OBJECT ORIENTED PATTERN IN JS

By : Kratika Chowdhary

FACTORY PATTERN

- Abstract process of creating object
- Encapsulate the creation of object with specific interfaces
- Did not address the issue of object identification

```
function createPerson(name, age, job){
    var o = new Object();
    o.name = name;
    o.age = age;
    o.job = job;
    o.sayName = function(){
        alert(this.name);
    };
    return o;
}

var person1 = createPerson("Nicholas", 29, "Software Engineer");

console.log(person1 instanceof createPerson) // false
```

CONSTRUCTOR PATTERN

- Custom constructors that define properties and methods for your own type of object.
- In this no object is explicitly created .
- The properties and method are assigned directly onto the this object.
- Downside to constructors is that methods are created once for each instance.

```
function createPerson (name, age, job) {
  this.name = name;
  this.age = age;
  this.job = job;
  this.speak =  function speak() {
    return this.name;
  }
}
 var Person1 = new createPerson('Nick', 24, 'Teacher')


 console.log( Person1 instanceof createPerson) // true
```

PROTOTYPE PATTERN

- Each function is created with a prototype property, which is an object containing properties and methods that should be available to instances of a particular reference type
- Negates the ability to pass initialization arguments into the constructor, meaning that all instances get the same property values by default.
- Due to its shared nature it is difficult to have separate ref value

CODE

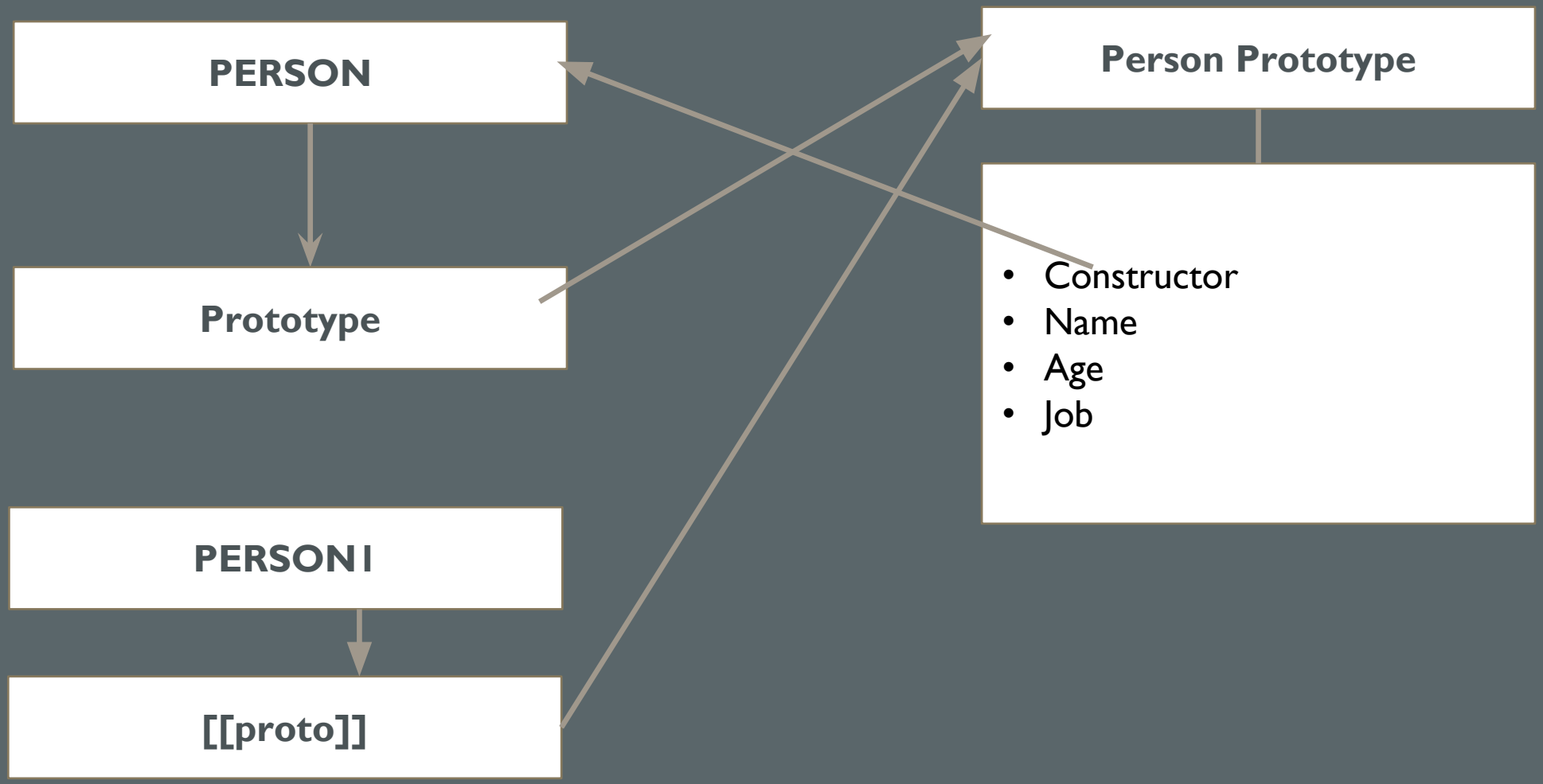
```
function Person(){
}

Person.prototype.name = "Nicholas";
Person.prototype.age = 29;
Person.prototype.job = "Software Engineer
Person.prototype.sayName = function(){
    alert(this.name);
};

var person1 = new Person();
person1.sayName();   //"Nicholas"

var person2 = new Person();
person2.sayName();   //"Nicholas“
```

HOW PROTOTYPE WORKS



HYBRID PATTERN

- Making use of best of both world that is Prototype and Constructor
- The constructor pattern defines instance properties, whereas the prototype pattern defines methods and shared properties.

```
function Person(name, age, job){
    this.name = name;
    this.age = age;
    this.job = job;
    this.friends = ["Shelby", "Court"];
}

Person.prototype = {
    constructor: Person,
    sayName : function () {
        alert(this.name);
    }
};

var person1 = new Person("Nicholas", 29, "Software Engineer");

console.log(person1 instanceof Person) // true
```

DYNAMIC PROTOTYPE PATTERN

- Developers coming from different OO based language find hybrid pattern confusing to avoid this

```
function Person(name, age, job){

    //properties
    this.name = name;
    this.age = age;
    this.job = job;

    //methods
    if (typeof this.sayName != "function"){

        Person.prototype.sayName = function(){
            alert(this.name);
        };

    }
}

var friend = new Person("Nicholas", 29, "Software Engineer");
friend.sayName();
```

PARASITIC CONSTRUCTOR PATTERN

- Fallback if other pattern fail
- Create a constructor that simply wraps the creation and return of another object while looking like a typical constructor.
- This is exactly the same as the factory pattern except that the function is called as a constructor, using the new operator.
- This pattern allows you to create constructors for objects that may not be possible otherwise.
- There is no relationship between the returned object and the constructor or the constructor's prototype; so cannot rely on instance of operator

```
function myArray() {
  var o = new Array();
  o.getOdd = function Odd() {
    return this.filter((x) => x % 2 !== 0)
  }
  return o;
}

var arr1 = new myArray();
arr1.push(3)
arr1.push(2)
arr1.push(5)
arr1.push(6)
console.log(arr1.getOdd()) //[3,5]
```

DURABLE CONSTRUCTOR PATTERN

- Objects that have no public properties and whose methods don't reference the this object.
- Best used in secure environment.
- No way to access any of its data members without calling a method.

```
function Person(name, age, job){

    //create the object to return

    var o = new Object();

    //optional: define private variables/functions here

    var lastName = 'Chowdhary'

     //attach methods

    o.sayName = function(){

        console.log(name+lastName);

    };

    return o;

}

var friend = Person("Nicholas", 29, "Software Engineer");

friend.sayName();  //"KratikaChowdhary"
```

THANK YOU

Linkedin.com/in/kratikac

Kratika0907@gmail.com