Modern Object-Oriented Programming 07/2016

Stanislaw Alam,

Dr. Ebrahimi,

C++

To say that object-oriented programming is a modern-day paradigm may be a misnomer—at least, the "modern" part. Although object-oriented programming and design is still the most traditional software development approach for engineering advanced software systems, it did not catch on in modern development until the 1980s. However, the idea of objects and the classes from which objects are derived may be traced to the ancient Greek schools of Plato and Aristotle.

In The Republic, Plato describes "forms," mathematical/geometrical structures that exist in an ethereal state. We can liken these structures to blueprints for real-world objects. For example, say an individual is building a cart. The individual then decides that the vehicle would be more efficient to move with a wheel. This wheel does not yet exist; it exists in the person's mind—as a concept, if you will. Plato would argue that the form of the wheel, i.e., the circle the wheel is derived from, does, in fact, exist. Once again, the idea exists. Let's liken the idea of the class to the context of object-oriented programming. Plato's theory of forms is out of the scope of this paper, but I hope you have gotten a general idea as to how it pertains to our discussion on object-oriented design. Now, let's say that the circle class is in the mind of the individual. To create a functional material object in the real world, i.e., something that is practical, the designer must create the object from raw materials—let's say, using wood he or she harvested from a tree. This wooden wheel will now be a physical object, just as the tree once was. We can say that the properties of the wheel are that it is wooden, it has a light brown color, and it is rigid. The individual could have also used a metal rim and rubber for constructing the wheel. If the latter had occurred, then the wheel would possess certain properties that the wooden wheel would not. These properties would contribute to the functionality of the cart, i.e., in the performance, effectiveness, and efficiency of said cart.

Let's see how we can use the metaphors above regarding object-oriented programming. It is very natural for human beings to think in terms of objects. According to studies conducted by Jean

Piaget, children begin to distinguish reality as consisting of objects. [1] This is a phase known as the "concrete operational stage." The ancient Greek philosopher Aristotle of Stagira postulated the "class." He used the concept of class to organize the many specimens of organisms he collected. This organizational structure, one in terms of class, lends itself to one of the tenets of object-oriented programming: inheritance. Aristotle took note that certain animals and plants have structures that are similar in form and function, e.g., the hoofed animals he classified in the class belonging to the hoofed animal vs. the animals of which had paws.

In modern object-oriented programming, it is appropriate to use inheritance. This allows for less code use and duplication of code. By defining a class with properties (form), an object may have methods (function) that are shared across the entire class. From this class, the superclass, other classes may inherit these same properties and functions. That is, once an object is created from such a class, it would inherit, for example, the form of a hoofed foot with the function of galloping.

Another tenet of object-oriented programming is polymorphism, which is derived from the Greek words for "many" and "shape." Aristotle may have documented the first classifications of life on Earth, as there are no surviving records from philosophers or natural scientists before him, but he did not distinguish that life forms may change over time depending on changes in the environment. In modern times, biologists have noticed that certain aquatic animals such as whales, dolphins, and manatees use their "arms" as fins, so much so that their arms ARE invariably fins. However, the underlying structures (i.e., the bones) are much closer to the arms of land-dwelling animals than they are to the bone structures of fish fins. Human beings could also use their arms to propel themselves through water. This is an excellent example of how an arm has polymorphic properties.

In object-oriented programming, a method may also be polymorphic in nature. Take, for example, the + symbol. This operative may be used to add numerical values and concatenate two or more string values.

In the study of biology, an organism is encapsulated from the environment. This allows the organism to maintain homeostasis. We only have to look at the cell as the basic building block of all life on Earth. The cell as an object that is encapsulated. Encapsulation is another tenet of

object-oriented programming. The environment in which a cell lives can be very harsh. Therefore, the cell has a cell membrane or cell wall to protect itself from elements in the environment. In multicellular organisms, cells are also differentiated by form and function. This differentiation gives them their utility for the organism. The cell has a function that is peculiar to the role of the organ it belongs to in a multicellular organism, e.g., a human. For this to occur, the cell is encapsulated with a membrane containing specific protein markers and protein gates that "float" on the phospholipid bilayer. Not only does this allow the cell to be recognized by other cells and the immune system, but these protein gates also allow the cells to be selectively permeable to elements within the extracellular space. We can liken this to "data hiding," which is the primary purpose of encapsulation as it pertains to object-oriented programming. Just as a molecular signal from one organ may act on specific individual cells (e.g., the thyroid-stimulating hormone), methods of an object should not alter the behavior of other objects of which they are not privy to. In modern object-oriented programming, we use the access modifier to accomplish this. We can use the private keyword for encapsulating data accordingly.

Object-oriented programming is one of the most popular paradigms of software engineering. Functional programming is another paradigm that is becoming popular. Overall, the use of these models is in assisting human beings in solving real-world problems. Software just as ideas are abstract. It also requires not only problem-solving skills but abstract thinking. It has become popular, almost a necessity in post-industrial societies for incoming individuals to the labor force to know/understand programming. It has become clear that individual members of post-industrial societies be proficient in their technical acumen concerning computer literacy. So much so that there is a push for programming to be taught in all secondary schools as a requirement. Today's leaders in the tech world, such as Mark Zuckerberg and Bill Gates, are the main proponents of these initiatives.

So far, in this essay, I tried to make analogies of object-oriented programming to other subjects such as Philosophy and Biology. For me, this important because I am tutoring my fellow students in their undergraduate studies. Biology is taught at every secondary school level. However, programming is not. Does biological science require some specialty of learning? Does programming need a particular type of learning and thinking that perhaps is not suitable for all students? Then why is Biology appropriate for all students? Biology, just like all sciences

requires critical thinking and problem-solving, also understanding abstract ideas like classes for the organization of life. However, biology is not taught in a vacuum. Today's high school students learn mathematics along with other disciplines. I strongly believe that students will excel in the subject that piques their interest(s). I think for students to excel in programming, they must have an emotional stake in the subject. Perhaps, the best way to grow their interest is through pedology. Indeed, I believe that every individual has a talent and use to society. Programming is a unique subject, and I think it is the accumulation and crystallization of all human pursuits. Programming is a combination of arts and sciences. It borrows something from both pure sciences such as mathematics and from the practical industries of the arts. If a teacher can spark interest in programming in a student whose primary interests lie in automobile mechanics, the teacher can show the student how he may be able to design a better engine by programming a microchip that can enhance the motor's performance. Learning is best done through association, and today, our society—and civilization as a whole—is software-based. I agree with Bill Gates and Mark Zuckerberg's Coding for Everybody initiative. Programming has to be fun; it has to be emotionally rewarding to the brain. I have experienced this reward myself. It is no different than having a eureka moment in physics or finishing a piece of music on a musical instrument. I also believe that exposing younger members of society to programming will also help them think logically. Programming, like mathematics, is a form of thinking. These days, I see young kids programming using simulation software such as Minecraft. Is programming for everyone? Yes, it is.