Introduction to Objects

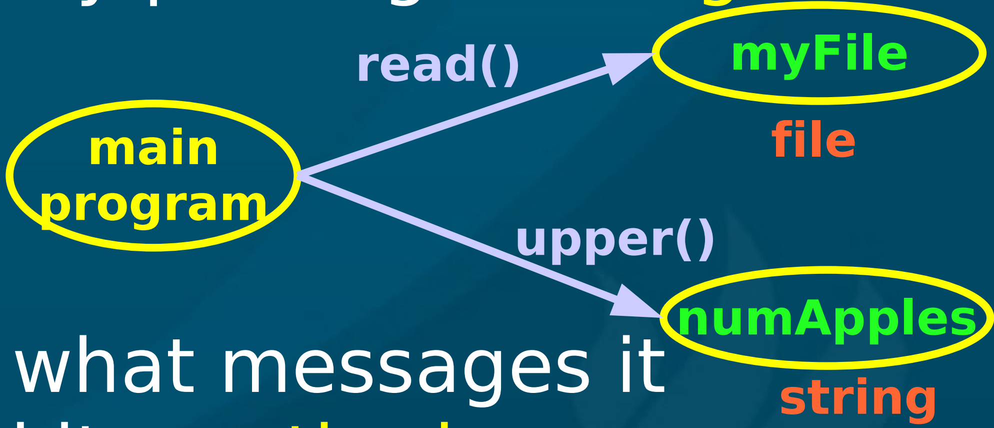
6 Nov 2009
CMPT140
Dr. Sean Ho
Trinity Western University

Object-oriented programming

- Procedural paradigm: “recipe” list of actions
  - Focus is on the procedures (verbs)
  - Variables, data structures get passed into procedures
  - **e.g.: `string.upper('hello')`**
- Object-oriented paradigm: collections of objects
  - Focus is on the data (nouns)
  - Messages get passed between objects
  - Procedures are methods belonging to objects
  - **e.g.: `'hello'.upper()`**

Everything is an object

- In object-orientation, all data are objects:
  - Variables, procedures, even libraries
- We make things happen by passing messages between objects
  - myFile.read(16)
  - appleName.upper()
- The object itself defines what messages it accepts: these are called its methods
  - e.g., files have read(), write(), etc.
    strings have upper(), len(), etc.



Methods and attributes

- Everything you can do with an object is encapsulated in its object definition
- Objects have attributes (local variables)
- Objects have methods (functions)
    - A collection of methods defines an interface
- Example: design an ADT for a Student:
    - Attributes: data stored with each Student
        - Name, ID#, phone #, GPA, course list, ….
    - Methods: operations involving a Student:
        - Register for course, change major, call dad for $$, …

Classes and instances

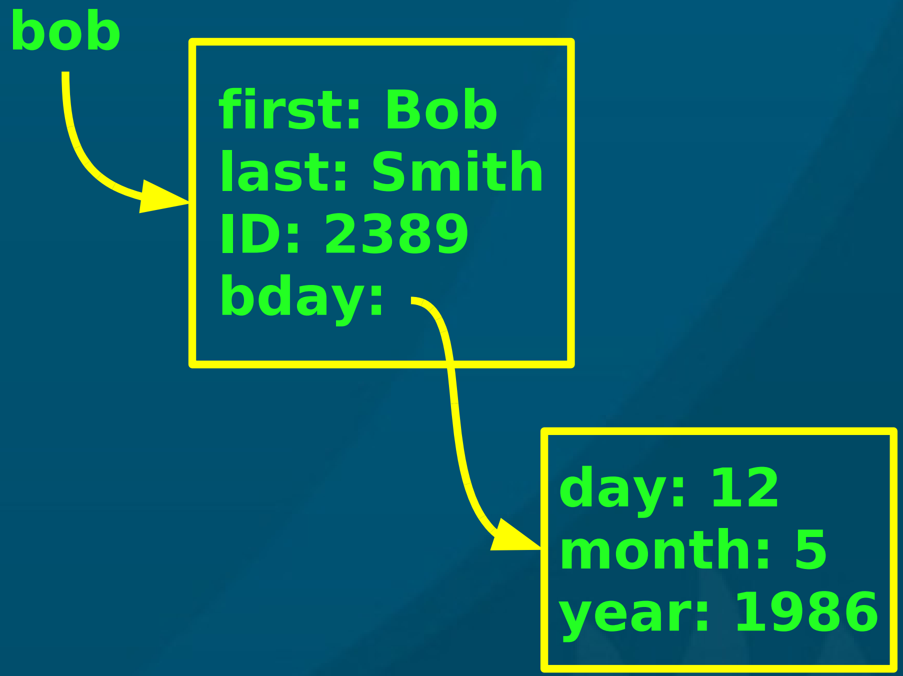
- We define (declare) object classes (types).
  A class is a user-defined type, containing:
  - Attributes: data stored in each object
  - Methods: operations involving the object
    - Constructor method: how to set up a new object
    - Destructor method: how to destroy an object cleanly
- Then instantiate the class (declare variables)
- e.g.: joe is a variable of type Student
  - joe is the instance; Student is the class

Example: declaring a class

- Define the Student type (capitalize class name):
  - **class Student:**
    - **def __init__(self):**
      - **self.firstName = ''**
      - **self.lastName = ''**
      - **self.GPA = 4.0**
- The __init__() method is the constructor
- First parameter of all methods is 'self'
  - Refers to current object
- Instantiate a new object of Student type:
  - **joe = Student()** **joe.firstName = “Joe”**

Objects may hold other objects

- class Date:
  - def __init__(self):
    - self.day = 0
    - self.month = 0
    - self.year = 0
- class Student:
  - def __init__(self):
    - self.firstName = ""
    - self.lastName = ""
    - self.birthdate = Date()



- Creating a new Student makes a new Date object:
  - bob = StudentRecord()
  - bob.birthdate.year = 1986

Constructor with parameters

- We can pass parameters to the constructor:
  - Usually for setting initial values of attributes:
    - class Student:
      - def __init__(self, f, l, g):
        - self.firstName = f
        - self.lastName = l
        - self.GPA = g
- Instantiate with name 'Joe Smith' and GPA=3.8:
  - joe = Student('Joe', 'Smith', 3.8)
- This now requires 3 parameters to constructor

Default parameters

- Functions may have default parameters:
  - **def double_me(x=0):**
    - **return x*2**
- Can call double_me() with 0 or 1 parameters:
  - **double_me() → returns: 0**
- Apply this to the constructor:
  - **class Student:**
    - **def __init__(self, f='', l='', g=4.0):**
      - **self.firstName = f**
      - **self.lastName = l**
      - **self.GPA = g**