Create Blueprint for Multiple Object

Friday, March 11, 2022 5:40 PM

- Javascript allows couple of way to create template to create multiple objects

Apporach #1 Factory Method

- Not really a JS feature.
- A simple programming pattern that we can use to create object

```
function createPerson(name, age){

    let person= new Object();
    person.name=name;
    person.age=age;

    person.show= function(){
        console.log(`Person [Name=${this.name}\tAge=${this.age}]`);
    };

    person.eat=function(food){
        console.log(`${this.name} eats ${food}`);
    };


    return person;

};

var people= [
                createPerson("Sanjay",50),
                createPerson("Prabhat", 40),
                createPerson('Shivanshi',15),
];

for(let person of people){
    person.show();
    person.eat('Lunch');
};
```

- We create a function that can
  1. Create one Object
  2. Attach all required property and methods
  3. Return that object

- We can call this method every time I want to create an object of a given type

Aside

- Javascript allows a trailing comma in lists and other comma separated elements like json object

- This is helpful
  - If we need to add another object later
  - Comment out last item of the list

Approach #2 constructor pattern

- Javascript provides a special syntax for a method to act as Object constructor

- A constructor is similar to a factory in the sense that it knows how to create object

- A constructor however
  1. Is a Javascript language feature
  2. It doesn't explicitly create The object
  3. It attaches all property and method to 'this' reference
  4. It doesn't return the object (because it has not created any)

Factory Method Style

```
const  createPoint= function(x,y){

    var p=new Object();

    p.x=x;

    p.y = y;

    p.toString=function(){
        return `Point(${this.x},${this.y})`;
```

Constructor Pattern

```
const  createPoint= function(x,y){

    var p=new Object();

    this.x=x;

    this.y = y;

    this.toString=function(){
```

```
    }

    return p;

}
```

To create an object of the point

Factory Method Style

- 'new' is already used inside factory method
- We simply call function to get 'new' object

```
var  p1 = createPoint(3,4);
```

A word on Naming Style

- I find "new" and "create" as redundant in the line

```
var p1 = new createPoint(3,4)
```

- We can drop the word 'create' from method name
- Conventionally a constructor function name should begin with upper case letter

```
const  createPoint= function(x,y){

    var p=new Object();

    this.x=x;

    this.y = y;

    this.toString=function(){
        return `Point(${this.x},${this.y})`;
    }

    return p;

}


var  p= new Point(3,4);
```

Difference between factory and constructor

- A factory uses 'Object' constructor
- Constructor uses a special named constructor
- Object knows what constructor created it.
- Constructor function can give special identity to an object which factory function can't

Object Created using Factory


```
        return
        `Point(${this.x},${this.y})`
        ;
    }

    return p;

}
```

Constructor Method

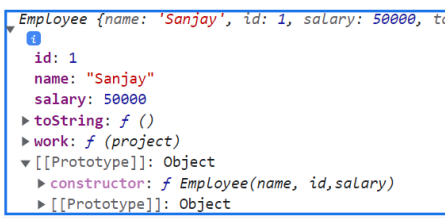
- We haven't called 'new' anywhere yet
  - We have to call it now
- We are not returning any object
  - It will be automatically returned.
- We will be using constructor method name instead of Object to create Object

```
var  p1 = new createPoint(3,4);
```

IMPORTANT!

- Javascript (ES5) has concept of construcgt
  - ES2015 also has the class concept

Object Created using Employee



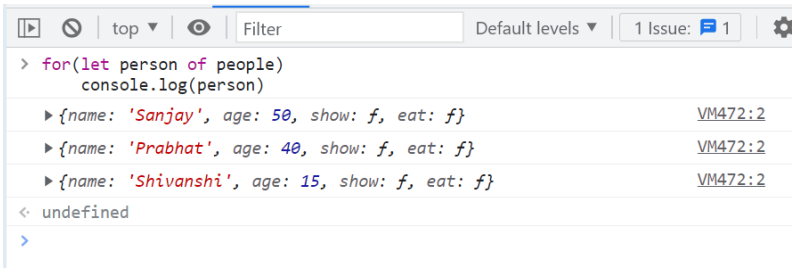
- Doesn't know what is the purpose of it's existence
- It has a generic constructor that is 'Object'

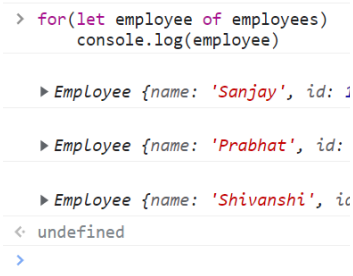
- Recognizes its own constructor
- We can check the constructor name to find o
  we are dealing with

Why is it important to know my constructor?

- Let us consider another issue first.
- In javascript every time we create a new object (either using factory or constructor) it duplicates all the fields using method

- If there are 10 person, there are 10 copies of eat function in memory
  - It reduces the performance and increases memory consumption





- While it makes sense that every person should have their own name or age, there is no point in having them separate eat or show as the logic is exactly same

Constructor Prototype

- Constructor method have special property called prototype
- If we add some information to the prototype, that information will be available in all object created using that constructor even if those objects are created before adding new information

- We generally add methods to the prototype

```
var p1 = new Person("Sanjay", 50);

//p1 has two methods —> show() and eat() at this point


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

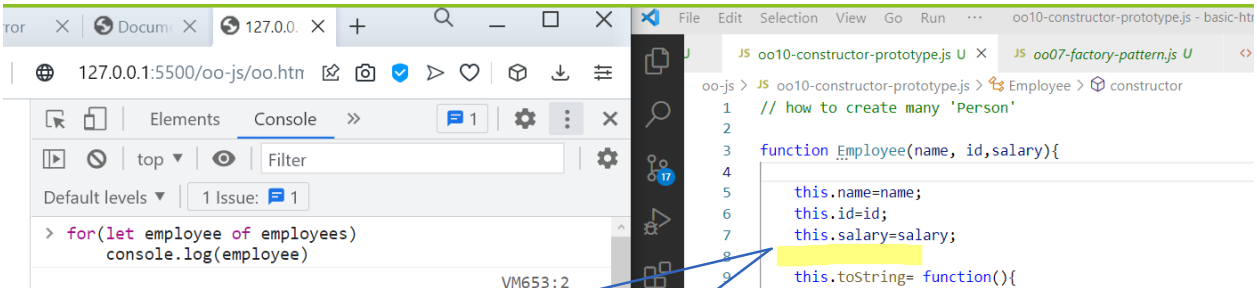
- Now p1 also can use a method 'dance' that is not part of p1 but part of Person prototype

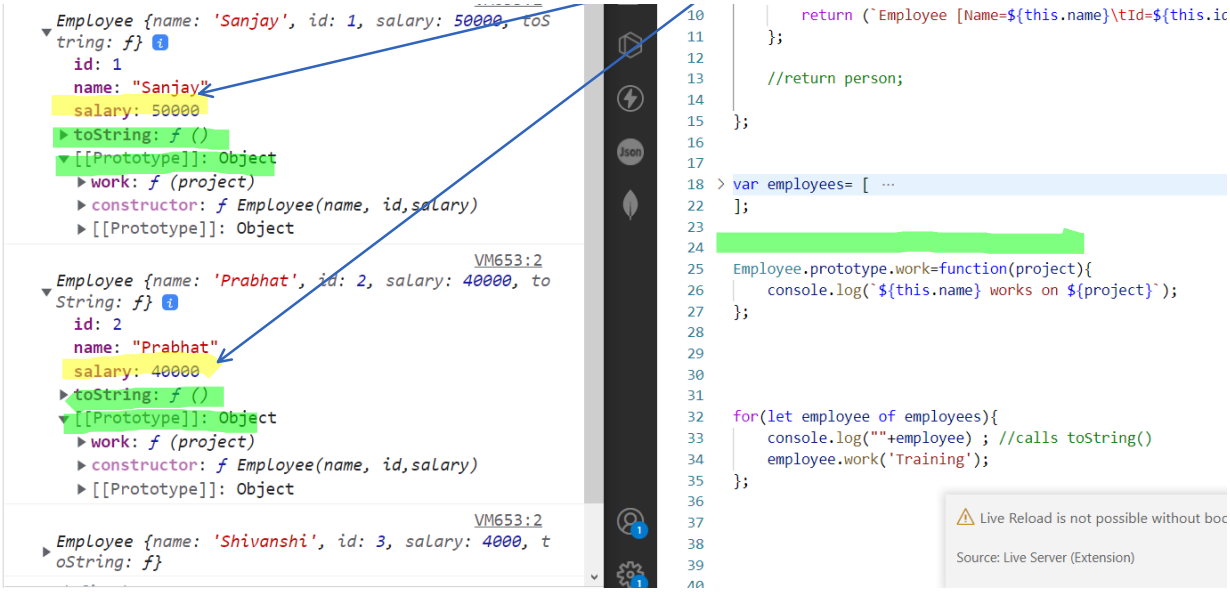
```
p1.dance();


var p2= new Person('Shivanshi', 15);


//both p1 and p2 has their own eat, and show
//but both share same function dance

p2.dance();
```






```
> employees[0].toString===employees[1].toString
< false
> employees[0].work===employees[1].work
< true
```