Inheritance, Part II

Reading: Sec 8.3, 8.5

Review

OOP — define new types of vars, operations on them

Element (name, symbol, atomic weight, ...)
Card (id, suit, rank)

Inheritance — key concept in OOP

*define a new class in terms of an existing class*

Conceptual benefits: easier to comprehend/design a program

Practical benefits: reuse existing code, don’t need to “reinvent the wheel”

Examples of Inheritance

Geometry: define a class named Shape

attributes include area, perimeter, etc

Polygon: a type of shape that has straight sides

specify number of sides when creating the object

Triangle, Rectangle, etc: types of polygons; maybe add new attributes (e.g. height of a triangle, diagonal of a rectangle
**Is-A**

Shape-Polygon-Triangle/Rectangle is an example of a type hierarchy

Top of the hierarchy: more general concepts
Bottom levels: more specific

CS jargon: these are examples of “is-a” relationships

A polygon *is a* type of shape. A triangle *is a* kind of polygon.

**More Examples**

From ICUP: Animals (a Bird is an animal, a Duck is a bird, ...)  
Building: a list of rooms [(w,d), (w,d), ...], method called area, ...  
House(Building): number of bedrooms, lot size, style, ...

**Has-A**

The Building example introduces a new type of relationship:

- a House object is-a Building (inherits area, ...)
- a House has-a room list

In a *has-a* relationship, one object has another type of object as an attribute (e.g. a list object is one of the instance variables of a Building or House object)
Queues

Sections 8.3 and 8.5 in the text show two ways to implement a Queue class

[see also Queues.py on class web sites]

A Queue is a linear data structure:

- initially empty
- items are added by enqueue, removed by dequeue

“first in, first out” (FIFO) property: items are removed in the order in which they were added

Other useful operations:

- len
- index operator

Straightforward implementation of the Queue class in Python:

- instance var named q to represent the items in a queue
  - initially []

- enqueue uses q.append() to add an item

- dequeue calls q.pop(0)

[demo]
Is-A or Has-A?

How would you characterize the relationship between a queue object and the list used to represent the queue?

If we make a queue object, can we
   find out how many objects are in the queue?
   access an item in the middle of the queue?
   iterate over a queue (without removing objects)?

A New Design

These additional operations are automatically inherited if we design our queue using Python’s list class as a base class

```python
class Queue(list):
...
```

[.demo]

In this new design, a Queue is-a list...

Note the new Queues do not have a list as an instance variable

Pitfalls

The new queue objects inherit everything from list, including some operations we don’t want (e.g. sort, reverse, insert, ...)

See the third version of Queue in Queues.py for one way to deal with this....