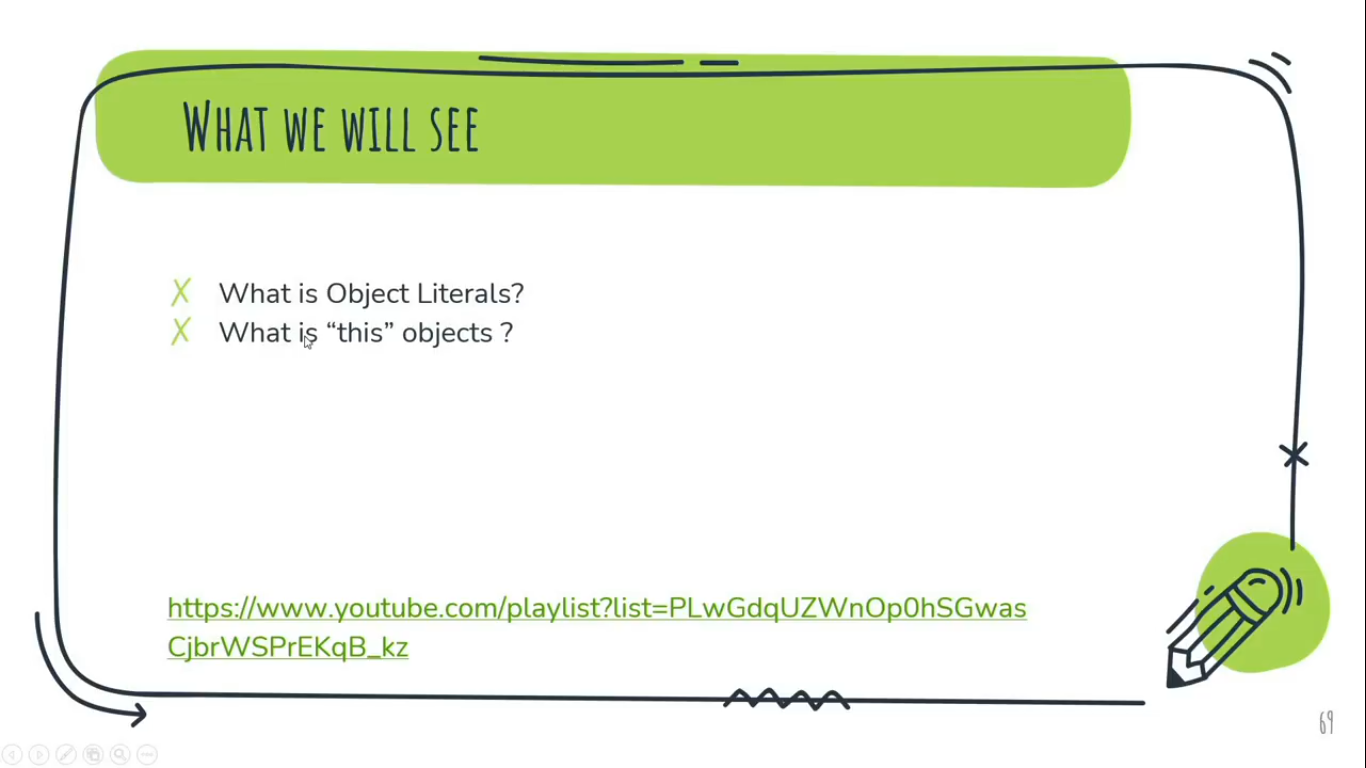
**OOPs (Object Oriented Programmings) in JS**

****

**Procedural Vs Object Oriented Programming**



In Procedural Programming,

1. Don’t Repeat yourself failed
2. Hard to debug and Manage

Code in Procedural Programming is like a Spaghetti-code which is totally messed up code for human Beings.

**Four Pillars of OOPs**

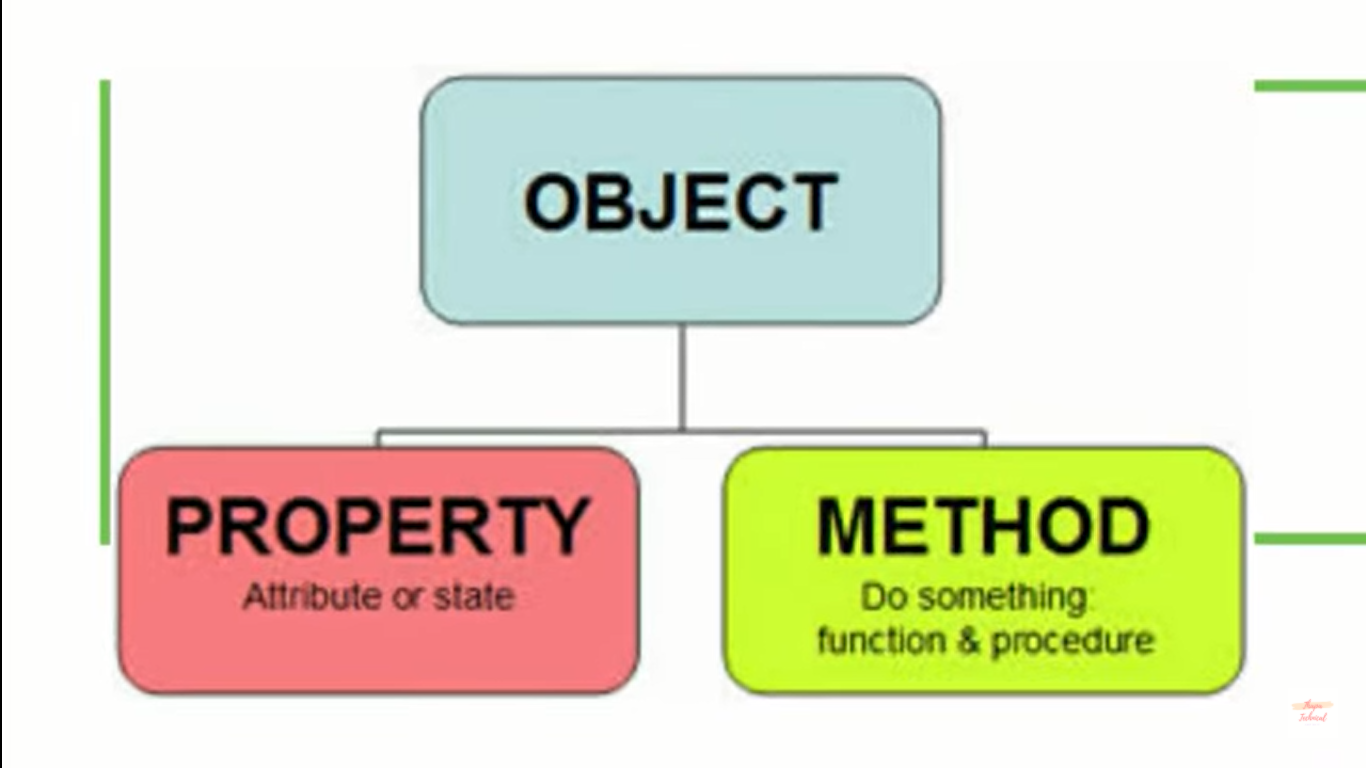
1. **Encapsulation:**Encapsulation means wrapping up data and member function (Method) together into a single unit i.e. class.  
   For the sake of understanding, we can consider It’s just like a capsule medicine where actual medicine filled inside a capsule.
2. **Abstraction:**Abstraction is the process of showing only essential/necessary features of an entity/object to the outside world and hide the other irrelevant information. For example - to open your TV we only have a power button. It is not required to understand how infra-red waves are getting generated in TV remote control. Or Do you ever worried about the function of .log() method of console object regarding how it function to print anything on the console.
3. **Inheritance:**Inheritance allows a class (subclass / child or derive class) to acquire the properties and behaviour of another class (super-class / parent class). It help to reuse, customize and enhance the existing code. So it helps to write a code accurately and reduce the development time.
4. **Polymorphism:**So polymorphism means “many forms”. A subclass can define it own unique behaviour and still share the same functionalities or behaviour of its parent/base class.  
   Ex:  
   class square(){ class circle(){  
   area() area()  
   } }  
     
   var S1 = new square(); var C1 = new square();  
   S1 - >area(); C1 - >area();

**Topics in OOPs:**

* **Objects**
* **Class**
* **Inheritance**
* **Constructor**

**Object**

**Q. What is Object Literal?**Object literal is simply a key:value or say Property:value pair data structure.



Storing Variables and Functions together in one container i.e. Object.  
Here, we can consider, container (which would be a variable ) as ‘Object’, Variable storing Value (String or any Data) as ‘Property’ and those Variable storing Function as ‘Method’ or directly as function (Because this method now going to work as function).

**1st way of initializing Properties & Methods in Objects before ES6**

Example:let bioData = {  
 name: "Chandan Kumar",

age: 20,

Branch: "CSFE",

message: function (){ ***//Defining method after ES6***

document.write(`Hey!, My name is ${bioData.name}, My Age is ${bioData.age}  
and My Branch is ${bioData.Branch}.`);

}

}

bioData.message();  
document.write(`</br> ${bioData.name}`);

**Output:**   
Hey!, My name is Chandan Kumar, My Age is 20 and My Branch is CSFE.  
Chandan Kumar

Note:

Here, bioData is a User-Defined Object is JavaScript where, name, age, Branch are the Properties and message is the method of this Object.

**2nd way of initializing properties & methods in Objects before ES6**

Example

let bioData = {} ***//One way of creating blank Object***

**//** let bioData = new Object(); ***//Another way of creating blank Object***

bioData.firstName = "Chandan"; ***//Creating Property and assigning values to it.***

bioData.message = ()=>{ ***//Creating Method and assigning a function to it.***

document.write(`Hey!, My name is ${bioData.name}, My Age is ${bioData.age} and My Branch is ${bioData.Branch}.`);

}

bioData.message();  
**Output:**   
Hey!, My name is Chandan Kumar, My Age is 20 and My Branch is CSFE.

**Object as value inside an Object:**

Example:

let bioData = {

myName: {

firstName: "Chandan",  
 lastName: "Kumar"

},

age: 20,

Branch: "CSFE",

message(){

document.write(`Hey!, My name is ${bioData.name}, My Age is ${bioData.age} and My Branch is ${bioData.Branch}.`);

}

}

document.write(bioData.myName.lastName);

Output:  
Kumar

**Q. What is this Object?**

The definition of “this” object is that it contain the current context.

The this object can have different value depending on where it is placed.

**#1** Example:  
console.log(this.alert(‘hi’));  
**//Here, this keyword is referring to the current context i.e. Window Object associating with its alert() method.  
Note: Window is a Global Object.**

**#2** Example:  
function myName(){  
console.log(this);  
}  
myName();  
**//Still the current context of ‘this’ object is window object because function ‘’myName” is calling from the Global scope or say Window object. Now this function would be consider as a method of window object. And by doing this *console.log(this.myName())*, it will return back the whole written above function.**

**#3** Example:  
var myname = “Chandan”; **//Here, initialization has done in Global Scope**   
function myName(){  
console.log(this.myname);  
}  
myName();  
**//Since, initialization of myname variable has done in Global Scope, so it will be the property of Global Object i.e. Window. Hence the output is Chandan.**

**#4** Example:  
const person = {  
age: 20,  
myName: {

fName:"Chandan",

lName:"Kumar"  
},

func () {  
console.log(this);  
}  
}

person.func();  
**//Here, the current context of ‘this’ object is person Object. Person Object containing Property as ‘age’, ‘myName’ and method as ‘func()’. ‘myName’ property has itself value as Object.**

**Note**

We can’t define method inside an object using ‘this’ keyword.

**For example:**

const person = {  
age: 20,  
myName: {

fName:"Chandan",

lName:"Kumar"  
},

func ()=> {  
console.log(this);  
}  
}

person.func();

**Here, the current context of ‘this’ object will be window object instead of person object.**

**Find the Output:-**

const Me ={

myName :{  
fName:"Chandan",  
lName:"Kumar",  
},

myAge:20,

execute: function(){  
document.write(`My Last Name is ${this.myName.lName} and my Age is ${this.myAge}.`);  
}  
}  
  
Me.execute();

**Output:**My Last Name is Kumar and my Age is 20.

**Note:**

* To get the **value** of key/property inside an Object:  
  If key as **String**, we’ll use (.)dot notation to get value,  
  *object.key*If key as **Number**, we’ll use array notation to get value,  
  *object[key]*
* To get the **function** of method inside an Object:  
  *object.method();*
* Mapping of an array, means to get the value/element of that array inside an object as key.  
  **Ex:**  
  let obj= {1:3};  
  obj[2]=5;  
  obj.ck = ‘king’;  
  console.log(obj);  
  console.log(obj[2]);  
  **Output:**{1:3, 2:5, ck:’king’}  
  5
* To get the only **keys** of the above array mapping in an object, we’ll use **forin** loop  
  for(let key in obj){  
  console.log(key);  
  }  
  **Ouput:**1  
  2  
  ck

**Predefined Methods & Properties of Object:-**

**hasOwnProperty() method:-**

This method return Boolean value by checking the given ‘key’ is present in the Object or not.

Return **true,** if Object has a property with given key.  
If not, return **false**.

Note: We can also use **in** identifier to check its presence.

For Ex:

function myFunction(a,b){  
 return a.**hasOwnProperty**(b);  
}  
myFunction({x: ’a’, y: ‘b’, z: ‘c’}, z);  
Output:  
true

function myFunction(a,b){  
 return b **in** a;

}  
myFunction({x: ’a’, y: ‘b’, z: ‘c’}, z);  
Output:  
true

**Object.keys() method:-**

This method return an array of Object’s keys.

**For Ex:**

**1st Solution**: Object.keys()  
function myFunction(a){  
 return Object.keys(a);  
}  
myFunction({x: ’a’, y: ‘b’, z: ‘c’});  
Output:  
[x, y, z] ***//Prototype: Array***

**2nd Solution**: Iterative way  
function myFunction(a){  
 arr = [];

for(key in obj){  
 arr.push(key);  
}

return arr;  
}  
myFunction({x: ’a’, y: ‘b’, z: ‘c’});  
Output:  
[x, y, z] ***//Prototype: Array***

**Note:**

To to delete a property from an object in Javascript, we can use the delete operator. Consider the following object:

const obj = {x:1,y:2};

To remove property y we would call:

delete obj.y;

The modified object would now be:

{x:1}

For Example:-

1st Solution:  
function myFunction(obj) {

delete obj.b;

return obj;

}  
myFunction({ e: 6, f: 4, b: 5, a: 3 });

**Output:**{e: 6, f: 4, a: 3}

**2nd Solution:** (Using Spread Operator)  
function myFunction(obj) {

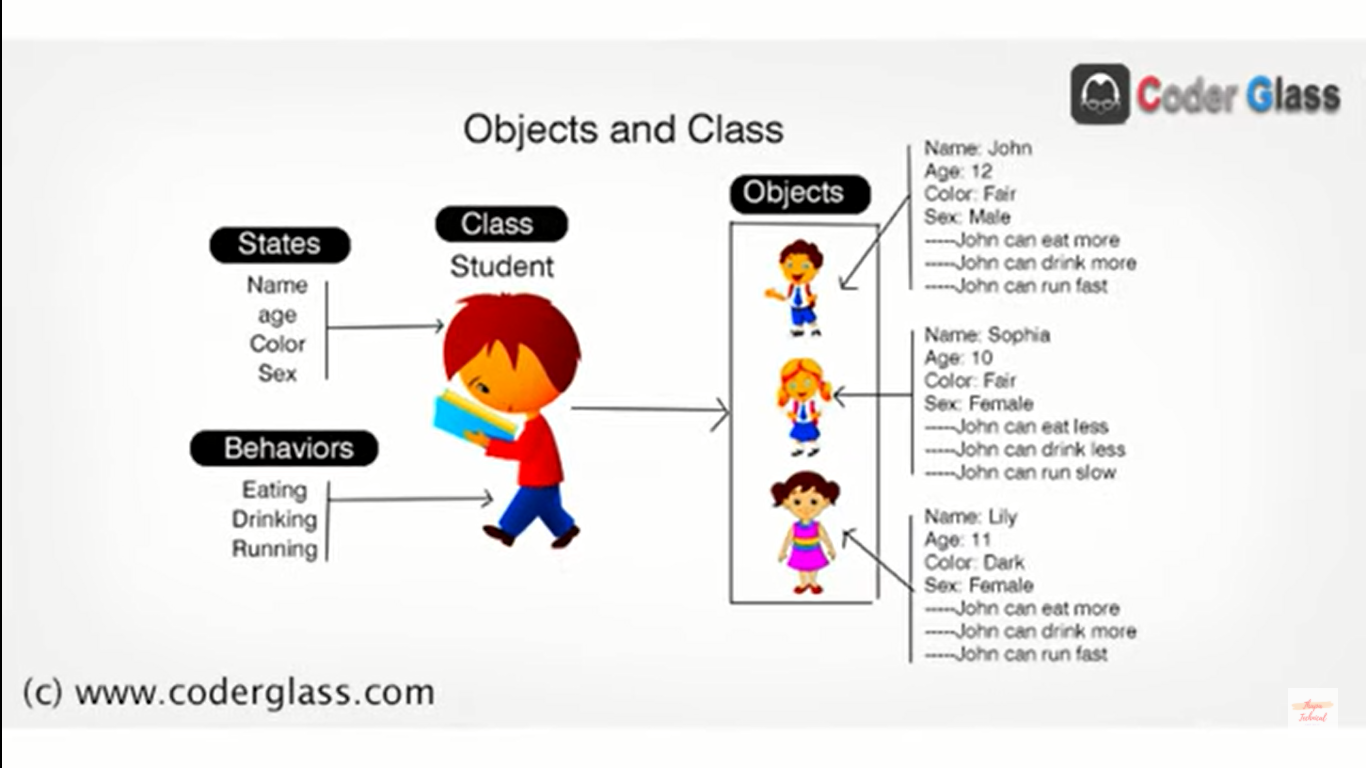
const { b, ...rest } = obj;

return rest;

}  
myFunction({ e: 6, f: 4, b: 5, a: 3 });

**Output:**  
{e: 6, f: 4, a: 3}

**Classes in JS**



In programming languages we often say “An object is an instance of a class”.  
This means that, using a class, I can create many objects and they all share methods and properties.

Since objects can be enhanced, there are many ways to create objects sharing methods and properties. But we want the simples one.

ES6 provides the keyword class, making it very easy to create a CLASS.

**A/C to Mozilla MDN page about Classes:**  
JavaScript classes, introduced in ECMAScript 2015, are primarily syntactical sugar over JavaScript’s existing prototype-based inheritance.

A class is a type of function, but instead of using the keyword function to inititate it, we use the keyword clas, and the properties are assigned inside a constructor() method.

**constructor()** is the pre-defined method of a class, & properties are assigned in it but not user-defined methods, which will be called automatically by the JavaScript without calling it manually. It accepts parameters.

**//for OPPs**  
A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one/same type.

**Example: //Class  
//Student Class**  
class Students{

**//constructor() is the pre-defined method of class 'Students', & properties are assigned in it but not user-defined methods, which will be called automatically by the JavaScript without calling it manually. It accepts parameters.**

constructor(name, age, rollNo){ *//Accepting Parameters*

this.myName= name; *//Here, this keyword is referring to the Students class as the current context. .myName is the key storing the value of name.*

this.Age= age;  
 this.myRollNo= rollNo;  
 }

bioData(){

console.log(`Hi I am a class method & this is my name: ${this.myName}, My Age: ${this.Age} & my Rollno: ${this.myRollNo}`);

}

}

let obj1 = new Students('Chandan', 21, 01); *//Here, we're calling class Students & Passing arguments by creating an instance of class Students using new keyword.*

let obj2 = new Students('Amar', 35, 09); //We can create multiple Objects using a single class.

console.log(obj1);  
obj1.bioData();  
obj2.bioData();

**Output:**Students {myName: 'Chandan', Age: 21, myRollNo: 1}

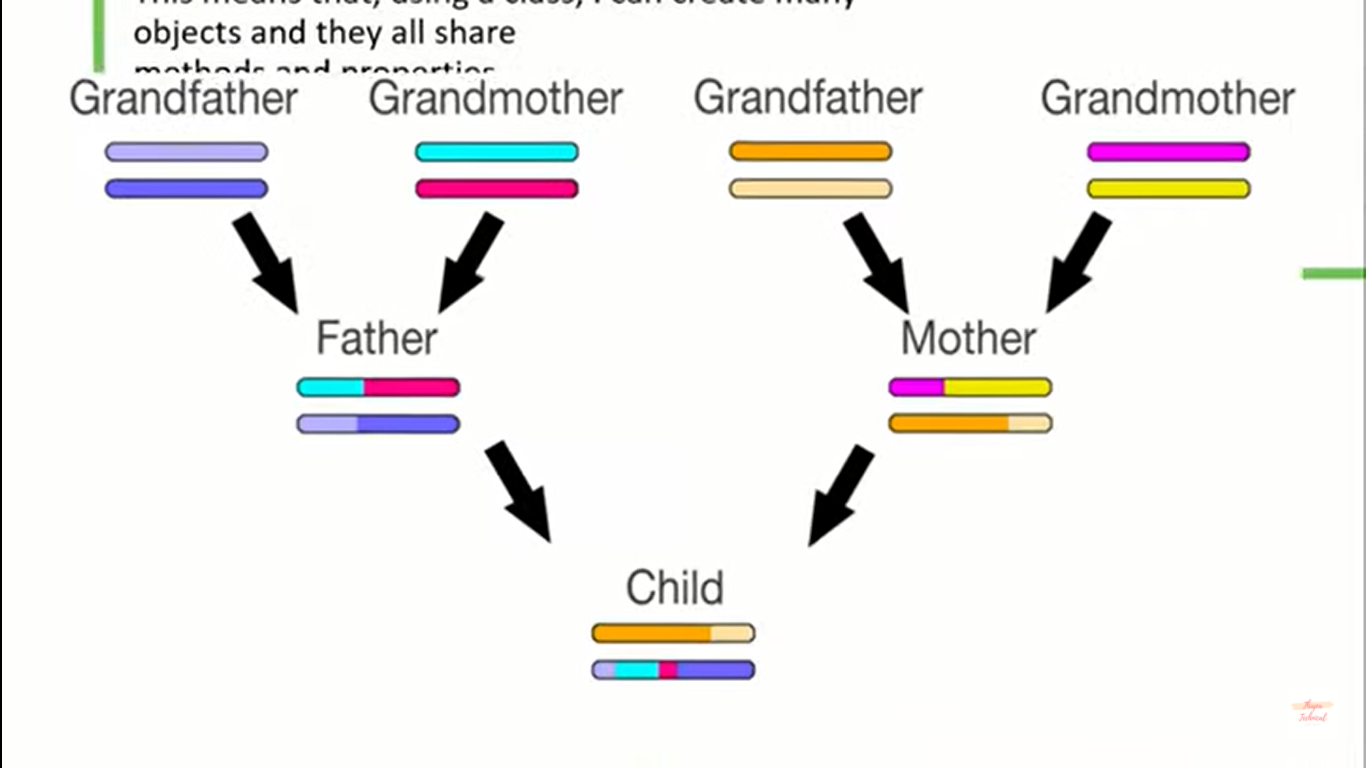
Hi I am a class method & this is my name: Chandan, My Age: 21 & my Rollno: 1

Hi I am a class method & this is my name: Amar, My Age: 35 & my Rollno: 9

**Note:**

* Whenever, we create an instance or say object (because object is the instance of class) of any class, by default constructor method or say function (because method itself consisting functions) will be called automatically by JS.

**Inheritance:**

****

If we’ve a *parent class* consisting same methods and properties, which can be easily inherited to a new derived class using ‘**extends**’ keyword .

**Instance Method Vs Static Method**

Instance Methods, means any child class can easily inherit instance method of parent class under their class but cannot inherit Static Methods.   
To create static method, we just have to add a ‘**static’** keyword before defining any method.

**Example: *Inheritance in class***

**//Student Class i.e. Parent Class (in reference of Player Class)**

class Students{

**//constructor() is the pre-defined method of class 'Students', & properties are assigned in it but not user-defined methods, which will be called automatically by the JavaScript without calling it manually. It accepts parameters.**

constructor(name, age, rollNo){ ***//Accepting Parameters***

this.myName= name; ***//Here, this keyword is referring to the Students class as the current context. .myName is the key storing the value of name.***

this.Age= age;  
 this.myRollNo= rollNo;

}

**//This is Instance Methods, means any child class can easily inherit instance method under their class but cannot Static Methods. To create static method, we just have to add a static keyword before defining any method.**

bioData(){

console.log(`Hi I am a class method & this is my name: ${this.myName}, My Age: ${this.Age} & my Rollno: ${this.myRollNo}. My favorite GAME is ${this.myGame} `);

}  
}

**// Player Class i.e. Child Class or Derived Class (after using extends or say inherit Students class it becomes child class)**

class Players extends Students{ ***//using extends keyword we can inherit the properties & methods of parent Class.***

**//Now in order to add some new Properties to the existing properties of parent class, we have to declare that property inside super constructor using super() method.**

constructor(name, age, rollNo, game){ ***//passing new parameter, to add new properties.***

super(name, age, rollNo);  **//passing parameter of parent class to access their properties in super() method.**

this.myGame= game;

}

playerBioData(){  
 super.bioData(); ***//to get the access of parent class's method using new method in new drived class*** }  
}

let obj1 = new Players('Chandan', 21, 01, 'Cricket');  **//Here, we're calling class Students & Passing arguments by creating an instance of class Students using new keyword.**

let obj2 = new Players('Amar', 35, 09, 'Football');  ***//We can create multiple Objects using a single class.***

console.log(obj1);  
obj1.bioData();  
obj2.bioData();

obj1.playerBioData();

**Output:**Players {myName: 'Chandan', Age: 21, myRollNo: 1, myGame: 'Cricket'}

Hi I am a class method & this is my name: Chandan, My Age: 21 & my Rollno: 1. My favorite GAME is Cricket

Hi I am a class method & this is my name: Amar, My Age: 35 & my Rollno: 9. My favorite GAME is Football

Hi I am a class method & this is my name: Chandan, My Age: 21 & my Rollno: 1. My favorite GAME is Cricket

**Recap:**

* Object Oriented Programming (Opps) always follow **DRY Concept** i.e. **D**on’t **R**epeat **Y**ourself
* Ex:-  
  **Objects**

*Chandan*  
1. Features or properties:- full name, height, weight  
2. Actions:- walk, eat  
*Jeff Bezos*  
1. Features or properties:- full name, height, weight  
2. Actions:- walk, eat

*Elon Musk*  
1. Features or properties:- full name, height, weight  
2. Actions:- walk, eat  
**Class**  
Since, here properties & actions of all Objects (Chandan, Jeff Bezos & Elon Musk) is Repeating same, which disobeying / violating the concept of OPPs i.e. Dry.  
So, class came into play to Obey this concept. Now all Objects are Man (Human Beings). Now the class would be Man consisting properties & actions would be same for all of their object.  
Chandan (man)  
Jeff Bezos (man)  
Elon Musk (man)   
  
class – (Grouping of Similar Object)  
  
Chandan (man) – Chandan is a object of man class,  
Jeff Bezon (man) – Jeff Bezon is a object of man class,  
Elon Must (man) – Elon Musk is a object of man class.

lets create another class namely, Software Engineer  
Now total Classes,  
Class (Man)  
1. Features or properties:- full name, height, weight  
2. Actions:- walk, eat

Class (Software Engineer)  
1. Features or properties:- full name, employee id, year of experience, expertise, height, weight  
2. Actions:- walk, eat, code, test, debug, deploy

Since, Both Man & software Engineer are Human Beings. Or say, Chandan is an Object of both Man & Software Engineer Class. So, some properties & action would same in Software Engineer class. But again here violation of DRY concept of OPPs has taken place.  
  
**Inheritance**So, to Obey the concept of DRY, Inheritance comes into play, which basically inherit the similar properties of parent class (Here Man) into new child/derive class (here Software Engineer)  
Hence,  
Class (Man)  
1. Features or properties:- full name, height, weight  
2. Actions:- walk, eat

Class (Software Engineer inherit man class)  
1. Features or properties:- employee id, year of experience, expertise   
2. Actions:- code, test, debug, deploy

* The standard of creating class is that, the first letter of the class should be upper case. Ex: class Player{}

90th video 16min