Lecture 30: Domain-Driven Design, Part 5

Kenneth M. Anderson

Object-Oriented Analysis and Design

CSCI 4448/6448 - Spring Semester, 2005

Supple Design

- In chapter 10, Evans reviews a few of the techniques that he uses to make his design "supple"

- supple, adj.
  - Readily bent; pliant.
  - Moving and bending with agility; limber.
  - Yielding or changing readily; compliant or adaptable.
- Results of <http://dictionary.reference.com/search?q=supple>

Supple Design, continued

- Those techniques are
  - Intention-Revealing Interfaces
  - Side-Effect Free Functions
  - Assertions
  - Standalone Classes
  - Closure of Operations
  - Conceptual Contours
- Lets look at the first three and briefly at the next two (Conceptual Contours is left as an exercise for the reader!)
- Evans shows the relationships of these techniques on page 245

Intention-Revealing Interfaces

- The interface of a class should reveal how that class is to be used
  - If developers don't understand the interface (and have access to source), they will look at the implementation to understand the class
    - At that point, the value of encapsulation is lost
- So, name classes and methods to describe their effect and purpose, without reference to their implementation
  - These names should be drawn from the Ubiquitous Language
  - Write a test case before the methods are implemented to force your thinking into how the code is going to be used by clients
- A simple example is shown on pages 247-249

Side-Effect-Free Functions

- Operations (methods) can be broadly divided into two categories
  - commands and queries
- Commands are operations that affect the state of the system
  - Side effects are changes to the state of the system that are not obvious from the name of the operation
  - They can occur when a command calls other commands which call other commands etc.; the developer invoked one command, but ends up changing multiple aspects of the system
- Queries are “read-only” operations that obtain information from the system but do not change its state

Side-Effect-Free Functions, cont.

- Operations that return results without producing side effects are called functions
  - A function can call other functions without worrying about the depth of nesting; this makes it easier to test than operations with side effects
- To increase your use of side-effect-free functions, you can
  - separate all query operations from all command operations
    - commands should not return domain information and be kept simple
    - queries and calculations should not modify system state
  - Use Value Objects when possible to avoid having to modify domain objects; operations on value objects typically create new value objects
- See example on pages 252 to 254

Assertions

- After you have performed work on creating as many side-effect free functions and value objects as possible, you are still going to have command operations on Entity objects
- To help developers understand the effects of these commands, use
  - intention-revealing interfaces AND
  - assertions
- Assertions typically state three things
  - the pre-conditions that must be true before an operation
  - the post-conditions that will be true after the operation
  - invariants that must always be true of a particular object
    - (these are typically not associated with any particular operation)

Example

- Many languages provide an assert mechanism
  - If not, you can move assertions for particular operations to test cases
- Here's an example of pre-conditions and post-conditions
- public void removeAttribute(String name) {
  - assert (name != null);
  - _atts.remove(name);
  - assert (!_atts.keySet().contains(name))
- }

- Another example on pages 256-259

Standalone Classes

- Interdependencies make models and designs hard to understand
  - They also make them hard to test
- We should do as much as possible to minimize dependencies in our models
  - Modules and Aggregates are two techniques already discussed for doing this; They don't eliminate dependencies but tend to reduce and/or limit them in some way
- Another technique is to identify opportunities for creating stand-alone classes for domain concepts
  - Such classes do not make use of any other domain concept
  - The Pigment Color class of the Paint example in the book is one such instance; it has clearly defined responsibilities and stands alone

Closure of Operations

- In Math, certain operations are "closed" with respect to a particular type of number; For instance, addition is closed under the set of integers; add any two integers and you get an integer
- When it fits, define operations such that their return type is the same as the type of its arguments; Such operations provides a high-level interface without introducing any dependency on other concepts
- Many Value Objects work in this way
  - PigmentColor.mixedWith() is one such operation
  - java.lang.BigDecimal has examples of others;
    - public BigDecimal add(BigDecimal value)
    - public BigDecimal multiply(BigDecimal value)

Closure Of Operations, cont.

- Even “partial” closure is good
  - This refers to the situation of an operation that “almost” meets the definition, such as
    - public BigDecimal divide(BigDecimal value, int roundingMode)
    - or where the arguments are the same as the host class but the return type is different
- The book shows one such example from Smalltalk (other scripting languages have similar mechanisms) on page 270

Semester in Review

- Fundamental OO Concepts
- Responsibility-Driven Design
- Use Cases
- Design Patterns
- Refactoring
- Test-Driven Design
- Domain-Driven Design
- OO Life Cycles

- Have a good summer!