OOJS

Carolina Caballero

Agenda

- Object Definitions
- Object Properties
- Object Methods
- Object Prototypes
- Getters and Setters
- Inheritance

Agenda

- Object Definitions
- Object Properties
- Object Methods
- Object Prototypes
- Getters and Setters
- Inheritance

Object Definitions

OOJS

- Means “Object Oriented JavaScript”
- Is a JavaScript library for working with objects.
- Features include inheritance, mixins, static inheritance and additional utilities for working with objects and arrays

JavaScript Objects

- **In JavaScript, almost "everything" is an object.**
  - Booleans can be objects (if defined with the **new** keyword)
  - Numbers can be objects (if defined with the **new** keyword)
  - Strings can be objects (if defined with the **new** keyword)
  - Dates are always objects
  - Maths are always objects
  - Regular expressions are always objects
  - Arrays are always objects
  - Functions are always objects
  - Objects are always objects

Variables Vs Objects

- JavaScript variables can contain single values:

```
var person = "John Doe";
```

- Objects are variables too. But objects can contain many values.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Object Properties

- A JavaScript Object is a collection of **properties** (name:value).

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

- Object properties can be: primitive values, other objects, and functions.

Object Methods

- Methods are **actions** that can be performed on objects.
- An **object method** is an object property containing a **function definition**.

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

Creating a JavaScript Object

- There are different ways to create new objects:
  - Using object literal
  - With keyword **new**
  - By constructor

Using an Object Literal

- This is the easiest way to create a JavaScript Object.
- Using an object literal, you both define and create an object in one statement.
- An object literal is a list of name:value pairs inside curly braces {}.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Using an Object Literal …

- Spaces and line breaks are not important.

```
var person = {
    firstName:"John",
    lastName:"Doe",
    age:50,
    eyeColor:"blue"
};
```

Exercise – like Object Literal

Company

Name: "Neitek Solutions"
Address: "Torre Diamante 1007 Local 724"
City: "Monterrey'
Telephone: "811 365 02 43"
BusinessType: "Computer Systems"

School

Name: "Tecnologico de Ciudad Madero"
Carreers:
"Ing Sistemas Computacionales",
"Ing. Electrica",
"Ing. Electronica"

Using the Keyword new

- This is the way to create a new JavaScript object with keyword new:

```
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

- This and previous example do the same.
- Both are limited, because they only create a single object.

Exercise – using **new**

Company

Name: "Neitek Solutions"
Address: "Torre Diamante 1007 Local 724"
City: "Monterrey'
Telephone: "811 365 02 43"
BusinessType: "Computer Systems"

School

Name: "Tecnologico de Ciudad Madero"
Carreers:
"Ing Sistemas Computacionales",
"Ing. Electrica",
"Ing. Electronica"

Using an Object Constructor

- The standard way to create an "object type" is to use an object constructor function:

```
function person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
}
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
```

- This function is an object constructor.

Using an Object Constructor…

- Once you have an object constructor, you can create new objects of the same type:

```
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
```

- File: Sample01.html

The *this* Keyword

- In JavaScript, the thing called **this**, is the object that "owns" the JavaScript code.
- The value of **this**, when used in a function, is the object that "owns" the function.
- The value of **this**, when used in an object, is the object itself.

JavaScript Objects are Mutable

- Objects are mutable: They are addressed by reference, not by value.

```
var x = person;  // This will not create a copy of person.
```

```
var person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"}

var x = person;
x.age = 10;           // This will change both x.age and person.age
```

- File: Sample02.html

Exercise – using constructor

Company1

Name: "Neitek Solutions"
Address: "Torre Diamante 1007 Local 724"
City: "Monterrey'
Telephone: "811 365 02 43"
BusinessType: "Computer Systems"

Company2

Name: "Yoga&You"
Address: "Prolongación San Alberto 417 Local 2"
City: "Monterrey'
Telephone: "81 2139 2246"
BusinessType: "fitnessCenter"

- Declare a function Company
- Make 2 variables for both companies
- Correct the *city in company2*

neitek_

Agenda

- Object Definitions
- Object Properties
- Object Methods
- Object Prototypes
- Getters and Setters
- Inheritance

Object Properties

JavaScript Object Properties

- Properties are the values associated with a JavaScript object.
- A JavaScript object is a collection of unordered properties.
- Properties can usually be changed, added, and deleted, but some are read only.
- You cannot use reserved words for property (or method) names. JavaScript naming rules apply.

Accessing JavaScript Properties

- The syntax for accessing the property of an object is:

```
objectName.property          // person.age
```

- Or

```
objectName["property"]       // person["age"]
```

- Or

```
objectName[expression]       // x = "age"; person[x]
```

- File: Sample03.html

Properties in a Loop

- The JavaScript for...in statement loops through the properties of an object.

```
var person = {fname:"John", lname:"Doe", age:25};

for (x in person) {
    txt += person[x];
}
```

- File: Sample04.html

Adding New Properties

- You can add new properties to an existing object by simply giving it a value.

```
person.nationality = "English";
```

- File: Sample05.html

Deleting Properties

- The **delete** keyword deletes a property from an object.
- The delete keyword deletes both the value of the property and the property itself.
- After deletion, the property cannot be used before it is added back again.
- The delete operator is designed to be used on object properties. It has no effect on variables or functions.
- The **delete** operator should not be used on predefined JavaScript object properties. It can crash your application.

Predefined Properties

- **arguments**: array with all arguments in the invocation
- **arguments.length:** number of arguments for function
- **arguments.callee** – the function it self
- **constructor**
- **prototype**

```
var myF = function (x) {
   alert('Hola Mundo');
   console.log ("arg.len=" + arguments.length);
   console.log ("arg[0]=" + arguments[0]);
   console.log (arguments.callee);
}

myF(421);
console.dir (myF.constructor);
console.dir (myF.prototype);
```

- File: Sample5a.html

Agenda

- Object Definitions
- Object Properties
- Object Methods
- Object Prototypes
- Getters and Setters
- Inheritance

Object Methods

JavaScript Methods

- JavaScript methods are the actions that can be performed on objects.
- A JavaScript **method** is a property containing a **function definition**.

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

Creating an Object Method

- You create an object method with the following syntax:

```
methodName : function() { code lines }
```

```
var person = {
    firstName: "John",
    lastName : "Doe",
    id       : 5566,
    fullName : function() {
       return this.firstName + " " + this.lastName;
    }
};
```

Accesing an Object Method

- You access an object method with the followina syntax:

```
objectName.methodName()
```

- To invoke the function:

```
name = person.fullName();
```

- To get the function definition:

```
name = person.fullName;
```

- Files: Sample06.html, Sample07.html

Adding new methods

- Adding methods to an object is done inside the constructor function:

```
function person(firstName, lastName, age, eyeColor) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
    this.eyeColor = eyeColor;
    this.changeName = function (name) {
        this.lastName = name;
    };
}
```

- File: Sample08.html

Predefined Methods

- **call** – invoke a function with an owner as the first argument, with call () you can use a method belonging to another object.
- **apply** – similar to call.

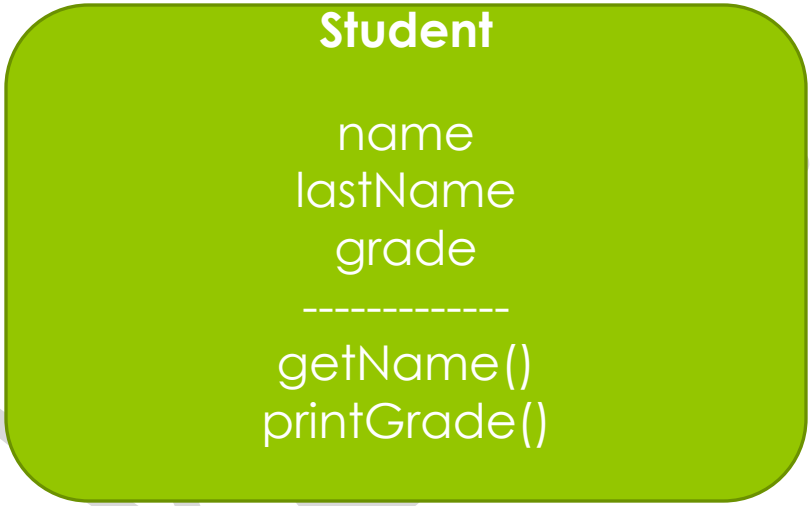
```
theFunction.apply(valueForThis, arrayOfArgs)
```

```
theFunction.call(valueForThis, arg1, arg2, ...)
```

- **toSource** – returns the source code as string
- **toString** – a string representing the object
- **valueOf** – primitive value of the specified object

- File: Sample08a.html

Exercise



- Create a funcion Student
- Implement getName and printGrade functions as you wish
- Create 2 vars for different students
- Print on screen getName and printGrade functions

neitek_

Agenda

- Object Definitions
- Object Properties
- Object Methods
- Object Prototypes
- Getters and Setters
- Inheritance

Object Prototypes

Object Prototypes

- Every JavaScript object has a prototype. The prototype is also an object.
- All JavaScript objects inherit their properties and methods from their prototype. Objects created using an object literal, or with new Object(), inherit from a prototype called Object.prototype.
- Objects created with new Date() inherit the Date.prototype.
- The Object.prototype is on the top of the prototype chain.
- All JavaScript objects (Date, Array, RegExp, Function, ...) inherit from the Object.prototype.

Metodos de las Funciones

- **apply** – Permite pasar argumentos más fácilmente
- **call** – Llama una función dentro de un contexto diferente.
- **toSource** – Regresa la fuente de la función como String.
- **toString**
- **valueOf**

Creating a Prototype

- The standard way to create an object prototype is to use an object constructor function:

```
function Person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
}
```

Creating a Prototype…

- With a constructor function, you can use the **new** keyword to create new objects from the same prototype:

```
var myFather = new Person("John", "Doe", 50, "blue");
var myMother = new Person("Sally", "Rally", 48, "green");
```

- The constructor function is the prototype for Person objects.
- It is considered good practice to name constructor function with an upper-case first letter.
- File: Sample09.html

Adding a Method to an Object

- Adding a new **property** to an existing object is easy:

```
myFather.nationality = "English";
```

  - File: Sample10.html
- Adding a new **method** to an existing object is also easy:

```
myFather.name = function () {
    return this.firstName + " " + this.lastName;
};
```

Adding Properties to a Prototype

- Wrong way:

```
Person.nationality = "English";
```

- Right way:

```
function Person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
    this.nationality = "English";
}
```

- File: Sample12.html

Adding Methods to a Prototype

- Define methods in construction function like this:

```
function Person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
    this.name = function() {return this.firstName + " " +
this.lastName;};
}
```

- File: Sample13.html

Using the prototype Property

- The JavaScript prototype property allows you to add new properties to an existing prototype:

```
function Person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
}
Person.prototype.nationality = "English";
```

- File: Sample14.html

Using the prototype Property…

- The JavaScript prototype property also allows you to add new methods to an existing prototype:

```
function Person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
}
Person.prototype.name = function() {
    return this.firstName + " " + this.lastName;
};
```

- File: Sample15.html

neitek_

Agenda

- Object Definitions
- Object Properties
- Object Methods
- Object Prototypes
- Getters and Setters
- Inheritance

Getters and Setters

Get&Set with methods

```
var address = {
  street: "No Street",
  city:   "No City",
  state:  "No State",

  get getAddress ()
  {
    return this.street + ", " + this.city + ", " + this.state;
  },

  set setAddress(theAddress)
  {
    var parts = theAddress.toString().split (", ");
    this.street = parts[0] || "" ;
    this.city   = parts[1] || "" ;
    this.state  = parts[2] || "" ;
  }
}
```

- File: Sample16.html

Get&Set with keywords

```
var address = {
  street: "No Street",
  city:   "No City",
  state:  "No State",

  get getAddress ()
  {
    return this.street + ", " + this.city + ", " + this.state;
  },

  set setAddress(theAddress)
  {
    var parts = theAddress.toString().split (", ");
    this.street = parts[0] || "" ;
    this.city   = parts[1] || "" ;
    this.state  = parts[2] || "" ;
  }
}
```

- File: Sample17.html

Get&Set with **call**

```
function Coordinates() {
  this.latitude  = 0;
  this.longitude =  0;
}
Object.__defineGetter__.call (Coordinates.prototype, "getCoords",
  function(){
   return "Lat: " + this.latitude + " Long: " +  this.longitude;
  }
);

Object.__defineSetter__.call (Coordinates.prototype, "setCoords",
  function (coords){
    var parts = coords.toString().split(", ");
    this.latitude  = parts[0] || "";
    this.longitude = parts[1] || "";
  }
);
```

- File: Sample18.html

Get&Set with defineProperty

```
function Point() {
  this.xPos  = 0;
  this.yPos  =  0;
}

Object.defineProperty (Point.prototype, "pointPos", {
   get: function(){
      return "X: " + this.xPos + " Y: " + this.yPos;
   },
   set: function(thePoint){
      var parts = thePoint.toString().split(", ");
      this.xPos = parts[0] || "";
      this.yPos = parts[1] || "";
   }
});
```

- File: Sample19.html

Get&Set - ECMAScript 5

```
var Circle = function (radius) {
  this._radius = radius;
}

Circle.prototype = {
  set radius(radius) { this._radius = radius; },
  get radius() { return this._radius; },
  get area() { return Math.PI * (this._radius * this._radius); }
}
```

- File: Sample20.html

Exercise

Person
name
lastName
age
---
toString()

- Declare Person object
- Implement 3 different ways of setters and getters.

Agenda

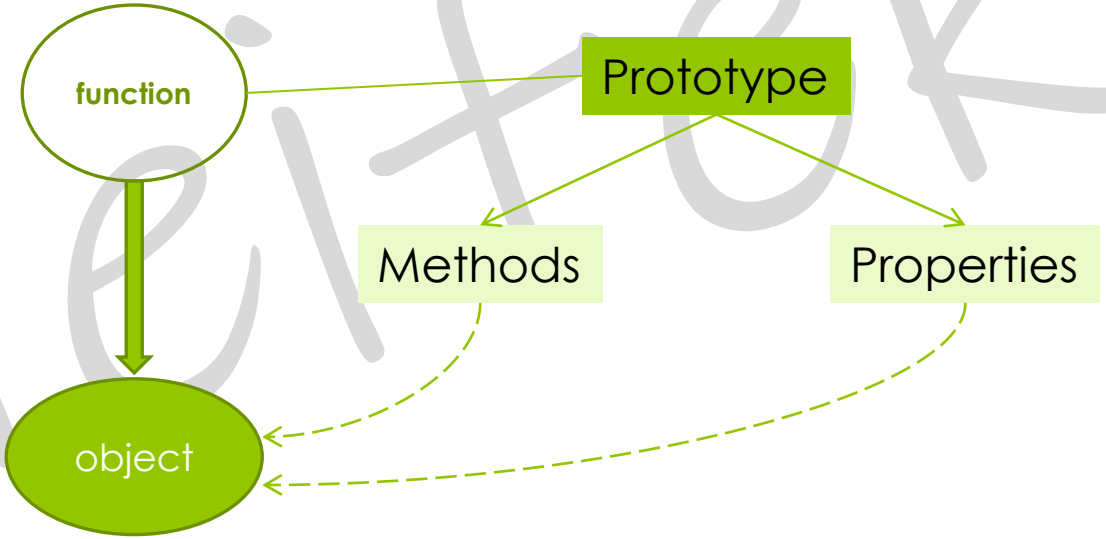
- Object Definitions
- Object Properties
- Object Methods
- Object Prototypes
- Getters and Setters
- Inheritance

Inheritance

Prototype inheritance

- An object has a number of properties. *This includes any attributes or functions(methods)*.
- An object has a special parent property, this is also called the prototype of the object(__proto__). An object inherits all the properties of its parent.
- An object can override a property of its parent by setting the property on itself.
- A constructor creates objects. Each constructor has an associated prototype object, which is simply another object.
- When an object is created, it's parent is set to the prototype object associated with the constructor that created it.

Prototype Concept

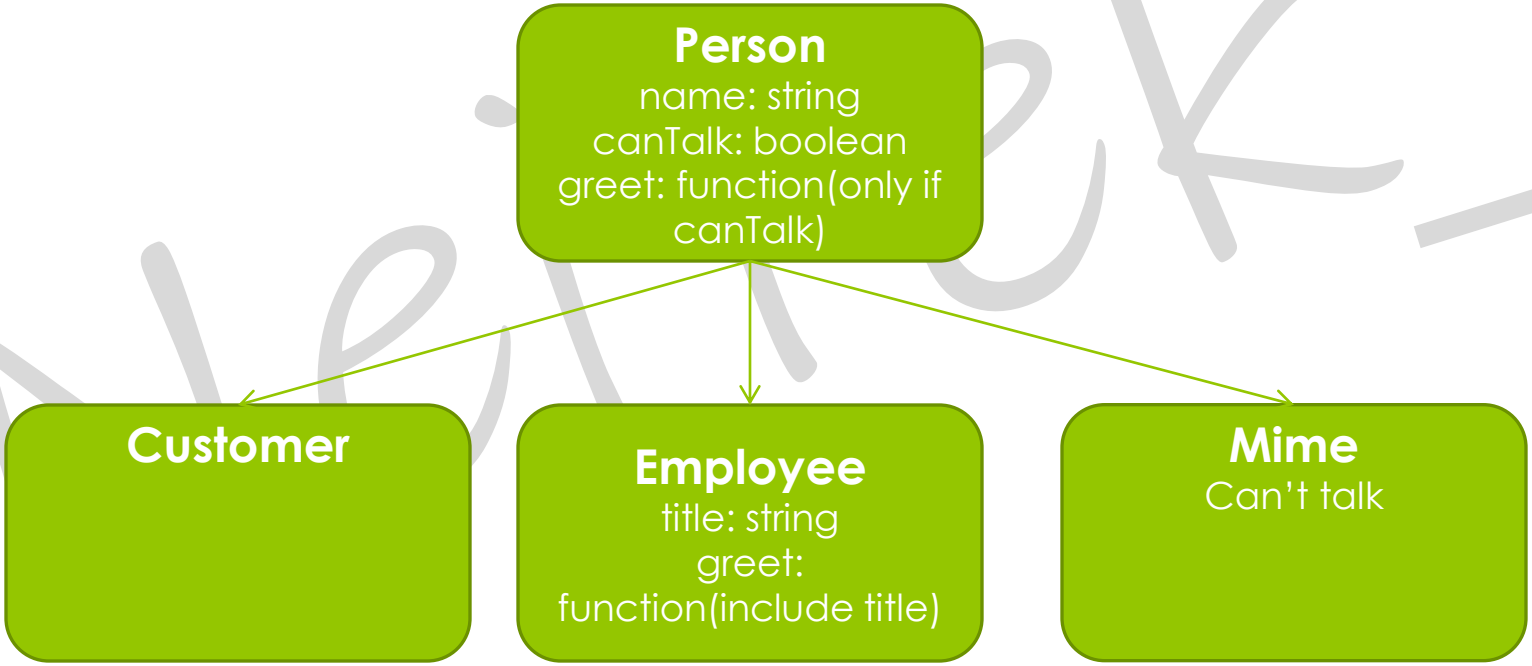


Inheritance

```
function A_Object() {
  this.name = "A";
  this.sayHi = function() {
    return "Hi, I'm " + this.name;
  }
}

function B_Object (aName){
  this.name = aName;
}

B_Object.prototype = new A_Object();
B_Object.prototype.constructor = B_Object;

a = new A_Object();
b = new B_Object("B");

document.write (a.sayHi() + "<br/>");
document.write (b.sayHi() + "<br/>");
```

- File: Sample21.html

Exercise



Using “Class” keyword

```
class A_Class {
  constructor (name) {
    this.name = name;
  }
  toString(){
   return  "A Class name  is: " + this.name;
  }
  static getInstance() {
     return new A_Class("A");
  }
}
```

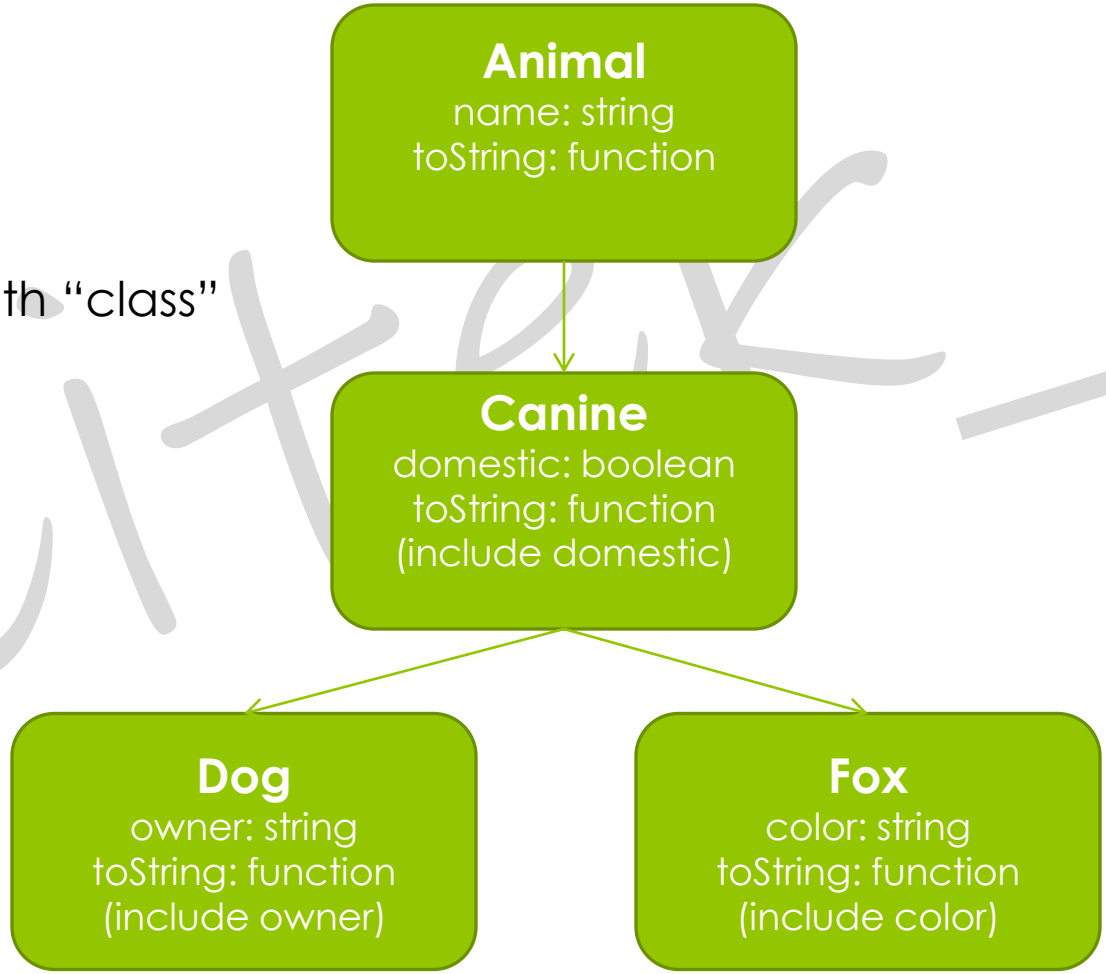
Using “Class” keyword

```
class B_Class extends A_Class{
  constructor (name, otherValue) {
    super (name);
    this.otherValue = otherValue;
  }
  toString(){
   return  "B Class name is: " + this.name
           + " otherValue: " + this.otherValue;
  }
}
```

- File: Sample22.html

Exercise

- Implement with "class" keyword



PREGUNTAS

neitek_