Joshua Koshy

Professor Ramirez

CS 1501

17th June 2017

Object-Oriented Programming

Object Oriented Programming is modeled after a change in perspective. We originally see programming as a logical procedure, in which we take the inputted data and process it. Rather than working off actions, there was a change in our perception, allowing us to manipulate “objects” instead. The idea behind it was that we are to list the objects within our problem as well as the relationships that define and relate them to one another. We can call this “data modeling”. This leads to the encompassing problem that programming is pushed to solve between these “objects”. Objects can be defined behind a specific set of objects, or what we call a “class” of objects. These classes define the types of data the object contains as well as the logic and structure that we can manipulate behind them. These logical sequences would be known as “methods”. We can break these traits down further and discuss what sets Object-Oriented Programming far from the rest, but first let’s briefly discuss how this type of programming came about in the first place.

SIMULA was the first developed Object-Oriented Programming language. Its goal was to create simulations while allowing non-specialists to be able to handle the language. It was back in mid-1960’s when Alan Kay gave birth to this language in a revision of SIMULA. Though SIMULA was credited for this moment, Alan Kay had borrowed many ideas from the internals of the older LISP programming language. These ideas include the notion of an object even existing within programming, with these objects holding their properties and attributes that define it. SIMULA was also able to borrow the ideas of automatic garbage collection, which helped the building and running of programs become much more efficient as well as allowed these new ideas to become central to the profession. SIMULA, more importantly, was also able to introduce the idea of classes (and subclasses) as well as instances of objects. These records of attributes were able to define objects distinctly in space in order to manipulate their properties through the messages and communication established between them.

With these traits, Object-Oriented Programming gives us programmers the means to accurately represent these worldly concepts and logic within our own space. Since the code is entwined with the object, we can allow the objects to live and exist within their own form as well as interact outside of it. The code and data are not separate, unlike that of other languages. They work together as one object, and act as one. This helps us to understand the idea of what an “object” is in these languages. A class is simply a blueprint of these objects, acting as a category that dictates the concept of the model. It shows exactly what criteria needs to be met and passed down to all objects within the class. Yet, it still allows for each object within the class to have it’s own unique methods and attributes.

Inheritance is another relationship that allows these objects to exist in different layers, with some encompassing the other. With the idea of subclasses, the parent class can allow behaviors to be passed down. And these subclasses can break down even further, with even more complex behaviors to be inherited. Exploring these attributes, we can also see how abstraction and encapsulation can help to house data in a unique manner. Objects can be told to only reveal whatever is necessary to the layers above and below. This allows communication to be very complex, with certain attributes not needing to be shown, for either security or simplicity’s sake. It allows for an object to operate cleanly and soundly.

More of the famous Object-Oriented programming languages include Java, Python, Ruby, C++, Visual Basic .NET, and more recently Objective-C. With this class, we’ve been working a lot on our assignments not only for this course, but many of our core courses here at the University of Pittsburgh with Java. We’ll see that many of these languages exist popularly as a backend-programming language because of structure and reliability Object-Oriented programming languages offer without sacrificing any of its speed or efficiency. Java exists mainly as a server-side programming language that is ideal for highly-viewed sites that experience a lot of traffic. It’s popular amongst enterprise-level applications and is a culmination of many different tools, labeled as “The Java Platform”. With a wide variety of open-source development tools and environments ready for programmers, it has still could keep up with competing languages with its speed and scalability. Though Java is built in many different tiers and updates, it remains easy to maintain and consistent in future versions. For large-scale websites, it’s able to keep business goals up with its reliable speed as well as develop beautiful applications in the process. The Java Virtual Machine can convert Java source code to machine code, making Java readable on whatever platform that is needed. Many other languages instead need the compiler to do that work for them. With it’s relative ease to learn as well as the giant ecosystem of developers, libraries, and frameworks, and APIs, Java has become a hotspot for programmers to delve into Computer Science and operate fully in a 100% object-oriented language. It’s also reliable due to it’s compatibility with many different platforms, including Android phones. Code can be written once and usually trusted to run anywhere the developer chooses. Java remains a universal with it’s large ecosystem expanding around the world. And with Object-Oriented Programming’s structure, programmers can stay at ease with how reliable, efficient, and comprehensible the work is.

Python is also widely used, and is a general back-end programming language that is highly valued by data-driven companies who need to analyze data and apply statistical technique into their business. It’s flexibility as well as it’s readability has pushed in into being very reliable. It can rival Java’s suitability for creating high-traffic web applications while retaining it’s focus on complex statistical functions. It’s powerful set of libraries allow for data manipulation, visualization, mining without many rewritten lines of code. It’s also known to operate on mostly any operating system without fail.

Ruby is similar to both Java and Python for it’s back-end help in projects, but it has become certainly trendy in the past couple of years, especially in the form of Ruby On Rails. It has expressed the idea that programmers should “do more with less” and has designed the language around that notion. With many “gems” packaged into various libraries, Ruby has become a huge competitor and has become a very dynamic, unique language, excelling in a full-stack framework. While remaining short and concise, it still remains scalable, like Python, and is able to remain very convenient for programmers.

C++ is a huge language that cannot be missed on this list, as it’s commonly known as the “grandfather”. It still remains vastly popular, especially in Microsoft’s .NET Framework as it’s very reliable speed keeps it productive. It’s key difference lies in the concept of “pointers”, which remains unique to C++. It allows for the developer to simplify their code further by gaining the location of the variable’s address space. This gives the developer further control over the variable rather than leaving it all to the compiler to handle. C++ allows for complex applications to run at very fast speeds. Though it’s compatibility is limited, it remains very popular within enterprises and databases for use.

In comparison to Object-Oriented programming languages, there exist another popular category of languages that rival the structure we’ve seen so far. They’re known as Procedural languages. C is a very popular example used when this category is mentioned. These languages allow the code to interact with data, but keeps them both separate from each other. It’s a much more linear system, where the procedures are simply bits of input and output. Though data and code isn’t bundled together like in Object-Oriented programming languages, we don’t have the need for global, shared data. Procedural languages, due to its structure, allows for some ease and transparency. However, it can also lead to difficulties. As programmer’s approach operating Procedural languages from a Top-Down approach, with each problem being broken down further and further down, these very problems suffer when the bigger problems face software maintenance. When one problem arises at the top of the structure, the bottom needs to all be tweaked and updated in order for the structure to survive. And so, the upkeep is a bit rough. Not to mention, when it comes to reusing components within software, Procedural languages are a lot more complex to operate when Object-Oriented languages allow for multiple objects to exist within the same layer.

Object-Oriented Programming is a very efficient and reliable structure that developers will continuously look back towards. It’s features and specifications make it clearly stand out as an intuitive and fundamental building block within our development of more complex problems. It’s structure allows us to scale to bigger and bigger forms of production, without suffering from any complexity (as long as the project is maintained properly).