**The basics of Object-Oriented Programming**

[OOP in Javascript: Basic Concepts and Implementation [Updated] (simplilearn.com)](https://www.simplilearn.com/tutorials/javascript-tutorial/oop-in-javascript)

Object-oriented programming is based on the concept of objects. In object-oriented programming data structures, or objects are defined, each with its own properties or attributes. Each object can also contain its own procedures or methods.

Software is designed by using objects that interact with one another. This offers various benefits, like:

* being faster and easier to execute;
* providing a clear structure for a program;
* making code easier to modify, debug and maintain; and
* making it easier to reuse code.

Object oriented languages are diverse, but the most popular ones are class-based languages in which objects are instances of classes. Significant object-oriented languages include Java, C++, C#, Python and Javascript.

The simplest way to explain object-orientated programming to a kid is to use something like a car as an example. A car has a model name, a colour, a year in which it was manufactured, an engine size and so on.

We would therefore create a Car object with the name, colour, engine size and year as attributes.

For every new car we use, we would use the car object. For instance, we can have a 2019 Blue BMW or a 2017 Red Audi. In each instance we would reuse the code contained in the original object.

The main principles of object-oriented programming are encapsulation, abstraction, inheritance and polymorphism. We will now deal with these four principles and how to explain them to a child.

**Encapsulation**

The principle of encapsulation entails that all the properties and methods of an object is kept private and safe from interference by other objects. In each object we can have both private and public variables and methods. Private variables and methods cannot be called or used by other objects, whereas public ones can.

To explain this, let’s again use our car example. The attributes such as colour, year and model would be private variables. They can just not be changed by other objects.

In turn we can have a public method called Start. Other objects, for instance a Person object, would be able to call this method.

We can use a computer game as an example that would be more relatable to children. Let’s look at a role-playing game as an example. In a typical role-playing game, we may have a main character or hero.

This hero will have several variables like name, outfit, hair colour and skin colour. The hero may also have methods like attack, walk, run and talk. Typically, these would be private variables and methods that can’t be modified by other objects in the game.

The hero may also have a variable Health and a method called Damage. These would typically be public variables and methods which can be modified by other objects. An enemy object would be able to call this method and change the value of the variable when it attacks the hero.

**Abstraction**

Abstraction can be seen as an extension of encapsulation. Oftentimes, programs are very large with thousands of lines of code. This is difficult to maintain, but abstraction helps with this.

Abstraction means that every object only exposes a high-level mechanism for using it. Thus, the code within, to a large extent becomes irrelevant to other objects interacting with the object.

Let’s use our car example again. As stated, it might have a Start method attached to it. This Start method may have some code inside it which provides what happens when the method is called. The object calling that method does not need to know it works or what code makes it work, but only that it works.

Using the game example, the Attack method may have many lines of code contained in it. This code can specify how the hero can be attacked and what effect the attack may have on the health of the hero.

The enemy object does not need to be aware how the attack works, just that it works and that it reduces the health of the hero.

**Inheritance**

As stated before, programs often contain thousands of lines of code which is complicated and difficult to maintain. Another problem often encountered is that we have similar objects. They can share some code or logic, but they are not exactly the same.

If we had to create a brand-new object for every object we use in our program it would lead to more code and complexity. In order to prevent this, we can use inheritance. With inheritance we extract the logic in one object, called the parent, to another object, called the child.

Using our car example, we can, for instance, extract the features like year, colour and model into another object. We can thus use our car object to create other objects like trucks, busses or vans due to their similar nature.

In our game example, we can have a variety of enemies. These enemies can all be child objects of a parent enemy object. Each enemy may have similar attributes, but also different attributes like outfit or weapon.

**Polymorphism**

We have now seen how inheritance enables us to use a parent object to define a child object. The problem is that the child might have a different way of implementing a method.

Polymorphism gives us a way to use an object exactly like its parent but keeping its own methods as they are.

As an example, let’s look at a Vehicle object. We can use this object to create other objects like a Car, a Truck or a Motorcycle. If the Vehicle has a Start method, it may be implemented by each child object differently. Polymorphism enables each child object to implement the Start method differently.

In our game, our enemies may each have an Attack method which it inherits from the parent, but each with its own implementation.

Object oriented programming may be complicated to learn, not only for adults, but also for children. However, the more you use examples and something that is relatable to children, the easier it will be to explain.

**OOPs Concepts:**

* Class
* Objects
* Data Abstraction
* Encapsulation
* Inheritance
* Polymorphism
* Dynamic Binding
* Message Passing

**1. Class:**

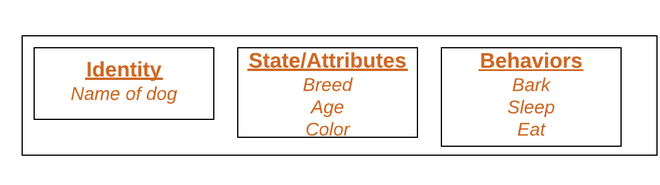
A class is a user-defined data type. It consists of data members and member functions, which can be accessed and used by creating an instance of that class. It represents the set of properties or methods that are common to all objects of one type. A class is like a blueprint for an object.

***For Example:***Consider the Class of Cars. There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range, etc. So here, Car is the class, and wheels, speed limits, mileage are their properties.

**2. Object:**

It is a basic unit of Object-Oriented Programming and represents the real-life entities. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated. An object has an identity, state, and behavior. Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other’s data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

For example “Dog” is a real-life Object, which has some characteristics like color, Breed, Bark, Sleep, and Eats.



*Object*

**3. Data Abstraction:**

Data abstraction is one of the most essential and important features of object-oriented programming. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation. Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

**4. Encapsulation:**

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. In Encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of their class in which they are declared. As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.

Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section, etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name “sales section”.

**5. Inheritance:**

Inheritance is an important pillar of OOP(Object-Oriented Programming). The capability of a class to derive properties and characteristics from another class is called Inheritance. When we write a class, we inherit properties from other classes. So when we create a class, we do not need to write all the properties and functions again and again, as these can be inherited from another class that possesses it. Inheritance allows the user to reuse the code whenever possible and reduce its redundancy.

**6. Polymorphism:**

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. For example, A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations. This is called polymorphism.

**7. Dynamic Binding:**

In dynamic binding, the code to be executed in response to the function call is decided at runtime. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time. Dynamic Method Binding One of the main advantages of inheritance is that some derived class D has all the members of its base class B. Once D is not hiding any of the public members of B, then an object of D can represent B in any context where a B could be used. This feature is known as subtype polymorphism.

**8. Message Passing:**

It is a form of communication used in object-oriented programming as well as parallel programming. Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function, and the information to be sent.