing structures without having to modify the elements that make up the structure
- All the behavior related to the core algorithm and its variations as applied to the various kinds of structure elements is rolled up into one class, which is distinct from the classes that make up the structure
- It is more difficult to add new classes to the structure

Tradeoffs in Using the Visitor Pattern

- The traversal logic can go in any of three places:
  - We could use an `Iterator` and invoke the `acceptVisitor()` method on each object returned, but only if all the classes in the structure are descended from a common root
  - We could put the traversal logic in the concrete `Visitor` classes
  - We could put the traversal logic in the `acceptVisitor()` methods on the structure elements

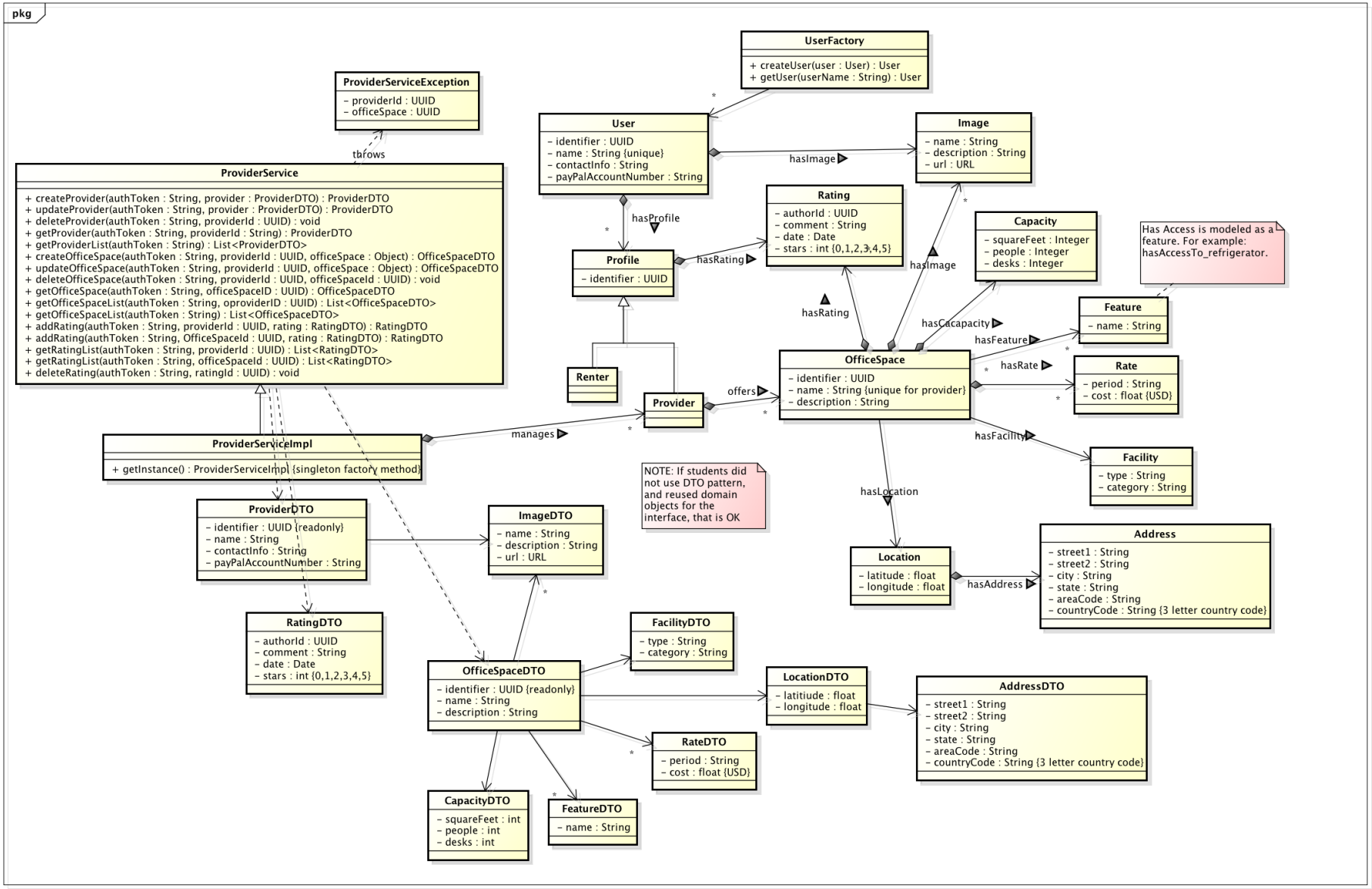
The Visitor Pattern in java.nio

- File systems are arranged hierarchically and lend themselves to algorithms that walk the tree of directories and files, for instance to calculate the disk usage or to do a recursive removal of a directory and all it contains
- This is ripe for the use of the Visitor Pattern. As of JDK 1.7, which introduced the java.nio package, Java has a specialized FileVisitor class
- You can find a good discussion of this and some examples at http://docs.oracle.com/javase/tutorial/essential/io/walk.html

Comments On Assignment 2

- Two things that are not given sufficient attention:
  - Designing your error handling strategy
  - Writing good javadocs
- More specific points on execution and code
  - Printing out a stack trace is a bad habit to get into
  - javadocs need to be more complete. For instance having a class description such as "The Transaction class represents a customer's transaction"
- Specific comments on design and documentation
  - Include a paragraph at the end of the overview that describes what's in the rest of the document.
  - Diagrams, such as class diagrams and sequence diagrams, require text to put them in context and to explain them
  - If required, use the 'loop' notation for sequence diagrams when you have to go through a list of objects

Provider Service Class Diagram



Notes on Assignment 3

- Assignment 3 is due on Thursday, October 30th
- Review the grade sheet to see what's expected
- Use the discussion forum for questions
- The problem statement for Assignment 3 is not prescriptive as to what the interfaces are and how they're implemented. This leads to many choices you have to make. Be sure to explain the choices you made based on the problem description and clarifications posted in the discussion forum

From the Grade Sheet for Assignment 3

- Your design document should provide

1. A description of the application of design patterns, e.g. Factory, Composite pattern and the Iterator pattern.
2. A class diagram showing the Collection Service classes
3. Sequence diagrams that show how the Search interface works.
4. Analysis of the error conditions that can occur.
5. A description of deviations from the assignment statement
6. The class and method level comments provide a good description of the role of each class

Context for Assignment 4

- Assignment 4 will implement the Authentication Service for use by the Provider and Renter Service.
- Discussion about how the SquareDesk services fit together.