Joanne Chen

Professor John C. Ramirez

CS1501

14th June 2018

This paper is an overview of the field of Object-Oriented Programming. It presents a brief historical perspective of the field, the primary features, some example languages associated with it and their key differences.

Object-Oriented Programming (OOP) is the term used to describe a programming approach based on objects and classes. The object-oriented paradigm allows us to organize software as a collection of objects that share behaviors and provide persistent services over time. This contrasts with conventional functional programming practice that only loosely connects between data and behavior. Conventional functional programming system consists of sequentially executed procedures, which are sequences of statements that specify a transformation from inputs to outputs. Principles of functional programming supports well-defined, defensible software giving reproducible results. Object-oriented programming, on the other hand, provides tools for defining the model objects clearly, adapting to new ideas and new forms of models. Since the 1980s the word 'object' has appeared in relation to programming languages, with almost all languages developed since 1990 having object-oriented features.

One of the earliest languages to begin moving away from procedures towards objects was Simula, developed in the 1960s by Ole-Johan Dahl and Kristen Nygaard at the Norwegian Computing Center (NCC)[1]. Kristen Nygaard started writing computer simulation programs in 1957. He saw a need for a better way to describe the heterogeneity and the operation of a system. And Ole-Johan Dahl joined him on his work January 1962. By May 1962, the main concepts for a simulation language were set. "SIMULA I" was born, a special purpose programming language for simulating discrete event systems. Later in February 1968, SIMULA 67, which extended with the concept of prefixing and other features to generalized process concept, was formally standardized on the meeting of the SIMULA Standards Group (SSG) [1]. In the few short years between the initial design for Simula and Simula 67, the combined necessities of computation, thought, and language completely rearrange the planned metaphor for describing the world and draw significantly closer to a fully object-oriented mode of programming [2].

The main ideas of object-oriented programming are simple and intuitive:

First, everything we compute with is an object, and objects should be structured to suit the goals of our computations. OOP treats data as a critical element in the program development. It ties data more closely to the function that operate on it and protects it from accidental modification from outside function. In so doing, the data obtained should correspond to the given model and satisfy its semantic rules. OOP decomposes a problem into several objects and then builds data and function around these objects.

Second, a key programming tool of OOP is a class definition. Objects belonging to same class share structure defined by properties which they all have. Class is relative to objects. A class describes the implementation and behavior of those objects. We call those objects instances of the class, as class provides a procedure that realizes the description by actually making an instance. For example, in a payroll system, a class could be Manager, and Pat and Jan could be two instances (two objects) of the Manager class. That refers to the creation of self-contained modules (Encapsulation) that bind processing functions to the data. Encapsulation ensures good code modularity, which keeps routines separate and less prone to conflict with each other.

Third, a class can inherit from a simpler superclass, such that an object of this class is also an object of the superclass. Inheritance allows a programmer to specify modified versions of a class and enables new classes to inherit the properties and methods of existing classes. For example, one could define a class for “animal” of which a “cat” would be a subclass, but it would inherit certain properties from the vehicle.

Forth, we can define methods that are only used when objects are of certain classes to compute with objects. Methods can only directly access data in their own object. To access the data within a different object, we must pass the whole object to any methods that need to act on that data. That is making a function or operator to act in different forms depending on the place they are present. For instance, we know that +, – operate on integer data type and is used to perform arithmetic additions and subtractions. But operator overloading defines new operations to these operators and make them operate on different data types.

Examples of programming languages with object-oriented programming (OOP) features include Java, C++, Python, Ruby, R, Smalltalk and etc. Smalltalk has been considered the first Object-Oriented language by many people. It was developed at the Learning Research Group at Xerox's Palo Alto Research Center in the early 1970s. In Smalltalk, everything is really an object. It is virtually impossible to write a program in Smalltalk that is not Object-Oriented. This is not the case for other languages that support objects, such as C++ and Java [2]. Languages that came after Smalltalk (like C++ and Java) have failed to realize the OOP ideals as well as Smalltalk in some degree.

C++ is a multi-paradigm language including object-oriented paradigm; however, it is less object-oriented than some other languages such as Python and Ruby. C++ has its roots in a project to simulate software running on a distributed system. C++ was originally implemented in 1982 for systems and applications programming, extending the procedural programming language C, which was designed for efficient execution. C++ can be used for developing System Software, operating systems, compilers, editors and data bases. It is suitable on reducing cost of software development.

Java was initially created to support network computing on embedded systems. Java has a syntax familiar to C programmers, but direct compatibility with C was not maintained. Java is a fully object-oriented programming language. It has all OOP features such as abstraction, encapsulation, inheritance and polymorphism. It is multi-threading and supports the development of distributed application whether using sockets or by invoking methods of remote objects. A Java program is compiled to a virtual machine (JVM). The loading of code is dynamic. The code produced is independent of machine architectures, being interpreted by a virtual machine.

Python is a multi-paradigm programming language. Object-oriented programming is fully supported, and many of its features support functional programming and aspect-oriented programming. Python's design offers some support for functional programming. It has filter(), map(), and reduce() functions; list comprehensions, dictionaries, and sets; and generator expressions. Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Python features a dynamic type system and automatic memory management. It has a large and comprehensive standard library.

In general, Object-oriented programming is a programming model organized around objects rather than actions. Object-oriented programming focuses on the objects that you want to manipulate, their relationships, and the logic required to manipulate them. The fundamental advantage of object-oriented programming is that the data and the operations that manipulate the data are both encapsulated in the object with a well-defined interface. Objects are the building blocks of an object-oriented program. Most modern programming languages support and encourage object-oriented programming.

References:

[1] Black AP. Object-oriented programming: some history, and challenges for the next fifty years. . 2013.

[2] Joque J. The Invention of the Object: Object Orientation and the Philosophical Development of Programming Languages. Philosophy & Technology. 2016;29(4):335-56.