

Object Oriented programming

And why you should use it



Radulescu alexandru

York St John University

**Object Oriented Programming** (OOP), initially spoken about in the 1960s and developed ever since, represents a stepping stone in the programming world due to its simplicity, ease of use and versatility. With its multitude of advantages over classic programming it has become a big deal for programmers, although it requires more intricate thinking to fully understand and use its power, contrary to the classic methods which are often long, less efficient but somewhat easier to implement.

**Object Oriented Programming** is a type of computer programming in which the programmers define data structures, data types and the functions they use. As such, they become “objects” and programmers can create relationships between them such as inheritance. Due to its code-block-like design, OOP can be used to break down a program and solve individual problems before uniting everything for the final solution.

**Objects** represent self-contained entities that consist of both data and methods to manipulate data. **Classes** are the “blueprint”, they contain the properties and methods, while objects are the representation of classes in a program. **Constructors** are methods used by classes to initialize variables, or make it easier for the programmer to create an object and assign values to important variables from the beginning. Also, different constructors may be created for the same class, but they must have different parameters in order to work properly.

**Polymorphism** is a fundamental principle of OOP which provides increased flexibility when using multiple classes. A function can very easily adapt to the context it’s used in (for example, if two classes A and B share a function that does the same thing, only the main class C would require that function. A and B can call the function from C directly and do not affect each other. Moreover, modifications can be made to the function in class C without A and B being affected.

**Encapsulation** is one of the fundamentals of object-oriented programming. It refers to the bundling of data with the methods that operate on that data. It is used to hide the values or state of data inside a class, preventing unauthorized direct access to them. Publicly accessible methods are generally provided in the class (so-called getters and setters) to access the values, and other client classes call these methods to retrieve and modify the values within the object. The idea that some of the objects data must only be accessed through a public interface, and not modified directly is what makes encapsulation unique and useful.

To use the data stored in an object, to either modify it, or even get that value, we make that data private and write a method inside the object to manipulate it, to prevent the ability of modifying accidentally outside of the object, in another class or in the main program. As such, it’s considered bad practice to retrieve data from an object and write code outside of it which modifies that data.

**Encapsulation** is a good idea for several reasons:

* The functionality is defined in one place, inside the object.
* It is defined in a logical place, where the core data is stored.
* Data can’t be directly modified outside of the object. It can only be modified using public methods.

**Inheritance** is the mechanism of basing an object or class upon another object (prototypical inheritance) or class (class-based inheritance), retaining similar implementation. An object created through inheritance (a “child object”) acquires all the properties and behaviors of the parent object. This allows programmers to create classes that are built upon existing classes, to specify a new implementation while maintaining the same behaviors (making the base class an interface that the children are based on), to reuse code and to simplify debugging: Instead of going through all of the classes and methods, you only need to go once through the main class (the parent object) and the children’s own code.

Inheritance is a way of arranging objects in a hierarchy from the most general to the most specific. An object which inherits from another object is considered to be a subtype of that object. An example might include Instructor and Student, each of which inherit from Person.

Inheritance is also a way of reusing existing code easily. If we already have a class which does *almost* what we want, we can create a subclass which will partially override some of its behavior, or even add more functionality.

**Abstraction** is one of three central principles of object oriented programming. Through the process of abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency. Abstraction is related to both encapsulation and data hiding. It is used when we want to make sure a “child object” will definitely have certain attributes, or methods, usually the most important ones which give it the basic functionality. This is done to lessen errors in code or mistakes that may appear if a programmer forgets to add a basic method.

In the process of abstraction, the programmer tries to ensure that the entity is named in a manner that will make sense and that it will have all the relevant aspects included and none of the less important ones. Since entities may have any number of abstractions, you may get to use them in another procedure in the future.

Overall, object oriented programming is a great alternative to classic programming and problem solving, due to the modular aspect of a program: it can be created in stages, with each problem being solved individually and implemented together in the main class. As such, writing code is less tiring, it’s easier to understand and makes more sense.

**REFERENCES**

<https://www.roberthalf.com/blog/salaries-and-skills/4-advantages-of-object-oriented-programming/>

<https://www.webopedia.com/TERM/O/object_oriented_programming_OOP.html/>

<https://www.geeksforgeeks.org/constructors-in-java/?ref=lbp/>

<https://medium.com/javascript-scene/the-forgotten-history-of-oop-88d71b9b2d9f/>

<https://press.rebus.community/programmingfundamentals/chapter/encapsulation/>

<https://whatis.techtarget.com/definition/abstraction>

+ The Moodle slides for some additional information