CITS2210
Object-Oriented Programming

Topic 1.
Introduction and Fundamentals:
Thinking Object-Oriented

Summary: This topic considers the fundamental concepts behind object orientation, and why they are so effective.

[These slides are based on those supplied by Tim Budd to complement chapter 1 of “An Introduction to Object-Oriented Programming”.]
Roadmap

In this topic we begin our exploration of object-oriented programming. Among the topics we will explore:

What is a paradigm? Why is this term used to describe the object-oriented approach to problem solving?

How does language influence thought?

What are some of the characteristics of the object-oriented way of thinking?

Conflicting Objectives

Along the way, I'll try to convince you the validity of the following two assertions:

- OOP is a revolutionary idea, totally unlike anything that has come before in programming languages
- OOP is an evolutionary step, following naturally on the heels of earlier programming abstractions

Both are true.
Why has OOP Remained Popular for so long?

OOP has been the dominant programming paradigm for more than twenty years. Why is it so popular?

- Proven record of success.
- Scales well from small problems to large
- Resonant similarity to techniques for thinking about problems in other domains.

Nevertheless, programming is still a task that requires skill and learning.

A New Paradigm

Object-oriented programming is often described as a new paradigm.

We start by considering the definition of this term:

**Par a digm n.** A list of all the inflectional forms of a word taken as illustrative example of the conjugation or declension to which it belongs. An example or model. [Late Latein *paradigma*, from Greek *paradeigma*, modern *paradeiknunai*, to compare, exhibit.]

What is the world does this have to do with computer programming languages?
Sapir-Whorf Hypothesis

In linguistics there is a hypothesis that the language in which an idea or thought is expressed colors or directs in a very emphatic manner that nature of the thought:

- Eskimo (or Innuit) languages and snow
- Arabic languages and camels

What is true of natural languages is even more true of artificial computer languages

Example from Computer Languages

A student working in DNA research had the task of finding repeated sequences of M values in a long sequence of values:

ACTCGGATCTTGCATTTGCAGACCTGGACTTTGCCA ...

Wrote the simplest (and therefore, most efficient?) program:

```
DO 10 I = 1, N-M
DO 10 J = I+1, N-M
FOUND = .TRUE.
DO 20 K = 1, M
20  IF X[I+K-1] .NE. X[J+K-1] THEN FOUND = .FALSE.
IF FOUND THEN ...
10  CONTINUE
```

Took a depressingly long time.
A Better Solution

A friend writing in APL found a much better solution by rearranging the data and sorting.

\[
\begin{align*}
A & \quad C & \quad T & \quad C & \quad G & \quad G & \text{positions 1 to } M \\
C & \quad T & \quad C & \quad G & \quad G & \quad A & \text{positions 2 to } M+1 \\
T & \quad C & \quad G & \quad G & \quad A & \quad T & \text{positions 3 to } M+2 \\
C & \quad G & \quad G & \quad A & \quad T & \quad T & \text{positions 4 to } M+3 \\
G & \quad G & \quad A & \quad T & \quad T & \quad C & \text{positions 5 to } M+4 \\
G & \quad A & \quad T & \quad T & \quad C & \quad T & \text{positions 6 to } M+5 \\
& \quad \ldots \\
T & \quad G & \quad G & \quad A & \quad C & \quad C \\
G & \quad G & \quad A & \quad C & \quad C & \quad \ldots \\
& \quad \ldots
\end{align*}
\]

Ran surprisingly quickly, thesis saved.

What lead to the discovery?

Why did the APL programmer find the better solution?

- Fortran programmer was blinded by a culture that valued loops, simple programs
- Sorting is a built-in operation in APL, good programmers try to find novel uses for sorting

The fundamental point is that the language you speak leads you in one direction or another.

But what about the Sapir-Whorf hypothesis, that says there are some thoughts you can express in one language that you cannot express in another?
Church's Conjecture

In computation we have the following assertion:

**Church's Conjecture:** Any computation for which there exists an effective procedure can be realized by a Turing machine language.

Anything can be done in any language, but it may simply be easier or more efficient to use one language or another.

Would YOU want to write an event-driven GUI interface in Turing machine?

Bottom line: Languages lead you, but do not prevent you from going anywhere you want.

Imperative Programming

So, what are the paradigms of programming?

Imperative programming is the ``traditional" model of computation.

- State
- Variables
- Assignment
- Loops

A processing unit is separate from memory, and ``acts" upon memory.
Visualization of Imperative Programming

Sometimes called the "pigeon-hole" model of computation.
Why Not Build a Program out of Computers?

Alan Kay thought about this conventional design of the computer, and asked why we constructed the whole out of pieces that were useless by themselves.

Why not build a whole out of pieces that were similar at all levels of detail? (Think of fractals).

Idea: A program can be build out of little computing agents.

Recursive Design

The structure of the part mirrors the structure of the larger unit.
Kay's Description of Object-Oriented Programming

Object-oriented programming is based on the principle of recursive design.

1. Everything is an object
2. Objects perform computation by making requests of each other through the passing of messages
3. Every object has its own memory, which consists of other objects.
4. Every object is an instance of a class. A class groups similar objects.
5. The class is the repository for behavior associated with an object
6. Classes are organized into singly-rooted tree structure, called an inheritance hierarchy.

We can illustrate these principles by considering how I go about solving a problem in real life.
Illustration of OOP Concepts -- Sending Flowers to a Friend

To illustrate the concepts of OOP in an easily understood framework, consider the problem of sending flowers to a friend who lives in a different city. Chris is sending flowers to Robin.

Chris can't deliver them directly. So Chris uses the services of the local Florist.

Chris tells the Florist (named Fred) the address for Robin, how much to spend, and the type of flowers to send.

Fred contacts a florist in Robins city, who arranges the flowers, then contacts a driver, who delivers the flowers.

If we start to think about it, there may even be other people involved in this transaction. There is the flower grower, perhaps somebody in charge of arrangements, and so on.
Agents and Communities

Our first observation is that results are achieved through the interaction of agents, which we will call objects.

Furthermore, any nontrivial activity requires the interaction of an entire community of objects working together.

Each object has a part to play, a service they provide to the other members of the community.

Elements of OOP - Objects

So we have Kay's first principle.

1. Everything is an object.

Actions in OOP are performed by agents, called instances or objects.

There are many agents working together in my scenario. We have Chris, Robin, the florist, the florist in Robins city, the driver, the flower arranger, and the grower. Each agent has a part to play, and the result is produced when all work together in the solution of a problem.
Elements of OOP - Messages

And principle number 2:

2. Objects perform computation by making requests of each other through the passing of messages

Actions in OOP are produced in response to requests for actions, called messages. An instance may accept a message, and in return will perform an action and return a value.

To begin the process of sending the flowers, Chris gives a message to Fred. Fred in turn gives a message to the florist in Robins city, who gives another message to the driver, and so on.

Information Hiding

Notice that I, as a user of a service being provided by an object, need only know the name of the messages that the object will accept.

I need not have any idea how the actions performed in response to my request will be carried out.

Having accepted a message, an object is responsible for carrying it out.
Elements of OOP - Receivers

Messages differ from traditional function calls in two very important respects:

- In a message there is a designated receiver that accepts the message
- The interpretation of the message may be different, depending upon the receiver

Different Receivers, Same Message, Different Actions

`/* Variables for three people */`  
`Florist fred;`  
`Friend elizabeth;`  
`Dentist ken;`  
`fred.sendFlowersTo(myFriend);  /* will work */`  
`elizabeth.sendFlowersTo(myFriend);  /* will also work */`  
`ken.sendFlowersTo(myFriend);  /* will probably not work */`

The same message will result in different actions, depending upon who it is given to.
Behavior and Interpretation

Although different objects may accept the same message, the actions (behavior) the object will perform will likely be different.

The determination of what behavior to perform may be made at run-time, a form of *late binding*.

The fact that the same name can mean two entirely different operations is one form of *polymorphism*, a topic we will discuss at length in subsequent chapters.

---

Elements of OOP - Recursive Design

3. Every object has its own memory, which consists of other objects.

Each object is like a miniature computer itself - a specialized processor performing a specific task.
Non-interference

It is important that objects be allowed to perform their task however they see fit, without unnecessary interactions or interference with other objects.

• "Instead of a bit-grinding processor ... plundering data structures, we have a universe of well-behaved object that courteously ask each other to carry out their various desires" -- Dan Ingalls.

• "Ask not what you can do to your data structures, but ask what your data structures can do for you"

Elements of OOP - Classes

4. Every object is an instance of a class. A class groups similar objects.

5. The class is the repository for behavior associated with an object.

The behavior I expect from Fred is determined from a general idea I have of the behavior of Florists.

We say Fred is an instance of the class Florist.

Behavior is associated with classes, not with individual instances. All objects that are instances of a class use the same method in response to similar messages.
Hierarchies of Categories

But there is more that I know about Fred then just that he is a Florist. I know he is a ShopKeeper, and a Human, and a Mammal, and a Material Objects, and so on.

At each level of abstraction I have certain information recorded. That information is applicable to all lower (more specialized) levels.
Class Hierarchies

```
Material Object
   /\        /
Living Thing    Non Living Thing
   |       |
Animal          Plant
   |       |
Reptile         Rock
   |       |
Mammal
   |       |
Human Being     Cat
   |       |
Dentist
   |       |
Ken
   |       |
Shopkeeper
   |       |
Flo
   |       |
Artist
   |       |
Beth
```

Intro OOP, Chapter 1, Slide 27
Elements of OOP - Inheritance

6. Classes are organized into a singly-rooted tree structure, called an inheritance hierarchy

Information (data and/or behavior) I associate with one level of abstraction in a class hierarchy is automatically applicable to lower levels of the hierarchy.

Elements of OOP - Overriding

Subclasses can alter or override information inherited from parent classes:

- All mammals give birth to live young
- A platypus is an egg-laying mammal

Inheritance combined with overriding are where most of the power of OO originates.
Computing as Simulation

- The OOP view of computation is similar to creating a universe of interacting computing objects
- Similar to the way in which a committee or club might be organized
- Also very similar to a style of simulation called discrete event-driven simulation
- Easily under estimated advantage of this view -- power of metaphor.

Metaphor and Problem Solving

Because the OOP view is similar to the way in which people go about solving problems in real life (finding another agent to do the real work!), intuition, ideas, and understanding from everyday experience can be brought to bear on computing.

On the other hand, common sense was seldom useful when computers were viewed in the process-state model, since few people solve their everyday problems using pigeon-holes.
``Unlike the usual programming method -- writing software one line at a time-- NeXT's ``object-oriented'' system offers larger building blocks that developers can quickly assemble the way a kid builds faces on Mr. Potato Head.''

Summary

- Object-oriented programming is not simply features added to a programming language. Rather, it is a new way of thinking

- Object-oriented programming views a program as a community of agents, termed objects. Each object is responsible for a specific task.

- An object is an encapsulation of state (data values) and behavior (operations).

- The behavior of objects is dictated by the object class.

- An object will exhibit its behavior by invoking a method (similar to executing a procedure) in response to a message.

- Objects and classes extend the concept of abstract data types by adding the notion of inheritance.