More on “this”, JS Inheritance

IT TALENTS
Training Camp

What is “this”? – code snippet #0

```
var fullname = 'John Doe';
var obj = {
   fullname: 'Colin Ihrig',
   prop: {
      fullname: 'Aurelio De Rosa',
      getFullname: function() {
         return this.fullname;
      }
   }
};

console.log(obj.prop.getFullname());

var test = obj.prop.getFullname;

console.log(test());
```

IT TALENTS Training Camp

What is “this”? – code snippet #1

```
function bike() {
  console.log(this.name);
}

var name = "Ninja";
var obj1 = { name: "Pulsar", bike: bike };
var obj2 = { name: "Gixxer", bike: bike };

bike();
obj1.bike();
obj2.bike();
```

What is “this”? – code snippet #2

```
var obj1 = {
  name: "Pulsar",
  bike: function() {
    console.log(this.name);
  }
}
var obj2 = { name: "Gixxer", bike: obj1.bike };
var name = "Ninja";
var bike = obj1.bike;

bike();
obj1.bike();
obj2.bike();
```

What is “this”? – code snippet #3

```
function bike() {
  console.log(this.name);
}

var name = "Ninja";
var obj = { name: "Pulsar" }

bike();
bike.call(obj);
```

What is “this”? – code snippet #4

```
var bike = function() {
  console.log(this.name);
}
var name = "Ninja";
var obj1 = { name: "Pulsar" };
var obj2 = { name: "Gixxer" };

var originalBikeFun = bike;
bike = function() {
  originalBikeFun.call(obj1);
};

bike();
bike.call(obj2);
```

IT TALENTS Training Camp

What is “this”? – code snippet #5

```
let person = {
    name: 'John Doe',
    getName: function() {
        console.log(this.name);
    }
};

setTimeout(person.getName, 1000);
```

IT TALENTS Training Camp

Binding the context

- The **bind()** method creates a new function that, when called, has its this keyword set to the provided value, with a given sequence of arguments preceding any provided when the new function is called.

```
let person = {
    name: 'Pesho',
    age: 25,
    getName: function () {
        console.log(this.name);
    }
};

let myFunc = person.getName.bind(person);

setTimeout(myFunc, 1000);
```

call() vs apply() vs bind()

- Use .bind() when you want that function to later be called with a certain context, useful in events.
- Use .call() or .apply() when you want to invoke the function immediately, and modify the context.
- Call/apply call the function immediately, whereas bind returns a function that, when later executed, will have the correct context set for calling the original function.

IT TALENTS
Training Camp

More about Prototypes

- Prototypes allow you to easily define methods to all instances of a particular object. The beauty is that the method is applied to the prototype, so it is only stored in the memory once, but every instance of the object has access to it.

- Lets see some practical examples of using prototypes
  - Attach method to the Array prototype
  - Attach method to the String prototype

More Object methods

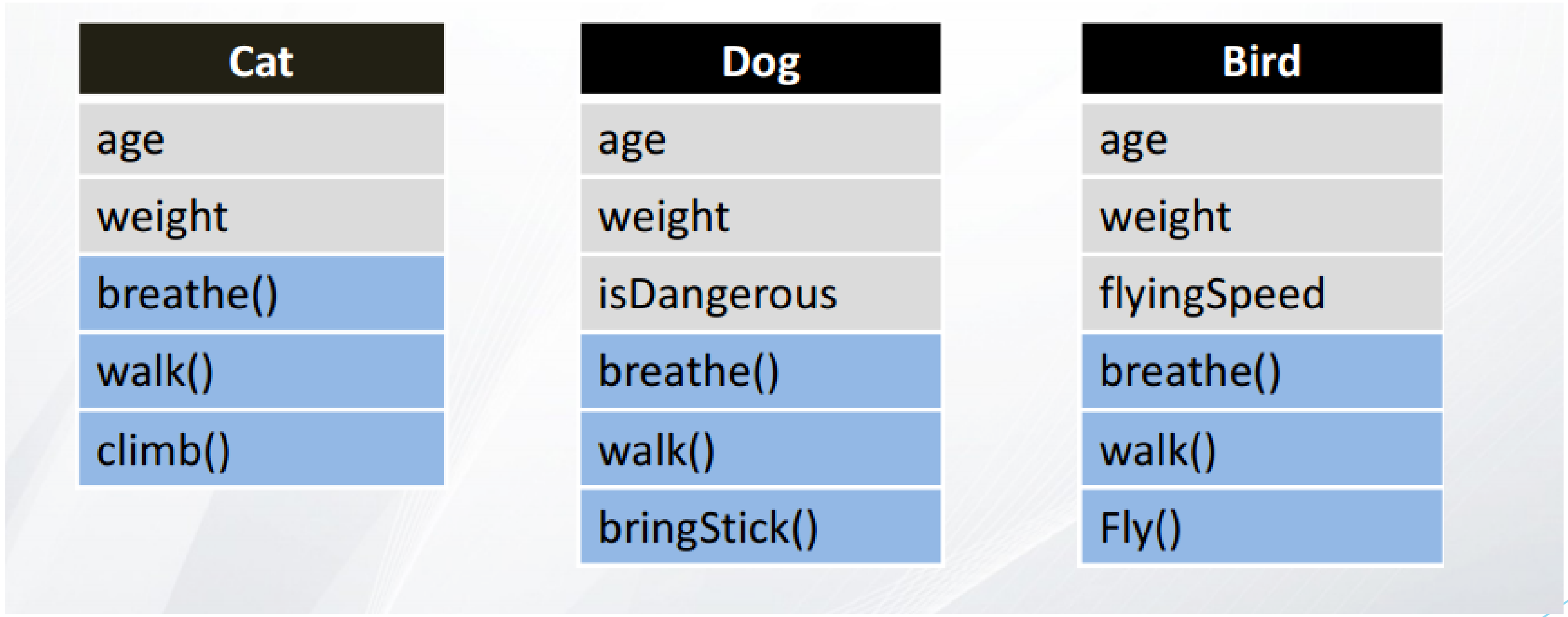
- **hasOwnProperty() -** method returns a boolean indicating whether the object has the specified property as its own property (as opposed to inheriting it).
- The **Object.assign()** method copies all enumerable own properties from one or more source objects to a target object. It returns the target object.

  /The easiest way to do a shallow copy of an Object/
- The **Object.create()** method creates a new object, using an existing object as the prototype of the newly created object.

Inheritance

Problem

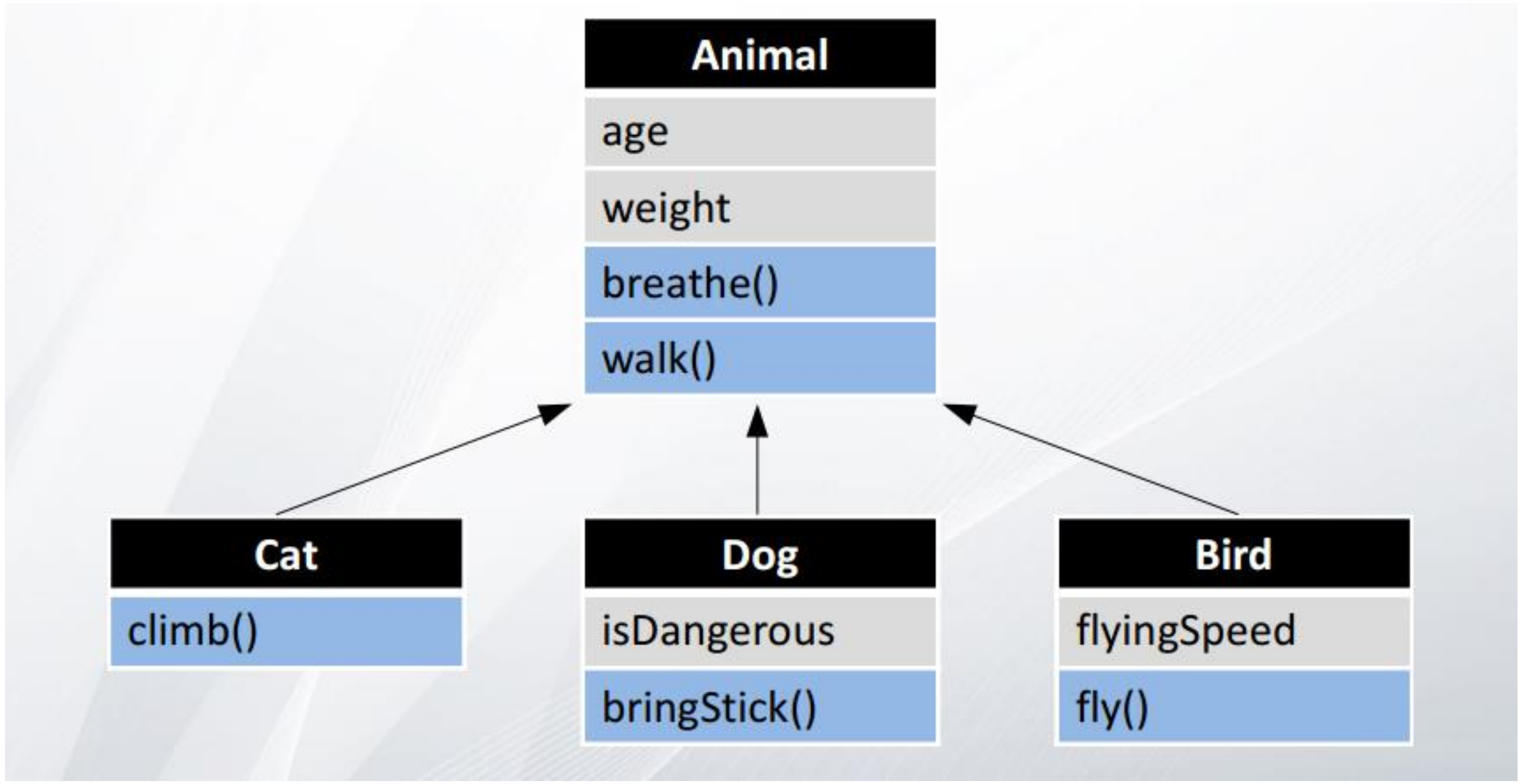
- Cats, dogs and birds have age and weight.
- They also can breathe and walk.
- Actually, they are animals, and every animal has age, weight and can breathe and walk.



Cat	Dog	Bird
age	age	age
weight	weight	weight
breathe()	isDangerous	flyingSpeed
walk()	breathe()	breathe()
climb()	walk()	walk()
	bringStick()	Fly()

Problem

- Is this the same?
- Yes, but the common logic is only in the class Animal



Inheritance in Javascript

- There are different types of inheritance which we can use.
  - Constructor stealing -> is a technique which we can use in order to call another constructor from within the initial constructor. Lets see an example:
  - Prototype chaining -> is a technique which we can use in order to chain the Prototypes.

Constructor Stealing

- Constructor stealing is achieved by using call or apply to pass the correct reference of “this”

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

function Student(name, age, score) {
    Person.call(this, name, age);
    this.score = score;
}

let pesho = new Student('Pesho', 20, 5);
```

Prototype Chaining

- Each object has a private property which holds a link to another object called its prototype. That prototype object has a prototype of its own, and so on until an object is reached with null as its prototype. By definition, null has no prototype, and acts as the final link in this prototype chain.
- The default prototype of every function is an empty object which is inherited from the prototype of Object
- You can break the prototype chain using the Object.create() function and attach the prototype you want to inherit

Inheritance looks ugly, right?

Don't judge so fast…

IT TALENTS
Training Camp

Welcome ES6 class

ES6+ new features

- Classes are just a syntactic sugar.
- We still don’t have real classes. They are using Prototypes under the hood
- It makes the code easier to write and understand
- Classes are not hoisted

Lets compare them:

```
var Shape = function (id, x, y) {
    this.id = id;
    this.move(x, y);
};
Shape.prototype.move = function (x, y) {
    this.x = x;
    this.y = y;
};
```

```
class Shape {
    constructor (id, x, y) {
        this.id = id
        this.move(x, y)
    }
    move (x, y) {
        this.x = x
        this.y = y
    }
}
```

IT TALENTS
Training Camp

Inheritance – ECMAScript 5

```
var Rectangle = function (id, x, y, width, height) {
    Shape.call(this, id, x, y);
    this.width  = width;
    this.height = height;
};
Rectangle.prototype = Object.create(Shape.prototype);
Rectangle.prototype.constructor = Rectangle;
var Circle = function (id, x, y, radius) {
    Shape.call(this, id, x, y);
    this.radius = radius;
};
Circle.prototype = Object.create(Shape.prototype);
Circle.prototype.constructor = Circle;
```

IT TALENTS
Training Camp

Inheritance - ECMAScript 6

```
class Rectangle extends Shape {
    constructor (id, x, y, width, height) {
        super(id, x, y)
        this.width  = width
        this.height = height
    }
}
class Circle extends Shape {
    constructor (id, x, y, radius) {
        super(id, x, y)
        this.radius = radius
    }
}
```