Object Oriented Programming In JS

In software engineering, anything that we have can be mapped to a blueprint or a vision. For example, in Amazon, every product has a similar structure, we can generalise this and make sure we don't repeat anything again and again.

Classes

Classes are blueprint on a set of real life entities.
Classes are going to represent how an entity should look and behave. When I say how do they look, I refer to the properties they posses. And when I say how they behave then I mean what actions can be performed on them.

Let's take an example:

Consider a `Product` class, inside this product class what do you think should be the properties of a product ? What is a property ? Property is just any feature that the product can contain.

- name -> Name of the product
- price -> Price of the product
- company -> Company the product belongs to
- reviews -> set of feedbacks / reviews the product got
- rating -> Rating of the product
- description -> Description of the product.\
- and more ...
  So above are the set of properties that a Product class can posses. In technical terms these properties are referred as `Data Members`. If two products are different from each other then atleast one of the data members should have a different value in them.

Now, what are the things we can do with a product ?

- We can buy a product
- Add product to cart
- We can display the details of a product
- We can wishlist a product
- We can rate a product
- We can add review to the product

- and more ....
  These are the actions which can be done on a product, which we can also consider as the behaviour of the product. And in technical terms these are referred as `Member functions`.

How to make a class in JS ?

In JS, we have a `class` keyword, that can help us to make a class and include the data members and member functions inside it.

Every class has a name associated to it as well, that we have to define once we declare the class.

```
class NameOfTheClass {
        // you can details like member functions and data member
}
```

Let's write a product class:

```
class Product {
        name;

        price;

        category;

        description;

        rating;



        addToCart() {

                console.log("Product added to cart");

        }



        removeFromCart() {

                console.log("Product removed from cart");
        }



        displayProduct() {

                console.log("Product displayed");

        }



        buyProduct() {

                console.log("Product bought");

        }



}
```

constructor : a very special member function of every class

Every class that we make in JS, has one special member function called as constructor. What is so special about it ?
So whenever we create an object using classes, the constructor is first method that is automatically called by JS. It is already available for any class you make. The default version of this constructor is also called `default constructor`.
We can change the implementation of the default constructor by writing one of our own. How ?
By doing something like this:

```
class Product {

        constructor() {
                // This is your custom constructor
        }
}
```

We will discuss more internals of constructor later.

Objects

Using classes the final entity that we will develop is called as `objects`. How can we create an object of a class ?
There is a keyword in JS called as `new` which can help us to create an object out of a class. Don't map the use case of `new` keyword in JS with other languages, it is very different from other languages.

```
let iphone = new Product();
```

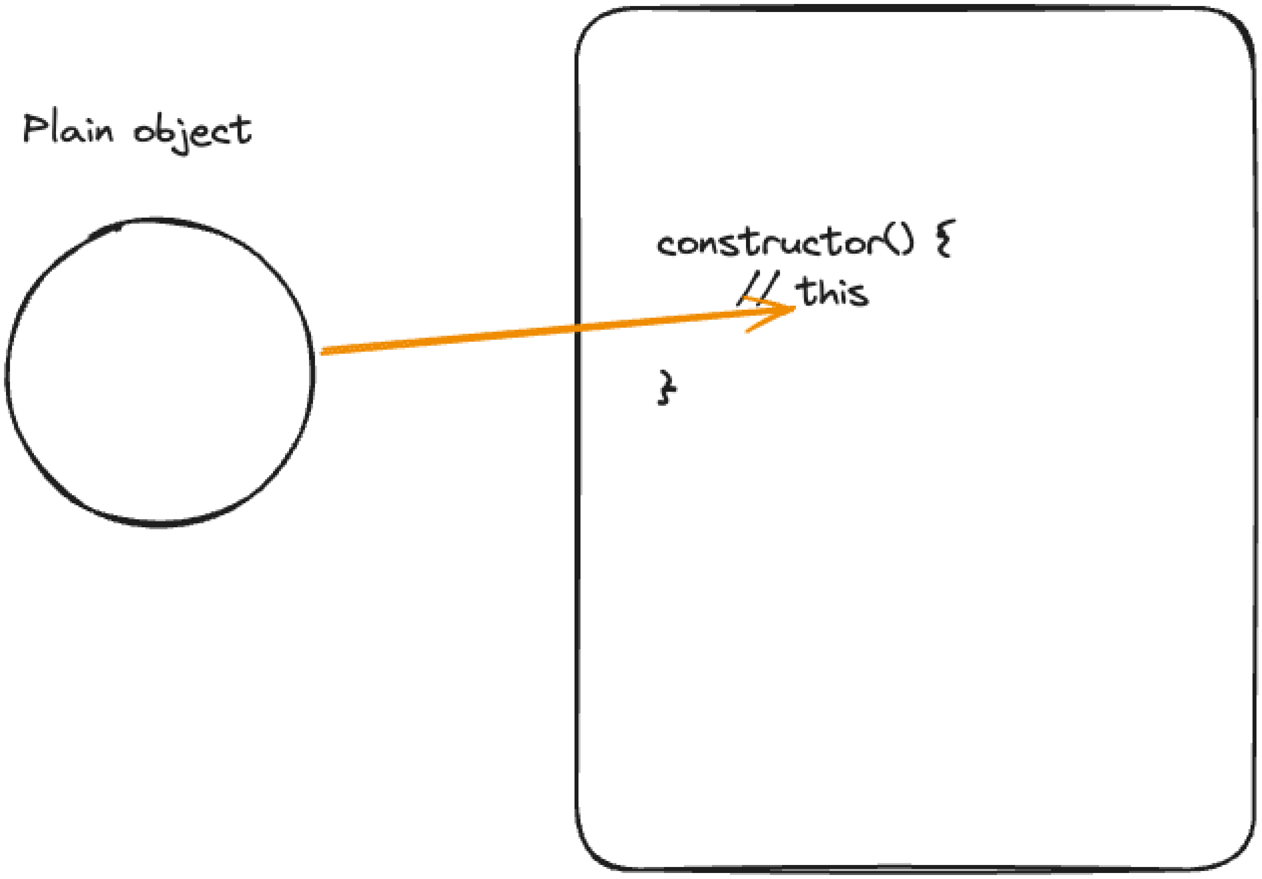
So, to create an object we write `new` and then mention the name of the class followed by a pair of parenthesis.

How new works ?

Every time we call new, it does the following 4 step procedure:\

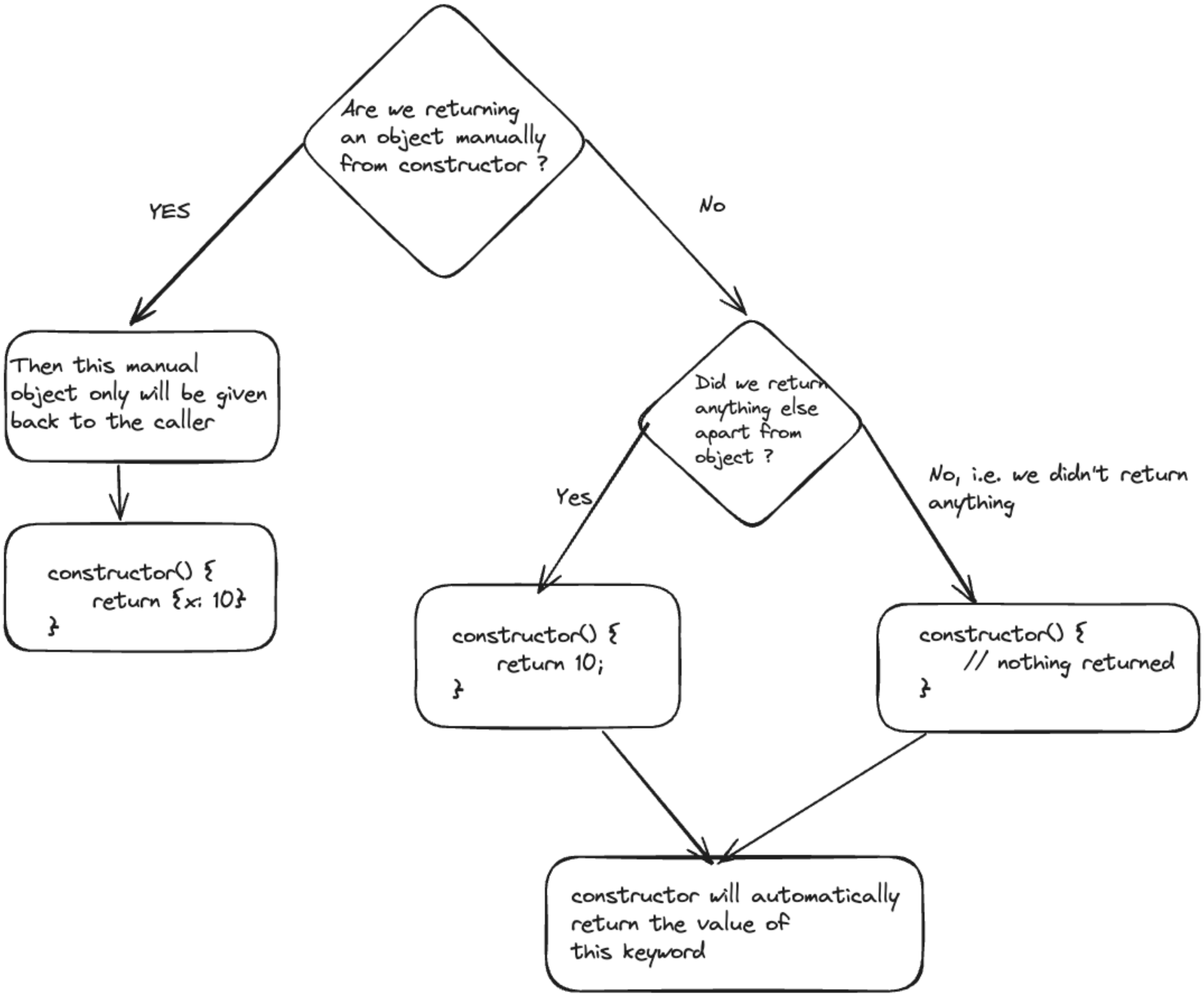
1. It creates a brand new plain & absolutely empty object.
2. It calls the constructor of the class and passes the newly created object (not as a parameter) but inside a keyword called as `this`. So constructor automatically has access to the this keyword and when we call new , then the this keyword has access to the plain object created in step 1 and constructor now can use the this keyword inside it. And then

whatever is logic of constructor it is executed.



3. In step 3, new triggers everything need to be done for prototypes to work (will discuss later).
4. Now, if from a constructor an object is manually returned then this manual returned object is stored in the called variable, otherwise in any other case i.e. either we don't return anything or return something apart from object, constructor doesn't care about it and

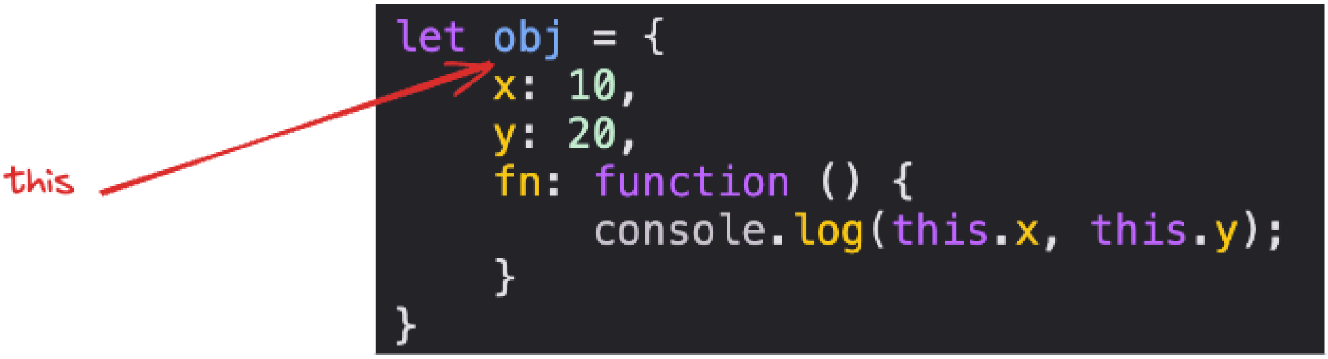
returns the value inside the this keyword.


Are we returning an object manually from constructor ?
YES
No
Then this manual object only will be given back to the caller
constructor() { return {x: 10} }
Did we return anything else apart from object ?
Yes
No, i.e. we didn't return anything
constructor() { return 10; }
constructor() { // nothing returned }
constructor will automatically return the value of this keyword

this keyword of JS

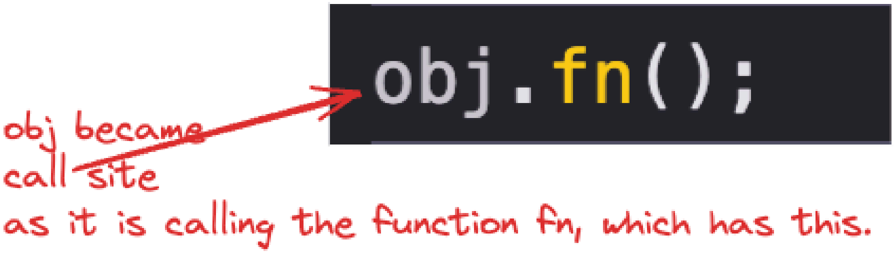
- this in JS works very differently than other languages.
- this as a keyword is available to be accessed in any function or even outside any function, and in classes as well.
- If we can use `this` keyword anywhere, then what's the value stored inside `this` ?

In most of the cases this refers to the call site.
Ahhh! What is a call site ?
Call site can be an object, or a position or may be even the new keyword. It refers to the entity which is calling this keyword.



```
let obj = {
    x: 10,
    y: 20,
    fn: function () {
        console.log(this.x, this.y);
    }
}
```

this



```
obj.fn();
```

obj became
call site
as it is calling the function fn, which has this.

It has an exception, this keyword will not refer to the call site if used inside an arrow function

In case of arrow functions, this is resolved using lexical scoping.

```
let obj = {
    x: 10,
    y: 20,
    fn: function() {
        const arrow = () => {
            console.log(this.x, this.y);
        }
        arrow();
    }
}
```

In this code, this is present inside the arrow function, hence we will resolve it lexically.

1. Is `this` defined in the scope of arrow function ? No
2. We go one scope up, i.e inside function fn.
3. Is `this` defined inside fn? yes, because fn is a normal function, we have a definition of this inside it which is the call site
4. Who is the call site ? Obj object which is responsible to trigger fn is the call site
5. Hence `this` refers to obj object and when we call arrow function we get output as `10 20`.

That means, when we make a new object using the `new` keyword, then `new` keyword creates a plain object and this plain object is the call site for the constructor hence, this keyword refers to the plain object altogether.