Jordan Carr

6/17/17

CS 1501

Understanding Object-Oriented Programming

Object-oriented programming is one of the pillars of modern-day programming. It provides an effective way of solving real-world problems more concretely than what earlier programming styles were able to achieve. This is what enables it to form the basis of a large portion of modern day software. The principles behind it form the foundation of many programming languages used today. Understanding the concepts of object-oriented programming, how it can be used, and what languages do differently to support it can give a software developer a greater understanding on how to solve problems within the field of computer science.

When software development was in its infancy the standard approach to writing code was to use procedural programming languages. In this style of programming, code is modularized based on the system’s processes (“Java” 3). For example, when ordering food at a restaurant its processes can be broken down into ordering the food, cooking the food, and among others serving the food. It was with the development of Simula 67 in the 1960’s and Smalltalk in the 1970’s and its idea of a class construct that object-oriented programming began to take shape. The 1980’s saw huge growth in the world of computers. Procedural languages like C were seen as flawed and too limited for the types of complex programs needed for the time (Sharma). However, it was not until the advent of Smalltalk-80 that the concept of object-oriented programming became widely accepted and utilized (“Java” 3). One of the first popular object-oriented languages was C++, developed by Bjarne Stroustrup as a doctoral thesis. It supported features like code reusability, extensibility, and polymorphism. Programmers quickly adapted it for its support of class inheritance (Sharma). C++ went on to heavily influence other object-oriented languages that are still in wide use today such as Java, Python, and C#. As the need for object-oriented languages increased, class based versions of imperative were developed. For instance, C was the basis for C++ and Objective C, and Pascal was the basis for Object Pascal (Harald xiv). Even to this day object-oriented programming languages are still widely used and developed for modern software applications.

Certain attributes of object-oriented programming distinguish it from other styles of programming. The main concept behind it is to consider complex entities which regroup data structures with procedures that allow them to be manipulated. Instances of these data structures are known as objects, and the structure that specifies them and acts as its blueprint are known as classes (Harald xiii). This combination of data and methods is also known as encapsulation. Object-oriented programming operates on the basis of class definitions and instances of these classes. This allows much closer representation of real-world problems than the procedural style since the world as we see it is defined by objects (“Java” 3). As an example of a real world problem take Mike, who is a homeowner, and Joe, who is a plumber. Mike detects that he has a leaking faucet and calls Joe to come fix it. Joe inspects the faucet and determines that he must replace it. Mike then pays Joe for his services. In this case both Mike and Joe can be seen as two different interacting objects, one of type homeowner and the other of type plumber. Each person has different procedures, also known as functions or methods. Mike’s functions include detect and pay, while Joe’s include fix and replace. Both work together to solve the problem of the leaking faucet. This also demonstrates another aspect of object-oriented programming: inheritance. Mike and Joe can be seen as two different types of the object person. Both inherit all of the attributes and methods of the parent class, but further elaborate on them to create specialized objects. In the case of Mike and Joe, both would inherit age, height, weight, and other attributes of a normal person. They would also inherit the normal functions of a regular person, such as eating and sleeping. Objects can also be attributes of other objects. For example, Mike could have a list of children as an attribute, each child being an instantiation of the person class. The procedures defined in class definitions usually utilize or modify these attributes in some way. Much of this data is abstracted from the user in order to make the data structure more accessible. Objects also have the ability to call themselves or their methods within themselves. This is also known as data hiding. It is in this way that object-oriented programming gives programmers a much closer representation of real world problems.

One example of an object-oriented programming language is Java. It was developed in 1995 as a secure style of object-oriented programming and quickly gained popularity. Java is a multiplatform language because it runs on a virtual machine that runs portable, compiled source files. These are known as class files, and every class in Java, whether it is a public class or a private inner class, gets its own. Everything in Java is based on classes, but the main class on which the program will run contains the main method, which is seen as more or less as class independent. Classes can contain both methods and attributes that help define what it is and how it can be manipulated (“Java” 18). Attributes can be primitive data types, such as integers or characters, or instances of other classes. Objects are instantiated via the class constructor and provides a description of the class that serves as the template for the objects (“Java” 20). However, Java can also have static methods, attributes and classes. These do not belong to individual instantiations of the class but to the class as a whole. For example, if there were multiple person objects, a static class variable would be used to keep track of how many persons there are. Anything static is called from the class itself and not from any specific instance of it. Java also supports private inner classes that different classes would not have access to. To demonstrate the use of these classes, consider the linked list. The data in the list would be stored in a private node class whose details would be abstracted from the user in order to hide the inner workings of the data structure. Java version 8 supports lambda functions, which are in essence methods that do not belong to any particular class.

Python is another example of an object-oriented programming language. In many aspects it is very similar to Java. It supports both inheritance and the idea of static and non-static objects like in Java, except they are known as class and instance objects. Class objects are created when references are made to the class without a class construction. On the other hand, instance objects are created when the constructor is called, creating an individual object instead of a single class object like above. Unlike Java, Python supports method objects, which are instantiated when the class is referenced with a function than is not a class constructor (“9”). For example, if xf is initialized from the object x’s method f, it will essentially store away that method for later use with that object x as its argument, even if the argument is not used. However, not everything in Python must be built within a class as in Java. Any code outside of a class definition is in essence similar to code existing in the main method in java. Another difference is that an object with an empty class declaration can serve as something similar to a struct in C or a record in Pascal, which can also be seen as a type of list (“9”).

To understand object-oriented programming it is useful to examine other languages that use objects but are not object-oriented. One would be JavaScript, a programming language designed to run on web browsers to perform client-side computation for web applications. Objects in JavaScript are more of a collection of properties than an object by the standard definition above. When initialized they can be set with a limited number of properties, but they can be added and removed at will. The values of said properties can be of any type, including other objects. Keys identify them, which is either a String or another symbol value (“JavaScript”). They can be seen as similar to objects in both Java and Python to a certain extent. In their simplest form objects can be just a collection of values that describe a data structure. However, unlike JavaScript attributes they cannot be added or removed in the same way. They can be left as a null value, depending on how the class is written, but not removed entirely. JavaScript functions are in some sense objects with the additional attribute of being callable, drawing some comparisons with the method object in Python and lambda functions in Java. These are some of the only ways that JavaScript objects are related to objects in other languages. It is known as an object-based programming language because it omits many of the key features that define object-oriented programming. One of which would be class inheritance. Objects also do not contain methods or functions in JavaScript, although they can be passed as arguments to other functions. In this sense, the object in JavaScript is closer to a struct in procedural languages such as C. It also makes them similar to hash maps or dictionaries in Python in the sense that data is stored and accessed via key and value pairs. Objects in JavaScript also do not have the ability to reference themselves in the sense that they cannot instantiate themselves within their own class definitions. Thus the differences between objects in JavaScript and those of other programming languages further demonstrate what defines object-oriented programming.

Works Cited

“9. Classes.” *Python Software Foundation*, 10 June 2017, <https://docs.python.org/3/tutorial/classes.html>.

“JavaScript data types and data structures.” Mozilla Developer Network, 16 June 2017, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data\_structures#Objects.

Poo, Danny, et al. *Object-oriented Programming and Java*. Springer, 2008.

Sharma, A.K. *Object-oriented Programming with C++*. Pearson India, 2014.

Wertz, Harald. *Object-oriented Programming with Smalltalk*. ISTE Press, 2015.