Winter 2014

Computer Science 1MD3
Introduction to Programming

Michael Liut (liutm@mcmaster.ca)
Ming Quan Fu (fumq@mcmaster.ca)
Brandon Da Silva (dasilvbc@mcmaster.ca)

www.michaelliut.ca/cs1md3

Contact Information

- Feel free to email any of us! Please **DESCRIBE YOUR QUESTION IN DETAIL** WITH YOUR **FULL NAME** & **STUDENT NUMBER**.
- We will reply to your email as soon as possible. You may not get a reply the day of the assignment due date or midterm .
  - Michael Liut (liutm@mcmaster.ca)
  - Brandon Da Silva (dasilvbc@mcmaster.ca)
  - Ming Quan Fu (fumq@mcmaster.ca)
    - Ming's Personal Office: ITB 206
- No Office hours, do you want to schedule a meeting with us?
  - Email us first to schedule a time. We do not have set office hours.
- Drop-in Centre not run by TAs, but by Engineering Department.

Assignments

Assign_2: **Wed Feb 26/14 BY 11pm**

Assign_3: **Tue Mar 13/14 BY 11pm**

Outline

- Review OO in last tutorials
- OO Programming in python
- OO Examples in detail

This is taken from [1]

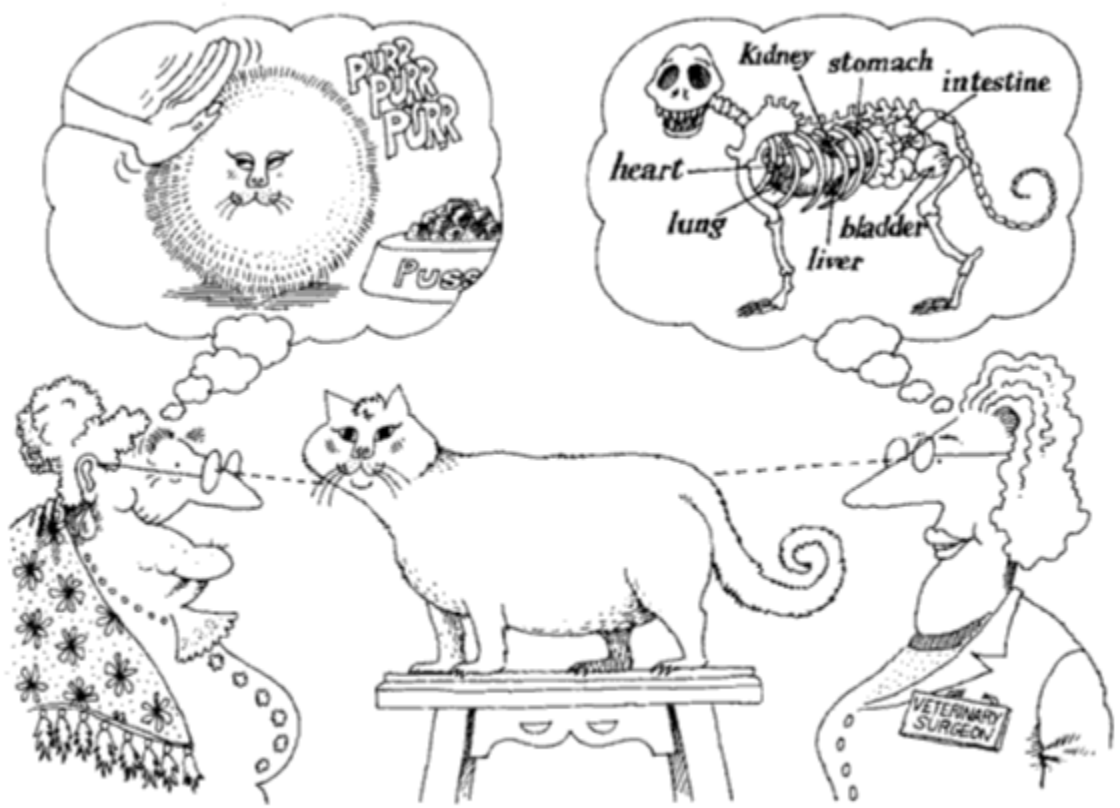
Review: Understanding OO

For all things object-oriented, the conceptual framework is the object model. There are four major elements of this model:

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

This is taken from [1]

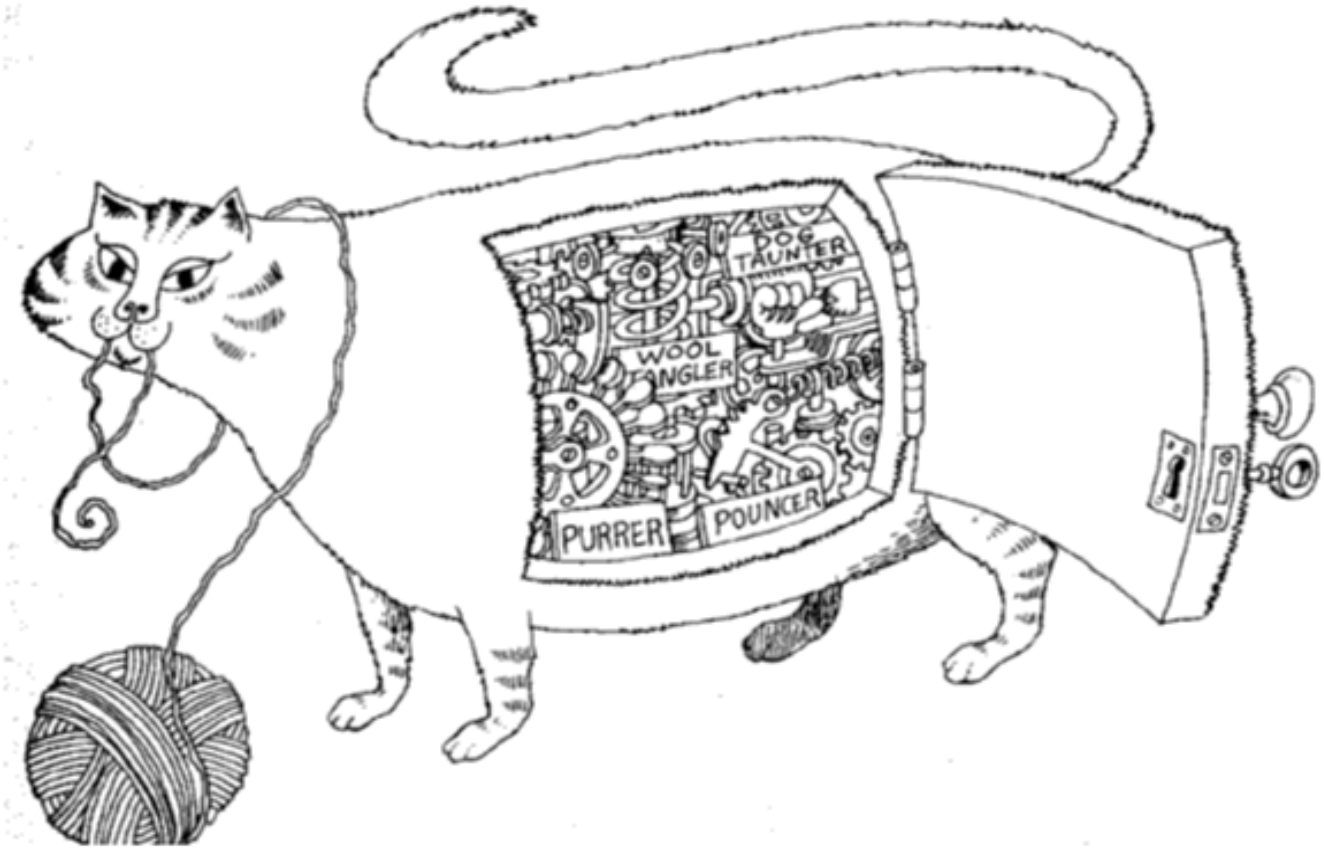
Review Understanding Class: Abstraction



Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

This is taken from [1]

Review: Understanding Class: Encapsulation



Encapsulation hides the details of the implementation of an object.

This is taken from [1]

Review Understanding Class: Modularity

Modularity packages abstractions into discrete units.

This is taken from [1]

Review: Understanding Class and Object: Hierarchy

This is taken from [1]

Overview of OOP Terminology

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables aren't used as frequently as instance variables are.
- Data member: A class variable or instance variable that holds data associated with a class and its objects.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

This is taken from [1]

OO Example in Python Class

```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print "Total Employee %d" % Employee.empCount

   def displayEmployee(self):
      print "Name : ", self.name,  ", Salary: ", self.salary
```

This is taken from [5]

Creating instance objects

- To create instances of a class, you call the class using class name and pass in whatever arguments its __*init*__ method accepts.

```
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
```

This is taken from [5]

Accessing attributes

- You access the object's attributes using the dot operator with object. Class variable would be accessed using class name as follows:

```
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```

This is taken from [5]

Complete Example

Now, putting all the concepts together:

```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print "Total Employee %d" % Employee.empCount

   def displayEmployee(self):
      print "Name : ", self.name,  ", Salary: ", self.salary

"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```

This is taken from [5]

Result and remove attributes of class and object

- When the above code is executed, it produces the following result:

```
Name :  Zara ,Salary:  2000
Name :  Manni ,Salary:  5000
Total Employee 2
```

- You can add, remove or modify attributes of classes and objects at any time:

```
emp1.age = 7  # Add an 'age' attribute.
emp1.age = 8  # Modify 'age' attribute.
del emp1.age  # Delete 'age' attribute.
```

This is taken from [5]

Class Inheritance child class

- Instead of starting from scratch, you can create a class by deriving it from a pre existing class by listing the parent class in parentheses after the new class name.
- The child class inherits the attributes of its parent class, and you can use those attributes as if they were defined in the child class. A child class can also override data members and methods from the parent.[5]

This is taken from [5]

Class Inheritance child class

- Derived classes are declared much like their parent class; however, a list of base classes to inherit from are given after the class name:

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
   'Optional class documentation string'
   class_suite
```

This is taken from [5]

Class Inheritance child class

```
class Parent:          # define parent class
   parentAttr = 100
   def __init__(self):
      print "Calling parent constructor"

   def parentMethod(self):
      print 'Calling parent method'

   def setAttr(self, attr):
      Parent.parentAttr = attr

   def getAttr(self):
      print "Parent attribute :", Parent.parentAttr

class Child(Parent): # define child class
   def __init__(self):
      print "Calling child constructor"

   def childMethod(self):
      print 'Calling child method'

c = Child()            # instance of child
c.childMethod()        # child calls its method
c.parentMethod()       # calls parent's method
c.setAttr(200)         # again call parent's method
c.getAttr()            # again call parent's method
```

This is taken from [5]

Class Inheritance child class

Result:

Calling child constructor

Calling child method

Calling parent method

Parent attribute : 200

This is taken from [5]

Overriding Methods

- You can always override your parent class methods. One reason for overriding parent's methods is because you may want special or different functionality in your subclass.

```
class Parent:        # define parent class
   def myMethod(self):
      print 'Calling parent method'

class Child(Parent): # define child class
   def myMethod(self):
      print 'Calling child method'

c = Child()          # instance of child
c.myMethod()         # child calls overridden method
```

This is taken from [5]

Overloading

```
class Vector:
   def __init__(self, a, b):
      self.a = a
      self.b = b

   def __str__(self):
      return 'Vector (%d, %d)' % (self.a, self.b)

   def __add__(self,other):
      return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)
v2 = Vector(5,-2)
print v1 + v2
```

This is taken from [5]

Data Hiding

An object's attributes may or may not be visible outside the class definition. For these cases, you can name attributes with a double underscore prefix, and those attributes will not be directly visible to outsiders.

This is taken from [5]

Data Hiding: example

```
#!/usr/bin/python

class JustCounter:
    __secretCount = 0

    def count(self):
        self.__secretCount += 1
        print self.__secretCount

counter = JustCounter()
counter.count()
counter.count()
print counter.__secretCount
```

This is taken from [5]

Data Hiding: example

- When the above code is executed, it produces the following result:

```
1
2
Traceback (most recent call last):
  File "test.py", line 12, in <module>
    print counter.__secretCount
AttributeError: JustCounter instance has no attribute '__secretCount'
```

Data Hiding: example

- Python protects those members by internally changing the name to include the class name. You can access such attributes as *object._className__attrName*. If you would replace your last line as following, then it would work for you:

```
..........................
print counter._JustCounter__secretCount
```

Data Hiding: example

- When the above code is executed, it produces the following result:

```
1
2
2
```

Reference

- [1] Book, OBJECT-ORIENTED ANALYSIS AND DESIGN

Grady Booch, Second Edition, SBN 0-8053-5340-2,1998

- [2] http://en.wikipedia.org/wiki/Computer_program
- [3] Keene. Object~Oriented Programming, p. 118.
- [4] Lea, D. Aug.st 12, 1988. Users Guide to GIVV C++ Library. Cambridge, MA: Free Software Foundation, p. 12
- [5] http://www.tutorialspoint.com/python/