


✓ 100 XP 

# Introduction to classes in TypeScript

5 minutes

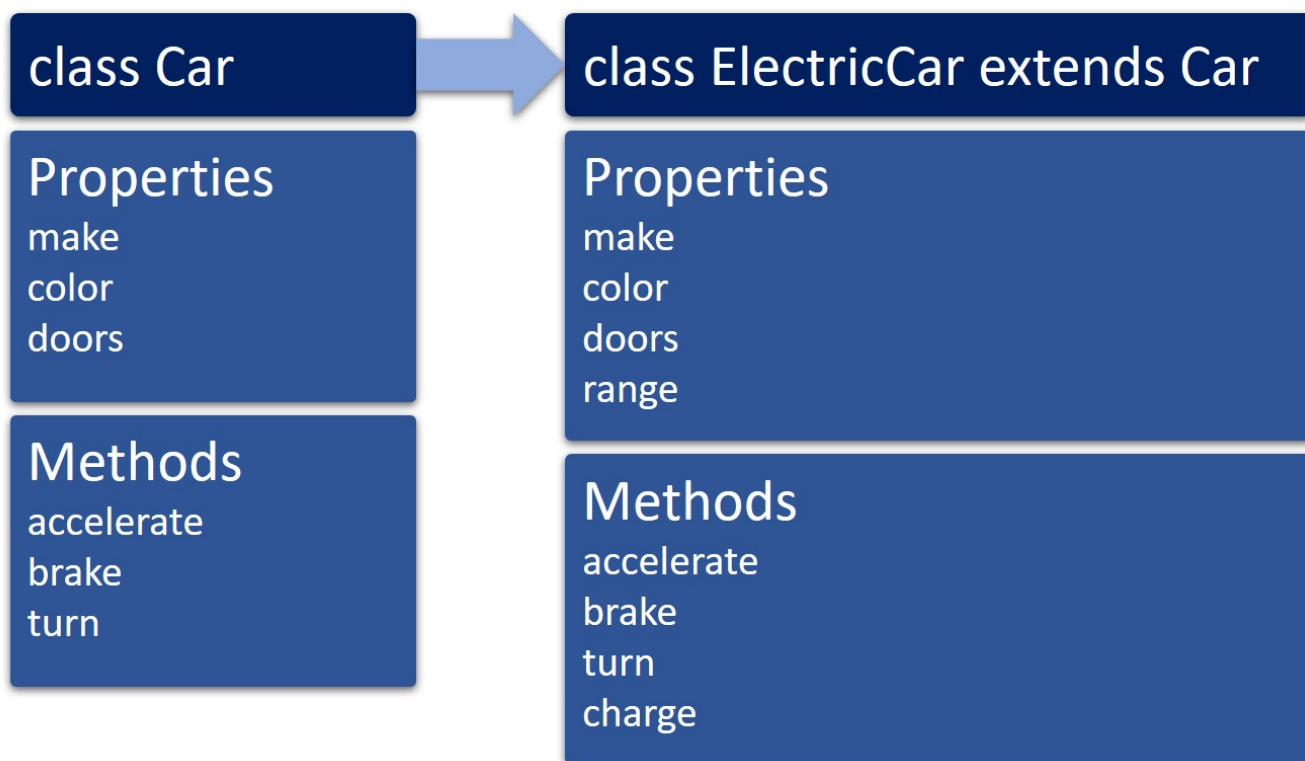
Classes enable you to express common object-oriented patterns in a standard way, making features like inheritance more readable and interoperable. In TypeScript, classes are yet another way to define the shape of an object, in addition to describing object types with interfaces and functions.

If you haven't worked with classes, it may be helpful to review a few basic concepts.

You can think of a class as a blueprint for building objects, like a car. A `car` class describes the attributes of a car, for example, the make, color, or number of doors. It also describes behaviors that the car can perform, like accelerating, braking, or turning.

But the `car` class is just a plan for building the car. You must build an instance of `car` from the `car` class before it becomes an object that you can assign property values to (like setting the color to blue) or call its behaviors (like applying the brakes.)

The `car` class can be reused to create any number of new `car` objects, each with their own characteristics. You can also extend the `car` class. For example, an `ElectricCar` class might extend `car`. It will have all the same attributes and behaviors of `car` but can also have its own unique attributes and behaviors, like its range and a charging operation.



The `Car` class includes the properties `make`, `color` and, `doors` and the methods `accelerate`, `brake`, and `turn`. When the `ElectricCar` class extends `Car`, it includes all of the properties and methods of `Car`, plus a new property called `range` and a new method called `charge`.

A class **encapsulates** data for the object. Data and behavior are included in the class but the details of both can be hidden from the person who is working with the object in code. For example, if you call the `turn` method of a `car` object, you don't need to know exactly how the steering wheel works, you just need to know that the car will turn left when you tell it to. The class serves as a black box where all the attributes and behaviors are only exposed through the properties and methods, limiting what a coder can do with it.

#### 💡 Tip

If you want to dig into object-oriented programming concepts in more detail, see the [Fundamentals of Classes](#) video.

## Class components

- **Properties**, also referred to as fields, are the data (or attributes) for the object. These are the defining characteristics of the object that you can set or return from your code.

- The `constructor` is a special function used to create and initialize objects based on the class. When you create a new instance of the class, the constructor creates a new object with the class shape and initializes it with the values passed to it.
- **Accessors** are a type of function that you use to `get` or `set` the value of properties. Properties can be read-only by simply omitting the `set` accessor in the class, or inaccessible by omitting the `get` accessor (the property will return `undefined` if you attempt to access it, even if it's assigned a value during initialization.)
- **Methods** are functions that define the behaviors or actions that the object can do. You can call these methods to invoke the behavior of the object. You can also define methods that are only accessible from within the class itself and are typically called by other methods in the class to perform a task.

## Design notes

You can create classes to model data, encapsulate functionality, provide templates, and numerous other uses. As such, the components listed above are **not** required in every class you create. You may only need methods and a constructor for a utility object, or solely properties to manage data.

### Note

Accessors typically only used when you need to control access to values, such as implementing validation, or to calculate values dynamically. If your class is strictly storing values and does not need the additional functionality provided by accessors you can use properties.

## Next unit: Exercise- Create a class

[Continue >](#)

How are we doing? ☆ ☆ ☆ ☆ ☆