Announcements

- Homework 5 due Wednesday 3/4 @ 11:59pm
- Project 3 due Thursday 3/12 @ 11:59pm
- Midterm 2 on Thursday 3/19 7pm-9pm
- Quiz 2 released Wednesday 3/4
  - Due Thursday 3/5 @ 11:59pm
- Object-oriented programming
- Similar to homework 5
- Guerrilla section this Sunday 3/1 on mutation

Lecture 15: Inheritance

2/27/2015
Guest Lecturer: Marvin Zhang

Some (a lot of) material from these slides was borrowed from John DeNero.

Inheritance

- Powerful Idea in Object-Oriented Programming
- Way of relating similar classes together
- Common use: a specialized class inherits from a more general class

class <new class>: (base class):
  ...
  The new class shares attributes with the base class, and overrides certain attributes
  Implementing the new class is now as simple as specifying how it’s different from the base class

Inheritance Example

class Account:
  
  ...  # Account.__init__

  class CheckingAccount(Account):
    
    ...  # CheckingAccount.__init__

    # Checking account:
    
    def __init__(self, name, balance=0, interest_rate=2%):
      super().__init__(name, balance)
      self.interest_rate = interest_rate

    def deposit(self, amount):
      super().deposit(amount)
      self.balance += amount
      self.interest += amount * interest_rate

    def withdraw(self, amount):
      super().withdraw(amount)
      self.balance -= amount
      self.interest -= amount * interest_rate

    def interest(self):
      super().interest()
      self.interest += self.balance * interest_rate

# Checking account:

>>> tom = CheckingAccount("Tom")

Attribute Look Up

To look up a name in a class:
1. If the name is in the attributes of the class, return the corresponding value
2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!
Designing for Inheritance

- Don’t repeat yourself! Use existing implementations
- Reuse overridden attributes by accessing them through the base class
- Look up attributes on instances if possible

class CheckingAccount(Account):
  withdraw_fee = 1
  def withdraw(self, amount):
    return Account.withdraw(self, amount)
Inheritance vs Composition (demo)

• Inheritance: relating two classes through specifying similarities and differences
  • Represents “is a” relationships, e.g. a checking account is a specific type of account
• Composition: connecting two classes through their relationship to one another
  • Represents “has a” relationships, e.g. a bank has a collection of bank accounts

Multiple Inheritance

• In Python, a class can inherit from multiple base classes
• This exists in many but not all object-oriented languages
• This is a tricky and often dangerous subject, so proceed carefully!

class SavingsAccount(Account):
    deposit_fee = 2
    def deposit(self, amount):
        return Account.deposit(self, amount - self.deposit_fee)

Multiple Inheritance Example

• Bank executive wants the following:
  • Low interest rate of 1%
  • $1 withdrawal fee
  • $2 deposit fee
  • A free dollar for opening the account!

class BestAccount(CheckingAccount, SavingsAccount):
    def __init__(self, account_holder):
        self.holder = account_holder
        self.balance = 1 # best deal ever

Multiple Inheritance Example

>>> such_a_deal = BestAccount('Marvin')
>>> such_a_deal.balance # instance attribute
1
>>> such_a_deal.deposit(20) # SavingsAccount
19
>>> such_a_deal.withdraw(5) # CheckingAccount
13

Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.

Moral of the story: inheritance (especially multiple inheritance) is complicated and weird. Use it carefully!