Class 04: Talking About Objects: Using UML

Held: Thursday, February 4, 2010

Summary: We begin thinking more deeply about object-oriented perspectives, focusing on the UML, a notation for representing object-oriented design.

Related Pages:

- EBoard.

Notes:

- Beautiful Code reading for Tuesday: Chapter 4.
- Other reading for Tuesday: Chapter one of Head First OOAD.
- I’m going to try a mostly-lecture format for our exploration of today’s beautiful code. Let me know what you think.
- Go to the summer research session today!
- EC: Friday’s CS Table: Pair Programming.

Overview:

- Python Followup.
- Beautiful Code: Multidimensional Arrays in NumPy.
- Thinking about object-oriented programming.
- UML: Classes and interfaces.
- UML: Processes and computation.
- UML: Q&A.

Python Continued: Binary Search

Beautiful Code: Multidimensional Arrays in NumPy

Preliminaries

- Background:
  - NumPy is a popular numerical package for Python
  - NumPy supports multi-dimensional arrays, which are useful
- The problem: We’re going to write a lot of code that does something with each element in a multi-dimensional array
  - Some of it will be written in C
  - Some of it will be written in Python
How can we make it efficient and elegant

Primary areas of chapter:
- Internal representation of multi-dimensional arrays in NumPy
- Ways to step through the elements of those arrays

Theses:
- The internal representation is cool
- Thinking at the Python level can help you program at the C level
- Modeling with iterators helps you think about problem solving
- Efficiency matters

Background: Storing Multidimensional Arrays

- For the basic multidimensional arrays (that is, ones that are not sliced), NumPy
  - Uses a contiguous chunk of memory
  - Lays the array out in what seems to be row-major order.
- For example, a 4x3x2 array would have the elements arranged as follows:
  - \([0,0,0]; [0,0,1]; [0,1,0]; [0,1,1]; [0,2,0]; [0,2,1]; [1,0,0]; [1,0,1], [1,1,0], [1,1,1]; ...\)
- This layout makes it fairly easy to determine where any element is.
  - For a one dimensional array of size A, element a is at offset a.
  - For a two dimensional array of size AxB, element \([a,b]\) is at offset a*B+b
  - For a three dimensional array of size AxBxC, element \([a,b,c]\) is at offset \((a*B+b)*C+c\) or \((a*B*C + b*C + c)\)
  - And so on and so forth

Background: Slicing Multidimensional Arrays

- Two basic strategies:
  - When you slice, make a copy
  - When you slice, build a structure that has a pointer to the original array and info about the slice.
- The sharing strategy saves space and time
- The sharing strategy potentially creates problems with aliasing
  - But some people like the aliasing
- The sharing strategy complicates the code you use to access elements (but not too much)
  - In essence, you need to take starting position and stride into account in the formula.

Back to the problem: Iterating arrays

- For a non-sliced array, it’s easy: You just repeatedly increment the pointer until you hit the end.
- For a sliced array, it’s a bit harder: You need to check where you are and do different things in different cases.
  - Straightforward but error prone
- You can also use recursion, but some people worry about the overhead of recursive calls (and the danger of overflowing the stack).
- So, what’s the big issue here?
  - Each time you need to iterate an array, you need to copy a big chunk of code.
Making it beautiful: Iterators

- An idea: Why not encapsulate the state of the iteration?
- That encapsulated state is an iterator.
- Iterators provide some simple operations, such as `next` and `get` and `end`.
- An object-oriented approach: We’ve turned a control structure into an object

A functional perspective

- We could also have turned the big chunk of code to copy into a higher-order function.
- But that tends to be less efficient.
- It would, however, allow us to avoid the overhead the author complains about for non-sliced arrays.
  - Hmmm ... should I show you code.

Questions and answers

- See the notes on the reading.

Thinking About Object-Oriented Programming

- What, in your mind, are the distinguishing features of object-oriented programming?
- Why are these features important?

The UML

- The UML == The Unified Modeling Language
- A visual notation for representing key issues in the design of an object-oriented program.
- Observation: We like to communicate things visually. But for a visual notation to work well, we have to agree upon what the symbols mean.
- Can be used prospectively or retrospectively
- Three traditional ways to think about the UML
  - We use the UML to sketch out classes and their relationships while thinking about design.
  - We use the UML to blueprint most of the details of a system.
  - We use the UML as a programming language to implement a system.
- Developed collaboratively by many of the big names in OOP in the early ages of the adoption of OOA&D by industry.
- We’ll focus mostly on it as a tool for sketching.
- That is, we’ll use the UML to communicate important things about our systems to each other, but we won’t try to dot every i and cross every t.
- Disclaimer: I don’t use the UML nearly enough in my own work.
The UML: Class Notation

What do we care about for classes? Here are some possibilities

- The basic components of the class:
  - The fields
  - The methods
- The role of the class in the inheritance hierarchy
- The other classes that the class uses (implicitly)

We will need notations for each of these things.

We may find that we need other notations, too.

The UML: Processes

Another common issue: What is the sequence of event calls as a particular task is accomplished.

Questions

Is the UML the only commonly used modeling language?

*Yes, it’s the most commonly used graphical modeling language for certain aspects of OO design, particularly for class relationships. That’s it’s intent. CRC Cards are popular for modeling activities, but that’s a separate issue.*

What is the interaction between modelling and the methodology used in the construction of software? There are an impressive variety of methodologies for object-oriented design alone, with no clear "best" option. Is the choice of a modeling language as subjective?

*Not really. The UML was designed to work for most methodologies.*

According to Wikipedia, UML standards have been criticized as "unintelligible geekspeak" (http://en.wikipedia.org/wiki/Unified_Modeling_Language#Criticisms). How should we as programmers balance the need for readability with the time-saving benefits of shorthand and certain code conventions and hacks? Is there a benefit in having some languages be relatively "unreadable" and other languages be extremely "readable", such that programmers have a wide range of language types to choose from?

*As you learned in Tutorial, it’s important to know who your audience is. Some people find Scheme unreadable. The UML standards are not written for a general programmer audience; these standards are intended to provide precise definitions of the different components of the language. You’ll find that any modern language standard is similarly dense.*

What is the purpose of having a machine-readable format for system structure? The implementation won’t be done by computers, so why is it necessary for it to be in a machine-readable form?
A lot of the implementation is straightforward enough that it can be done by computers. In fact, there are those who promote using the UML as a programming language.

Plus, it’s a lot easier to edit stuff in machine-readable form.

Can you explain how profiles and stereotypes work for UML extension?

No.

I viewed the members list at http://www.uml.org/. My question is how can we trust a standard developed by a group with members of companies voting on its board to produce an unbiased modeling language?

Trust? If there are enough companies involved, they will generally build a language that provides no one member with a competitive advantage.

The UML is a very highly regarded tool to programmers, but what are some of its drawbacks?

There’s a lot to the UML. If you’re someone who believes that you need to know the whole language to use it, there’s a lot to grasp. And, as is often the case in large systems, different people may learn different subsets, leading to it being less of a lingua franca.

Copyright © 2010 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/2.5/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.