Inheritance Throughout Computer Science

Nathan Blankenship

Student, Montana Tech

11/1/2021

Computer Science is an incredibly diverse field, packed to the brim with different programming languages, styles, and workflows. These unique programming elements are often further differentiated by being specifically tailored to a certain problem (or family of problems) in the workforce. It may seem surprising then to find that there are many principles and basic ideals of programming that are considered ubiquitous; constructs so useful that they have become an essential part of programming as a whole. Perhaps one of the most recognized constructs - and one of the most widely used - is Inheritance.

Within computer science, Inheritance is defined as: "Basing an object or class upon another object or class, retaining similar implementation". For members of the Computer Science field, this jargon makes good sense, but for anyone who isn't familiar with programming, the above definition would likely mean very little. Therefore, a deeper dive into the history, meaning, and functionality of Inheritance is necessary.

The fundamental unit of Inheritance is the Class. Classes are data structures within Imperative programming that hold attributes and methods that are unique to the class, but shared by any instantiation(s) of the class, which are called objects. These classes and their relationships are the cornerstone of the aptly named Object-Oriented Programming, a subset of tools and programming languages including C++, Python, and Java. Formally, Object-Oriented Programming is defined as: "a computer programming model that organizes software design around data, or objects, rather than functions and logic" (Gillis, A. S., & Lewis, S. 2021). In summary, Inheritance is a facet of Object-Oriented Programming that relates classes to one another.

A logical next question to ask is 'how does Inheritance relate classes in practice?' Inheritance within Object-Oriented languages takes similar (but never identical) approaches:

classes may be created (on their own OR) by referencing another class/classes in order to *inherit* attributes and methods -that is to say, one class will have (often limited) access to the data and functions within another class. Different texts, languages, and professionals define this relationship as either a parent-child class relation or a superclass-subclass relation. This paper will use parent and child to refer to the class being inherited from, and the class inheriting, respectively.

The history of Inheritance is uniquely tied to one man: Alan Kay (1940-). Kay is remembered (along with myriad other achievements) as the father of the term “Object-Oriented Programming”, first using it while attending graduate school at the University of Utah. This term would go on to encapsulate one of the largest fields of programming languages, including those that take advantage of Inheritance. In other words, Inheritance is a *child* of Object-Oriented Programming, to use an appropriate term.

Kay then joined the team that would write Smalltalk, one of the first languages considered to be in the Object-Oriented family today (legitimately everything in Smalltalk is an object). However, Kay was not a part of the team that would eventually introduce Inheritance as we know it today to Smalltalk.

Interestingly, Alan Kay somewhat regrets the way Object Oriented Programming has evolved through the years, saying “Object-Oriented Programming to me means only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things.” (Elliott, 2019). To Kay, the specific nature of Object-Oriented Programming in modern languages has become too restrictive upon the versatility of his idea. By Kay’s definition, Inheritance is a non-essential part of Object-Oriented Programming, as long as objects can communicate, encapsulate and retain data, Object Oriented Programming has been achieved.

However, given the wealth of languages that use Inheritance, and the similarities between these languages' implementations of Inheritance, it is still very much worth discussing and learning.

Depending on the language, inheritance can take several forms. Described above is Single Inheritance, wherein one child adopts the characteristics of one parent. There are several kinds of inheritance -each suited for different situations- throughout Object-Oriented languages.

Python and C++ for example, feature the same set of 5 main Inheritance categories (formal definitions from Singh, 2020):

1. Single Inheritance, as described above
2. Multiple Inheritance, wherein "A class inherits more than one parent class"
3. Multi-Level Inheritance: a class which is itself a child can become a parent (multiple 'generations' of inheritance are allowed)
4. Hierarchical Inheritance: One parent class is allowed to have multiple children
5. Hybrid Inheritance: A combination of the above types of inheritance; a multi-level child can still be (one of the) parent(s) of another new class, for example.

NOTE: Hierarchical inheritance seems to sometimes be considered a subset of multiple Inheritance. That is to say, some sites claim that multiple inheritance encapsulates a multi-parent multi-child inheritance environment, and Hierarchical inheritance is specifically the ability of a parent class to have multiple children. Other places define it exactly as above.

Multi-Level inheritance is the logical next step to take from single (also frequently referred to online as simple) inheritance. After all, if parents can exist, grandparents should be able to as well, by simple logic. Of course, computers can be somewhat more picky, so the presence of simple inheritance does not imply the presence of multi-level inheritance in any language.

Multi-level inheritance adds complexity and flexibility, so it follows naturally that it also adds the need for more care in coding. Particularly, when a method is called within a class, if that class has parent(s) with the same function signature, things get interesting. If the child class has a method with that name, the child's version of the method will run. Otherwise, each parent is checked going 'up', until a method with the right signature is found (THE FIRST TIME).

Some languages allow you to specify where to look in ambiguous cases like this. Python, for example, has the super() function, which will designate only the first-level parent of a given class. For most applications, however, it is most useful to remember that this behavior exists, and try to write code that avoids it altogether.

From multi-level inheritance, the next addition to Inheritance's toolset is multiple inheritance. Multiple inheritance allows one parent class to give attributes and methods to multiple children, hence the name.

There seems to be quite a lot of contention, confusion, or perhaps just differing definitions regarding multiple inheritance online. Particularly, the distinction between multiple and hierarchical inheritance. The initial definition of inheritance types used in this paper defines multiple inheritance as one *parent's* ability to have several *children*. The same source goes on to define hierarchical inheritance as one *child* with the ability to inherit from more than one *parent*.

Many other sources have muddied this definition by grouping both behaviors under multiple inheritance, however. Indeed, multiple inheritance seems to have become the umbrella under which all of this behavior is often defined, but only when taken in the general sense. The only truly common thread among all the language-specific articles and lessons used to compile this paper is the phrase ‘multiple inheritance’, so this paper will consider hierarchical inheritance to be a branch of multiple inheritance (unless otherwise specified, which seems to be frequently), to avoid any confusion brought on by conflicting definitions.

The last major category of inheritance is not in itself a new category, but rather a flexible combination of previous types of inheritance, known as hybrid inheritance. It is important to specify that hybrid inheritance allows the combination of simple, multiple, and multi-level inheritance, because a given programming language may have one or more of these types of inheritance *without* the ability to combine them. Hybrid inheritance, then, is only an increase in the applicability of previously mentioned methods, rather than being a new method in and of itself. For example, a child class with two parents could become (one of) the parent(s) of another class later, without issue.

One thing that has become abundantly clear in researching inheritance is that, like many constructs in computer science, its applications vary wildly between languages. While an exhaustive search of all languages that contain Object-Oriented Programming would be a herculean feat, this paper seeks to at least answer a few of the general questions one might have if one wanted to try inheritance in a modern-day programming language. As so many have done before, this paper will begin by looking at Java.

Java is a programming language that features more than one type of Inheritance, boasting single, Multi-Level, and Hierarchical Inheritance(one parent can host multiple children). Notably, however, Multiple Inheritance is not allowed in Java (at least not directly through the class keyword). This also excludes the direct use of Hybrid Inheritance in Java.

NOTE: Java does specify the difference between their definitions of multiple and hierarchical inheritance clearly, and in line with the initial definition given in this paper. To avoid loss of information I have retained these. In the slideshow accompanying this paper, it seemed unwise to introduce contradictions like this at all, so multiple inheritance is used to encapsulate both.

Inheritance is very easy to see in Java, thanks to the use of the extends keyword. For example, if one wanted to allow class B access to class A's methods and attributes in Java, they would type '**public class A extends B {** ', for example. This implementation of inheritance is good for beginners because of this visual flag within code (the extends keyword will always and only be related to inheritance).

Another useful modern interpretation of inheritance exists within the C++ language. C++ boasts all the same inheritance types as Java, along with this paper's definition of multiple inheritance (both parents and children can have multiple links), and can therefore also support hybrid inheritance. Syntactically, if a programmer wanted to encapsulate simple inheritance by passing attributes and methods from class A to B, they would write ' **class B : public A {** ', for example. Here it becomes more clear why the 'extends' keyword in Java was favorable for its syntax; C++ inheritance is much easier to miss when re-reading code. That said, the increased usability in C++ (hybrid and multiple inheritance) provides an argument in its favor.

The last language to be examined in this paper will be Python, whose inheritance is just as robust as C++, that is to say, both languages support the basic types of inheritance, along with a hybrid capability. Python is, unsurprisingly, the simplest of these languages syntactically to achieve inheritance, it only uses parenthesis. For example, class B as a child of class A in python looks like ' **class B(A):** '.

To summarize, Object-Oriented Programming is a huge and incredibly complex facet of programming that has become the go-to programming style for much of the planet. Inheritance exists as an afterthought to Object-Oriented Programming, and even still, is quite complex and nuanced across many languages and interpretations. Hopefully, in any case, this paper provides the reader with a window into, and some understanding of, inheritance as a concept and as a tool when programming.

Works Cited

Inheritance and Polymorphism. Isaac Computer Science. (n.d.). Retrieved November 1, 2021, from https://isaaccomputerscience.org/concepts/prog_oop_inheritance_polymorphism?examBoard=all&stage=all.

Singh, H. (2020, December 29). *Inheritance in object oriented programming: Inheritance in oops*. Analytics Vidhya. Retrieved November 1, 2021, from https://www.analyticsvidhya.com/blog/2020/10/inheritance-object-oriented-programming/.

Elliott, E. (2019, May 13). The Forgotten History of Oop. Medium. Retrieved October 20, 2021, from https://medium.com/javascript-scene/the-forgotten-history-of-oop-88d71b9b2d9f

Gillis, A. S., & Lewis, S. (2021, July 13). *What is object-oriented programming (OOP)?* SearchAppArchitecture. Retrieved November 1, 2021, from https://searchapparchitecture.techtarget.com/definition/object-oriented-programming-OOP.

C++ Inheritacne-- Healthy Addiction Retrieved October 20, 2021, from https://www.pinterest.com/pin/607986018422022272/.

Inheritance and Polymorphism. Isaac Computer Science. (n.d.). Retrieved November 1, 2021, from https://isaaccomputerscience.org/concepts/prog_oop_inheritance_polymorphism?examBoard=all&stage=all

Maia, L. (2020, December 3). Python Tips Series: Multiple Inheritance. Retrieved November 13, 2021, from https://www.maiango.com/post/python-tips-series-multiple-inheritance

Inheritance in C++. GeeksforGeeks. (2021, June 1). Retrieved November 13, 2021, from https://www.geeksforgeeks.org/inheritance-in-c/.