Chapter 9

Object-Oriented Design

Section 9.3

Class Hierarchy Design

Questions:
- What are the dimensions of a class hierarchy?
- How can a set of classes be factored into related groups that form the class hierarchy? How can this factoring be expressed?
- Is there a process to use in forming a class hierarchy?

Class Hierarchy Dimensions

“Rationalizing” a Class Hierarchy

A class hierarchy is justified by a “rationalization” that reveals the categorization underlying the class hierarchy.

The word “rationalization” in this sense means to provide a “rational” (reasoned, principled) explanation for the hierarchy’s structure and organization.

Rationalizations aid in documenting and learning the hierarchy.

A Rationalized Hierarchy

A Process to Design a Class Hierarchy

Iteratively perform these steps:
- **Analyze**: study all of the classes with the same immediate parent class (or all classes at the beginning) looking for common and differentiating properties
- **Classify**: divide into groups - all things in the same group have the same common properties, things in different groups have a discriminating property
- **Rationalize**: create a base class for each group
Designing Complex Logic

- Complex decision-making scenarios are difficult to:
  - state precisely
  - understand clearly
  - program correctly
  - design for extensibility

Designing Complex Logic

The problems with complex logic can be overcome by using:
- a graphical, semi-formal design notation (state transition diagram, finite state automata), and
- a technique to produce an extensible object-oriented structure from the diagrams
Form of a State-Transition Diagram

Example: Step 1

Example: Step 2

Example: Step 3

Example: Step 4

Design Representations

Graphically depict the essential elements of a class and the nature of its relationships with other classes.
A Class

Representing Inheritance

Representing Aggregation

Other Relationships

Annotations

A Complete Example
Design Patterns

- Q: How does a California road engineer design a new bridge for cars/trucks?
- A: Search library of old California bridge designs for similar case.
- If we abstracted common cases of bridges, we’d have design patterns.

Elements of a Design Pattern

- name - simple, descriptive term: suspension bridge
- intent - brief description of aim: keep cars from falling into water
- motivation - concrete example: bridge can be built high so boats can go underneath
- applicability - where is it useful? span distances longer than 3/4 mile with few footings
- structure - “blueprint” diagram
- participants - each class in pattern: steel cables, footings, roadbed, ...

Example Pattern: ‘Composite’

Structural pattern
Represent part-whole relationships in tree hierarchy
Uniform treatment of individual objects
Example: graphics object:
  leaves: line, circle, …
  composites: canvas, panel, … (contain 1 or more composites or leaves)
Composite: Application

- Part
  - PartType GetType()
  - int GetIdentifier()
  - int GetTime()
  - float GetQuality()
  - int GetNumberOfRejections()
  - void SetRejectStatus(bool)
  - bool GetRejectStatus()

- CompoundPart
  - AddPart(Part)
  - int GetNumberOfParts()
  - Part Decompose()

Chain of Responsibility

- Behavioral pattern
- Pass a request down a chain of objects
  - Handle a received message
  - Handle a user input event
- Request may stop when handled, or continue down to each object in chain

Chain of Responsibility: Structure

Chain of Responsibility: Application

- MessageReceiver
  - MessageReceiver*
  - void SetMessageHandler(MessageReceiver*)
  - void Receive(Message*)

- MessageHandler
  - MessageHandler*
  - void SetSuccessor(MessageReceiver*)
  - void Handle(Message*)

- StartGameHandler
  - void Handle(Message*)

- PlayerPositionHandler
  - void Handle(Message*)
Observer

• Behavioral pattern
• When changed, Subject notifies Observers

Observer: Structure

Observer: Sequence Diagram

Observer: Application

// called when unit completes a part
For all o in observers
  o.Update(this)