What is Inheritance

Inheritance in most class-based object-oriented languages is a mechanism in which one object acquires all the properties and behaviors of another object. JavaScript is not a class-based language although *class* keyword is introduced in ES2015, it is just syntactical layer. JavaScript still works on *prototype* chain.

Classical Inheritance vs Prototypal Inheritance

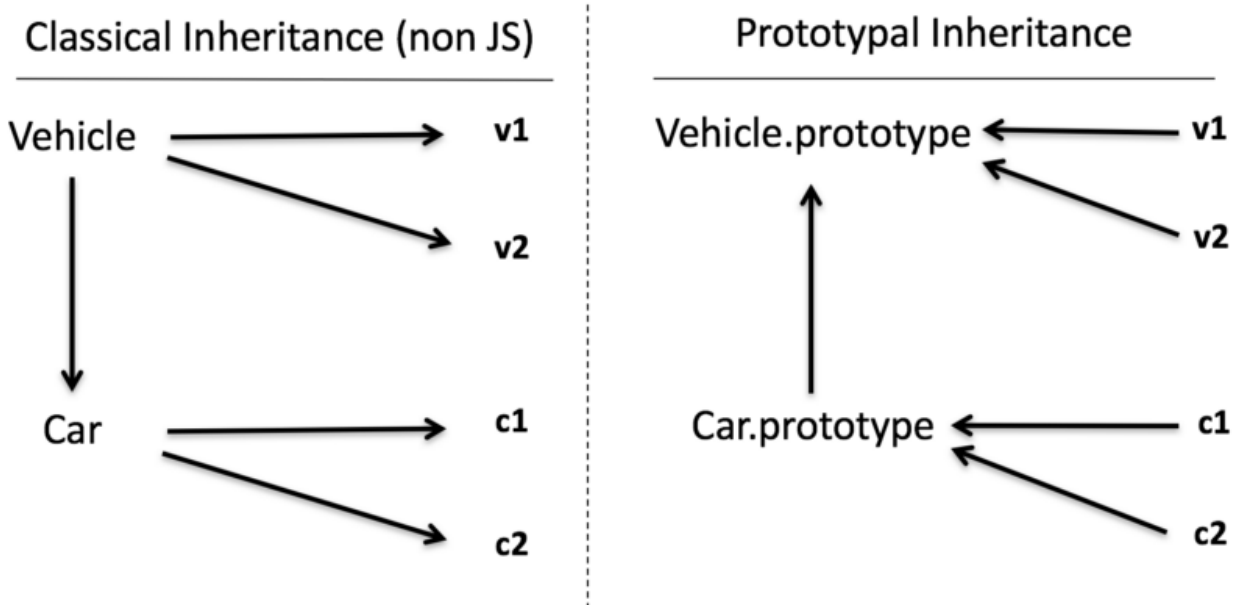


diagram of inheritance

Classical Inheritance (non-javascript)

- `Vehicle` is parent class and `v1`,`v2` are the instances of `Vehicle`

- `Car` is child class of `Vehicle` and `c1`, `c2` are instances of `Car`
- In classical inheritance it creates a copy of the behavior from parent class into the child when we extend the class and after that parent and child *class* are separate entity.
- Similarly, another copy of the behavior happens when we create an instance or object out of the class and both are separate entity again.
- It's like car is manufactured from the vehicle and car blueprints but after that both are separate entity because they are not linked because It's a copy. That's the reason all arrows going downwards (property and behavior flowing down)

Prototypal Inheritance (Behavior delegation pattern)

- `v1` and `v2` are linked to `Vehicle.prototype` because it's been created using *new* keyword.
- Similarly, `c1` and `c2` is linked to `Car.prototype` and `Car.prototype` is linked to `Vehicle.prototype`.
- In JavaScript when we create the object it does not copy the properties or behavior, it creates a link. Similar kind of linkage gets created in case of extend of class as well.
- All arrows go in the opposite direction compare to classical non-js inheritance because it's a behavior delegation link. These links are known as prototype chain.
- This pattern is called *Behavior Delegation Pattern* which commonly known as ***prototypal inheritance*** in JavaScript.

You may go through article JavaScript — Prototype to understand the ***prototypechain*** *in depth.*

Example of prototypal inheritance

- Usage of `Object.create()` to achieve classical inheritance.
- In the below code snippet, `Car.prototype` and `Vehicle.prototype` is linked with help of `Object.create()` function.

```
// Vehicle - superclass
function Vehicle(name) {
  this.name = name;
}
// superclass method
Vehicle.prototype.start = function() {
  return "engine of "+this.name + " starting...";
};
// Car - subclass
function Car(name) {
  Vehicle.call(this,name); // call super constructor.
}
// subclass extends superclass
Car.prototype = Object.create(Vehicle.prototype);
// subclass method
Car.prototype.run = function() {
  console.log("Hello "+ this.start());
};
// instances of subclass
var c1 = new Car("Fiesta");
var c2 = new Car("Baleno");
// accessing the subclass method which internally access
superclass method
c1.run();   // "Hello engine of Fiesta starting..."
c2.run();   // "Hello engine of Baleno starting..."
```

- In the above code, object `c1` gets access to method `run()` and method `start()` because of below prototype chain. As per below diagram, we can see object `c1` does not have these methods but it has links to go upwards.
- keyword `this` in above code is nothing but current execution context of each method which is `c1` and `c2`.

*To understand keyword **this** in details, you can go through article [JavaScript—All about this and new keyword](#).*

Diagrammatic representation of above code

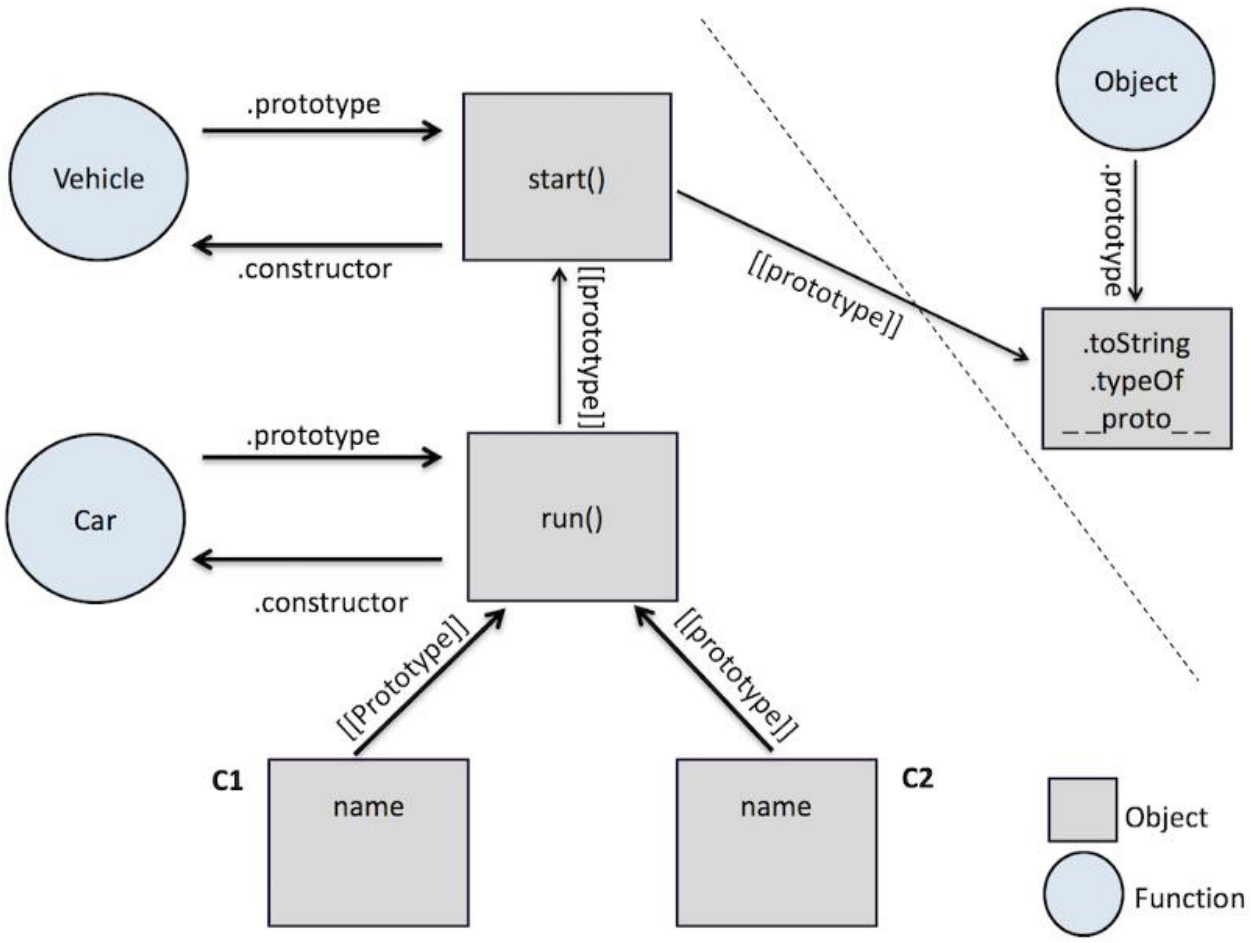


diagram of javascript inheritance

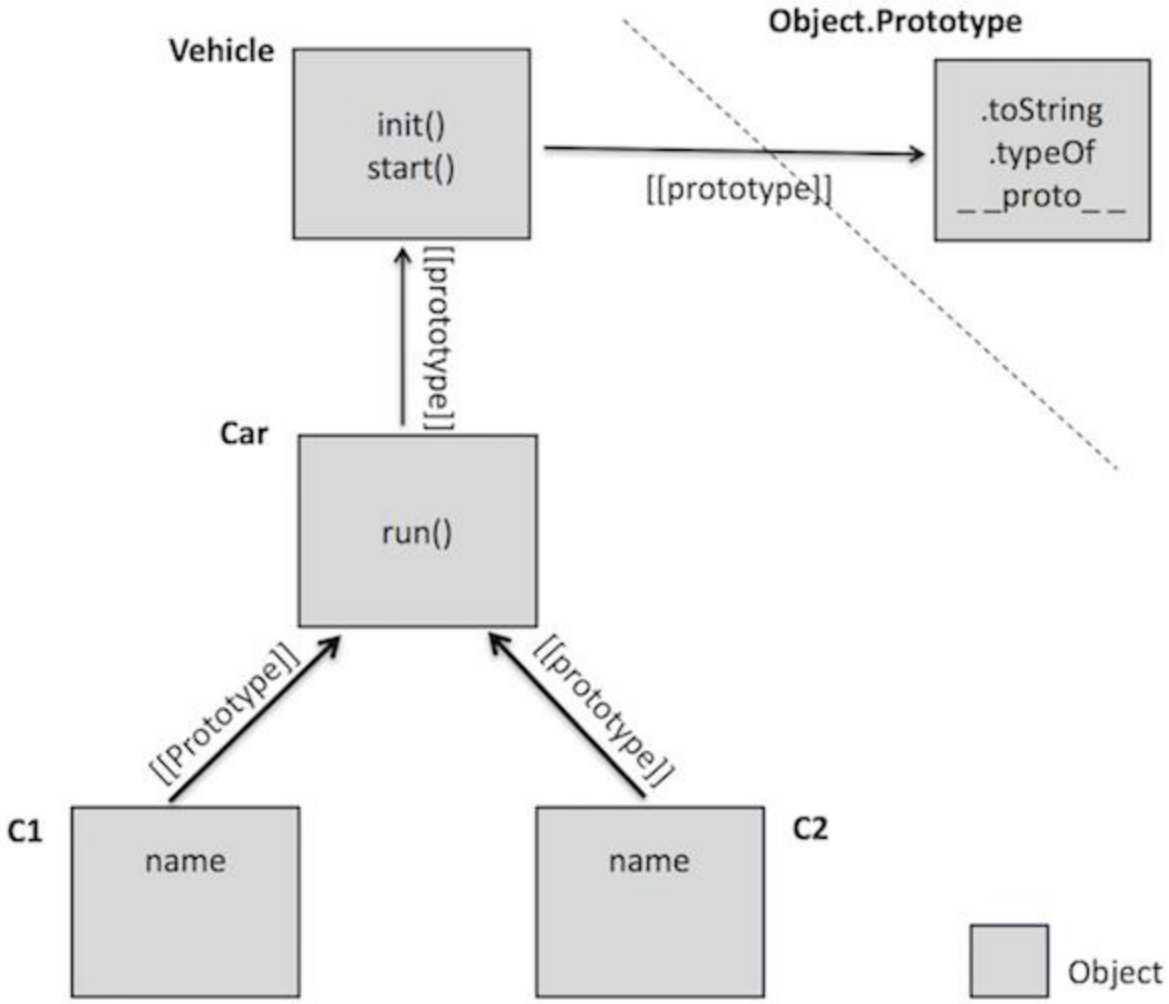
Objects linked to other objects

- Now we will simplify the previous example code of inheritance with focusing only on objects and object linkages.
- So we will try to
  remove *.prototype*, *constructor* and *new* keyword, will be thinking about only objects.
- We will be using `Object.create()` function to create all the linkages between objects.

Below is simplified code of previous example.

```
// base object with methods including initialization
var Vehicle = {
  init: function(name) {
    this.name = name;
  },
  start: function() {
    return "engine of "+this.name + " starting...";
  }
}
// delegation link created between sub object and base object
var Car = Object.create(Vehicle);
// sub object method
Car.run = function() {
  console.log("Hello "+ this.start());
};
// instance object with delegation link point to sub object
var c1 = Object.create(Car);
c1.init('Fiesta');
var c2 = Object.create(Car);
c2.init('Baleno');
c1.run();   // "Hello engine of Fiesta starting..."
c2.run();   // "Hello engine of Baleno starting..."
```

Diagrammatic representation of above code



objects linking

- We can see now, how we have removed the complexity of *new*, all the *.prototype,*
  constructor functions and *call* method everything and still achieved the same result.
- Only thing which matters is object `c1` linked to another object and then linked to another object again and so on.
- This is called object delegation pattern as well.

Summary

It is important to understand the prototypal inheritance and prototype chain before using these in code to avoid complexity.