CS211 Lecture: Review of Fundamental Concepts and Java

last revised August 11, 2006

Objectives:
1. To review key OO concepts discussed last semester, and their implementation in Java.

Materials:
1. Fundamental Concepts Exercises distributed to students to work on at previous class
2. Solutions to the above.
3. Projectable of Booch p. 77
4. Polymorphism demo programs in java and C++ (static and dynamic binding versions) - code to show and to run

I. Introduction

A. Last semester, you were introduced to the key concepts of object orientation (OO) through learning the Java programming language.

B. The goal of this session is to review those concepts and their implementation in Java, by reviewing some key vocabulary terms. (One mark of a person who is well educated in a field of learning is precise use of the vocabulary of the field.).

II. Basic Vocabulary

A. In order to understand object-oriented methods, it is necessary to clearly understand some fundamental terms.

1. Unfortunately, in the world of OO some terms are fairly universal while in other cases different programming language communities use different terms for the same thing. Thus, in some cases we will use more than one term for the same basic idea.

2. Last class you were given a handout of exercises to work through. Let’s walk through them.

WALK THROUGH FIRST PART OF EXERCISES HANDOUT (FIRST MATCHING SECTION)
B. There are a few key ideas related to these terms we want to review

1. Object
   
a) What is an object?  ASK CLASS
   
   An object is a something we create to model a portion of the "world" a given software system must deal with.

b) An object has three critical properties:
   
   ASK CLASS
   
   (1) Stored data, or state
   
   (2) Behaviors, or operations
   
   (3) Identity

   PROJECT - Booch P. 77


   In the OO world, the terms message and method are used when describing operations performed by an object.

   What is a message?  What is a method?  How are they related and how do they differ?  ASK

   a) In an object-oriented system, objects interact with one another by means of messages.  For example, when a customer withdraws money at an ATM, the ATM object involved may send a message to the appropriate account object requesting the withdrawal of funds.

   b) Each class defines methods for dealing with various messages.  Thus, the bank account object will have a withdraw method that deals with the withdraw message.  (Note that the name of the message and the method are the same).

   c) The reason for the distinction is that different objects may have different methods for handling a given message, and thus may respond to the same message in different ways.  Thus, for example, the withdraw method for a simple bank account object might signal an error if the account receives a withdraw message whose amount might overdraft the account.  But a bank account that has overdraft protection might use a different method to respond to the same message - one that automatically transfers money from somewhere else to cover the overdraft.
d) Note, then, that one object requests another object to do something by sending it a **message**. The message consists of a message **name** and (possibly) a list of parameters. The object responds to the message by using its own **method** for doing so, and may return a result back to the sender of the original message.

3. Class

   a) What is a class?  **ASK CLASS**

      A class is a “blueprint” for building individual objects (called its **instances**). The class to which an object belongs determines its properties - e.g. what operations it can perform (behaviors it can exhibit) and what information is remembered as part of its state.

   b) Typically, a class is specified in terms of two things:  **ASK CLASS**

      (1) An **interface** (what clients of objects that belong to the class are allowed to know and rely upon)

      (2) An **implementation** (which provides the services specified in the interface in a way that a client of the class need not know.)

   c) What is the convention typically used in OO Software for class names?

      **ASK**

      Each class is given a name (typically beginning with an Uppercase letter, and then using mixed case). In Java, ordinarily, each class resides in a separate source file whose name is the same as the name of the class, followed by .java.

      In contrast, objects and attributes of objects have names beginning with a lowercase letter.

4. Classes can be either abstract or concrete

   a) What is the difference?  **ASK CLASS**

      A concrete class is one to which objects can actually belong. An abstract class serves to generalize the properties of some number of concrete classes - but no object will actually belong to it without belonging to one of its concrete subclasses.
b) WALK THROUGH SECOND PART OF EXERCISES HANDOUT (OPEN-ENDED QUESTIONS.)

c) EXERCISE E6 - part a - p. 36

5. Encapsulation:

A class serves to "encapsulate" the properties of related objects.

a) What do we mean by encapsulation? ASK

Typically, the state of a particular object is not directly accessible - it cannot be examined or altered by just anyone. Rather, the state is only accessible through the operations which the class provides for that purpose.

b) In Java, visibility modifiers are used to control what parts of a class are considered to be part of its interface and what are considered to be part of its implementation.

c) WALK THROUGH THIRD PART OF EXERCISES HANDOUT (VISIBILITY MODIFIERS QUESTION)

d) EXERCISE E7 - part b - p. 37

e) Java also has package (default) visibility, which is used when there is a need for classes to closely cooperate with one another either by virtue of being part of some subsystem. (package).

6. Polymorphism.

What does the term “polymorphism” mean? ASK

The essence of polymorphism is that OO systems provide mechanisms that allow different objects to do the same thing in different ways - and the knowledge of how to handle this resides with the object.

a) That is, different classes may have different methods for responding to the same message.

b) In Java, when a message is sent to an object, the method that handles it is the appropriate version for that particular object - e.g. suppose we have the following: [ PROJECT CODE ]
class Foo
{
    public void speak()
    { System.out.println("Huh"); } 
}

class Bar extends Foo 
{
    public void speak()
    { System.out.println("Hello"); } 
}

class Baz extends Foo 
{
    public void speak()
    { System.out.println("Goodbye"); } 
}

together with the following main method in some main class:

public static void main(String [] args)
{
    Foo f1 = new Bar();
    Foo f2 = new Baz();
    f1.speak();
    f2.speak();
}

What will the output be?

ASK

DEMO - java Polymorphism

Hello
Goodbye

c) The reason why this works as it does is because Java uses dynamic binding. When the speak() message is sent to each object, the correct version of the speak() method for that object is called, based on the actual type of the object.
Note that f1 is declared to be of class Foo, but is actually of one of its subclasses, Bar - so the speak() method of Bar is used, printing "Hello".

In similar fashion, f1 is declared to be of class Foo, but is actually of one of its subclasses, Baz - so the speak() method of Baz is used, printing "Goodbye".

d) It is worth being aware of the fact that not all OO programming languages use dynamic binding by default. For example, consider the C++ equivalent to the above program (where the only changes are to replace Java specific code with C++ equivalents)

*SHOW:* Polymorphism.cc

**DEMO IT**

Explanation for output: C++ uses static binding by default, so the methods appropriate to the declared types of f1 and f2 are used - in both cases, the version of speak() in class Foo.

e) In C++, it is possible to get dynamic binding instead:

*SHOW:* PolymorphismDynamic.cc

**ASK:** What's different?

virtual before declaration of speak() in class Foo

**DEMO IT**

C. One interesting question we will deal with in OO design is the matter of thinking about how the various classes comprising a system are related to one another. As we shall see, many kinds of relationships are possible - but one of the most important is the relationships that give rise to a class hierarchy involving inheritance.

1. GO OVER FINAL GROUP OF EXERCISES ON HANDOUT
2. EXERCISE E10 (p. 43)
3. EXERCISE E12 - part a (p. 43)
4. (BONUS) EXERCISE E19 (p. 53)