100 XP

Exercise - Instantiate a class

5 minutes

At this point, you have a class named `Car` that has three properties, and you can get and set the value of those properties. It also has four methods. Now, you can instantiate the `Car` class using the `new` keyword and pass parameters to it, creating a new `Car` object.

Continue working in the Playground.

1. Below the class declaration, declare a variable called `myCar1` and assign a new `Car` object to it, passing in values for the `make`, `color`, and `doors` parameters (make sure that the `doors` parameter is assigned an even number.)

```
TypeScript

let myCar1 = new Car('Cool Car Company', 'blue', 2);  // Instantiates the Car object with all parameters
```

2. You can now access the properties of the new `myCar1` object. Enter `myCar1.` and you should see a list of the members defined in the class, including `color` and `_color`. Select **Run** to return the value of both properties to the console. What happens? Why?

```
TypeScript

console.log(myCar1.color);
console.log(myCar1._color);
```

3. The member `_color` represents the property defined in the class, while `color` is the parameter that you pass to the constructor. When you refer to `_color`, you're accessing the raw data for the property, which returns `'blue'`. When you refer to `color`, you're accessing the property through the `get` or `set` accessor, which returns `'The color of the car is blue'`. It's important to understand the difference between the two because you often do not want to allow direct access to the property without doing some validation or other work on the data before getting or setting it. You'll learn about using access modifiers to control the visibility of class members later in the unit.

4. Recall that the `set` block for the `doors` parameter tests the value to determine if it is even or odd. Test this by declaring a variable called `myCar2` and assigning a new `Car` object to it, passing in values for the `make`, `color`, and `doors` parameters. This time set the value of the doors parameter to an odd number. Now, select **Run**. What happens? Why?

```
TypeScript
let myCar2 = new Car('Galaxy Motors', 'red', 3);
```

5. Although you passed an odd number to `doors`, it compiles and runs without errors because no data validation occurs in the `constructor`. Try setting the value of `doors` to another odd number (for example, `myCar2.doors = 5`) and test it. This should invoke the `set` block and throw an error. If you want to perform this validation step when the `Car` object is initialized, you should add a validation check to the `constructor`.

```
TypeScript
constructor(make: string, color: string, doors = 4) {
    this._make = make;
    this._color = color;
    if ((doors % 2) === 0) {
        this._doors = doors;
    } else {
        throw new Error('Doors must be an even number');
    }
}
```

6. Test the optional parameter `doors` by omitting it from the object initialization.

```
TypeScript
let myCar3 = new Car('Galaxy Motors', 'gray');
console.log(myCar3.doors);  // returns 4, the default value
```

7. Test the methods by sending the return values to the console.

```
TypeScript
```

```
console.log(myCar1.accelerate(35));
console.log(myCar1.brake());
console.log(myCar1.turn('right'));
```

Next unit: Access modifiers

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆