ES2015 class

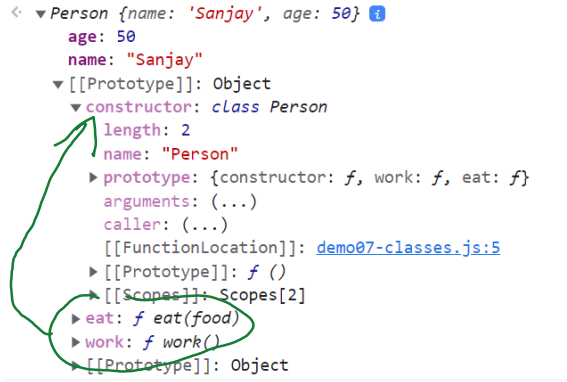
Monday, March 14, 2022 4:48 PM

- ES2015 introduces the concept of classes.
- They are alternative of constructor function
- They can be considered as different syntax for constructor
- It is to make JavasSript closer to other langauges like c++/java/C#

```
class Person{

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

    work(){
        console.log(`${this.name} is working`);
    }

    eat(food){
        console.log(`${this.name} is eats ${food}`)
    }
}

let p= new Person("Sanjay",50);
p.eat('Lunch');
```

Note:

- constructor is simply called constructor
  - In most language the constructor name is same as that of class
- Fields are declared directly in constructor and not in the class.
- function inside the class don't have functionprefix
- Class functions are directly added to the constructor prototype



The two codes below are identical in their work

ES2015

```
class Person{

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

    work(){
        console.log(`${this.name} is working`);
    }

    eat(food){
        console.log(`${this.name} is eats ${food}`)
    }
}

let p= new Person("Sanjay",50);
p.eat('Lunch');
```

ES 5

```
function Person( name, age){
    this.name=name;
    this.age=age;
}

Person.prototype.work=function(){
    console.log(`${this.name} is working`);
}

Person.prototype.eat=function(food){
    console.log(`${this.name} is eats ${food}`)
}

let p= new Person("Sanjay",50);
p.eat('Lunch');
```

Static members of class

- A class can have static methods
- These methods belong to the class
- These methods are called using class name referece

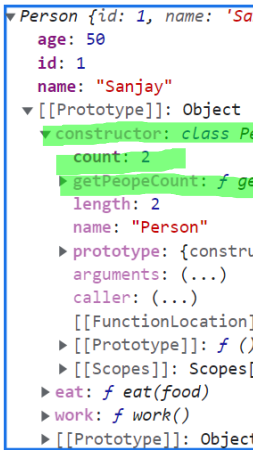
```
class Person{

    static count=0; //a single member to keep a track all person object and give them an id

    static getPeopeCount(){ return Person.count;}

    constructor( name, age){
```

- Fields and methods can be de
- Always referred using class na
- For static field there would be in

```
10          this.id=++ Person.count; //increase count and add to id
11          this.name=name;
12          this.age=age;
13      }
14
15      work(){
16          console.log(`${this.name} is working`);
17      }
18
19      eat(food){
20          console.log(`${this.name} is eats ${food}`)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

D:\MyWorks\Corporate\202203-infogain-react\basic-html-css-js>node "d:\MyWorks\Corporate\202203-infogain-r
\demo08-static.js"
Person.getPeopeCount() 0
p1 Person { id: 1, name: 'Sanjay', age: 50 }
p2 Person { id: 2, name: 'Shivanshi', age: 15 }
Person.getPeopeCount() 2

D:\MyWorks\Corporate\202203-infogain-react\basic-html-css-js>
```

- Static is attached to the const directly and NOT to the const prototype.



Class Property

- Properties are functions that are used like ordinary fields
- They can be great to validate the value before setting
- They may be useful to define readonly properties

```
class Person{
    constructor( name, age){

        this.Name=name;
        this.Age=age;
    }

    get Name(){
        return this.name;
    }
    set Name(value){
        if(value.length<2)
            throw new Error("Invalid value. Name must be at least 2 chars");
        this.name=value;
    }

    get Age(){
        return this.age;
    }

    set Age(value){
        if(isNaN(value) || value<0)
            throw new Error("Invalid age. Must be age >= 0")

        if(value<this.age)
            throw new Error(`${this.name} can't get younger now`);

        this.age=value;
    }
}
```

- Get functions are called when we try to access the fie

```
Person person= new Person( "Sanjay", 50);

Console.log ( person.Age );
```

```
person.Age = 60;
```

- **It calls the function person.Age(60);**
- **Method call looks like assignment**

Inheritance

- One class can inherit the property from another class
- Inherited class gets all the properties and methods of the class it is inheriting (referred as super class)
- It defines an "is a type of relationship"

```
class Employee extends Person{

    constructor(id, name, age, salary){
        super(name,age); //calls Person constructor

        this.id=id;
        this.salary=salary;
```

- A new class extends an existing (super) class using
- Now all property of the Person is available to Emp

```
    }

    work(){
        console.log(`${this.name} works for a salary of ${this.salary}`);
    }

}
```

- Employee constructor MUST pass the properties t using super call
- Super call if present must be first statement in the

instanceof operator

- JavaScript supports **instanceof** operator to check if a given value is of a given type or not

```
var  emp = new Employee(1, "Sanjay", 50, 50000);

console.log( emp instanceof Employee);  //true

console.log( emp instanceof Person);  //true. Person is super class

console.log( emp instanceof Book);  //false

console.log( emp instanceof Object);  //always true. Everything is an object
```