Object-oriented programming (OOP) is a programming paradigm that began in the 1960s and is based on the concepts of classes and objects. A class serves as a blueprint for a given data type and an object is an instance of a particular class. Classes often encapsulate data, called fields, operations to manipulate the fields, called methods, and sometimes even other classes, called inner classes. This class structure provides degrees of abstraction, which means that users of a class don’t necessarily need to know how a class is implemented in order to use it [1]. Abstraction is thus a key concept of many of the features of OOP and makes it a key distinction from procedural programming, another programming paradigm. In procedural programming, data and operations to manipulate data are considered separate entities [2]. From the first incarnation of OOP, Simula 67, to more modern variants such as Java and Python, OOP continues to play an important role in shaping the way that programs are written and the ways that programmers think about solving problems using software.

OOP has its roots in the programming language Simula 67 – a language developed by Kristen Nygaard and Ole-Johan Dahl in 1967 and based on ALGOL 60 and Simula I. Simula 67 was the first language to implement co-routines using a data structure called a class. Classes serve as one of the primary forms of data abstraction and are a primary feature in OOP. In addition, Simula 67 introduced the ideas of objects (instances of a given class), inheritance (when a subclass “inherits” the attributes of its parent class), and static binding (disambiguating identifiers at compile time) [3].

One of the next big advancements was with Smalltalk, a language developed by Alan Kay, and later Adele Goldberg, at Xerox PARC from 1969-1972. Smalltalk was the first full implementation of an OOP language and included data abstraction, inheritance, and dynamic type binding. It also pioneered the idea of using a graphical user interface (GUI) as a primary way of interacting with computers [4].

From the mid-1970s to the 1990s there was a proliferation of OOP languages. Some notable languages were Ada (U.S. Department of Defense, 1975-1983), C++ (Bjarne Stroustrup at Bell Labs, 1980-1983), Objective-C (Tom Love and Brad Cox, 1984), Python (Guido van Rossum, 1990), Java (James Gosling at Sun Microsystems, 1995), and Ruby (Yukihiro Matsumoto, 1996). Each of these languages helped to increase awareness of the OOP paradigm, increase use of its concepts, and refine the primary features of what constitutes OOP today [4]. As these and other languages evolved over the past fifty-one years, so have the primary features of object-oriented programming.

Most modern OOP languages have many features in common with non-OOP languages. Some of these common features include: variables, primitive data types (such as integer and character types), arrays, references, control statements, recursion, and abstract data types. However, OOP also consists of several unique concepts (not all of which are implemented in every OOP language) which will be discussed here in more detail. Some of these features are: classes, objects, fields, methods, encapsulation, abstraction, access control/restrictions, modularity, reusability, composition, inheritance, polymorphism, and interfaces [1].

Probably the two most important features are encapsulation and abstraction. As previously explained, encapsulation is the idea of combining data and operations which apply to a set of data together into a new data type called a class. Classes and their objects form the first level of hiding, or abstraction, that OOP languages provide. Abstraction in terms of OOP means hiding data and or other implementation details inside of a given class from users of the class. This hiding can be done by only providing a class file and not the given source code (hiding implementation), or by hiding certain fields, methods, and/or inner-classes from outside classes (and sometimes subclasses) using access control. In OOP, access control has 3 main levels: public (meaning a field or method can be accessed inside and outside of the given class), protected (meaning it can be accessed by other classes within the same package or by subclasses of that class), and private (meaning that it’s only accessible within that class) [1].

The other two main concepts in OOP are the ideas of modularity and reusability of code. Modularity in general is the concept of creating modules (objects) and then combining those modules together to form a new program. Objects by their nature are modular and enable OO programs to reuse pre-written classes in new programs without having to reimplement those data types (classes) or know the implementation details (abstraction) as long as proper documentation for use is provided with a given class. While entire classes can be reused in this way, OOP also allows for the reuse of parts of a class through the concepts of composition, inheritance, and polymorphism [1].

Composition is using objects of other predefined classes in order to create a different functionality in a new class. This type of class creation means that the new class “has a” relationship to the predefined class, but cannot directly access private or protected members of the predefined class. Think of it like adding different ingredients together to make a new type of ice cream. The new ice cream has the flavors of the ingredients, but it doesn’t share anything else in common with them. Inheritance on the other hand allows for the creation of subclasses and for the implementation of interfaces. Subclasses are created by extension of a predefined class (referred to in this relationship as a superclass). This means that a subclass can add new functionality that the superclass didn’t have, but also use the public and protected methods and fields of the superclass without having to implement them directly (again providing abstraction). In this type of class creation, the new subclass “is a” specific type of the superclass. A concrete example of inheritance would be the definition of a car and its relationship to other vehicles. A car is a subclass of vehicles, and shares things in common (e.g. wheels) with other vehicles. A car is a vehicle, but not all vehicles are cars. The relationship is one way [1]. A second type of superclass and subclass relationship is that between an interface class and a subclass of an interface. Interfaces are classes that consist of method headers (and sometimes fields) and are used to give a consistent way of accessing collections of mixed data types. The interface does this by forcing its subclasses to implement its method headers. These subclasses of an interface utilize the concept of polymorphism (many forms). Polymorphism in a superclass/subclass setting means overriding a superclass’s method (or method header) with a new one. This concept of overriding can be done with any type of subclass, not just those of interfaces, and relies on dynamic (runtime) binding. The other meaning for polymorphism is having multiple methods with the same name, but that have different method signatures (number and or type of parameters) in the same class [1].

In order to see how OOP languages can differ in their implementations, it is helpful to consider four main properties of a language: its readability, writability, reliability, and cost (usually in terms of time) [4]. Two very popular OOP languages which can be easily compared are C++ and Java [5]. Both languages are imperative in nature and in fact, Java is based on C++.

The first difference between the two is in terms of readability. While both languages share the common ancestor C, C++ is a very large and complex language in part because it has components of OOP and procedural programming. Java on the other hand is significantly simplified (e.g. it removed structures, unions, and pointers) which helped to improve its readability and make the language simpler overall. The next difference is in terms of writability. While both languages allow for various levels of abstraction, because Java is a simpler language it doesn’t allow for as much expressivity and has a lower writability than C++. Java’s biggest benefit though is in terms of reliability - the language was designed to eliminate some of the unsafe features of C++ (e.g. pointers and array bounds checking). The trade off with enforced safety checks in Java is that Java programs run slightly slower and have a higher cost (CPU time) compared to C++ programs [4].

As with natural languages, OOP languages will continue to grow and adapt with time. Some may fall out of favor and new ones may come along to meet some need that no existing languages adequately fulfill. As the brief history of OOP over the past fifty-one years has shown, the languages that have endured the longest are those that were able to make changes that improved reliability and decreased costs.

# Bibliography

|  |  |
| --- | --- |
| [1] | J. C. Ramirez, "CS 0445 - Data Structures [PowerPoint slides]," University of Pittsburgh, 2018. [Online]. Available: http://people.cs.pitt.edu/~ramirez/cs445/. [Accessed April 2018]. |
| [2] | T. Gaddis, Starting out with Java: From Control Structures through Objects, 6th edition, ISBN: 978-0-13-395705-1, Boston, Massachusetts: Pearson Education, 2016, pp. 19-21. |
| [3] | O.-J. Dahl, "The Roots of Object Orientation: The Simula Language," in *Software Pioneers: Contributions to Software Engineering, ISBN 978-3540430810*, M. Broy and E. Denert, Eds., Heidelberg, Springer, Berlin, Heidelberg, 2002, pp. 78-90. |
| [4] | R. W. Sebesta, Concepts of Programming Languages, 11th edition, Global edition, ISBN 978-1-292-10055-5, Harlow, Essex: Pearson Education UK, 2016, pp. 94-114. |
| [5] | TIOBE, "TIOBE Index for October 2018," TIOBE, October 2018. [Online]. Available: https://www.tiobe.com/tiobe-index/. [Accessed 19 October 2018]. |