Lecture 26: Design Patterns (part 2)

Kenneth M. Anderson
Object-Oriented Analysis and Design
CSCI 6448 - Spring Semester, 2003

Last Lecture

- Design Patterns
  - Background and Core Concepts
  - Examples
    - Singleton, Factory Method, and Adapter

Goals of Lecture

- Cover Additional Design Patterns
  - State
  - Iterator
  - Flyweight
  - Decorator
  - Observer
  - Composite

State

- Intent
  - Allow an object to alter its behavior when its internal state changes
- Motivation
  - TCPConnection example
  - A TCPConnection class must respond to an open operation differently based on its current state: established, closed, listening, etc.

State, continued

- Applicability
  - Use State when
    - an object's behavior depends on its state
    - operations have large, multipart conditional statements that depend on the object's state
- Participants
  - Context
    - defines the interface of interest to clients
    - maintains an instance of a ConcreteState subclass
  - State
    - defines an interface for encapsulating the behavior associated with a particular state of the Context
  - ConcreteState
    - each subclass of State implements a different behavior that implements the correct behavior for a particular state

State, continued

- Structure
  - Page 306 of Design Patterns
- Collaborations
  - Context delegates state-specific requests to the current ConcreteState object
  - A context may pass itself as an argument to the State object handling the request
  - Context is the primary interface of clients
  - Either Context or ConcreteState subclasses can decide which state succeeds another and under what circumstances

State, continued

- Consequences
  - State localizes state-specific behavior and partitions behavior for different states
  - State makes state transitions explicit
  - State objects can be shared
- Example
  - We saw an example of the state pattern back in Lecture 20

Iterator

- Intent
  - Provide a way to access the elements of an aggregate object (e.g. a collection class) sequentially without exposing its underlying representation
- Also Known As
  - Cursor
- Motivation
  - A collection may have multiple ways of being "traversed"; Iterator lets you keep traversal operations out of the core collection interface

Iterator, continued

- Applicability
  - Use the Iterator pattern
    - to access an aggregate object's contents without exposing its internal representation
    - to support multiple traversals of aggregate objects
    - to provide a uniform interface for traversing different aggregate structures (that is, to support polymorphic iteration)
- Participants
  - Iterator
    - defines an interface for accessing and traversing elements
  - ConcreteIterator
    - implements Iterator interface and keeps track of current position within collection
  - Aggregate
    - defines an interface for creating an Iterator (factory method)
  - ConcreteAggregate
    - implements the factory method

Iterator, continued

- Structure
  - page 259 of Design Patterns
- Collaborations
  - A ConcreteIterator keeps track of the current object in the aggregate and can compute the next object in the traversal
- Consequences
  - The Iterator pattern supports multiple traversals for each collection (e.g. inorder, preorder, postorder for trees)
  - Iterators simplify Aggregate interface
  - More than one traversal can occur on a single collection at once; as long as the traversal is read-only

Iterator, continued

- Implementation
  - The Iterator interface in the Java Collection classes
    - java.util.Iterator (interface)
    - java.util.List (interface)
    - java.util.LinkedList (class)
    - java.util.ListIterator (interface)
      - implementing subclass is private within List class

Flyweight

- Intent
  - Use sharing to support large numbers of fine-grained objects efficiently
- Motivation
  - Imagine a text editor that creates one object per character in a document
  - For large documents, that is a lot of objects!
    - but for simple text documents, there are only 26 letters, 10 digits, and a handful of punctuation marks being referenced by all of the individual character objects