What is Object-Oriented Programming?

Adapted from: https://www.educative.io/blog/object-oriented-programming
Original text by Erin Doherty - May 20, 2024

Object-Oriented Programming (OOP) is a programming paradigm in computer science that relies on the concept of **classes** and **objects**. It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects. There are many object-oriented programming languages, including JavaScript, C++, Java, and Python.

A **class** is a blueprint that creates more specific objects. Classes often represent broad categories, like `Car` or `Dog` that share **attributes**. These classes define what attributes an instance of this type will have, like `color`, but not the value of those attributes for a specific object.

Classes can also contain functions called **methods** that are available only to objects of that type. These functions are defined within the class and perform some action helpful to that specific object type.

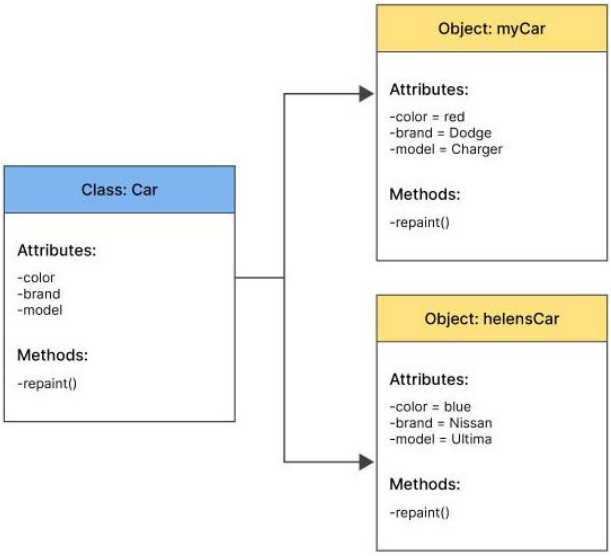
For example, our `Car` class may have a `repaint` method that changes the `color` attribute of our car. This function is only helpful to objects of type `Car`, so we declare it within the `Car` class, thus making it a method.

Classes are used as a blueprint to create individual **objects**. These represent specific examples of the class, like `myCar` or `goldenRetriever`. Each object can have unique values to the properties defined in the class.

For example, say we created a class, `Car`, to contain all the properties a car must have, `color`, `brand`, and `model`. We then create an instance of a `Car` type object, `myCar` to represent my specific car.

We could then set the value of the properties defined in the class to describe my car without affecting other objects or the class template.

We can then reuse this class to represent any number of cars.



Class blueprint being used to create two Car type objects, myCar and helensCar

Benefits of OOP for software engineering

- OOP models complex things as reproducible, simple structures
- Reusable, OOP objects can be used across programs
- Polymorphism allows for class-specific behavior
- Easier to debug, classes often contain all applicable information to them
- Securely protects sensitive information through encapsulation

How to structure OOP programs

Let's take a real-world problem and conceptually design an OOP software program.

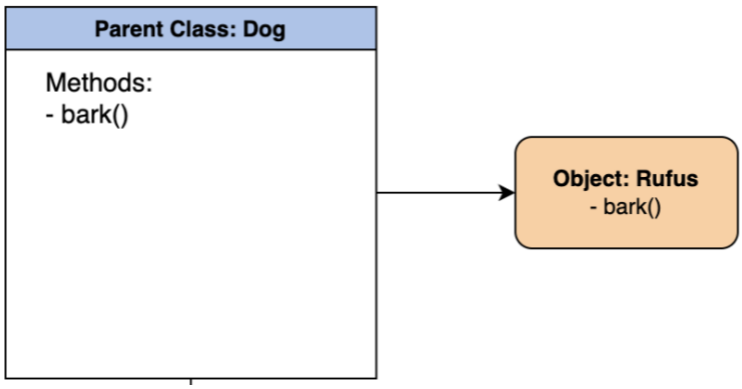
Imagine running a dog-sitting camp with hundreds of pets where you keep track of the names, ages, and days attended for each pet. How would you design simple, reusable software to model the dogs?

With hundreds of dogs, it would be inefficient to write unique entries for each dog because you would be writing a lot of redundant code.

Grouping related information together to form a class structure makes the code shorter and easier to maintain.

In the dog sitting example, here's how a programmer could organize the program: create a class for all dogs as a blueprint of information and behaviors (methods) that all dogs will have, and create objects from the class that represent individual dogs.

The diagram below represents how to design an OOP program by grouping the related data and behaviors together to form a simple template:



The `Dog` class is a generic template containing only the structure of data and behaviors common to all dogs as attributes. We can create objects like `Rufus`.

Building blocks of OOP

Next, we'll take a deeper look at each of the fundamental building blocks of an OOP program used above:

- Classes
- Objects
- Methods
- Attributes

Classes

In a nutshell, classes are essentially **user-defined data types**. Classes are where we create a blueprint for the structure of methods and attributes. Individual objects are instantiated from this blueprint.

Classes contain fields for attributes and methods for behaviors. In our `Dog` class example, attributes include `name`, `birthday`, and `attendance` , while methods include `bark()` and `updateAttendance()`.

Remember, the class is a template for modeling a dog, and an object is instantiated from the class representing an individual real-world item.

Objects

Objects are, unsurprisingly, a huge part of OOP! Objects are **instances of a class created** with specific data. For example, in the code snippet below `rufus` is an instance of the `Dog` class.

```
// instantiate an object of the Dog class, and individual dog named Rufus
rufus = Dog("Rufus", "2/1/2017")
```

In this code:

- A new object is created named `rufus`. A constructor method works behind the scenes.
- The constructor initializes `name` and `birthday`, and assigns values to them (`"Rufus"` and `"2/1/2017"` respectively)

Terms / Vocabulary

Term	What is it?	Information Contained	Actions	Example
Classes	Blueprint	Attributes	Behaviors defined through methods	`Dog`
Objects	Instance	State, Data	Methods	`rufus`

Attributes

Attributes are the information that is stored. Attributes are defined in the `Class` template. When objects are instantiated, individual objects contain data stored in the attribute fields.

The state of an object is defined by the data in the object's attribute fields. For example, a puppy and a dog might be treated differently at a pet camp. The birthday could define the state of an object and allow the software to handle dogs of different ages differently.

Methods

Methods represent behaviors. Methods perform actions; methods might return information about an object or update an object's data. The method's code is defined in the class definition.

When individual objects are instantiated, these objects can call the methods defined in the class. In the code snippet below, the `bark` method is defined in the `Dog` class, and the `bark()` method is called on the `rufus` object.

```
// calling bark() on the rufus object
rufus.bark()
```

Methods often modify, or update data. Methods don't have to update data though. For example, the `bark()` method doesn't update any data because barking doesn't modify any of the attributes of the `Dog` class: `name` or `birthday`.

The `updateAttendance()` method adds a day the `Dog` attended the pet-sitting camp. The `attendance` attribute is important to keep track of for billing owners at the end of the month.

Methods are how programmers promote reusability and keep functionality encapsulated inside an object. This reusability is a great benefit when debugging. If there's an error, there's only one place to find it and fix it instead of many.