Basic Concepts of Object-Oriented Programming

Elom Kponmassi

January 6, 2022

Frostburg State University

Executive Summary

This document describes and explains some of the basic concepts of object-oriented programing. It covers their definitions, uses, and makes them easier to understand for less experienced readers by using real world examples. The report starts with an over-all definition of object-oriented programing, classes, and objects. It then elaborates on the more complex concepts of encapsulation, abstraction, inheritance, and polymorphism. It finally ends by touching upon data structures, what they are, their uses and some examples of them.

Table of Contents

Introduction

Object Oriented Programming, also know as OOP, is one of the most used programming paradigms or types. As the name implies, this type of programming focuses on using the concepts of objects and classes to build comprehensible software. “It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects” (Doherty, 2020). These blueprints function like molds that you predefine, so that you only need to put in the wanted material to get the result that you want. It also makes it easier in the future if you want to reuse the mold with other materials. Object Oriented Programming is widely popular because it is fast, easy to execute, and creates applications that can be reused, thus saving on development time (W3Schools).

Classes and Objects

In OOP, a class can be understood as a concept. For example, when you think of a smartphone, you have this image or concept of a smartphone in your mind, of what a smartphone is. In OOP, that concept is a class. It defines what something is. Now when you hold your phone in your hand, you’re holding an actual phone that is an object. An object is an instance of a class. In programming, you can create the class “Phone” that is the concept or blueprint of a phone. It defines what a phone is, and what properties every phone has, like a

brand, model, and year it was released. “A class defines the attributes and behaviors that all objects created with this class will possess.” (Weisfeld, 2019, p. 16). After creating the class, you can then create multiple “Phone” objects which would all have their own different brand, model, and year. For example, you could have a “Phone” object that has brand: Samsung, model: Galaxy S21, year: 2021 and another “Phone” object that has brand: Apple, model: iPhone 13 Pro, year: 2021. The two are different objects (instances) of the same class “Phone”. Classes and objects also contain functions or methods. “A function is a block of code that performs a task” (Seaton, 2020). In our phone example, that could be turning on the phone or making a call. These are functions that every phone has, so they would be included in the class. So, every object of that class would have these functions.

Encapsulation

One of the four Principles of OOP is Encapsulation. “The notion of encapsulation […] refers to the bundling of data, along with the methods that operate on that data, into a single unit” (Sumo Logic, What is Encapsulation?). In most cases, that unit is a class. The data being the attributes (brand and model from our phone example) and the methods being the functions. Encapsulation also means keeping data away from users’ access. For security reasons, it is best not to let users access or change some information. Things like social security or credit card numbers are best-left unknow by the general public. This kind of data are usually made private so only the system can access them, and that concept is called encapsulation

Abstraction

The second Principle of OOP that we will discuss is Abstraction. The main goal of abstraction is to make programs less complex by hiding unnecessary details from the user (Janssen, 2017). It makes a program better to use if users don't know what is going on in the background. Abstraction means making it easier for users by not bombarding them with information they do not need to know about. For example, when a user pushes the power button, all they see is the phone or machine turning on. They don't need to know about the interactions that are going on inside the machine or the code that was written to make it all happen.

Inheritance

Another Principle of OOP is Inheritance. The role of inheritance is to create class hierarchies, where classes and objects inherit properties and functions from their parent or super class (Koenig-Bautista, 2019). Sticking to the phone example, if we have the class "Phone", we can also make another class "Android" and make it a subclass of the "Phone" class. The "Phone" class would then be the parent of the "Android" class and the "Android" class would be called its child class. And as a subclass, "Android" would inherit the properties and functions of its parent class "Phone". We can then add more specific properties and functions to the subclass, like calling Google Assistant for the "Android" class or calling Siri for an "IOS" class. So, if we were to make an "Android" object called Galaxy S21 and an "IOS" object called iPhone 13, both objects would be of the "Phone" class and have the same functions from that class, for example the turn on function. However, they would be from different subclasses and

have the functions that are exclusive to those subclasses. The Siri function would be exclusive to the iPhone 13 object and unavailable on the Galaxy S21 object and vice-versa for the Google Assistant function. So, inheritance is very useful when it comes to organizing a hierarchy and reusing data and functions.

Polymorphism

There are many definitions for polymorphism depending on the field of study. But in computer science, it can be defined as "the ability of different objects to respond in a unique way to the same message" (Sumo Logic, Polymorphism). This ties in a little with inheritance but what it means is that two objects can give you different results for the same function. For example, if the "Phone" class has a function "Color" that gives you the color of the phone object, you can execute that function on two different phone objects and get two different results. One phone object could be blue and the other red, even though you used the same function. That is what Polymorphism means. You have to write your code in such a way that it works correctly on different objects without having to change the function itself.

Data Structures

Another major topic in talking about Object Oriented Programming is data structures. Because programmers have to deal with so much data, they need a way to store and manipulate it, so they come up with data structures. "A data structure is a specialized format for organizing, processing, retrieving and storing data" (Loshin, 2021). In layman's terms, they are just a way to store a lot of data. And when you are dealing with so much data, you want to

have a way to arrange it all so you can recognize where everything is. Some of the most basic data structures are Arrays, Stacks, Queues and Trees.

Arrays

"An array is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key" (Busbee & Braunschweig, (2018). Basically, an array is a list of numbered elements and these numbers are called an indexes. The first element has an index of zero (0), the second and index of one (1) and so on. This makes it easier to store large lists and you can access each element by its index.

Stacks

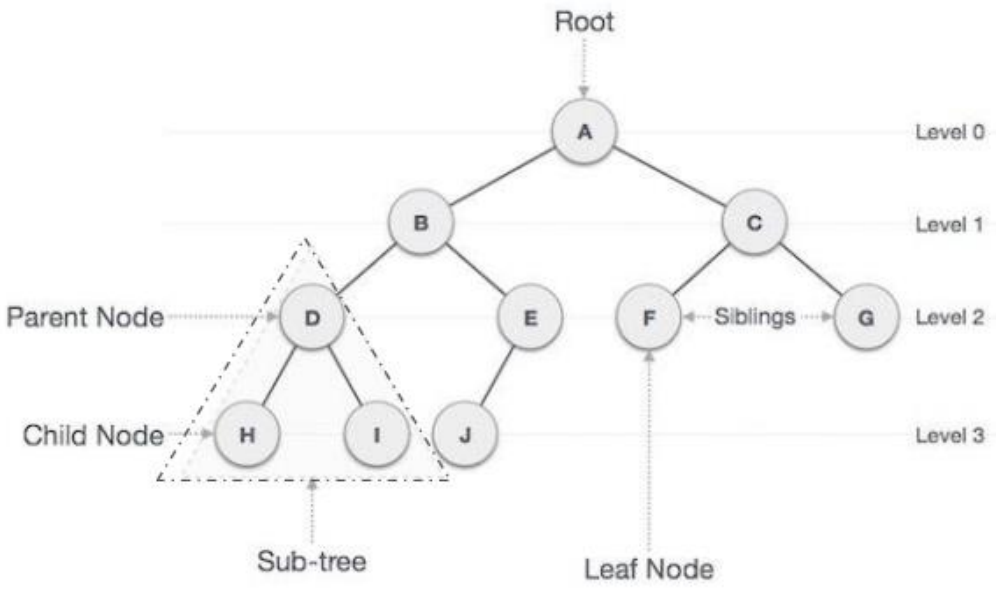
Stacks are also used to store lists, but they work a little differently. Faithfull to their name, stacks store their elements on top of each other. Think of a stack of plates, every new plate is put on the top and every plate that is taken is also taken form the top. This is how stacks work in programming. The newest element is put on the top and every time you take an element, it has to be from the top. This is also called Last In, First Out or LIFO.

Queues

Queues are very similar to stacks, but they operate the other way around. Think of a queue at your local Starbucks, every new customer joins the queue from the bottom, and it is the customer at the top of the queue that is served first and who gets to leave. The same way, when it comes to a queue in programming, the newest element is put at the bottom and every time you take an element, it has to be from the top. This is also called First In, First Out or FIFO.

Trees

Trees are a data structure that are mostly used to organize hierarchies. “A tree is a hierarchical data structure defined as a collection of nodes. Nodes represent value and nodes are connected by edges” (Great Learning Team, 2020).



Trees work similarly to family trees, where whatever is on top is the parent of whatever is below it. They are one of the most common data structures and are very efficient when it comes to searching a specific element. Instead of going through the whole tree one element after the other, you just have to figure out if the node you’re looking for is at the right or left of its parent node, halving the search time.

Conclusion

To finalize, there are a lot of difficult concepts in Object-Oriented Programming some more complex than others. However, they are not impossible to understand. They might

require a lot of thinking and practice to get used to, but they are worth the while. This document only covers the tip of the iceberg when it comes to Object-Oriented Programming concepts, but it is a good place to start. Programming is one the best documented fields of study and I would encourage everyone to delve deeper into it if they are, in any way, interested in Computer Science.

References

Doherty, E. (2020). *What is object-oriented programming? OOP explained in depth.*

https://www.educative.io/blog/object-oriented-programming

W3Schools *C++ What is OOP?.*

https://www.w3schools.com/cpp/cpp_oop.asp

Weisfeld, M. (2019). *Object-oriented thought process* (3rd ed.). Addison Wesley Professional.

Seaton, J. (2020) *What Is a Function in Programming?*

https://www.makeuseof.com/what-is-a-function-programming/

Sumo Logic *Encapsulation.*

https://www.sumologic.com/glossary/encapsulation/

Janssen, T. (2017) *OOP Concept for Beginners: What is Abstraction?*

https://stackify.com/oop-concept-abstraction/

Koenig-Bautista, A. (2019). *What is Inheritance? A Simple OOP Explanation*

https://medium.com/@andrewkoenigbautista/inheritance-in-object-oriented programming-d8808bca5021

Sumo Logic *Polymorphism.*

https://www.sumologic.com/glossary/polymorphism/

Loshin, D. (2021). *data structures*

https://searchsqlserver.techtarget.com/definition/data-structure

Busbee, K. L., & Braunschweig, D. (2018). *Programming Fundamentals : A Modular Structured Approach* (2nd ed.). Creative Commons Attribution-ShareAlike.

Great Learning Team (2020). *Understanding Trees in Data Structures https://www.mygreatlearning.com/blog/understanding-trees-in-data-structures/*