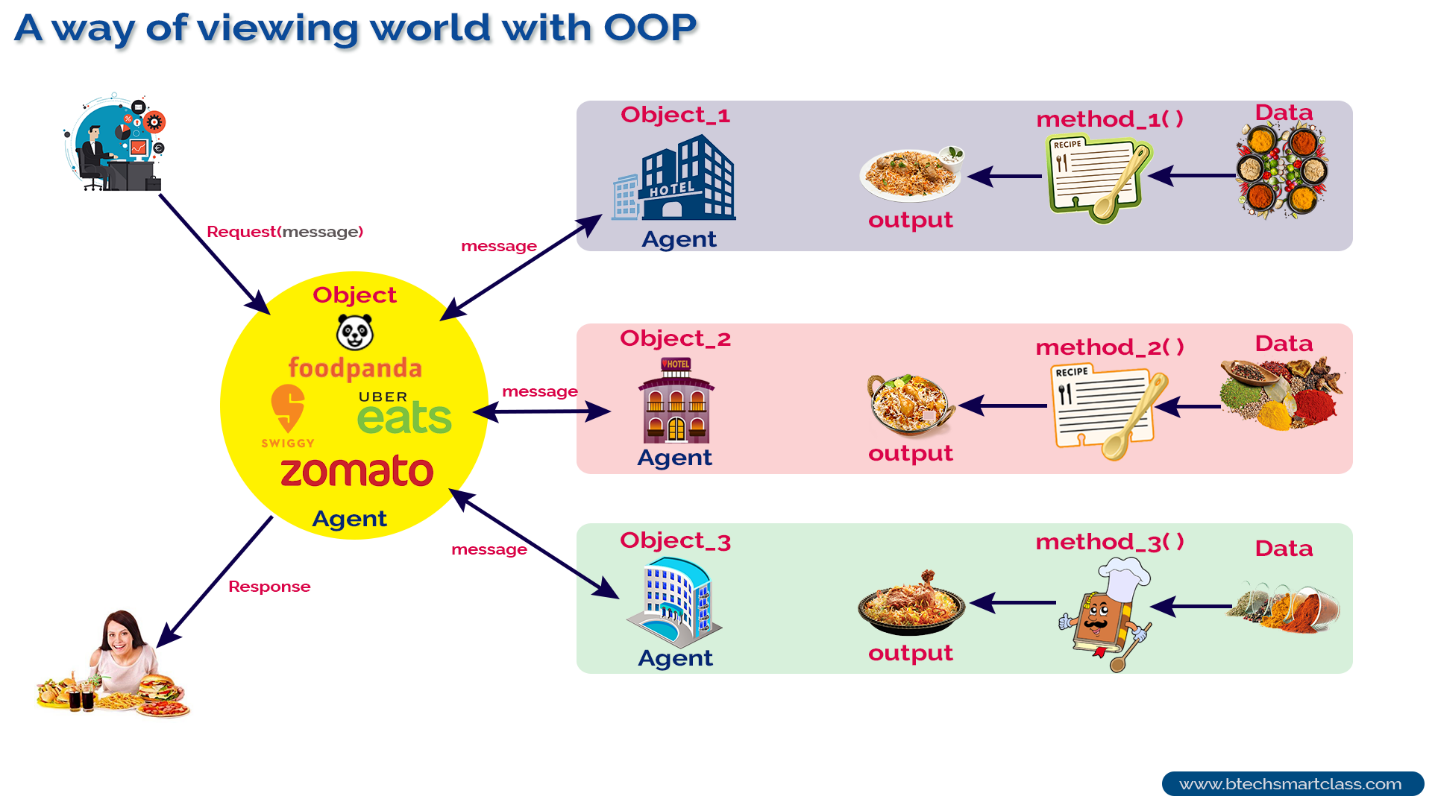
**A way of viewing world agent**

A way of viewing the world is an idea to illustrate the object-oriented programming concept with an example of a real-world situation.

Let us consider a situation, A person sitting in the office and wish to get food to his family members who are at home from a hotel. Because of the distance from office to home, there is no possibility of getting food from a hotel himself. So, how do we solve this issue?

To solve the problem, he/she will call Zomato (an **agent** in food delivery community), tell them the variety and quantity of food and the hotel name from which he wish to deliver the food to his/her family members. Look at the following image.



**Agents and Communities**

To solve food delivery problem, he/she used a solution by finding an appropriate agent (Zomato) and pass a message containing his/her request. It is the responsibility of the agent (Zomato) to satisfy the request. Here, the agent uses some method to do this. He/ She does not need to know the method that the agent has used to solve the request. This is usually hidden.

So, in object-oriented programming, problem-solving is the solution to the problem which requires the help of many individuals in the community.

**An object-oriented program is structured as a community of interacting agents, called objects. Where each object provides a service (data and methods) that is used by other members of the community.**

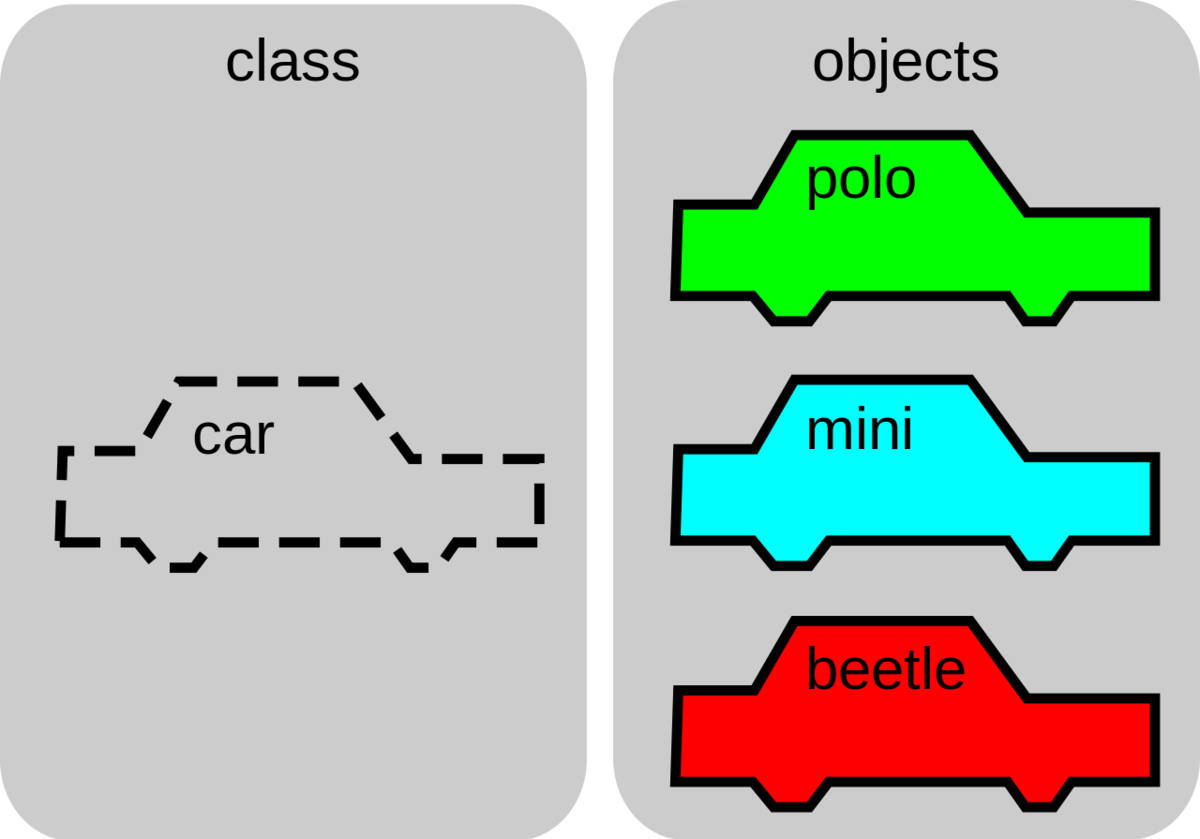
In our example, the online food delivery system is a community in which the agents are Zomato and set of hotels. Each hotel provides a variety of services that can be used by other members like Zomato, myself, and my family in the community.

**In object-oriented programming, every action is initiated by passing a message to an agent (object), which is responsible for the action. The receiver is the object to whom the message was sent. In response to the message, the receiver performs some method to carry out the request. Every message may include any additional information as arguments.**

**Classes and Types of Classes(Varieties of Classes)**

In object-oriented programming, a **class** is a blueprint for creating **objects** (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behavior (member functions or methods).

The user-defined objects are created using the class keyword. The class is a blueprint that defines a nature of a future object. An **instance** is a specific object created from a particular class. Classes are used to create and manage new objects and support **inheritance**—a key ingredient in object-oriented programming and a mechanism of reusing code.

[*[1]*](https://brilliant.org/wiki/classes-oop/#citation-1)

The image above shows how a Car object can be the template for many other Car instances. In the image, there are three instances: polo, mini, and beetle. Here, we will make a new class called Car, that will structure a Car object to contain information about the car’s model, the color, how many passengers it can hold, its speed, etc. A class can define types of operations, or methods, that can be performed on a Car object. For example, the Car class might specify an accelerate method, which would update the speed attribute of the car object.

Class Types: -

Classes in oops can have several different forms of responsibility, and thus, are used for different purposes. The following categories cover a large no. of cases:

* Data Managers
* Data sinks or data source
* View or observer classes
* Facilitator or helper classes

Data Manager (Data or State classes): - These are the classes with the principal responsibility of maintaining data or state information.

Data sinks or data source: - These are classes that generate data, or that accept data and process them further, such as a class performing output to a disk or file. These classes do not hold the data for any period of time, but generates it on demand or processes it when called upon.

View or Observer class: - The important role of most application is the display of information on an output device such as a terminal screen. Because the code for performing this activity is often complex, frequently modified, and largely independent of the actual data being displayed, it is good programming practice to isolate display behavior in classes other than those that maintain the data being displayed. Such role of a class can be considered as view or observer class.

Facilitator or Helper Class: -

These classes maintain little or no state information themselves but assist in the execution of complex tasks. For ex., in displaying a playing card image we use the services of a facilitator class that handles the drawing of lines and text on the display device.