Object-Oriented Programming

William Kurek

*Department of Computer Science*

*The University of Pittsburgh*

# Abstract

Since it was first introduced by Simula in the 1960s, Object-oriented programming has become a staple among programming paradigms. With the direct support of classes, encapsulation, polymorphism, and inheritance through the concept of objects, Object-oriented programming languages (OOPLs) have a wide variety of implementations. Therefore, it has been widely accepted since its debut. This paper begins with a brief history of OOPLs, followed by a detailed description of its primary features with a focus on polymorphism and inheritance. The paper will then explore some of the most common and widely used OOPLs, including Java and C++, and the key differences between them.

# Introduction

In the early days of commuting, programs became very difficult to maintain as they grew in complexity and size. Due to dependencies and the separation of data and functionality, a minor alteration at any point in an application would cause a dispersion of errors throughout the entire program, potentially sabotaging an entire software system. Object-oriented programming is a programming paradigm that was introduced to deal with this prominent issue. The paradigm presented mechanisms which minimized dependencies by promoting modular design and reusability. Specifically, it combined both data and its respective functionality into a single object, which could be changed and manipulated without disrupting the entire program.

# Key Features

While the universal definition of Object-oriented programming is still widely debated, there are key concepts which all OOPLs implement and they are defined as follows:

## Objects are the rudimentary units of OOPLS and they represent any entity that a program must handle. These entities prominently replicate real world objects including persons, places, or things. During execution, objects interact with one another to complete the task at hand, while maintaining their own address and space in memory. Objects are a type of variable known as class.

## Classes group together objects with shared attributes. This grouping of objects binds the data and its respective functions together. Once a class has been created, any number of objects belonging to that class can be defined. While classes are user-defined data types, their behavior is identical to that of built-in data types.

## Encapsulation is the idea of wrapping of data and functionality into a single entity is called encapsulation. Data stored in a class is not accessible by outside functions. Data is only accessible to the functions defined within the same class. The restriction of data access hides data implementation details from user.

**Data Abstraction** is the concept of hiding implementation and background details from the outside world. In other words, only the information essential to the user, without any other details, is provided.

**Inheritance** is the concept of reusing, or inheriting, a previously defined class to create a new class with identical states and behaviors. Because it promotes code reusability, inheritance is one of Object-oriented programming’s most beneficial attributes.

**Polymorphism** is the process of forming a single function that can be utilized by multiple different argument types, each functioning in a different way. In other words, it is the capability of different objects to use the same name.

# Brief History

The first programming language to utilize objects was Simula 67. Simula 67, developed by Kristen Nygaard and Ole-Johan Dahl in Norway, was a programming language intended for creating simulations. While working on simulations that dealt with ships, Nygaard and Dahl discovered they could categorize ships into different groups, each having its own class which produced unique data and behavior. Simula 67 was a revolutionary creation and was an inspiration for an abundance of programming languages. Then, Xerox PARC, in their programming language Smalltalk, introduced the term “object oriented programming” to the world of commuting in the 1970s. Inspired by Simula 67, Smalltalk utilized objects as the basis for computation, but with an emphasis on dynamics. In other words, unlike other frequently used languages in that time, objects could be created, mutated, or deleted. The concepts of inheritance and object messaging were also introduced by Smalltalk. Soon after, even existing languages began to integrate the object-oriented programming paradigm. Specifically, Bjorn Stroustrup modified the popular language C to incorporate object-oriented concepts. This resulted in the creation of the C++ language, which was the first OOPL to see wide commercial use. Overall, the groundbreaking creation of Simula, and its’ influence on Smalltalk, paved the way for creation of essential, modern OOPLs. These languages include Java, Python, Ruby, Eiffel, and C++.

**Comparing Languages**

There are many programming languages today which implement the object-oriented paradigm. Although all OOPLs demonstrate object-oriented concepts, they are still quite different in numerous ways. A comparison of two of the most common OOPLs today is presented below.

**C++**

As stated previously, C++ is a version of C which implements statically-typed, object-oriented elements. Because C was designed primarily for systems programming, C++ allows low level hardware access and direct access to memory. Consequently, C++ pointers can be used modify explicit memory locations. While C++ provides multiple inheritance, generic programming, and exception handling, it does not provide garbage collection. Rather, C++ supports destructors which are automatically summoned when an object is erased. Further, in C++, the programmer must specify the explicit class of an object. This is known as compile-time binding, and it ensures optimal runtime efficiency. But, compile-time binding also reduces class reusability. Also, C++ allows both method and operator overloading.

**Java**

Java is an OOPL, introduced by Sun Microsystems, designed to minimize implementation dependencies. It is a statically-typed OOPL, with syntax like C++. But, unlike C++, Java does not implement low-level programming concepts such as pointer arithmetic. Also, Java runs on the java virtual machine, further making it safe and portable. As a result, Java ensures type security by having garbage collection, which allows the relocation of referenced objects. Furthermore, Java is extremely portable because of its ability to be executed on any given computer, regardless of the operating system. But, a trade-off of portability is an increase in runtime, compared to that of native applications. Next, Java provides both single and multiple inheritance. Multiple inheritance can be achieved using interfaces. Also, Java allows method overloading, but does not enable operator overloading.

**References**

* Weisfeld, Matt. "The Evolution of Object-Oriented Languages." *The Evolution of Object-Oriented Languages - Developer.com*. N.p., 31 Mar. 2005. Web. 17 June 2017.
* Wodehouse, Carey. "What Is Object-Oriented Programming?" Hiring | Upwork. N.p., 15 Jan. 2017. Web. 17 June 2017.