

Python Programming

Week Five, Class Two
OOP in python, cont.

Outline

- Review
- Patterns in software
 - Where they come from
 - What they are
 - How they work, in practice
- Examples

Inheritance

- True or False:
 - If class A has method `foo`, then if class B inherits from class A, it also has a method called `foo`?
- How do you use the `class` keyword use to indicate that class C inherits from two other classes A and B ?
- Multiple inheritance means _____.
- What issue discussed last lecture creates problems when a class inherits from multiple other classes? How do you solve it?

UML

- Stands for U____ M____ L_____.
- Draw in UML
 - Class A has method "doSomething()" that takes one argument – a string, and returns a string
 - Class B inherits from A
 - Class C inherits from A and B
 - Class C and contains zero or more instances of B contained

Patterns in Software

- Christopher Alexander, an architect, wrote about patterns in architecture
- He described “the quality without a name” that some buildings possess
 - Creates a feeling of livability
 - No internal contradictions – form and use are in harmony
 - Easier to understand what this means via counter-examples.
 - A classroom contains valuable equipment which requires protection from theft; the theft protection prevents students from entering and using the equipment.
- Alexander proposed communities can create their own livable environments by combining architectural patterns
 - He catalogued these in his writing.

Patterns in Software

- Software engineers realized that some object-oriented software systems also have “the quality without a name”
 - Sometimes, when working with some-one else’s code, you feel as though you are in the living room of an old friend.
 - Everything works they way you expect.
 - All your intuitions about the system turn out to be true.
 - It’s easy to understand how the entire systems works.
 - Other times, you feel you are staggering about in the dark in house filled with breakable, sharp objects. You can’t change anything without fear of destroying the system.
- Richard Gabriel writes about “the quality without a name” in his book Patterns in Software.



Patterns in Software

- Inspired by Alexander, Beck and Cunningham proposed thinking about software designs as higher-level patterns at the 1987 OOPSLA Conference
 - OOPSLA: Object-Oriented Programming, Systems, Languages, and Applications (Association for Computing Machinery)
- Design Patterns: Elements of Reusable Object-Oriented Software
 - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (sometimes called the “Gang of Four” or “The Four Friends”)
 - Very influential, each chapter describes a pattern observed in software.



A design pattern has

- A name – “a handle we can use to describe a design problem .. in a word or two.”
- A problem – a real-life situation where applying the pattern makes sense.
- A solution - the elements and aspects of the pattern that solve the problem.
- Consequences – results and trade-offs of applying the solution.



Design Patterns

- Some patterns are very common.
 - Software engineers use the names of patterns as shorthand during design discussions
- Each has strengths and weaknesses
- Best way to understand them is to examine a few
- As you do more programming
 - you will observe the same patterns or structures re-occur
 - It will get easier to re-use them and talk about them.



Pattern categories

- Creational – how to create new object instances
 - Abstract Factory, Builder, Factory Method, Prototype, **Singleton**
- Structural – how to organize a system
 - Adapter, Bridge, **Composite**, Decorator, Façade, Flyweight, Proxy
- Behavioral – how objects interact
 - Chain of Responsibility, Command, Interpreter, **Iterator**, Mediator, Memento, **Observer**, State, Strategy, Template Method, Visitor



Singleton

- “Ensure that a class only has one instance, and provide a global point of access to it.”
- What is an example you already know about ... from python?



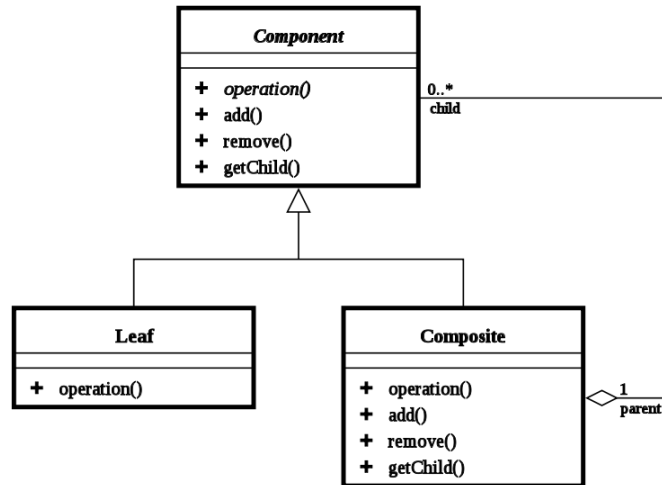
The os object

- The `os` object is a singleton.
- Any given instance of the python interpreter can only interact with one operating system at a time.
- It makes good sense to have a Singleton object representing the operating system.



Composite – Structural

- Use when containers and the things they contain need to have some of the same methods.



Composite – an example

- In genomics, we often need to perform computations on **gene models**, which are hypotheses about how transcription and splicing happens
- Gene models have parts, with start and end positions, such as **exons**
- Gene models also have start and end positions
- Composite pattern works great for representing gene models as objects in python

GeneModel, Range, Exon

- Everything is a Range
- GeneModel is a Range, has Ranges (Exons)

See board



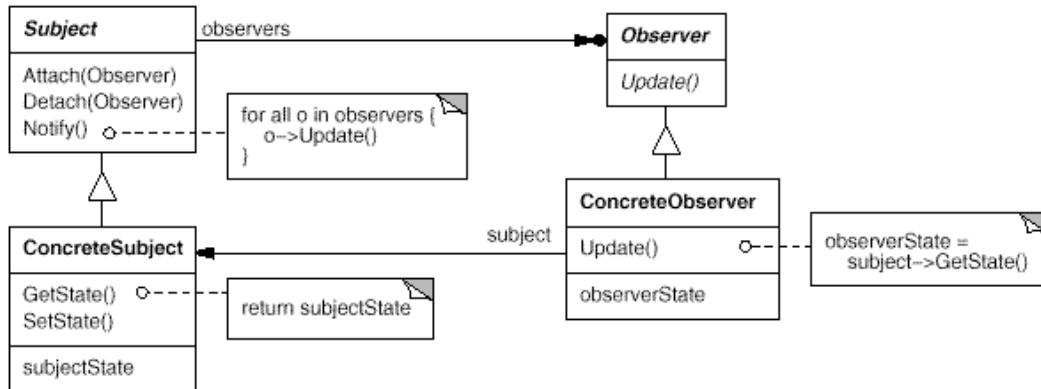
Observer - Behavioral

- “Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated accordingly.”
- Used a lot in graphical user interfaces (GUI)
- You’ll see this again when we cover developing GUIs in Java, which uses the Observer pattern to manage user interactions with graphical components, called “widgets” (menus, windows, etc.)



Observer - Behavioral

- Observers added to Subjects, that notify their Observers when something important happens



Iterator - Behavioral

- “Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.”
- Often used when you need to process a large amount of data, one element at a time, and you can’t read everything into memory.
- For example, a gigantic “fasta” file from a sequencing experiment
 - GA II generates 10 million reads per lane
 - And there are 8 lanes per flow cell (one control)
 - Probably you can’t read them all into memory at once.



Iterator

- Using an iterator in python:

```
import urllib

def openURL(url="http://www.transvar.org"):
    fh = urllib.urlopen(url)
    return fh

if __name__ == "__main__":
    while 1:
        line = fh.readline()
        if not line:
            fh.close()
            break
        else:
            print line
```

Iterator

- Each invocation of `readline` returns the next line of text. You can't go back.
- Sometimes called a *stream*.

```
>>> import iterator_example as i
>>> fh = i.openURL('http://www.uncc.edu')
>>> fh.readline()
'<div class="hide" ><!-- Required for an IE bug with css based flyout menu-->&nbsp;  </div>\n'
>>> fh.readline()
'<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">\n'
>>> fh.readline()
'\n'
>>> fh.readline()
'-----\n'
>>> fh.readline()
'Pulling from New Coldfusion render site\n2002-2009 (c) University of North Carolina at Cha
rlotte\n'
>>> fh.readline()
' Developer: David McIntosh - Manager of Web Services - ITS\n***** IMPORTANT NOTE *****\n'
>>> fh.readline()
' WARNING: Please DO NOT edit this page directly. \n'
```

No homework over the weekend

- On Wed, your first project
- Object-oriented programming assignment:
 - Implement Gene, Range, Exon, etc using the Composite pattern
- Use them in a program to identify overlapping ESTs from a sequencing experiment
- Next week: regular expressions and unit testing.