Lecture 16: Using Classes Effectively

Announcements

Prelim pickup
If you have not picked up your prelim you can pick it up after lecture.

Spring Break
Have a great break! No office hours or consulting hours during break.
Important!

**YES**

```python
class Point(object):
    """Instances are 3D points"
    x [float]: x coord
    y [float]: y coord
    z [float]: z coord"
```

3.0-Style Classes
Well-designed

**NO**

```python
class Point:
    """Instances are 3D points"
    x [float]: x coord
    y [float]: y coord
    z [float]: z coord"
```

“Classic” Classes
No reason to use these
Designing types

• One definition of a *type*: a set of objects with the operations on those objects.
  - int—set: integers; ops: +, −, *, /, …
  - Time—set: times of day; ops: time span, before/after, …
  - Worker—set: all possible workers; ops: hire, pay, promote, …
  - Rectangle—set: all axis-aligned rectangles in 2D; ops: contains, intersect, …

• When you define a class, you are (should be) thinking of a “real type” you want to create
  - Python gives you the tools to do this, but doesn’t do it for you
  - Physically, any object can take on any value
  - Discipline is required to get what you want
Making a class a type

1. Think about what values you want in the set
   - What attributes? What values can they have?
2. Think about what operations you want
   - Often influences the previous question
   • To make (1) precise: write a class invariant
     - A statement we promise ourselves to keep true after every method call
   • To make (2) precise: write specifications of methods
     - A statement of what the method does and what it expects (preconditions)
• Write your code to make these statements true!
Planning out a class

```python
class Time(object):
    """
    Instances represent times of day.
    Instance variables:
        hour [int]: hour of day, in 0..23
        min [int]: minute of hour, in 0..59
    """

    def __init__(self, hour, min):
        """
        The time hour:min.
        Pre: hour in 0..23; min in 0..59
        """

    def increment(self, hours, mins):
        """
        Move this time <hours> hours and <mins> minutes into the future.
        Pre: hours [int] >= 0; mins in 0..59
        """

    def is_pm(self):
        """
        Returns: this time is noon or later.
        """
```

class invariant
States what attributes are present and what values they can have.
A statement that will always be true of Time instances.

method specification
States what the method does.
Gives preconditions stating what is assumed to be true of the arguments.
class Rectangle(object):
    """Instances represent rectangular regions of the plane.
    Instance variables:
        t [float]: y coordinate of top edge
        l [float]: x coordinate of left edge
        b [float]: y coordinate of bottom edge
        r [float]: x coordinate of right edge
    For all Rectangles, l <= r and b <= t.
    """

def __init__(self, t, l, b, r):
    """The rectangle [l, r] x [t, b]
    Pre: args are floats; l <= r; b <= t"

def area(self):
    """Return: area of the rectangle."""

def intersection(self, other):
    """Return: new Rectangle describing intersection of self with other."""
Planning out a class

```python
class Hand(object):
    """Instances represent a hand in cards. Instance variables:
    cards [list of Card]: cards in the hand
    This list is sorted according to the ordering defined by the Card class.
    """

def __init__(self, deck, n):
    """Draw a hand of n cards.
    Pre: deck is a list of >= n cards""

def is_full_house(self):
    """Return: True if this hand is a full house.""

def discard(self, k):
    """Discard the k-th card.""
```

class invariant
States what attributes are present and what values they can have.
A statement that will always be true of Hand instances.

method specification
States what the method does.
Gives preconditions stating what is assumed to be true of the arguments.
Implementing a class

• All that remains is to fill in the methods. (All?!)

• When implementing methods:
  ▪ Assume preconditions are true
  ▪ Assume class invariant is true to start
  ▪ Ensure method specification is fulfilled
  ▪ Ensure class invariant is true when done

• Later, when using the class:
  ▪ When calling methods, ensure preconditions are true
  ▪ If attributes are altered, ensure class invariant is true
Implementing an initializer

```python
def __init__(self, hour, min):
    """The time hour:min.
    Pre: hour in 0..23; min in 0..59""
    self.hour = hour
    self.min = min
```

Instance variables:
- hour [int]: hour of day, in 0..23
- min [int]: minute of hour, in 0..59
Implementing a method

Instance variables:
- hour [int]: hour of day, in 0..23
- min [int]: minute of hour, in 0..59

```python
def increment(self, hours, mins):
    """Move this time <hours> hours and <mins> minutes into the future.
    Pre: hours [int] >= 0; mins in 0..59""

    self.min = self.min + min
    self.hour = (self.hour + hour +
                 self.min / 60)
    self.min = self.min % 60
    self.hour = self.hour % 24
```

This is true to start

What we are supposed to accomplish

This is also true to start

You put code here

This should be true at the end
The view from outside

• Invariants and preconditions serve two purposes
• They are tools for you, as the author, to think through your plans in a disciplined way
• They communicate to the user* of the class how they are allowed to use it
• Together they are the interface of the class
  ▪ interface between two programmers
  ▪ interface between two parts of the program

* …who might well be you!

---

**interface** | ˈɪntərˌfæs| noun
1. a point where two systems, subjects, organizations, etc., meet and interact: the interface between accountancy and the law.
   • *chiefly Physics* a surface forming a common boundary between two portions of matter or space, e.g., between two immiscible liquids: the surface tension of a liquid at its air/liquid interface.
2. **Computing** a device or program enabling a user to communicate with a computer.
   • a device or program for connecting two items of hardware or software so that they can be operated jointly or communicate with each other.

— The Oxford American Dictionary