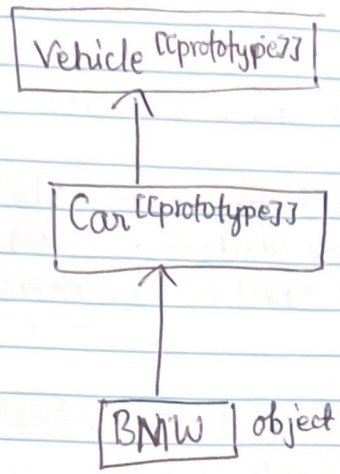
Object Prototype:-

If we have two instances of same constructor which uses the same property, it is better to keep the uniform property in a single place, and make the instances refer from that place. It prevents the property taking space in system's memory. This is where prototype comes in.

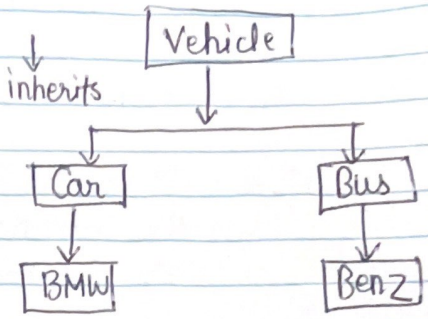
When it comes to inheritance, JS only has one construct: objects. Each object has a private property which holds a link to another object called its prototype. That prototype object has a prototype of its own, and so on until an object is reached with null as its prototype. Null has no prototype, and acts like the final ~~chain~~ link in this prototype chain.

* If an object and its prototype has same property, the property in the prototype is not visited when tried to access the property name in that object. This is called Property shadowing.

Prototypical Inheritance



Classical Inheritance



- The power of prototype is that we can use/reuse the set of properties if they should be present on every instance - especially for methods. For example:

```
const boxes = [
    { value: 1, getValue() { return this.value } },
    { value: 2, getValue() { return this.value } }
    ]
```

This is redundant and unnecessary, so we can move the method into the [[prototype]].

```
const boxPrototype = {
        getValue() {
            return this.value;
        },
    };

const boxes = [
    { value: 1, __proto__ : boxPrototype },
    { value: 2, __proto__ : boxPrototype }
    ]
```