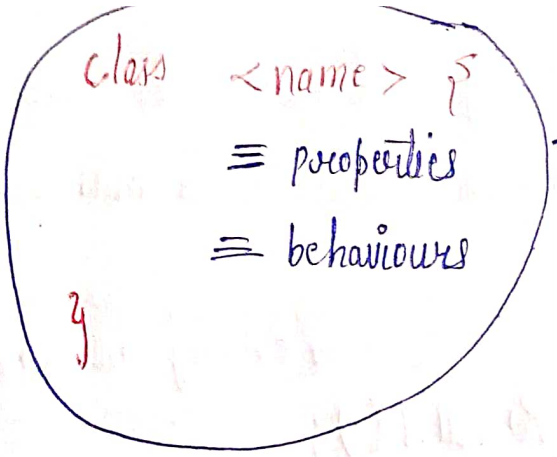

→ Using this blue print, we can create real life entities.

→ To create and manage these entities, we have two keywords. "new", "this".

* "this" Keyword

Except one case, this always refers to the calling site / context.

"this" keyword refers to calling / context, from what context or from what reference, we are calling the "this" keyword actually makes a difference.

→ Inside arrow function "this" does not refers to calling le context.

* "new" Keyword.

Whenever there is a class, or there is a blue print that we have created and with that blue print, we want to actually initialized a brand new instance, that's where we use "new" keyword.

example :—

```
constructor () { }
```

Now, the "new" keyword created a brand new plain javascript object and we are calling the constructor. for example :—

```
const p = new Product();
```

→ calling the constructor

So this constructor is getting called with respect to brand new plain javascript object.

So, here calling context is a brand new plain empty javascript object.

So, if we use 'this' keyword inside constructor, like

```
constructor () {
    this.....
}
```

So, what is this going to point to?

→ 'this' keyword is actually going to point same brand new plain empty object.

("this" will not going to point the name of the class)

example: —

```
class Product() {
    name;
    price;
    description;

    constructor (n, p, d) {
        this.name = n;
        this.price = p;
        this.description = d;
    }
}

const p = new Product("sachin", 10, "I am cool");
console.log(p);
```

We don't need to mention these values in the class.

Point to the plain empty object

So, 'this' is pointing to the plain empty object. So, we in plain empty object, we are manually creating a plain key-value pair, where 'key' is "name" and 'p' is 'value' whatever we are passing inside them.

Now, constructor is a method like a function and in a function we generally returned something.
what if we start returning something from inside the constructor.

let's suppose we are returning "10".

example:—

```
constructor ( n, p, d ) {
    this.name = n;
    this.price = p;
    this.description = d;
    return 10;        // primitive → no effect
}
```

Actually the movement we returned "10" nothing will happen. Because, inside a **constructor** if we use the returned keyword with the **primtive**, then there is no effect. Because **constructor** is meant to do something with an **object**.

Whereas, instead of returned "10" if we do our return a plain object, like

```
return { };
```

we will get a plain object in the console.

And if we don't return anything, it is equal in saying return **this**.

```
return this; // If we don't return anything, it is
             equal in saying "return this"
```

Note:- In Javascript, a class may have only one constructor.

```
function Product(n, p, d) {
    this.name = n;
    this.price = p;
    this.des = d;
}

const p = new Product("bag", 100, "cool new bag");
console.log(p);
```

This kind of syntax is called as function constructor.

- Classes are just wrapper over function. whatever we do with classes, we can achieve by using function also.

* **Abstraction :-**

It refers to the process of hiding the implementation details of an object and only exposing only the essential features to the outside world.

* **Encapsulation :-**

It refers to the practice of wrapping data and methods into a single unit.

⇒ We can use the '#' symbol to define private variables in a class. like :—

```
class Person {
    #name ;    // private variable
    #age ;     // private variable

    constructor ( name, age ) {
        this.#name = name ;
        this.#age = age ;
    }

    getname () { // public method to access private variable
        return this.#name ;
    }

    getAge () { // public method to access private variable
        return this.#age ;
    }
}

const person = new Person ("John", 30);

console.log ( person.getname() ); // "John"

console.log ( person.#name ); // syntax Error :
Private field '#name' must be declared in an enclosing class,
```