



Software Development

[Video Tutorials](#) [Articles](#) [Ebooks](#) [Free Practice Tests](#) [On-demand Webinars](#) [Live Webinars](#)

[Home](#) [Resources](#) [Software Development](#) [C++ Tutorial for Beginner](#) [Basic Concepts of Object Oriented Programming \(OOPs\) in C++: Encapsulation, Polymorphism & More](#)

Basic Concepts of Object Oriented Programming (OOPs) in C++: Encapsulation, Polymorphism & More

Lesson 12 of 24

By [Harsh Bhardwaj](#)

Last updated on Jul 25, 2022

62940

[Previous](#)

[Next](#)

[Tutorial Playlist](#)

Table of Contents

[What Are OOPS Concepts In C++?](#)

[Why Do You Need Object-Oriented Programming?](#)

Basic Object-Oriented Programming (OOPS) Concept in C++

Advantages of OOPs

Conclusion

The main intent of introducing the [C++ programming language](#) was to add object-oriented features to the [C language](#). OOPs offer several benefits or advantages to the designer and user, and there are various areas where OOPs play an essential role, including user interface design, simulation, and modeling, etc. This article on OOPs concepts in c++ will help you understand and learn all about object-oriented programming.

Post Graduate Program: Full Stack Web Development

in Collaboration with Caltech CTME

ENROLL NOW

What Are OOPS Concepts In C++?

[OOPs, or Object-oriented programming](#) is an approach or a [programming](#) pattern where the programs are structured around objects rather than functions and logic. It makes the data partitioned into two memory areas, i.e., data and functions, and helps make the code flexible and modular.

Object-oriented programming mainly focuses on objects that are required to be manipulated. In OOPs, it can represent data as objects that have attributes and functions.

Why Do You Need Object-Oriented Programming?

The earlier approaches to programming were not that good, and there were several limitations as

well. Like in procedural-oriented programming, you cannot reuse the code again in the program, and there was the problem of global data access, and the approach couldn't solve the real-world problems very well.

In object-oriented programming, it is easy to maintain the code with the help of classes and objects. Using inheritance, there is code reusability, i.e., you don't have to write the same code again and again, which increases the simplicity of the program. Concepts like encapsulation and abstraction provide data hiding as well.

Now have a look at some basic concepts of C++ OOPs.

Basic Object-Oriented Programming (OOPS) Concept in C++

There are some basic concepts that act as the building blocks of OOPs.

- Classes & Objects
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

So now, you will understand each of these concepts in detail.

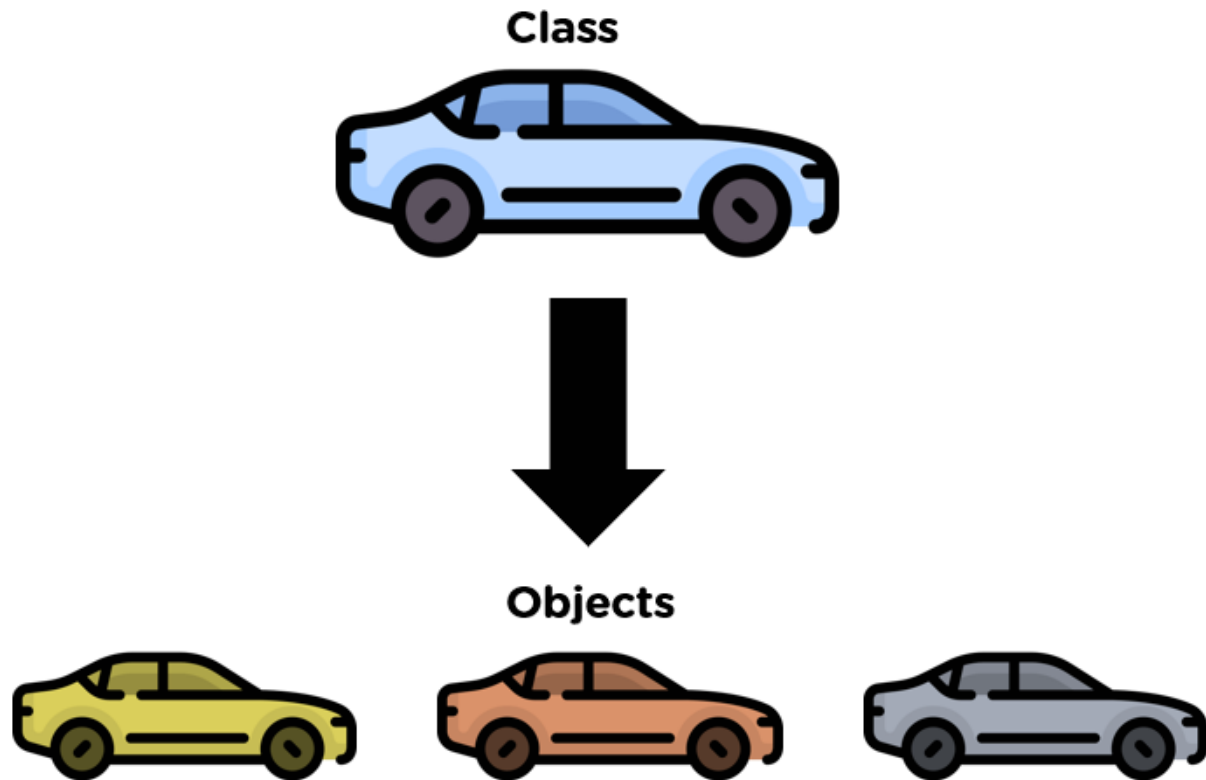
- **Object**

An **Object** can be defined as an entity that has a state and behavior, or in other words, anything that exists physically in the world is called an object. It can represent a dog, a person, a table, etc.

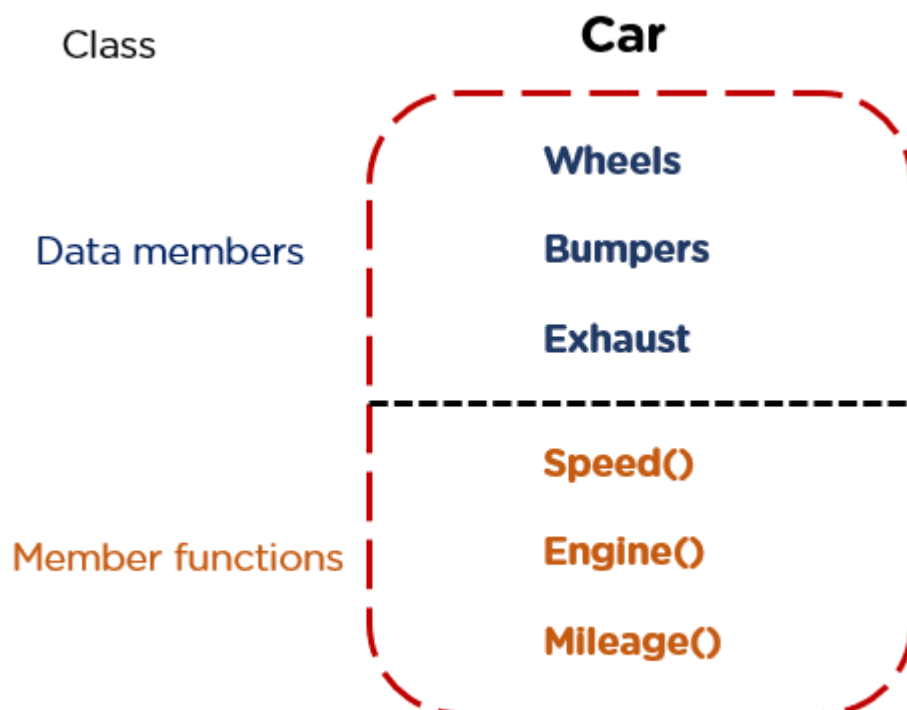
An object means the combination of data and programs, which further represent an entity.

- **Classes**

Class can be defined as a blueprint of the object. It is basically a collection of objects which act as building blocks.



A class contains data members (variables) and member functions. These member functions are used to manipulate the data members inside the class.



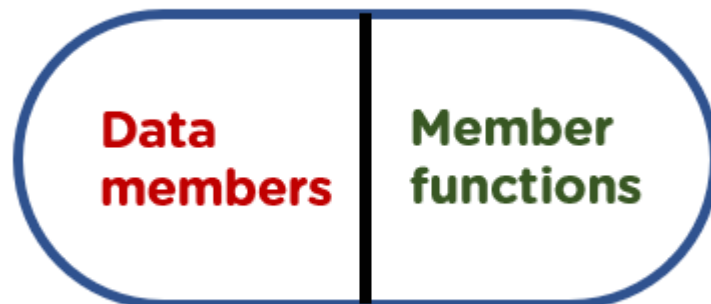
Full Stack Web Developer Course

To become an expert in MEAN Stack

[VIEW COURSE](#)

- **Abstraction**

Abstraction helps in the data hiding process. It helps in displaying the essential features without showing the details or the functionality to the user. It avoids unnecessary information or irrelevant details and shows only that specific part which the user wants to see.



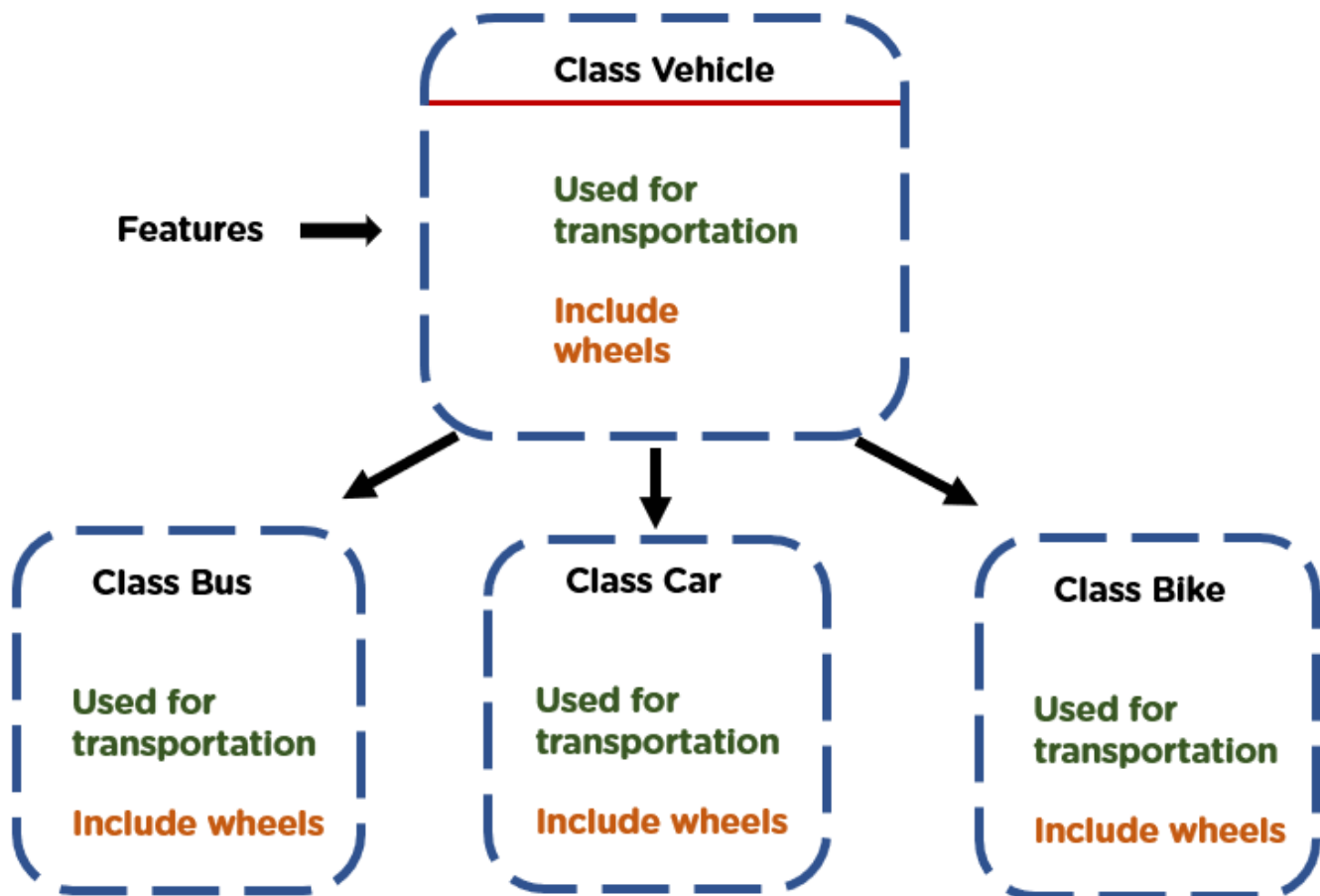
- **Encapsulation**

The wrapping up of data and functions together in a single unit is known as encapsulation. It can be achieved by making the data members' scope private and the member function's scope public to access these data members. Encapsulation makes the data non-accessible to the outside world.



- Inheritance

Inheritance is the process in which two classes have an is-a relationship among each other and objects of one class acquire properties and features of the other class. The class which inherits the features is known as the child class, and the class whose features it inherited is called the parent class. For example, Class Vehicle is the parent class, and Class Bus, Car, and Bike are child classes.



Let's have a look at an example of inheritance.

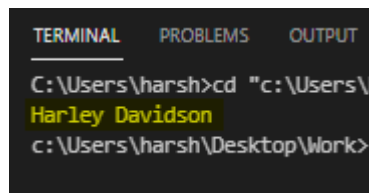
```
2  #include <iostream>
3
4  using namespace std;
5
6  class Parent
7  {
8      public:
9      string name1="Harley";
```

```
9   string name1= "Harley" ;
10  };
11
12  class Child: public Parent
13  {
14      public:
15          string name2="Davidson";
16  };
17
18  int main()
19  {
20      Child ch;
21      cout << ch.name1 + " " + ch.name2;
22
23      return 0;
24  }
```

As we can see, there are two classes named parent and child. In line 12, the child class inherits the parent class. Initially, you created an instance of the child class, through which you are calling name1 and name2 variables; both of these string variables are holding Harley and Davidson, respectively.

Since child class is inheriting parent class, when you call using the object of the child class, you get the result name1 as Harley and name2 as Davidson.

Below is the output.



```
TERMINAL  PROBLEMS  OUTPUT
C:\Users\harsh>cd "c:\Users\
Harley Davidson
c:\Users\harsh\Desktop\Work>
```

New Course: Full Stack Development for Beginners

Learn Git Command, Angular, NodeJS, Maven & More

[ENROLL NOW](#)

- Polymorphism

Polymorphism means many forms. It is the ability to take more than one form. It is a feature that provides a function or an operator with more than one definition. It can be implemented using function overloading, operator overload, **function overriding**, virtual function.

Let's have a look at an example where we are implementing function overriding.

```
27 #include <iostream>
28 using namespace std;
29
30 class Animal {
31 public:
32     void speed() {
33         cout << "Who is more faster\n" ;
34     }
35 };
36
37 class Cheetah : public Animal {
38 public:
39     void speed() {
40         cout << "cheetah says im faster \n" ;
41     }
42 };
43
44 class Dolphin : public Animal {
45 public:
46     void speed() {
47         cout << "Dolphin says im faster \n" ;
48     }
49 };
50 int main() {
51     Animal a;
52     Cheetah c;
53     Dolphin d;
54
55     a.speed();
56     c.speed();
57     d.speed();
58     return 0;
59 }
```


As you can see in the above example, there are three classes. Class Animal is the parent class, Class Cheetah, which is the derived class, and Class Dolphin is again the derived class of Animal class.

All three classes have a [function](#) of the same name, i.e., speed, but the definition of this speed function is different in all three classes. Inside the main function, firstly, you are invoking the speed function using the object of the parent class, then using the object of child class Cheetah, you are again invoking the function, and similarly, you are invoking the function using the object of dolphin class.

Below is the output of the above program. You can see that when you call the function using the object of the parent class, it invokes the function of the parent class. But when you call the function using the object of the child class, it overrides the parent class function and prints the function of the child class.



Now go through the advantages of C++ OOPs.

Advantages of OOPs

There are various advantages of object-oriented programming.

- OOPs provide reusability to the code and extend the use of existing classes.
- In OOPs, it is easy to maintain code as there are classes and objects, which helps in making it easy to maintain rather than restructuring.
- It also helps in data hiding, keeping the data and information safe from leaking or getting exposed.
- Object-oriented programming is easy to implement.

Master front-end and back-end technologies and advanced aspects in our [Post Graduate Program in Full Stack Web Development](#). Unleash your career as an expert full stack developer.

Get in touch with us NOW!

Conclusion

After reading this tutorial on OOPS Concepts in C++, you would have understood why you need Object-oriented programming, what C++ OOPs are, and the basic concepts of OOPs like polymorphism, inheritance encapsulation, etc. You also learned about the advantages of C++ OOPs along with examples of polymorphism and inheritance.

If you are perhaps looking to build a [career in software development](#), check the [Post-Graduate Program in Full Stack Development](#) by Simplilearn. It can prove to be the ideal solution to help you build your career in the right way.

Do you have any questions regarding this tutorial on OOPS concepts in C++? If you do, then please put them in the comments section. We'll help you solve your queries. To learn more about C++ OOPs, click on the following link: [C++ OOPs](#)

Happy learning!


About the Author


 [Harsh Bhardwaj](#)

Avijeet is a Senior Research Analyst at Simplilearn. Passionate about Data Analytics, Machine Learning, and Deep Learning.

[View More](#)

Recommended Resources

 **Software Developer Resume:**
A Comprehen...

 **Predictions and Trends**
2020: DevOps and S..

© 2009 -2022- Simplilearn Solutions

Disclaimer

PMP, PMI, PMBOK, CAPM, PgMP, PfMP, ACP, PBA, RMP, SP, and OPM3 are registered marks of the Project Management Institute, Inc.

*According to Simplilearn survey conducted and subject to [terms & conditions](#) with Ernst & Young LLP (EY) as Process Advisors