CSc 110, Autumn 2016

Lecture 31: Encapsulation

Adapted from slides by Marty Stepp and Stuart Reges
Abstraction

Don't need to know this

Can focus on this!!

An x64 processor is screaming along at billions of cycles per second to run the XNU kernel, which is frantically working through all the POSIX-specified abstraction to create the Darwin system underlying OS X, which in turn is straining itself to run Firefox and its Gecko renderer, which creates a Flash object which renders dozens of video frames every second because I wanted to see a cat jump into a box and fall over.

I am a god.
Encapsulation

• **encapsulation**: Hiding implementation details of an object from its clients.

  • Encapsulation provides *abstraction*.
    • separates external view (behavior) from internal view (state)
  • Encapsulation protects the integrity of an object's data.
Private fields

• A field can be made invisible to outsiders
  • No code outside the class can access or change it easily.

  __name

• Examples:

  self.__id
  self.__name

• Client code sees an error when accessing private fields:
Accessing private state

• We can provide methods to get and/or set a field's value:
  
  # A "read-only" access to the __x field ("accessor")
  def get_x(self):
      return self.__x

  # Allows clients to change the __x field ("mutator")
  def set_x(self, new_x):
      self.__x = new_x

• Client code will look more like this:
  
  print("p1: (" + str(p1.get_x()) + ", " + str(p1.get_y()) + ")")
  p1.set_x(14)
Benefits of encapsulation

• Provides abstraction between an object and its clients.

• Protects an object from unwanted access by clients.
  • A bank app forbids a client to change an Account's balance.

• Allows you to change the class implementation.
  • Point could be rewritten to use polar coordinates (radius $r$, angle $\theta$), but with the same methods.

• Allows you to constrain objects' state (invariants).
  • Example: Only allow Points with non-negative coordinates.
A Point object represents an (x, y) location.

class Point:
    self.__x
    self.__y

def __init__(self, initial_x, initial_y):
    self.__x = initial_x
    self.__y = initial_y

def distance_from_origin(self):
    return sqrt(self.__x * self.__x + self.__y * self.__y)

def get_x(self):
    return self.__x

def get_y(self):
    return self.__y

def set_location(self, new_x, new_y):
    self.__x = new_x
    self.__y = new_y

def translate(self, dx, dy):
    self.__x = self.__x + dx
    self.__y = self.__y + dy
Client code, version 4

def main9):
    # create two Point objects
    p1 = Point(5, 2)
    p2 = Point(4, 3)

    # print each point
    print("p1: (" + str(p1.get_x()) + ", " + str(p1.get_y()) + ")")
    print("p2: (" + str(p2.get_x()) + ", " + str(p2.get_y()) + ")")

    # move p2 and then print it again
    p2.translate(2, 4)
    print("p2: (" + str(p2.get_x()) + ", " + str(p2.get_y()) + ")")

OUTPUT:
p1 is (5, 2)
p2 is (4, 3)
p2 is (6, 7)