Exercise- Create a class

5 minutes

To create a class, define its members: properties, a `constructor`, accessors, and methods.

Let's create a new class called `Car`. You can use the `Car` class on its own to create basic `Car` objects, or you can extend the `Car` class to create new classes for specific types of cars, like a `GasCar` or an `ElectricCar` class. These classes will inherit the properties and methods of the `Car` class, as well as have their own properties and methods.

1. Open the Playground and remove any existing code.
2. Create a new `class` by using the `class` keyword followed by the class name, `Car`. By convention, class names are PascalCase. Let's also add some comments to make it easier to add the class members in the correct places.

TypeScript

```
class Car {
    // Properties

    // Constructor

    // Accessors

    // Methods

}
```

Declare the class properties

You can think of the properties of a class as the raw data that is passed to the object when it's initialized.

The properties of the `Car` class are those that apply to any car, regardless of the specific make or model. For example, these properties may include the make of the car, the color, and the

number of doors. Because you're working in TypeScript, you can also apply type attributes to the properties.

1. Declare the three properties for the `Car` class: `_make: string`, `_color: string`, and `_doors: number`.

TypeScript

```
// Properties
_make: string;
_color: string;
_doors: number;
```

Define the class constructor

Classes in TypeScript create two separate types: the instance type, which defines what members an instance of a class has, and the `constructor` function type, which defines what members the class `constructor` function has. The `constructor` function type is also known as the "static side" type because it includes static members of the class.

Using a `constructor` can simplify classes and make them easier to manage when you're working with many classes.

A `constructor` function initializes the properties of the class and has three parts:

- The `constructor` keyword.
- A parameter list, which defines the parameters that will be passed to the new object when a new instance is created. When defining the parameter list, remember that:
  - It's not required to define a parameter for every property in the class.
  - As with all TypeScript functions, the parameters can be required or optional, have default values, or be rest parameters. (This is a key difference from JavaScript.)
  - The parameter names can be different from the property names. Keep in mind that these names will appear in Intellisense when you work with objects of this type so use names that are sufficiently descriptive.
- The property assignments. Each statement assigns the value of a parameter to the value of a property. To indicate that you're accessing a member of the class (in this case, the property), apply the `this.` keyword.

A class may contain at most one `constructor` declaration. If a class contains no `constructor`

declaration, an automatic constructor is provided.

Continue defining the `Car` class in the Playground.

1. Create the `constructor` for the `Car` class. Start with the `constructor` keyword and then define the parameters and types that will be passed to the new `Car` object when a new instance is created. For the `Car` class, define one parameter for each of the three properties and annotate it with the type. Make the `doors` parameter optional with a default value of `4`.

2. Inside the code block for the `constructor`, assign a parameter value to each property (for example, `this._make = make`). In this case, set it to the value of the associated parameter but note that you can assign any expression that returns the required type.

```
TypeScript
```

```
// Constructor
constructor(make: string, color: string, doors = 4) {
    this._make = make;
    this._color = color;
    this._doors = doors;
}
```

Tip

The underscore (_) before the property name isn't required in the property declaration but it provides a way to distinguish the property declaration from the parameters that are accessible through the constructor, while still tying the two together visually.

Define the accessors

While you can access the class properties directly (they're `public`, by default), TypeScript supports getters/setters as a way of intercepting access to a property. This gives you finer-grained control over how a member is accessed on each object.

To `set` or return the value of the object's members from code, you must define `get` and `set` accessors in the class.

Continue defining the `Car` class in the Playground.

1. Define a `get` block for the `make` parameter that returns the value of the `_make` property.

```
TypeScript

// Accessors
get make() {
    return this._make;
}
```

2. Define a `set` block for the `make` parameter that sets the value of the `_make` property to the value of the `make` parameter.

```
TypeScript

set make(make) {
    this._make = make;
}
```

3. You can also use `get` and `set` blocks to validate data, impose constraints, or perform other manipulation of the data before you return it to the program. Define `get` and `set` blocks for the `color` parameter, but this time, return a string concatenated to the value of the `_color` property.

```
TypeScript

get color() {
    return 'The color of the car is ' + this._color;
}
set color(color) {
    this._color = color;
}
```

4. Define `get` and `set` blocks for the `doors` parameter. Before returning the value of the `_doors` property, verify that the value of the `doors` parameter is an even number. If not, throw an error.

```
TypeScript

get doors() {
```

```
    return this._doors;
}
set doors(doors) {
    if ((doors % 2) === 0) {
        this._doors = doors;
    } else {
        throw new Error('Doors must be an even number');
    }
}
```

Define the class methods

You can define any TypeScript function within a class and call it as a method on the object or from other functions within the class. These class members describe the behaviors that your class can perform and can perform any other tasks required by the class.

Continue defining the `Car` class in the Playground.

1. Define these four methods for the `Car` class: `accelerate`, `brake`, `turn`, and `worker`. You'll notice that there's no `function` keyword. This isn't required or allowed when defining functions in a class, so it helps you keep the syntax succinct.

TypeScript

```
// Methods
accelerate(speed: number): string {
    return `${this.worker()} is accelerating to ${speed} MPH.`
}
brake(): string {
    return `${this.worker()} is braking with the standard braking system.`
}
turn(direction: 'left' | 'right'): string {
    return `${this.worker()} is turning ${direction}`;
}
// This function performs work for the other method functions
worker(): string {
    return this._make;
}
```

Next unit: Exercise - Instantiate a class

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆