Chapter 9

Object-Oriented Design and High-Level Programming Languages
9.1 OO Methodology

**OO Design**

A problem-solving methodology that produces a solution to a problem in terms of self-contained entities called *objects*.

**Object**

A thing or entity that makes sense within the context of the problem.

E.g., a *student*, a *car*, *the time*, *the date*.
Object-Oriented Design

World View of OOD

Problems are solved by

– isolating the objects in a problem,
– determining their properties and actions (responsibilities), and
– letting the objects collaborate to solve a problem

What? Say again!
Object-Oriented Example

You and your friend are making dinner

**Objects**: you, friend, dinner

**Classes**: you and friend are people

People have name, eye color, ...

People can shop, cook, ...

**Instance of a class**: you and friend are instances of class People, you each have your own name and eye color, you each can shop and cook

You collaborate to make dinner
Object-Oriented Lingo

**Class** = description of a *group* of similar objects  
**Object** = *instance* of a class = concrete example of an object in the class  
**Classes** contain:  
- Properties (attributes) *(name, eye color)*  
- Behaviors (responsibilities) *(shop, cook)* of the class  
**Method** = A named algorithm that defines behavior *(shop, cook)*  
- Python: `my_list.append(42)`
Two types of design

Top-Down Design decomposes problems into **tasks**

Object-Oriented Design decomposes problems into collaborating **objects**

Yes, but how?
OOD Steps

– **isolate** the real-world objects in the problem
– **abstract** the objects with like properties into groups (classes)
– **determine** the responsibilities of the group in interacting with other groups
OOD Example

Think of design as a mapping from real world objects to classes of objects

```
class Date:
    d = '00-00-0000'  #format MM-DD-YYYY
```
class Date

dogBirthdate
myBirthdate
marriageDate

Description
Instances

class Date:
d = '00-00-0000'  #format MM-DD-YYYY

dogBirthDate = Date()
myBirthDate = Date()
marrigeDate = Date()}
We call an object's interactions with other objects its responsibilities.

Create itself
Know the state of its fields
Compare itself to another date
Return a date a number of days hence
Responsibilities become **methods** associated with the classes and objects

```java
class Date
    getMonth
    getDay
    getYear
```

- `dogBirthdate.getMonth()`
- `dogBirthdate.getDay()`
- `dogBirthdate.getYear()`

- `myBirthdate.getDay()`
- Etc.

- `marriageDate.getYear()`
- Etc.
Aren’t we putting the carriage in front of the horses?
class Date:
    d = '00-00-0000'  # format MM-DD-YYYY
    def initMonth(self):
        mo = input('Enter month (MM):')
        self.d = str(mo) + self.d[2:]
    def getMonth(self):
        return self.d[0:1]

How do we use the class in a program?
class Date:
    d = '00-00-0000' # format MM-DD-YYYY
    def initMonth(self):
        mo = input('Enter month (MM):')
        self.d = str(mo)+ self.d[2:]
    def getMonth(self):
        return self.d[0:1]

dogBD = Date()
dogBD.initMonth()
print dogBD.getMonth()

>>> Enter month (MM):04
4
>>>
Do not confuse the “responsibilities” of real-life people with the “responsibilities” of the objects representing those people!

Example: the zoo-keeper

- In real-life, (s)he has the responsibility to feed the animals, clean the cages, etc.

- In an OO program: `zooKeeper.getSSN()`, `zooKeeper.getListOfAnimalsCaredFor()`, etc.
OOD Methodology

Four stages to the decomposition process

– **Brainstorming** to locate possible classes

– **Filtering** the classes to find duplicates or remove unnecessary ones

– **Scenarios** are tried to be sure we understand collaborations

– **Responsibility algorithms** are designed for all actions that classes must perform
Let’s examine the problem-solving process for creating an address list

Brainstorming and filtering
– Circling the nouns and underlining the verbs is a good way to begin

Create a list that includes each person’s name, telephone number, email, and address. This list should then be printed in alphabetical order. The names to be included in the list are on scraps of paper and business cards.
OOD Example
Design tool: CRC Cards

Class-Responsibility-Collaboration cards = notational device to record information about a class, what is must do and with whom it must collaborate.

We’ll talk about **inheritance** next time.

<table>
<thead>
<tr>
<th>Class Name:</th>
<th>Superclass:</th>
<th>Subclasses:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td>Collaborations</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Other useful responsibilities can be added depending on the application, e.g. sending emails, checking out books from library, etc.
Can you think of any other useful responsibilities?
Hint: think of Python strings.
Can you think on other useful responsibilities?  
Hint: Think about the list algorithms from Ch.8
Responsibility Algorithms

Person Class

Initialize

name.initialize()
Write "Enter phone number; press return."
Get telephone number
Write "Enter email address; press return."
Get email address

Print

name.print()
Write "Telephone number: " + telephoneNumber
Write "Email address: " + emailAddress

Tells name to initialize itself
Tells name to print itself
Responsibility Algorithms

Name Class

Initialize

"Enter the first name; press return."
Read firstName
"Enter the last name; press return."
Read lastName

Print

Print "First name: " + firstName
Print "Last name: " + lastName
OO Problem Solving and Implementation Phases

(a) Problem-solving phase

(b) Implementation phase

Problem space of objects

Abstract to classes (descriptions of objects)

Class type definitions

Program space of objects
To do in notebook for next time:

- Answer questions 1-10, 77-80

- Read section 9.2 “Translation Process” and answer questions 11-17
QUIZ Translation Process (9.2)

- Explain the difference between a compiler and an interpreter.
- Is Bytecode compiled or interpreted?
- What is JVM?
- How is JVM similar to PEP/8?
9.3 Programming Language Paradigms

Imperative Paradigm

Program describes the processing, i.e. what needs to be done

Declarative Paradigm

Program describes the goal, i.e. the results that need to be attained
Imperative Languages

• Procedural
  – Characterized by sequential instructions
  – A program in which statements are grouped into a hierarchy of subprograms
  – Fortran, C, C++, Python

• Object-oriented model
  – Program consists of a set of objects and the interactions among the objects
  – Python, C++, Java, Smalltalk, Simula
Imperative Languages

C++ is as a **procedural language** with some OO features

Java is an **OO language** with some procedural features

Python has **both** procedural and OO features.
Declarative Languages

• **Functional**
  – Based on the mathematical concept of a function
  – Lisp, Scheme, and ML

• **Logic**
  – Based on principles of symbolic logic
  – Types of statements
    • declares facts about objects and relationships
    • defines rules about objects
    • asks questions about objects
  – PROLOG
Scheme – Everything is a function!

Prefix notation

#;> (* 3 4)
12
#;> (+ (* 5 4) (+ 1 4))
25
#;> (length '(2 4 6 8 10))
5
#;> (max 2 5 1 3)
5

There is no assignment!

And no variables!
scheme

;;; (define factorial
;;; (lambda (n)
;;; (if (= n 0) 1 (* n (factorial (- n 1))))))
;;; (factorial 7)
;;; 5040

The only way to implement loops is by using recursion!
PROLOG – There is no I/O!

Pets to owners
owns(mary,bo).
owns(ann,kitty).
owns(bob,riley).
owns(susy,charlie).

?-owns(mary,bo)
yes
?-owns(bo,mary)
no
?-owns(susy,bo)
no
?-owns(ann, Cat).
Cat = kitty
yes

?-owns(Name, charlie).
Name = susy
yes

?-owns(Name, toto).
no

Upper case is variable; lower case is constant
Prolog – another example
(not in text)
Prolog – another example
(not in text)
Prolog – another example
(not in text)

%gender facts
female(pam). % Pam is female
female(liz).
female(ann).
female(pat).
female(alexia).

male(tom). % Tom is male
male(bob).
male(tom_jr).
male(jim).
male(john).
male(james).
male(dean).

%derived relations
mother( X, Y) :-
parent( X, Y),
female( X). % X is the mother of Y if X is a parent of Y AND X is female
Prolog – another example (not in text)

grandparent( X, Z) :- % X is a grandparent of Z if
    parent( X, Y), % X is a parent of Y and
    parent( Y, Z). % Y is a parent of Z

%define grandma
%define grandpa

sister( X, Y) :- % X is a sister of Y if
    parent( Z, X), % X and Y have the same parent and
    parent( Z, Y), % X and Y have the same parent and
    female( X), % X is female and
    not( X = Y). % X and Y are different

%define brother

%indiv. work: define aunt and uncle
Prolog – another example (not in text)

%recursive relations

ancestor(X, Z) :- % Rule ancl: X is an ancestor of Z
parent(X, Z).

ancestor(X, Z) :- % Rule anc2: X is an ancestor of Z
parent(X, Y),
ancestor(Y, Z).

%indiv. work: define descendant

?-ancestor(tom, jim).
yes

?-owns(tom, james).
no
%parent relation
parent(pam, bob).
parent(tom, bob).
parent(pam, liz).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
parent(liz, john).
parent(liz, james).
parent(dean, john).
parent(dean, james).
parent(dorian, jim).
parent(pam, tom_jr).
parent(tom, tom_jr).
parent(bob, dan).
parent(liz, alexia).
parent(dean, alexia).

%gender facts
female(pam).
female(liz).
female(ann).
female(pat).
female(alexia).

male(tom).
male(bob).
male(tom_jr).
male(jim).
male(john).
male(james).
male(dean).
male(dan).

sister(X, Y) :-
  parent(Z, X),
  parent(Z, Y),
  female(X),
  not(X = Y).

?-sister(liz, Name)
?-sister(pat, Name)
9.4 Functionality of Procedural Languages

Variables
Input/Output
Control Structures
• loops
• decisions
Subprograms (a.k.a. functions)

READ FOR NEXT TIME
9.5 Functionality of OO Languages

Encapsulation
A language feature that enforces information hiding

Classes
Templates for similar objects (people, names, etc.)

Inheritance
A property that allows a class to inherit the data and actions of another class

Polymorphism
A ability to handle the ambiguity of duplicate names
Class Definition
Skip the pseudocode examples in the text, we’ll concentrate on Python syntax

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

Source: http://www.tutorialspoint.com/python/python_classes_objects.htm
Figure 9.4 Class person
To get objects of a class, we must create (instantiate) them:

```python
emp1 = Employee("Smith", 1000)
emp2 = Employee("Jones", 1200)
```
Inheritance

A construct that fosters reuse by allowing an application to take an already-tested class and derive a class from it that inherits the properties the application needs.

Polymorphism

The ability of a language to have duplicate method names in an inheritance hierarchy and to apply the method that is appropriate for the object to which the method is applied.
Example of inheritance and polymorphism

Polymorphism: The \texttt{Print()} methods are now different, but we’re still calling them by the same name!

- \textbf{Student}
  - GPA
  - Print()

- \textbf{Faculty}
  - Rank
  - Print()
"Example based on http://en.wikipedia.org article"

class Pet:
    def __init__(self, name):
        self.name = name

class Cat(Pet):
    def talk(self):
        return 'Miaou!'  # Get it? :)

class Dog(Pet):
    def talk(self):
        return 'Woof! Woof!'

# main program
pets = [Cat('Missy'), Cat('Scratchy'), Dog('Lassie')]
for p in pets:
    print(p.name + ': ' + p.talk())

>>> Missy: Miaou!
Scratchy: Miaou!
Lassie: Woof! Woof!
Inheritance and Polymorphism

Inheritance and polymorphism work together

How?

They combine to allow the programmer to build useful hierarchies of classes that can be put into a library to be reused in different applications.
Chapter review questions

• Distinguish between functional design and object-oriented design

• Describe the stages of the object-oriented design process

• Apply the object-oriented design process

• Name, describe, and give examples of the three essential ingredients of an object-oriented language

• Understand how the constructs of top-down and object-oriented design are implemented in programming languages
Ethical Issues

Gambling on the Internet

*Have you ever visited an Internet gambling site?*

*Should Internet gambling be outlawed?*

*Should Internet gambling be legalized and regulated?*

*Should Internet gambling be taxed?*
Who am I?

I am best known for structured programming. Can you define it?

I am also known for my wit. Can you recall some of my witty sayings?
Review questions

Answer in notebook!

• 25 – 46
• 73 only Python
• 76 only a. (Python)