**Fundamentals**

**Object-oriented programming (OOP)** is a programming paradigm based on the concept of **objects** that interact with each other to perform program functions. Each object can be characterized by a state and behavior. An object’s current state is represented by its **fields**, and an object’s behavior is represented by its **methods**.

**Basic principles of OOP**

There are four basic principles of OOP. They are **encapsulation, abstraction, inheritance,**and **polymorphism.**

* **Encapsulation** ensures bundling (=encapsulating) of data and the methods operating on that data into a single unit. It also refers to the ability of an object to hide the internal structure of its properties and methods.
* **Data abstraction**means that objects should provide a simplified, abstract version of their implementations. The details of their internal work usually aren't necessary for the user, so there's no need to represent them. Abstraction also means that only the most relevant features of the object will be presented.
* **Inheritance**is a mechanism for defining parent-child relationships between classes. Often objects are very similar, so inheritance allows programmers to reuse common logic and at the same time introduce unique concepts into the classes.
* **Polymorphism** literally means "having many forms" and is a concept related to inheritance. It allows programmers to define different implementations for the same method. Thus, the name (or interface) remains the same, but the actions performed may differ. For example, imagine a website that posts three main types of text: news, announcements, and articles. They are somewhat similar in that they all have a headline, some text, and a date. In other ways, they are different: articles have authors, news bulletins have sources, and announcements have a date after which they become irrelevant. It is convenient to write an abstract class with general information for all publications to avoid copying it every time and store what is different in the appropriate derived classes.

These are the key concepts of OOP. Each object-oriented language implements these principles in its own way, but the essence stays the same from language to language.

**Objects**

The key notion of OOP is, naturally, an **object.**There are a lot of real-world objects around you: pets, buildings, cars, computers, planes, you name it. Even a computer program may be considered as an object.

It's possible to identify some important characteristics of real-world objects. For instance, for a building, we can consider the number of floors, the year of construction, and the total area. Another example is a plane, which can accommodate a certain number of passengers and transfer you from one city to another. These characteristics constitute the object's **attributes** and **methods**. Attributes characterize the states or data of an object, and methods characterize its behavior.

**Classes**

Often, many individual objects have similar characteristics. We can say these objects belong to the same **type** or **class**.

A class is another important notion of OOP. A class describes a common structure of similar objects: their fields and methods. It may be considered a template or a blueprint for similar objects. An object is an individual **instance** of a class.

In accordance with the principle of encapsulation mentioned above, any class should be considered as a black box, that is, the user of the class should see and use only the interface part of the class, namely, the list of declared properties and methods, and should not delve into the internal implementation.

Let's look at some examples below.

**Example 1. The building class**

**An abstract building for describing buildings as a type of object (class)**

Each building has the same attributes:

* Number of floors (an integer number);
* Area (a floating-point number, square meters);
* Year of construction (an integer number).

Each object of the building type has the same attributes but different values.

For instance:

* Building 1: number of floors = 4, area = 2400.16, year of construction = 1966;
* Building 2: number of floors = 6, area = 3200.54, year of construction = 2001.

It's quite difficult to determine the behavior of a building, but this example demonstrates attributes pretty well.

**Example 2. The plane class**

Unlike with a building, it is easy to define the behavior of a plane: it can fly and transfer you between two points on the planet.

**An abstract plane for describing all planes as a type of object (class)**

Each plane has the following attributes:

* Name (a string, for example, "Airbus A320" or "Boeing 777");
* Passenger capacity (an integer number);
* Standard speed (an integer number);
* Current coordinates (they are needed to navigate).

Also, it has a behavior (a method): transferring passengers from one geographical point to another. This behavior changes the state of a plane, namely, its current coordinates.

**Objects and classes**

In OOP, everything can be viewed as an object; a class, for example, is also an object. Programs are made up of different objects that interact with each other. Object state and behavior are usually combined, but this is not always the case. Sometimes we see objects without a state or methods. This, of course, depends on the purpose of the program and the nature of the object.

For example, there is such a thing as an interface. Not a user interface, but a class that only serves to be inherited from in order to guarantee an interface to its descendant classes. It is a stateless class. Structures exist in C++ for historical reasons. Now a structure in C++ is also a class, but once upon a time, a structure had only properties and did not have any methods – a type for storing data and nothing else. These are special cases, and they are sometimes useful.

**Conclusion**

Object-oriented programming is akin to the work of an architect. As an architect, you create templates called **classes**and build individual instances of these classes, known as **objects**. Similar to the inhabitants of a building, users of a class interact solely with its **fields**and **methods**, without needing to delve into the internal implementation details. This principle is commonly known as **encapsulation**.

OOP is a key part of many programming languages, including C++, C#, Java, Kotlin, Python, and Ruby. Mastery of OOP enables you to write tidy, easily updatable code, an essential skill for every coder. So, it's time to start practicing!

Encapsulation is about hiding the internal data of objects from the world.

Abstraction is about giving only the most relevant features of the object to the user.

Polymorphism is about defining different logic of the same method.

Try to differentiate between actions (methods) and states (attributes)