Object Oriented Programming

Object oriented or OO programming is a paradigm of software programming developed in the early and mid 1990s, which places the emphasis on data and data types before the functions and methods that are applied to that data [1]. The more natural way to programme for humans is to use procedural or functional programming where the ideology places emphasis on the methods and functions first and foremost, and the code is written as a list of instructions that the computer must execute in order [2]. OO programming has advanced software reliability and reuse, often allowing the software to be better suited to our understanding of the real world, which the programmes may be reacting to, or trying to control [2].

Many of the most widely used programming languages support OO programming, to a greater or lesser extent, typically in combination with procedural programming [3]. Such examples include C++, C#, Java, Python, PHP, Ruby, Perl, Objective-C, and Swift [4]. Generally, all OO programming language use abstraction, encapsulation, inheritance, and polymorphism to provide greater flexibility, modularity, and reusability in developing software [5]. The philosophy behind and rationale for using OO programming as well as the advantages and disadvantages of using such a programming paradigm will be discussed further in this essay.

Before we discuss the details of OO programming, we must first identify what objects actually are. Objects in the programming sense are generally objects in real life, and very often software engineers design their programmes using a noun analysis to decide on the classes that must be included in the software [6,7]. In addition, a verb analysis allows engineers to decide on the methods that must be written to act upon the data within classes [7]. In the more abstract sense, a class is a blueprint of elements that are required to make up an object, which ultimately equates to instance variables for what all objects or lines of data within that particular class must possess [4].

The overarching theory behind OO programming is that procedures and data structures are bundled together so that an object, which is an instance of a class, operates on its own data structure [5]. In addition, each class and object is encapsulated so that other classes and objects cannot get access to data stored in private variables within another object, without using so-called getter and setter methods [4]. However, by using these methods, data stored in private field variables within a class can be seen and changed, putting into question the very concept of encapsulation [1]. Nonetheless, data must be shared between classes, and encapsulation allows the software developer to better control who can get access to data outside of classes in an ordered and predetermined way.

Another significant feature of OO programming is Inheritance, which is a formal relationship between classes that allows methods and class constructors to be passed from a parent class to a child class, meaning that code can be reused without having to re-type it [3]. This has the advantage that if that code needs to be modified, it only has to be modified once in one location, and the effect is seen in the child classes without having to do anything further [7]. Not all OO languages allow multiple inheritance patterns such as C# and Java [1], which ultimately means that classes must be carefully arranged in a tree-like diagram for inheritance to work correctly, and sometimes code must be re-typed if a particular pattern of inheritance cannot be achieved [6].

Numerous OO programming languages have interface features that are an abstract system, which provide a contract between the interface and the classes that implement the interface, making sure that all of the classes implement all the methods given within that interface [4]. The major benefit of using interfaces is that they can simulate multiple inheritance since classes can implement multiple interfaces [5]. As a continuation of inheritance and interfaces, polymorphism, meaning taking many forms, means that the execution of a method of a given name may be different depending on the class of the object for which the method is invoked [2]. This allows the software developer to programme more generally than specifically. In particular, polymorphism enables code that process objects that share the same superclass,

either directly or indirectly, as if they were all objects of the superclass, which can simplify the software programme greatly [4].

Using Java as an example, all methods must be associated with a class [3]. They can either act upon the data stored in that class by acting on each instance of a class, or function as static methods, in which each instance of a class is not used as input into the method [6]. This is not true for all OO languages, since Python for example allows users to write methods that are not associated within a class, which is essentially equivalent to a static method [1]. This static method concept allows the programmer to create procedural or functional-like code within a predominantly object-oriented language [5].

The greatest advantage of OO programming is the in-built modularity of the paradigm, which allows multiple software developers to divide and conquer the project by being able to work on different aspects of the same project at the same time [7]. In addition, it also means that large sections of code can be re-used in other projects without too much effort [5]. This benefit also extends to maintaining and creating future updates for the software, where in theory only the sections that need to be updated need to be modified, and all of the other sections will be unchanged [1]. However, this requires an extensive planning phase so that the software is designed extremely well from the outset, and before any code is actually written, otherwise this modulatory is not useful and programmes can end up with immensely tangled code or so-called spaghetti code very easily [7].

The major disadvantage with OO programming is the initial outlay in the design phase of the project. Each class and object must be clearly defined and the inheritance pattern must be worked out before any coding can begin [7]. In addition, because every class must be its only separate file, the programme size and complexity can become very large for even minor projects, and must be carefully managed for major projects [2]. Furthermore, for people who are new to OO programming there can be a steep learning curve at the beginning to conceptualize the idea of objects, classes, inheritance, and interfaces, and how this is exactly translated into practice [1].

The most criticised feature of OO programming is the speed in which the programmes load and run [1]. This depends of course on the hardware that is running the software, as well and the OO language in question, but by virtue of the intrinsic nature of OO programmes, including the interpretation or compiling into bytecode, OO programmes are generally slower than procedural or functional programmes [2]. Furthermore, software programmes written in OO languages generally have more lines of code compared to procedural and functional languages, where all of these instructions need to be interpreted, compiled, and run [3]. Therefore, software developers must make an important decision whether efficiency is an important factor of the function of their programme [7].

As an alternative to OO programming, functional and procedural programming is often used in situations that rely heavily on mathematical calculations that aim to model mathematical functions rather than object bases situations [2]. In an attempt to make Java a more general language, lambda expressions or functions were added to version 1.8 of Java in 2014 to allow a more functional style of programming to be achieved in an OO language [4].

The choice of programming paradigm and language greatly depends on the task that must be tackled by the software. Some projects lend themselves more toward OO programming techniques, whereas other and best attempted using functional or procedural programming [1]. Forcing the application of an OO programming language onto a situation that should be coded in a procedural or functional style will result in very inefficient programmes [2]. Therefore, no one language or paradigm is suitable for all situations and careful considerations about the ultimate function of the programme must be addressed very early in the planning phase. However, OO programming languages has developed over the past 25 years to be more general purpose, allowing the languages to be used in more varied situations, which has ultimately led to its popularity and as the programming paradigm of choice for many people.

References

[1] Budd T. An Introduction to Object-Oriented Programming. 3rd edition. Cambridge: Pearson; 2001.
[2] Reynolds C, Tymann P. Schaum's Outline of Principles of Computer Science (Schaum's Outline Series). New York, NY: McGraw-Hill Education; 2008.
[3] Nair PS. Java programming fundamentals: problem solving through object oriented analysis and design. Boca Raton, FL: CRC Press; 2008.
[4] Deitel P, Deitel H. Java How to Program (Early Objects). 10th edition. Cambridge: Pearson; 2014.
[5] Liang YD. Introduction to Java Programming, Comprehensive Version. 10th edition. Cambridge: Pearson; 2013.
[6] Eckel B. Thinking in Java. 4th edition. Upper Saddle River, NJ: Prentice Hall; 2006.
[7] Sommerville I. Software Engineering. 10th edition. Cambridge: Pearson; 2015.