CSC108H Lecture 27

Dan Zingaro

November 14, 2012

Inheritance

- Many times, a new class will have much in common with an existing class
- Copying-and-pasting from the old class, and then making a few changes, is a bad idea
  - What if you then discover a bug in the original class?
  - What if you find that there are many classes you'd like to create from the original one?
- **Inheritance** allows a new class to specialize an existing class by specifying only what is different between them
- The class that inherits is called a subclass, and the class that is inherited from is its superclass

Is-a

- Inheritance models is-a relationships
- The subclass "is-a" subset of the superclass
- e.g.
  - The superclass could be `Ship`, representing all ships
  - The subclass could be `InvisibleShip` that "is-a" `Ship` but cannot take damage

Has-a

- Be careful not to use inheritance to model has-a relationships
- For example, consider `Segment` (line segments) and `Point` (points)
- Is a point a line segment? — No
- Is a line segment a point? — No
- Inheritance cannot be used here
- A line segment "has" two points, not "is" two points
- Has-a relationships are modeled with instance variables, not inheritance

ConcepTest

In the following pairs of words, the first is the subclass and the second is the superclass. Which of them is a correct example of inheritance?

- A. dog, cat
- B. dog, animal
- C. animal, dog
- D. dog, tail
- E. None of the above

ConcepTest

In the following pairs of words, the first is the subclass and the second is the superclass. Which of them is a correct example of inheritance?

- A. school, building
- B. school, student
- C. student, school
- D. school, computer
- E. None of the above

Inheriting from a Class

- So far all classes have started with `class Name(object):`
- To inherit from another class, use that class' name in the parentheses instead of `object`

```
class InvisibleShip(Ship):
  ...
```

When calling a method on an `InvisibleShip`:

- If the method exists in `InvisibleShip`, it is called
- Otherwise, the one in `Ship` is called

ConcepTest

```
class A(object):
  def __init__(self, x):
    self.x = x

  def __str__(self):
    return str(self.x)


class B(A):
  def __init__(self, x):
    self.x = x * 2

b = B(5)
print(b)
```

What is the output of this code?

- A. 5
- B. 10
- C. 510
- D. 105

Unit-Testing

- A docstring is not the place for a full test suite
- Instead, we create a separate file that contains all of our test cases, and run them with `unittest`
- We subclass `unittest.TestCase` to make a class of test cases
- Each method in this class contains one test case
- Inheriting from `TestCase` gives us access to methods used for testing
  - `assertEqual`, `assertTrue`, `assertFalse`

Testing Dictionary Inversion

- Remember back to our dictionary-inversion function
- It returns a dictionary where values are lists
- The order of elements in these lists does not matter, but different list orderings make == return `False`
- In our tests for this function, we can't just compare the returned dictionary with the expected dictionary
- Instead, check that the keys are the same, and that values are the same irrespective of order
- Also, check that the function does not modify the dict parameter

Testing Functions that Return None

```
def merge_dict(d1, d2):
  ''' (dict of ({object: int}, dict of {object: int})) -> NoneType
  Add key/value pairs from d2 into d1. If a key from d2 already
  appears in d1, the new value in d1 is the sum of the values. If a
  key appears only in d1 or d2, then the new value in d1 is the
  original value from the dictionary that contained this key.
  d2 is unchanged.
```

- Define dicts `d1` and `d2`
- Call `merge_dict(d1, d2)`
- Assert three things: `merge_dict` returned `None`, `d1` is as expected, `d2` is unchanged