EXPT 10 Introduction to Object Oriented Programming
Date:

What is Object-Oriented Programming?

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.

OOP languages are not necessarily restricted to the object-oriented programming paradigm.

Some languages, such as JavaScript, Python, and PHP, all allow for both procedural and object-oriented programming styles.

A **class** is an abstract blueprint that creates more specific, concrete objects.

Classes often represent broad categories, like `Cat` 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.

Class templates are used as a blueprint to create individual **objects**.

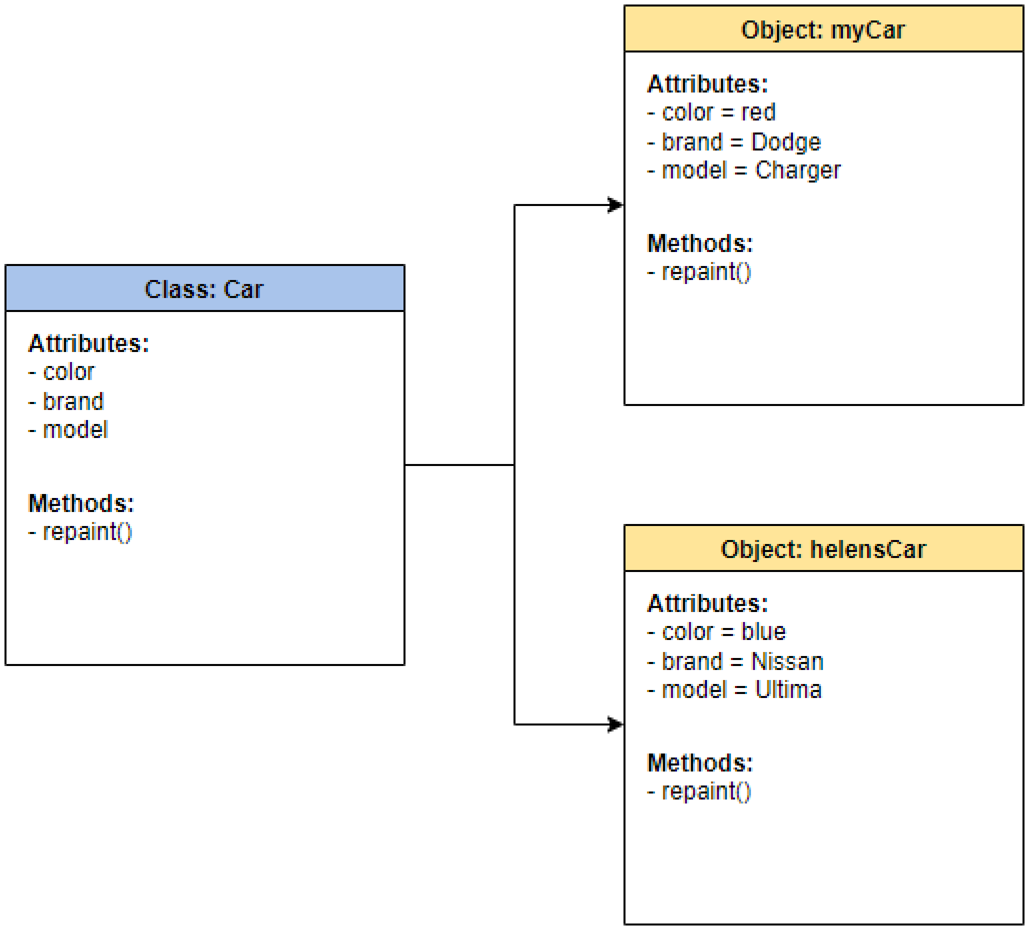
These represent specific examples of the abstract class, like `myCar` or `goldenRetriever`.

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.



How to structure OOP programs

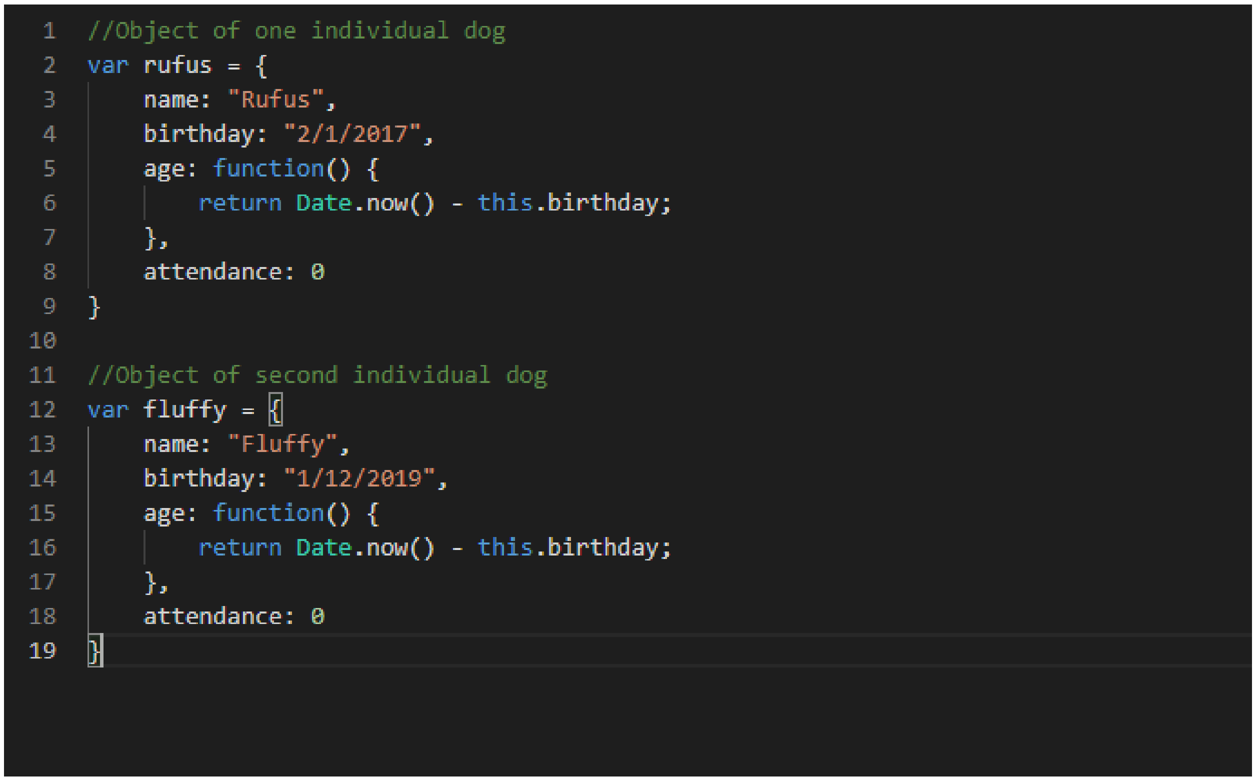
Let's take a real-world problem and conceptually design an OOP software program.

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.

Below we see what that might look like with objects rufus and fluffy.



```javascript
//Object of one individual dog
var rufus = {
    name: "Rufus",
    birthday: "2/1/2017",
    age: function() {
        return Date.now() - this.birthday;
    },
    attendance: 0
}

//Object of second individual dog
var fluffy = {
    name: "Fluffy",
    birthday: "1/12/2019",
    age: function() {
        return Date.now() - this.birthday;
    },
    attendance: 0
}
```

As you can see above, there is a lot of duplicated code between both objects.

The age() function appears in each object. Since we want the same information for each dog, we can use objects and classes instead.

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 think about organizing an OOP:

1. **Create a class for all dogs** as a blueprint of information and behaviors (methods) that all dogs will have, regardless of type. This is also known as the **parent class**.

2. **Create subclasses** to represent different subcategories of dogs under the main blueprint. These are also referred to as *child classes*.
3. **Add unique attributes and behaviors** to the child classes to represent differences
4. **Create objects from the child class** that represent dogs within that subgroup

The diagram below represents how to design an OOP program by grouping the related data and behaviors together to form a simple template and then creating subgroups for specialized data and behavior.

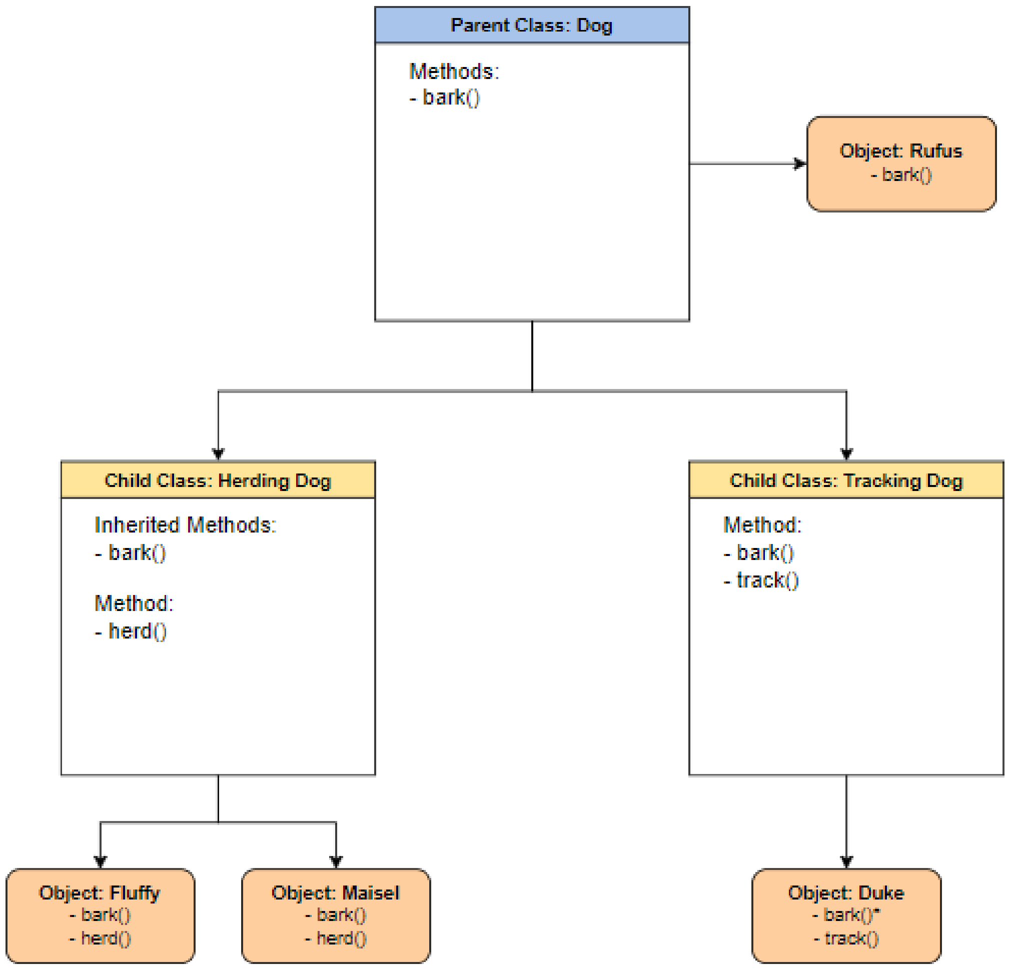
The `Dog` class is a generic template containing only the structure of data and behaviors common to all dogs as attributes.

We then create two child classes of `Dog`, `HerdingDog` and `TrackingDog`.

These have the inherited behaviors of `Dog` (`bark()`) but also behavior unique to dogs of that subtype.

Finally, we create objects of the `HerdingDog` type to represent the individual dogs `Fluffy` and `Maisel`.

We can also create objects like `Rufus` that fit under the broad class of `Dog` but do not fit under either `HerdingDog` or `TrackingDog`.



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`, while methods include `bark()` and `updateAttendance()`.