CPS122 Lecture: Identifying Objects and Classes

last revised January 30, 2022

Objectives:

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

Materials:

1. Quick-Check questions and answers for ch. 4
2. Warp and Woof Diagram Projectable
3. Textbook Exercises 4.1, 4.2, 4.3
4. ATM System example on the web.
5. AddressBook use cases for activity

I. Introduction

A. Today, we are continuing to to talk about analysis, where the goal is to *understand* a problem. We are not yet focusing on *how* we will solve the problem - it is still our goal to understand *what* it is we are trying to do.

B. 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.

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

   b) 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 - though the actual form will not be identical.

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.

C. 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 minimize coupling - i.e. the degree to which one module depends on another.

a) 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.

b) 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!

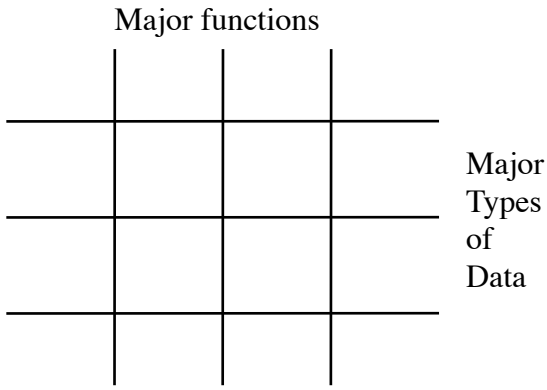
3. While any approach to solving a large problem necessarily involves decomposition, an object-oriented approach to software development diverges dramatically from the structured approach in terms of how it approaches this.

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, producing student schedules, producing course lists, etc.

b) As today's chapter in the book discussed, the older 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. 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.

This can be depicted as follows - it's like the "warp and woof" of cloth:



PROJECT

c) Thus, a key part of solving any problem is identifying the classes that naturally model its domain.

4. In object-oriented decomposition, we identify the objects that are implied by the problem - but then we actually create the classes that those objects belong to - and then the program creates the objects as it runs.

D. Recall from the reading that it is important to consider not only the individual objects, but also how the objects relate to one another.

What are the three ways discussed in the book?

ASK

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

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

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

b) There is a stronger form of aggregation known as composition or containment. 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.

Examples: We might say a CourseRoster is an aggregation of Students - we can meaningfully say that a CourseRoster has Students or that a Student is part of a CourseRoster; but we cannot say that the relationship is exclusive, since a Student is generally part of multiple CourseRosters, and Students don't cease to exist when a course is over!

However, we would say at a Book is composed of Chapters - assuming that a Chapter is only part of one Book, comes into existence when the Book is written, and would cease to exist if the Book ceased to exist.

c) It turns out that the way we represent aggregations/compositions and associations is so similar, that it is best to actually consider aggregation to be a form of association

3. Inheritance/generalization: a relationship between CLASSES, **not** INDIVIDUAL OBJECTS.

a) We will say a lot more about this later.

b) This is the relationship implied by phrases like “is a”.

Examples: the relationship between classes like "Student" and "Person" - but not the relationship between classes like "Person" and "Arm". (The latter would be aggregation).

c) However, we don't use this for the relationship between an object and the class it belongs to, even though we might use the phrase "is a" - e.g. though the object "Joe" might belong to the class "Student", we don't say the relationship is inheritance - rather, it is class membership, because "Joe" is an individual object, not a class.

4. Some things to note about these.

   a) It is vital to note the difference between association/aggregation on the one hand, and inheritance/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 association 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) The distinction between simple aggregation and composition is usually clear because of the very strong, exclusive nature of the latter.

5. Do the following as think-pair-share

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

   b) Athlete: Team: Aggregation (it is meaningful to say "a Team is made of Athletes", but it is not reasonable to say an Athlete is a Team, or that an Athlete can be used anywhere a Team is needed, nor is it reasonable to say that an Athlete can only belong to one and only one Team and ceases to exist when the season is over!)

   c) Husband, Wife: Association (it is not reasonable to say a Husband is a Wife, or that a Husband is a part of a Wife - and a Husband doesn't cease to exist if his Wife dies).

   d) Wing, Airplane: Composition - it is meaningful to say "A wing is part of an airplane", and the relationship is an exclusive one - we don't think of a wing being part of more than one airplane, or of having an existence independent of the airplane it is part of.

E. Two key ideas we will keep coming back to:

1. The notion of responsibility driven design. Every object fulfills a certain responsibility.

   Example: In the address-book system, a Person object represents the information of a single person, and the AddressBook object represents an aggregation of Persons.

2. The notion of single task object - sometimes called the STO principle. If one kind of object has two distinct tasks to perform, it probably should be two different kinds of object.

   Example: in the address-book system, it would not be appropriate to task a single kind of object with both representing an persons and an aggregation of person.s

II. **Go Over Quick Check Questions**

III. **Exercises to do in groups of 4**

A. Exercise 4.1

B. Exercise 4.5

IV. **Object 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 identified will necessarily be part of any system that solves problems in that domain.

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

C. Exercises in Groups of 4

1. Exercise 4.3

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

   a) What are the key concepts?

   ASK

   (1)An individual bicycle

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

   (3)A customer

   b) How are these concepts related to one another?

   ASK

   (1)A specialist bicycle is a kind of bicycle - Generalization - it is meaningful to say "a specialist bicycle is a bicycle"

   (2)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 multiplicity

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

(b) 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)

(3) 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.

V. **Object 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. Activity in Groups of 4: Apply noun-extraction to use cases for AddressBook system.

VI. Class Identification is not Once-for-All

It is important to recognize that identifying objects and the classes they imply 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.

C. The classes identified during analysis will often carry forward into the implementation of the system - with others added as well to support the implementation per se. (A 5:1 expansion is not uncommon.)

1. The classes discovered by analyzing the problem domain are called analysis classes.

2. The classes used to actually implement the system are called design classes or implementation classes.

3. One of the advantages of the "seamless development" aspect of OO is that the analysis classes generally form an important subset of the classes actually used to implement the system.