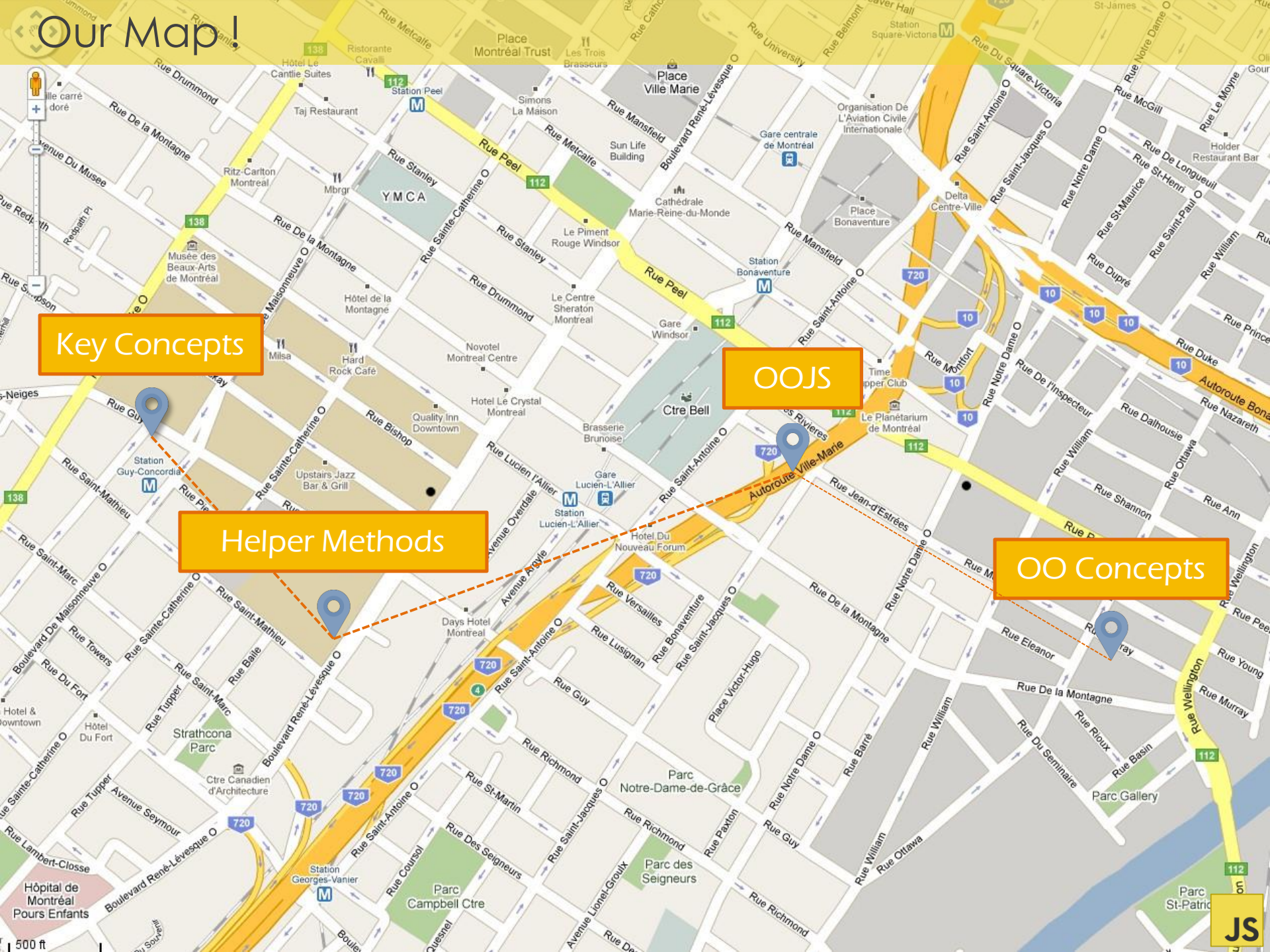
level

Object Oriented JavaScript

Is JavaScript an object oriented language or not ?

JS

Our Map !



Key Concepts

```
var str = 'ahmed';          //str is primitive

var len = str.length;       //str is now an object
```

```
function personName (firstName) {
  var nameIntro = "This person name is ";

   function addLastName (lastName) {
       return nameIntro + firstName + " " + lastName;
    }

  return addLastName;
}
```

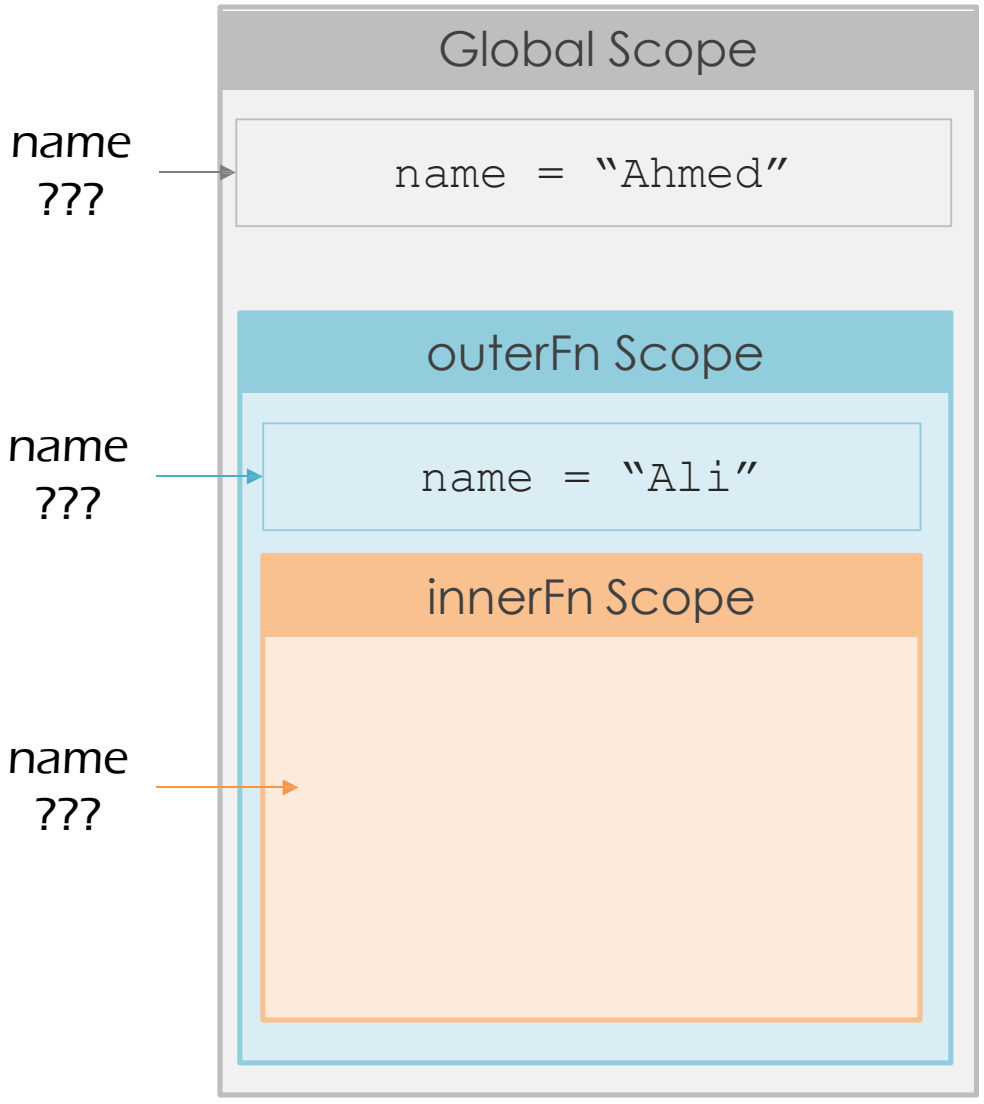
```
var addName = personName("Ibrahim");  //firstName = Ibrahim
addName("Mohsen");
//This person name is Ibrahim Mohsen
```

Key Concepts | Lexical Scope

```
var name = "Ahmed"

function outerFn(){
    var name = "Ali"
    function innerFn(){
        console.log(name)
    }
    innerFn()
}
outerFn()
console.log(name)
```

Output:

```
Ali
Ahmed
```



```
var employee = {
        'myName': "Ahmed",
        'getName': function(){
                console.log( this .myName);
        }};
person.getName();
employee.getName();
employee.getName();
```

Output

```
> "Ahmed"
> person is undefined
> "Ahmed"
```

```
window.myName = "Ahmed";
window.getMyName = function(){ console.log(this.myName);}
var person  = {
        'myName': "Mohamed",
        'getName': window.getMyName
};
person.getName();
person.getName();
```

Output

```
> undefined
> "Mohamed"
```

Key Concepts | context

```
var myName = "Ahmed";
var getMyName = function(){ console.log(this.myName);}
var Person   = {'myName': "Mohamed",
                'getMyName': function(){
                                window.getMyName()
                             }
               };
Person.getMyName();
```

window

Output

> "Ahmed"

Window context

myName = "Ahmed"

Person context

myName = "Mohamed"

And That is a problem,

But We will solve it later.

Helper Methods | call

```
fun.call(thisArg [, arg1[, arg2[, ...]]])
```

Example

```
var name = "Ahmed";

function getMyData (age , job){
        console.log(this.name+" is"+ job);
}

var Person = {
        name : "Mohamed"
}
getMyData(14, "a student");
            Ahmed is a student
getMyData.call(Person,24, "an engineer");
             Mohamed is an engineer
```

Key Concepts | context problem solving

```
var myName = "Ahmed";
var getMyName = function(){ console.log(this.myName);}
var Person   = {'myName': "Mohamed",
                'getMyName': function(){
                                window.getMyName.call(this)
                             }
               };
Person.getMyName();
```

person

Output

> "Mohamed"

Problem solved

Window context

myName = "Ahmed"

Person context

myName = "Mohamed"

Helper Methods | apply

```
fun.apply(thisArg [, [arg1,arg2,…] ])
```

Example

```
var name = "Ahmed";

function getMyData (age , job){
        console.log(this.name+" is"+ job);
}

var Person = {
        name : "Mohamed"
}

getMyData(14, "a student");
            Ahmed is a student

getMyData.apply(Person,[24, "an engineer"]);
             Mohamed is an engineer
```

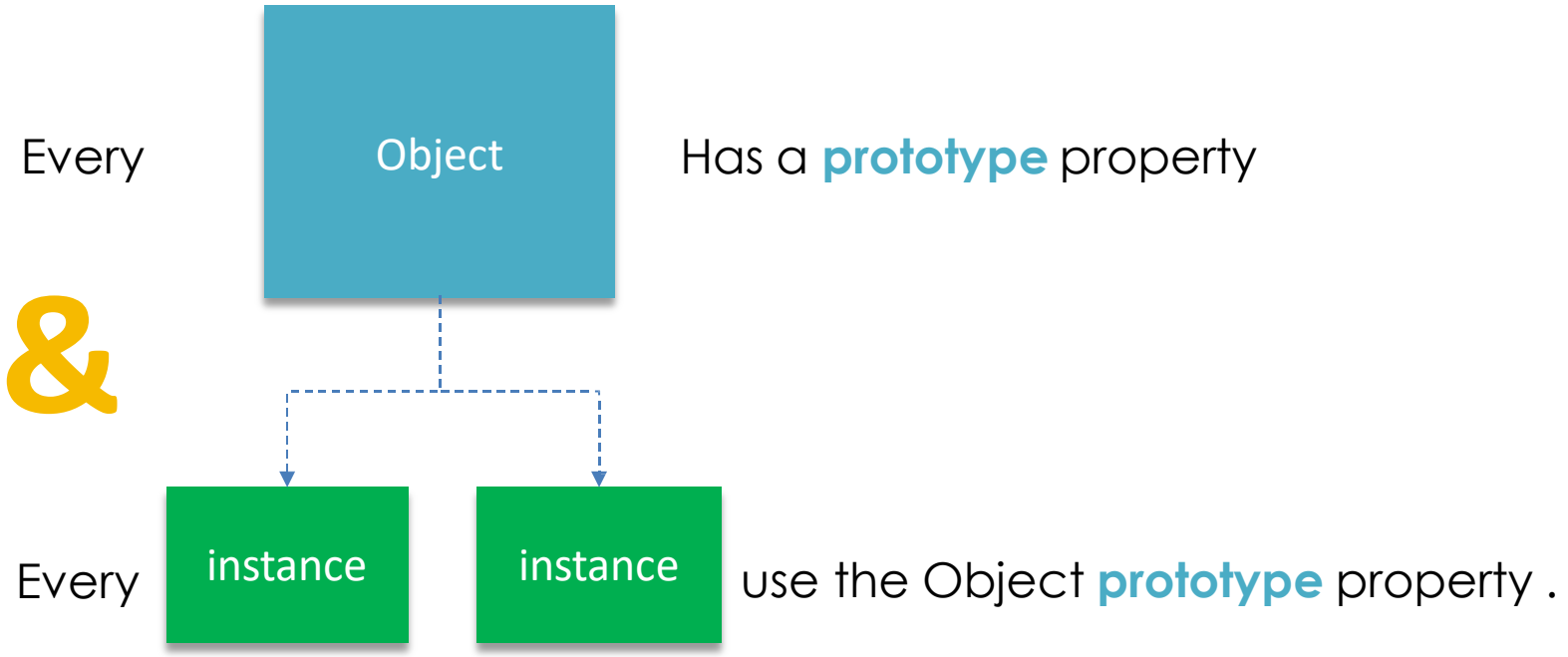
Helper Methods | bind

```
fun.bind(thisArg [, arg1[, arg2[, ...]]])
```

Example

```
var age = 11;

function getAge (){
        console.log("My Age is "+ this.age);
}

var Person = { age : 14}
var getPersonAge = getAge.bind(Person);
getAge();
            My Age is 11
getPersonAge();
            My Age is 14
```

Key Concepts | **prototype** property



Example

- Create a new instance of String Object.

```
var str = new String("ITI");
```

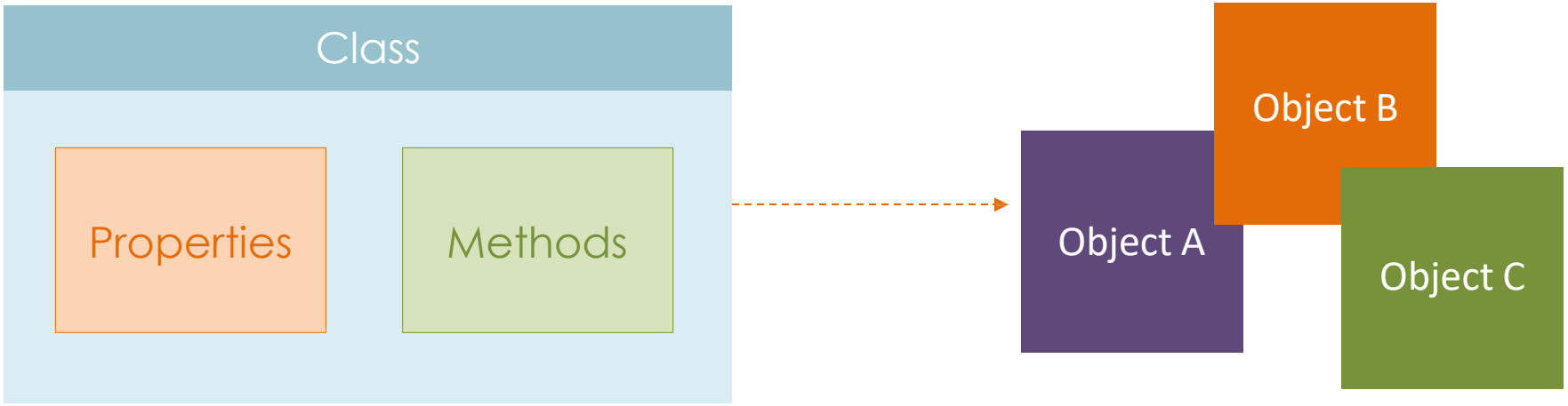
- Use a method that existed in the inherited prototype property of String Object.

```
str.toLowerCase();
```

Object Oriented Keywords (OOJS)

JS

A **class** is a template definition of an object's properties and methods.



JavaScript Implementation

```
var Animal = function () {};
var penguin = new Animal();

var elephant = new Animal();
```

penguin

elephant

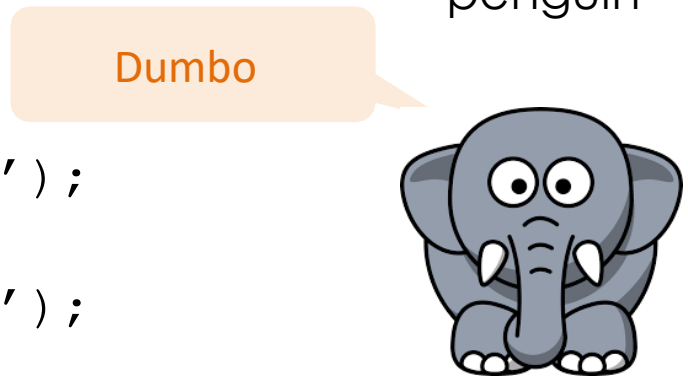
A method called at the moment an object is instantiated.

JavaScript Implementation

```
var Animal = function (name) {
        this.name = name;
};
```

penguin

```
var penguin = new Animal('amigos');

var elephant = new Animal('Dumbo');
```



elephant

Object **Property** is an object characteristic, such as name.

Object **Method** is an object capability, such as walk.

JavaScript Implementation

```
var Animal = function (name) {
        this.name = name;
        this.sayMyName =
        function(){
                console.log("My Name is "+this.name);
        };
};
var penguin = new Animal('Amigos');
penguin.sayMyName();
```

penguin

Object Oriented Concepts

JS

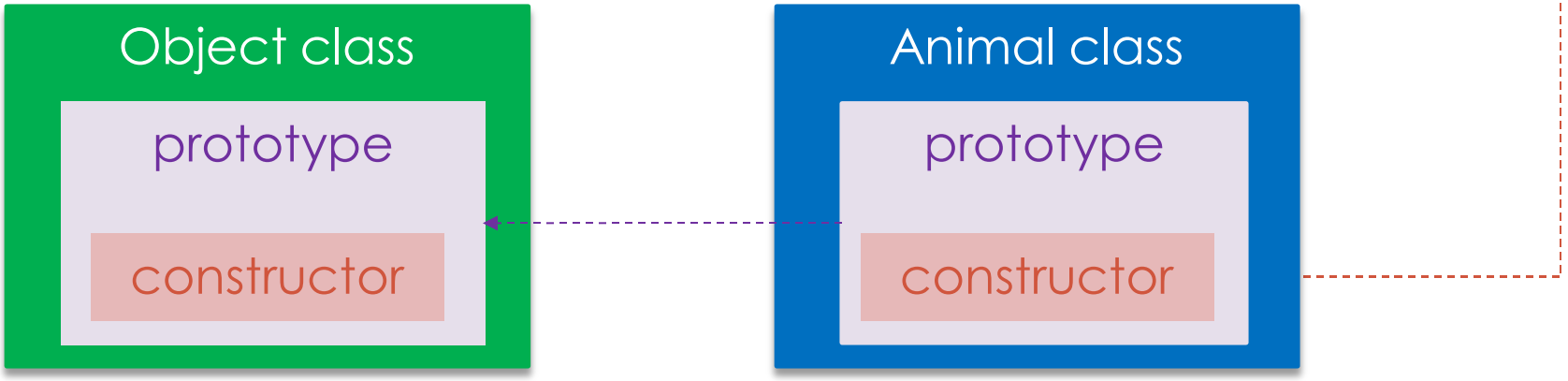
Inheritance

A class can inherit characteristics and capabilities from another class.

JavaScript Implementation

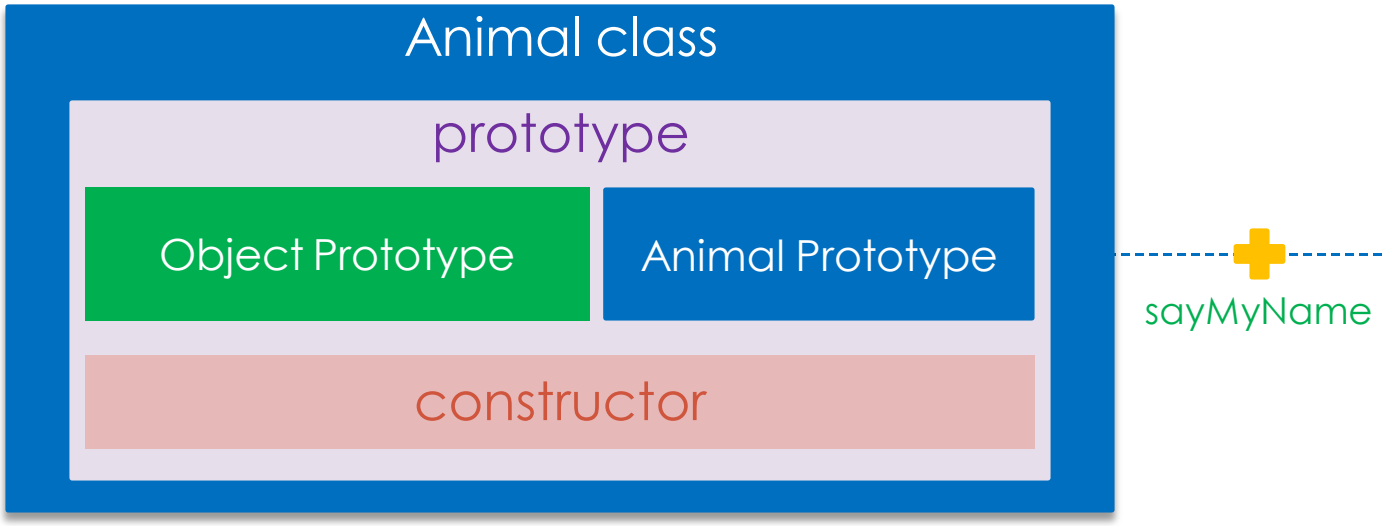
1 Create Parent Class:

```
var Animal = function (name) {
        this.name = name;
};
```



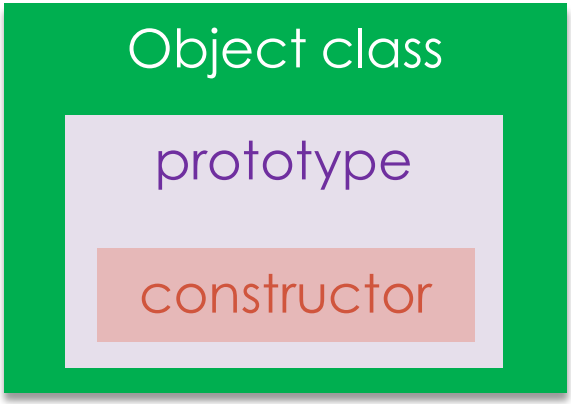
2 Use prototype property to define properties and methods:

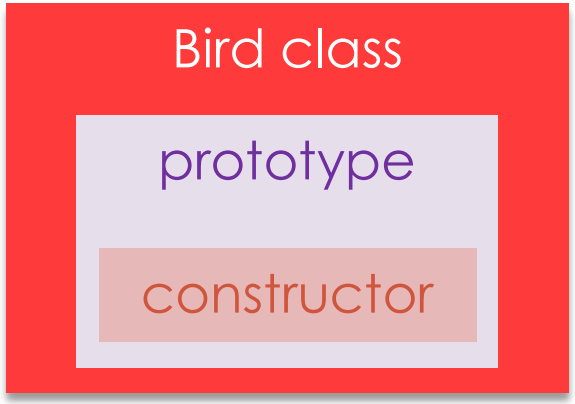
```
Animal.prototype.sayMyName = function(){
        console.log("My Name is "+this.name);
}
```



3 Create Child Class:

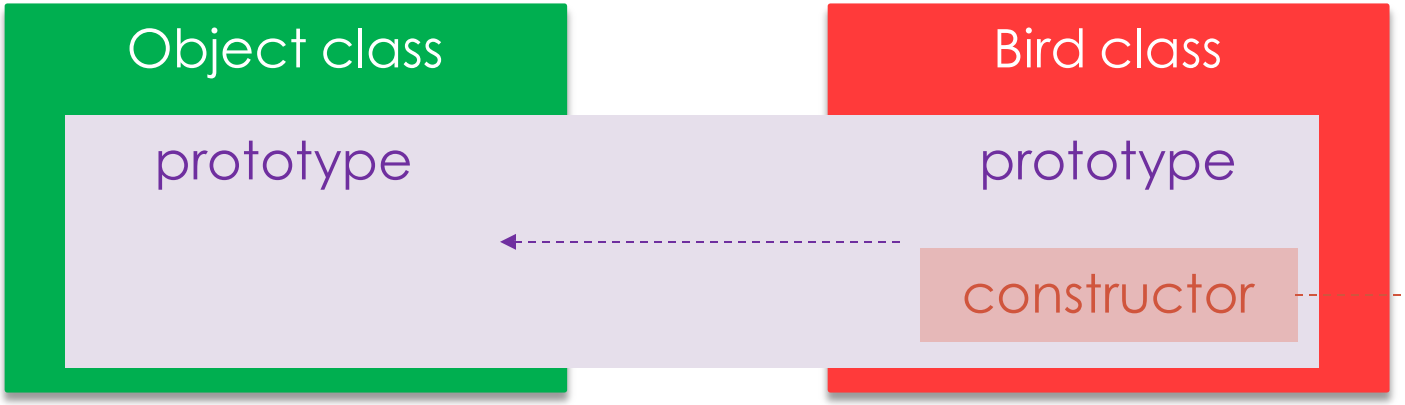
```
var Bird = function (name, canFly) {
        Animal.call(this,name);
        this.canFly = canFly;
};
```





3 Create Child Class:

```
var Bird = function (name, canFly) {
        Animal.call(this,name);
        this.canFly = canFly;
};
```



But we want **Bird** class to inherit from **Animal** class not **Object** class

???

```
Object.create(proto [,properties])
```

Example

```
var Person = function(pname){
        this.pname = pname;
}
var Student = Object.create(Person.prototype);
//OR
var Student = new Person("islam");
//So, What's the difference between new or Object.create
```

Fight No#1

JS

new

It creates a new instance of the class

Example:

```
var Person = function(pname){
        this.pname = pname;
}
var Student = new Person("Ali");
//Student is an instance of
Person So It take the prototype
of it.
```

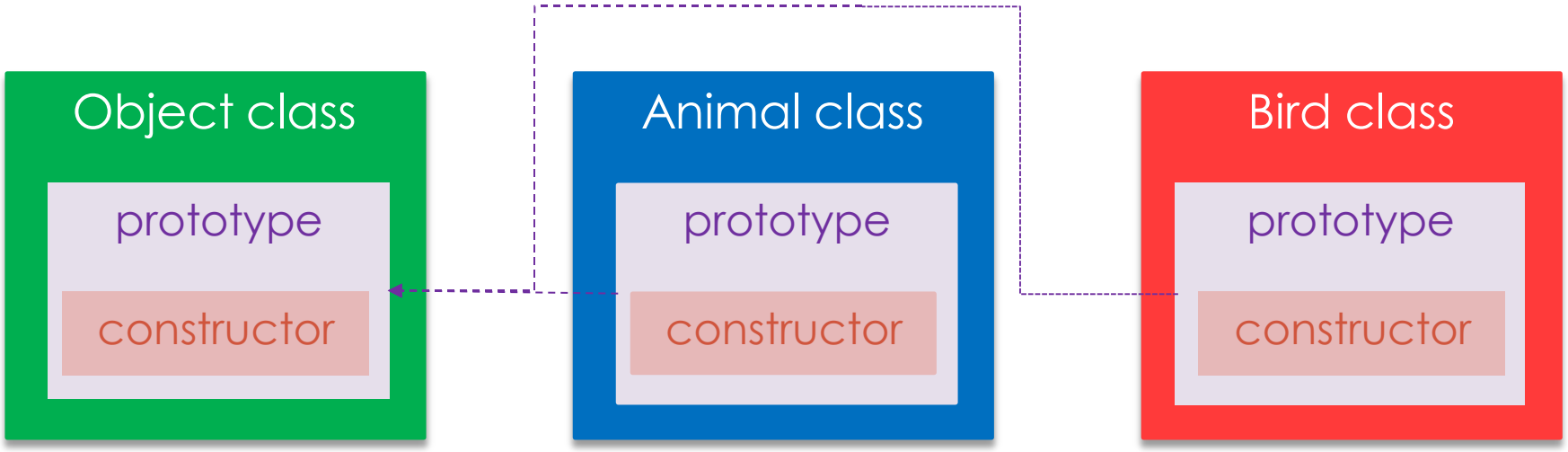
Object.create

method creates a new object with the specified prototype object.

Example:

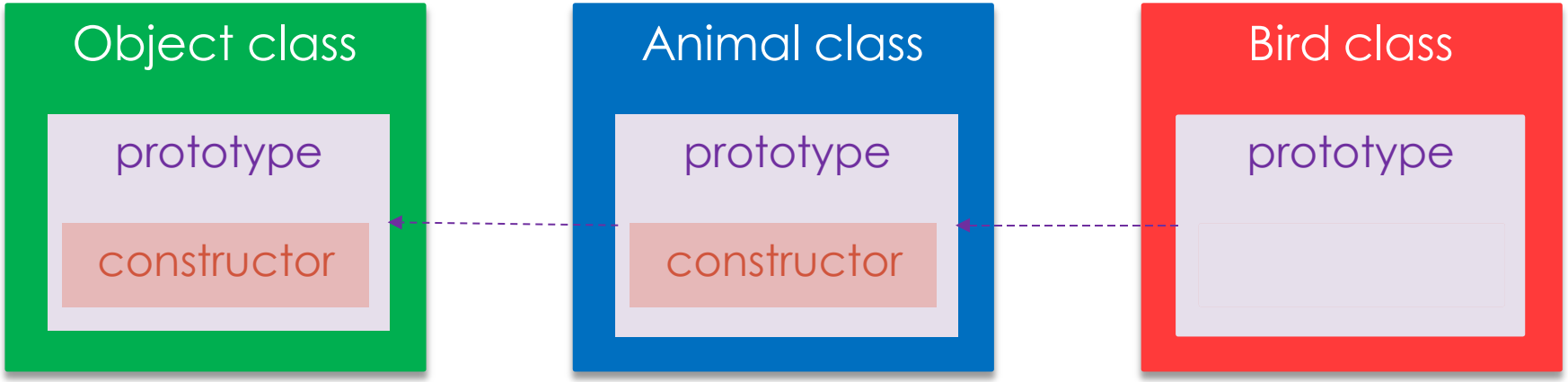
```
var Person = function(pname){
        this.pname = pname;
}
var Student =
Object.create(Person.prototype);
//Student is a customized object
with prototype of Person.
```

4 Create Child Prototype that inherit from Parent prototype:



4 Create Child Prototype that inherit from Parent prototype:

```
Bird.prototype = Object.create(Animal.prototype);
```

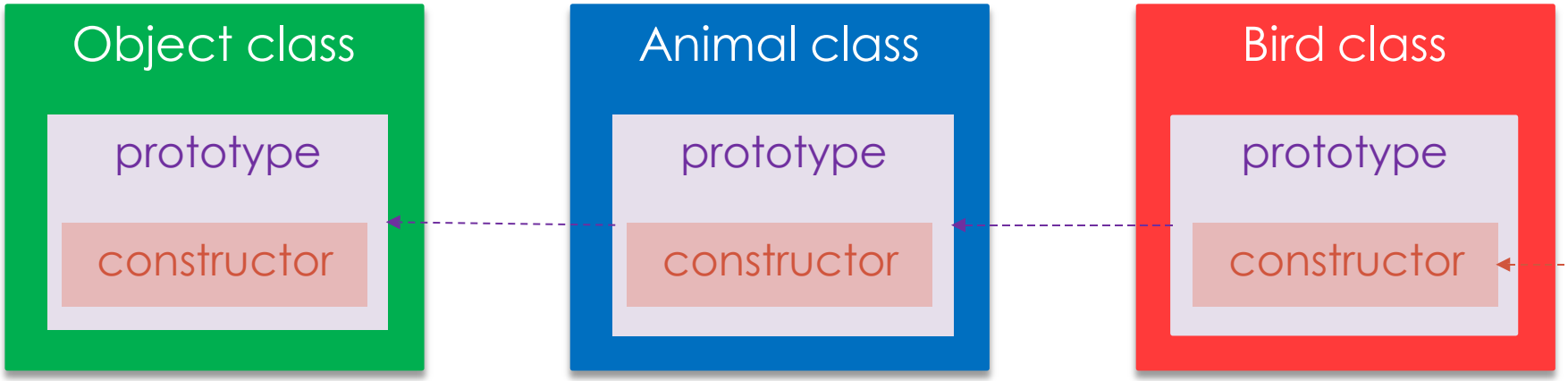


4 Create Child Prototype that inherit from Parent prototype:

```
Bird.prototype = Object.create(Animal.prototype);
```

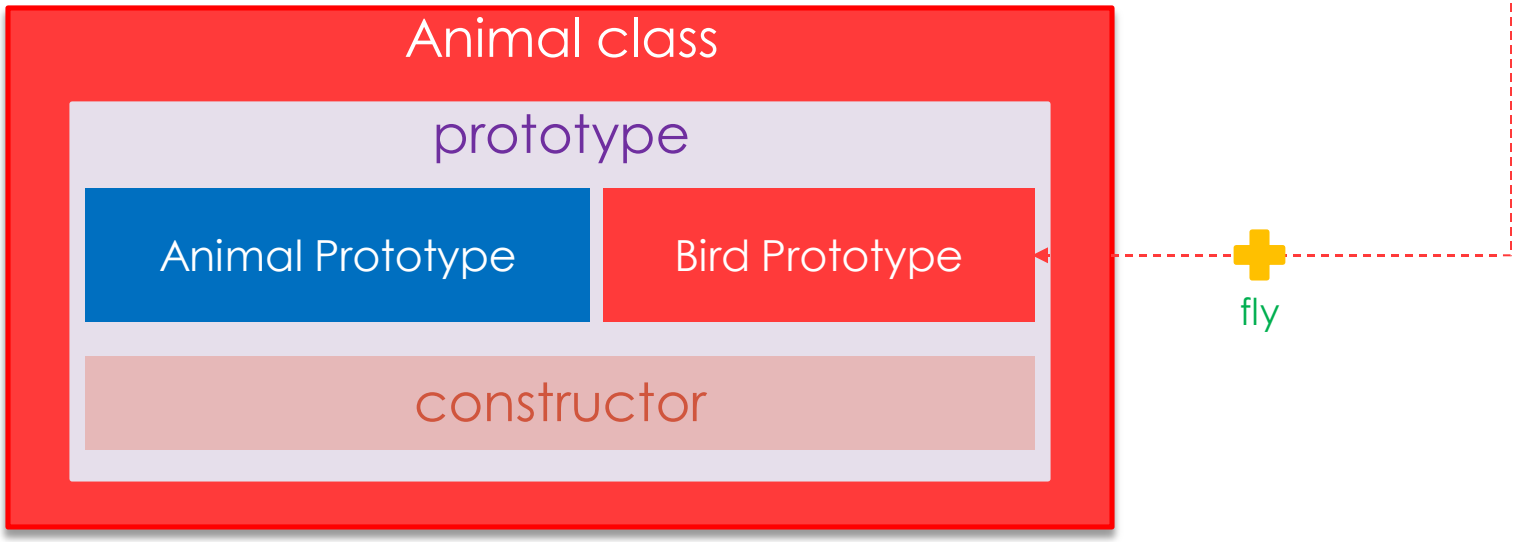
5 Create child constructor:

```
Bird.prototype.constructor = Bird;
```



6 Add Child Own Properties and Methods:

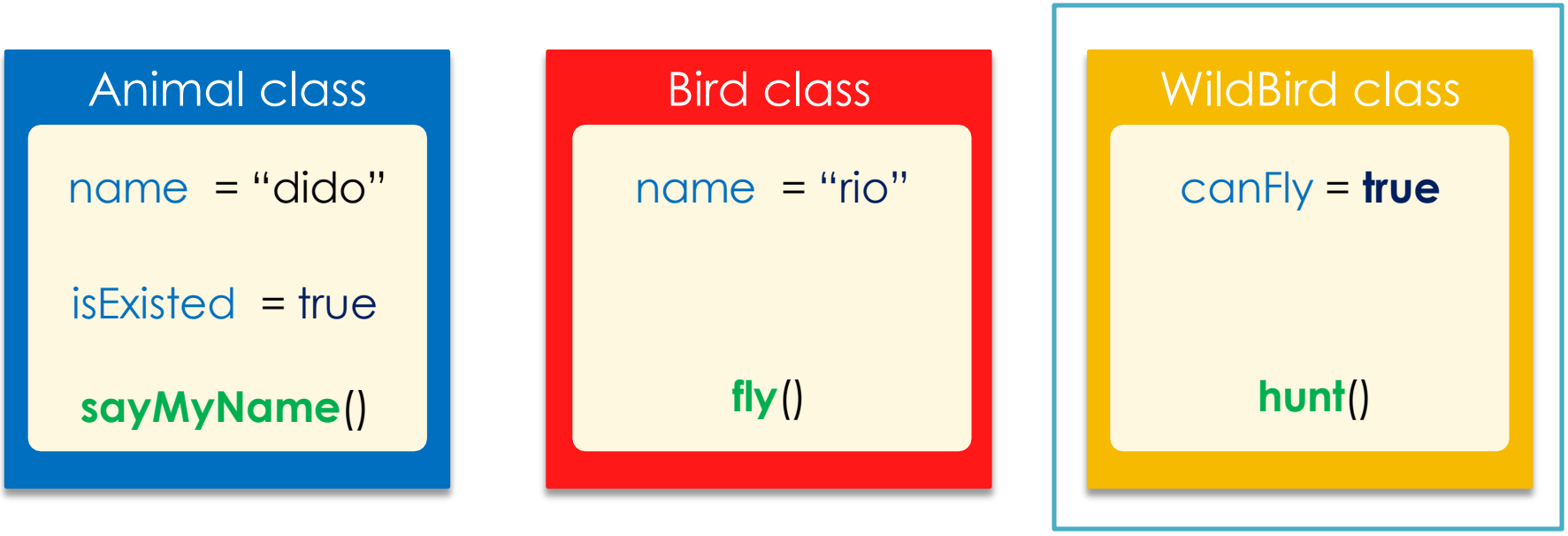
```
Bird.prototype.fly = function(){
        if(this.canFly) { console.log( "I fly :D" ); }
        else { console.log( "I can't fly :( " ); }
};
```



7 Let's Try:

```
var penguin = new Bird('amigos',false);

penguin.sayMyName();

Penguin.fly();
```

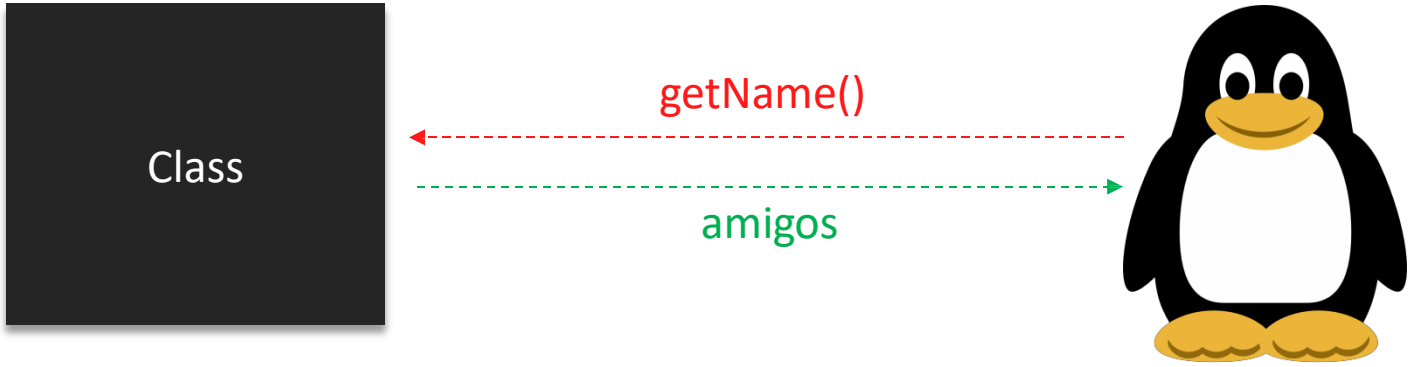
Amigos




```
var Eagle = new WildBird(true);
console.log(Eagle.name); // Rio
console.log(Eagle.isExisted); // true
Eagle.hunt(); //Hunt!!;
```

Encapsulation

Encapsulation is the packing of data and functions into one component (for example, a class) and then controlling access to that component .



Helper Methods | defineProperty

??!

```
Object.defineProperty(obj, prop, descriptor)
```

Descriptor is an object that describe the characteristics of the property.

Descriptor consists of:

configurable	**true** if the descriptor may be changed.
enumerable	**true** if and only if this property **shows up** during enumeration.
writable	**true** if the prop's value may be changed using **=**.
value	The **value** associated with the property.
get	A function which serves as a **getter** for the property.
set	A function which serves as a **setter** for the property.

```
Object.defineProperty(obj, prop, descriptor)
```

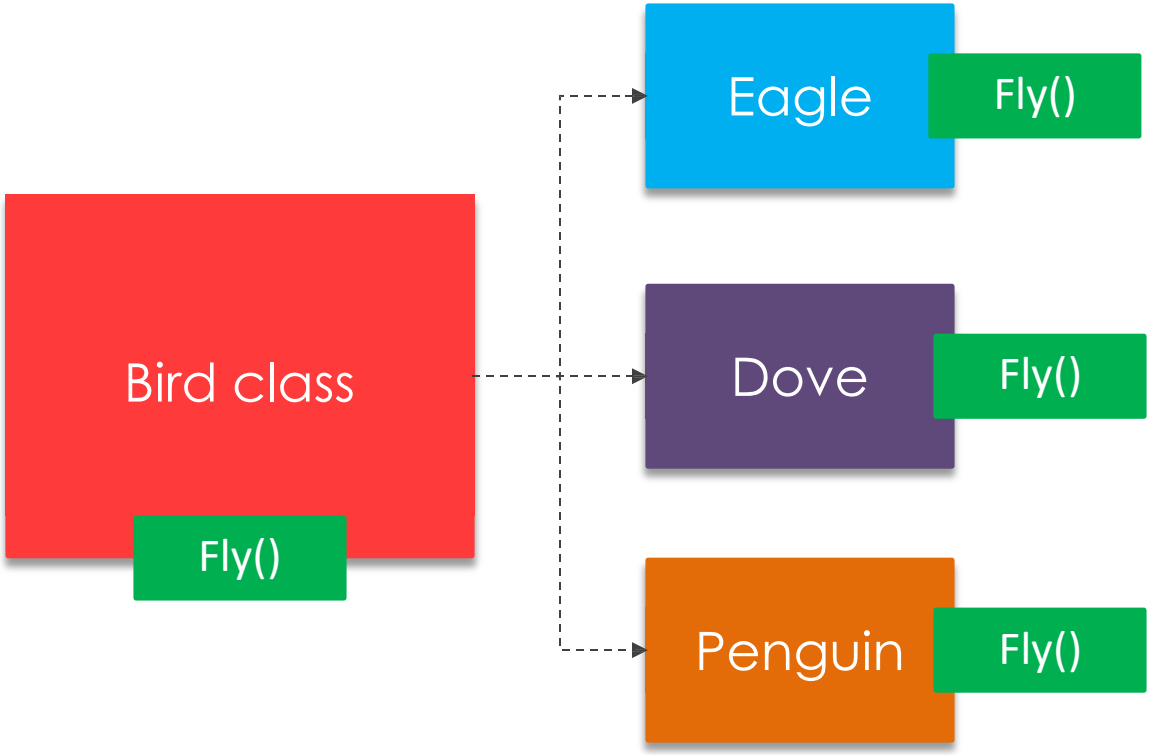
Example

```
var obj = {};

Object.defineProperty(obj, 'id',{
        enumerable: false,
        writable:false,
        configurable:false,
        value: "my-id"
});

Object.defineProperty(obj, 'id',{
        get: function(){return id},
        set: function(newVal){id = newVal},
        configurable:true,
});
```

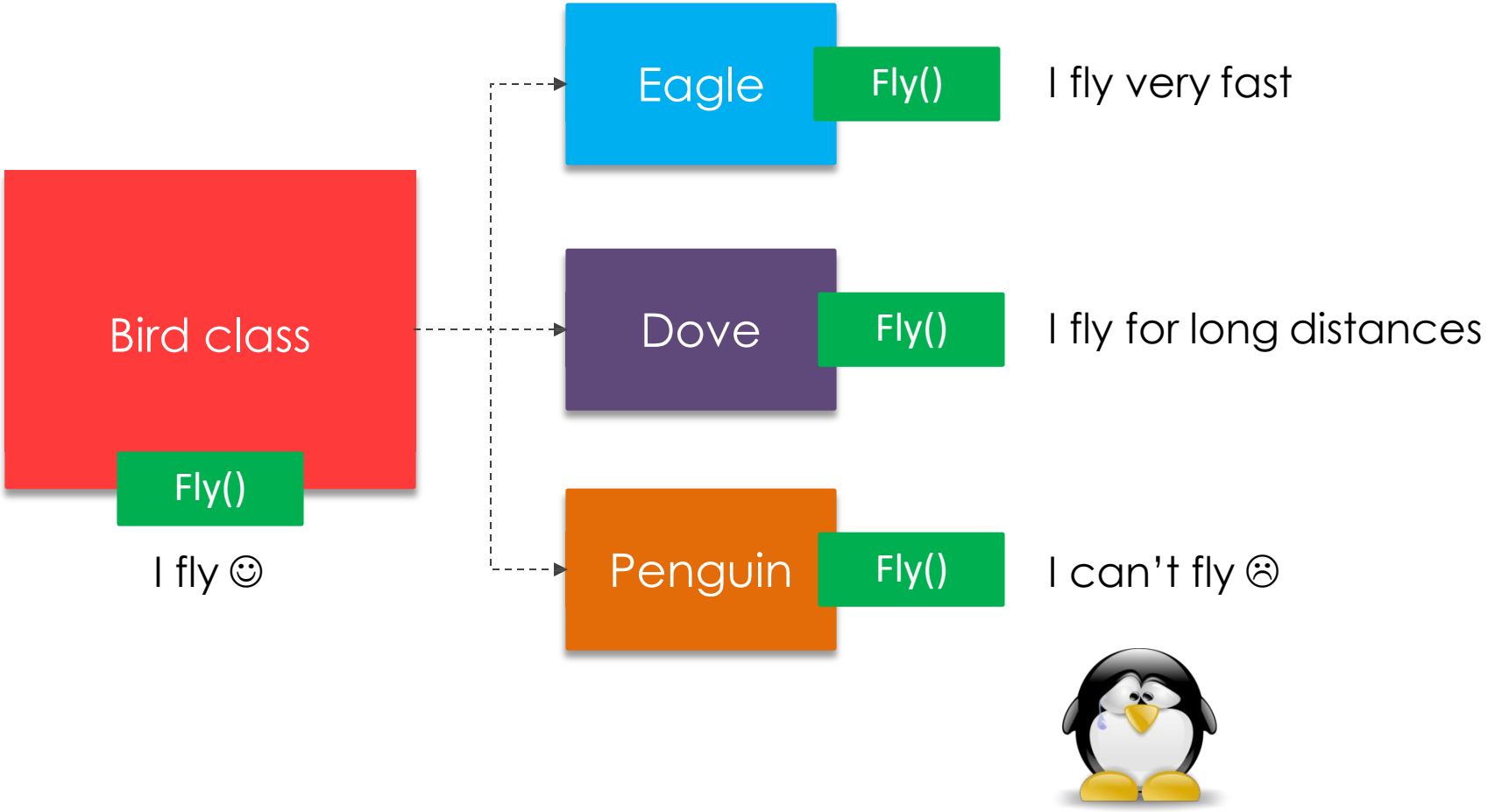
Polymorphism

JS

Poly means "*many*" and **morphism** means "*forms*". Different classes might define the same method or property.



Poly means "*many*" and **morphism** means "*forms*". Different classes might define the same method or property.



1 Overriding the parent method and don't implement it:

```
Bird.prototype.sayMyName = function(){
    console.log( "Hi everyone, My Name is"+this.name );
};
```

2 Overriding the parent method with implementing it inside the new one:

```
Bird.prototype.sayMyName = function(){
    Animal.prototype.sayMyName.call(this);
    console.log( "Hi everyone, My Name is"+this.name );
};
```

Report

Can we do overloading in JavaScript ?

If **Yes**, Tell me **How??**

If **No**, Tell me **Why??**

Note:

Support Your Answer by Examples.