CS211 Lecture: Identifying Objects and Classes

last revised August 27, 2007

Objectives:

1. To show how to identify the major objects/classes for a problem

Materials:

1. ATM System example on the web.
2. AddressBook use cases

I. Introduction

A. Today, we are continuing to to talk about analysis, where the goal is to *understand* a problem. Actually, there are two kinds of things we need to thing about at this point.

1. Application analysis is concerned with understanding the requirements of a particular problem.

   The development of use cases, as we discussed last time, falls into this category. We seek to understand how someone will use our software.

   The specification of initial functional tests likewise falls into this category. Spelling out such tests helps us to better understand what must take place.

2. Domain analysis is concerned with understanding the particular application domain of which a specific problem is a part.

   For example, if you were developing a system involved with student registration for courses, the domain you would need to be familiar with includes concepts like students, courses, course offerings, sections, enrollments etc; as well as the relationships between them.

   To illustrate this, what is the difference between a course offering and a section?

   ASK

   Why is this difference important?

   ASK

When a student registers for a multiple section course, whether there is room depends on the specific section; each section has its own roster; and the specific professor is responsible for assigning the student's final grade.

a) In an OO approach to problem solving, we use the same concepts (objects and classes) for analyzing a domain as we will later use for developing a solution to a problem in that domain.

b) This stands in sharp contrast with the traditional structured approach, which uses quite different approaches for understanding a domain and for developing a solution to a particular problem in the domain.

B. At the heart of any problem-solving approach is the idea of <u>decomposition</u> - breaking a large problem up into smaller pieces.

1. An old joke: "How do you eat an elephant?"

   ASK

   One bite at a time

2. Problems of any significant size require the effort of more than one person - in fact, major software projects may involve thousands. One wants to decompose a problem into <u>modules</u> that are as independent as possible, so that different people can work on them without getting in each other's way. The technical term for this is that we want to <u>minimize coupling</u> - i.e. the degree to which one module depends on another.

   Example: Some books are collections of articles by different authors - that is, the books are decomposed into chapters, with each having its own author. This works reasonably well.

   Suppose, instead, that the book were decomposed by pages, with one author responsible for page 1, 5, 9, 13..., a second responsible for page 2, 6, 10, 14 ... etc. Obviously, this would result in chaos!

3. While any approach to solving a large problem necessarily involves decomposition, an object-oriented approach to software development diverges dramatically from the structured approach in terms of how it approaches this.

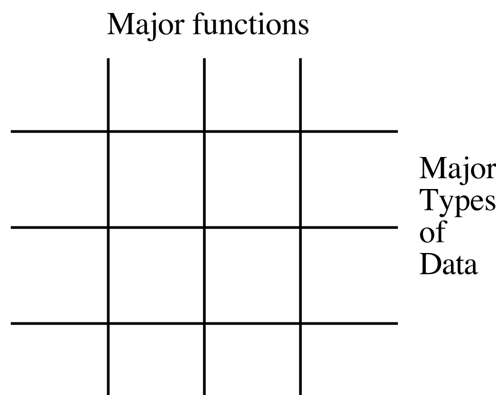
a) Any software system can be viewed in two ways - one can focus on the data that is being manipulated, or the functionality that manipulates the data.

   Example: consider software used for student registration. The data being manipulated includes information about individual students, information about individual courses, and information about enrollment in courses - what students are in what course. The functionality includes things like enrolling a student in a course, dropping a student from a course, printing student schedules, printing course lists, etc.

b) The structured approach decomposes the system according to its functionality - e.g. major pieces in the decomposition would be “enroll student”, “drop student”, “print schedule”, “print course list”, etc.

c) The object oriented approach decomposes the system according to its data - e.g. the major pieces in the decomposition would be “student” and “course”, related by an “enrolled in” relationship.

4. Picture the system as being like a piece of woven cloth, with threads running both horizontally and vertically in it. The structured approach focuses on the threads running one way; the object-oriented approach focuses on the threads running the other way.



5. Several of the Quick-Check Questions from the book got at thie issue of what advantage does the OO approach have over the structured approach - i.e. why is it better to base the design by decomposing in terms of data, rather than functionality?

a) “List three problems associated with the structured approach”

ASK

(1) The key problem has to do with modularity.

In a functional approach, the different pieces of the decomposition are highly dependent on one another - e.g. in a student registration system the authors of the “enroll”, “drop”, “student schedule” and “course list” modules must agree among themselves as to how they are going to represent an enrollment, since “drop” must undo what “enroll” has done and “student schedule” and “course list” depend on what it has done.

(2) Maintenance - due to the high coupling between modules, a change made to one can have a ripple effect on others

(3) Testing - it is not possible to test the modules in isolation from one another - e.g. how does one test “student schedule” without “enroll” first working? How does one test “enroll” without being able to see the results via “student schedule” or “course list”?

(4) Software re-use - it is hard to re-use an individual module in another system, because the modules are so dependent on one another.

b) “List four qualities that are desirable in a software construct.”

ASK

(1) Autonomous - not highly dependent on other constructs

(2) Cohesive - with a single, well-defined purpose

(3) Easy to understand

(4) Easily adapted to accomodate changing requirements

c) “What is meant by the term seamless development?”

ASK

In traditional software development methodologies, the line between analysis and design is fairly sharp, because very distinct notations are used for documentation at each stage. [Analysis considers the structure of the data, but design is based on the functionality]. In OO methodologies, the line is less distinct, because similar notations can be used at both stages. The seamlessness of OO tools at the transitions between the various phases of the software lifecycle is a strength of the OO paradigm.

II. Class Identification Based on Domain Analysis

A. It is often possible to develop a model of the general domain of which a particular problem is a part in terms of objects and classes. The objects and classes thus identifier will necessarily be part of any system that solves problems in that domain.

B. It is important to consider not only the individual objects, but also how the objects relate to one another.

1. Quick check question l: "List three types of relationships between classes. Briefly describe each"

   ASK

   a) Association: objects of the two classes have some sort of relationship and can communicate with one another.

   b) Aggregation: a stronger relationship in which there is an "ownership" or whole-part relationship between the objects (as opposed to association where the two objects can be thought of as peers).

      In describing an aggregation, one will typically use phrases like "has a" or "is a part of"

      (Note: there is a strong form of aggregation called composition or containment which we will discuss below)

   c) Inheritance/generalization: a relationship between CLASSES, **not** INDIVIDUAL OBJECTS. In describing generalization, one will typically use phrases like "is a".

2. We will discuss these concepts in more depth in a later lecture.

   a) For now, it is vital to note the difference between association/aggregation on the one hand, and generalization on the other. The key distinction is rooted in a concept known as *the law*

of substitution: we can legitimately say that class A is a generalization of class B if and only if wherever an A is required, A B can be used.

b) While there is a very sharp distinction between generalization, on the one hand, and assocation or aggregation on the other, the distinction between association and aggregation is not always as clear; sometimes, a reasonable case can be made for either.

c) Examples of relationships: (ASK for each, discuss reasons)

Person, Student: Generalization (satisfies law of substitution - it is meaningful to say "a Student is a Person")

Course Roster, Student Aggregation (it is meaningful to say "a Course Roster is made of Students")

Course, Student Probably simple Association - maybe Aggregation.

Book, Chapter Aggregation - it is meaningful to say "A chapter is part of a book"

3. Quick Check question q: "What is the difference between aggregation and composition?"

   ASK

   The essence is that the relationship is exclusive: the part belongs to exactly one whole, and cannot exist apart from the whole, and the parts live and die with the whole

   Example: Course Roster, Student is an aggregation but cannot be regarded as a composition, because students are, in general, enrolled in multiple courses, and a student can exist even if not enrolled in any courses.

   Example: Book, Chapter: Composition is reasonable in this case - unless one wants to allow a Chapter to have a separate existence (as might be the case with certain kinds of reprinting)

C. At this point, we are interested in identifying classes which are part of the <u>problem</u> domain. Later, we will extend this to include classes that are part of the <u>solution</u> domain for a specific problem.

D. EXAMPLE: Let's develop an OO model of the domain underlying the "Wheels" system in the book.

1. What are the key concepts?

   ASK

   a) An individual bicycle

   b) A specialist bicycle (e.g. racer, mountain bike ...)

   c) A customer

2. How are these concepts related to one another?

   ASK

   a) A specialist bicycle is a kind of bicycle - Generalization - it is meaningful to say “a specialist bicycle is a bicycle”

   b) A given bicycle can be hired by a given customer.

      Association - it is not meaningful to say “a bicycle is a customer or vice versa; it is not meaningful to say “a bicycle is a part of a customer or vice versa”.

      In the case of association, one can also consider <u>multiplicity</u>

      (1) Any given bicycle can only be hired by one customer at a time.

      (2) But a given customer can hire multiple bikes at a given time (e.g. a family)

         Thus, this association is 1 : many from customer to bicycle (more on this later)

   c) A customer can have reservations for one or more bicycles at some time in the future.

      Probably association

      Note that a bicycle can be reserved for multiple customers (at different times), so the association is many : many.

E. Another example: the domain underlying the ATM example system

1. What are the key concepts?

   ASK

2. How are these concepts related to one another?

   ASK

III. Class Identification Based on Noun Extraction

Another approach to identifying classes that is sometimes simplistic, but yet is often useful, is called noun extraction. The basic idea is this: read over the system requirements/use case flows of events, and note the nouns that appear.

A. Some of the nouns that appear - especially the ones that appear frequently - will turn out to refer to objects that need to be represented by classes in the final system.

B. Other nouns will turn out to be other things., including:

1. Attributes of objects, rather than objects in their own right. An important skill to develop is being able to distinguish the two. Recall that objects have three essential characteristics:

   ASK

   a) Identity.

   b) State (often complex - i.e. involving more than a simple value).

   c) Behavior

      Examples: ASK for Wheels examples of attributes that are not objects in their own right, and why

      things like customer name, bicycle rental rate, date due, etc. are attributes

2. Actors or other objects that are outside the scope of the system.

   Example: ASK for Wheels examples

   Receptionist

   Note that since we don't have to build models of these, they do not need to be represented by classes inside the system.

C. Finally, sometimes there may be a generalization-specialization relationship between nouns - implying an inheritance relationship between the corresponding classes.

D. Apply noun-extraction to use cases for AddressBook system.

IV. Class Identification is not Once-for-All

It is important to recognize that identifying classes is not something we do once and then never change. As the design process proceeds, we should be prepared to:

A. Add additional classes that we discover the need for

B. Reconfigure classes identified previously as we develop a clearer sense of what their responsibilities will be.