Class Inheritance

To create a class inheritance, use the `extends` keyword.A class created with a class inheritance inherits all the methods from another class:

Example

Create a class named "Model" which will inherit the methods from the "Car" class:

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}

let myCar = new Model("Ford", "Mustang");
document.getElementById("demo").innerHTML = myCar.show();
```

- The `super()` method refers to the parent class.
- By calling the `super()` method in the constructor method, we call the parent's constructor method. This gets us access to the parent's properties and methods.

Inheritance is useful for code reusability: reuse properties and methods of an existing class when you create a new class.

Getters and Setters

Classes also allow you to use getters and setters.

It can be smart to use getters and setters for your properties, especially if you want to do something special with the value before returning them, or before you set them.

To add getters and setters in the class, use the `get` and `set` keywords.

Example

Create a getter and a setter for the "carname" property:

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  get cnam() {
    return this.carname;
  }
  set cnam(x) {
    this.carname = x;
  }
}

const myCar = new Car("Ford");

document.getElementById("demo").innerHTML = myCar.cnam;
```

Note: even if the getter is a method, you do not use parentheses when you want to get the property value.

The name of the getter/setter method cannot be the same as the name of the property, in this case `carname`.

Programmers generally use an underscore character _ before the property name to separate the getter / setter from the actual property:

Example

You can use the underscore character to separate the getter/setter from the actual property:

```
class Car {
  constructor(brand) {
    this._carname = brand;
  }
  get carname() {
    return this._carname;
  }
  set carname(x) {
    this._carname = x;
  }
}
const myCar = new Car("Ford");
document.getElementById("demo").innerHTML = myCar.carname;
```

To use a *setter*, use the same syntax as when you set a property value, without parentheses:

Example

Use a setter to change the carname to "Volvo":

```
class Car {
  constructor(brand) {
    this._carname = brand;
  }
  get carname() {
    return this._carname;
  }
  set carname(x) {
    this._carname = x;
  }
}

const myCar = new Car("Ford");
myCar.carname = "Volvo";
document.getElementById("demo").innerHTML = myCar.carname;
```

Unlike functions and other JavaScript declarations, you must declare a class before you can use it:

Example

```
//You cannot use the class yet.
//myCar = new Car("Ford") will raise an error.

class Car {
  constructor(brand) {
    this.carname = brand;
  }
}

//Now you can use the class:
const myCar = new Car("Ford");
```