ES 2015

Dr. Peiwei Mi

UNIT IV – OBJECT AND OOP

- OOP Review
- Objects
- OOP
- Implementing OOP in ES5

OBJECT-ORIENTED PROGRAMMING REVIEW

- Object
  - A powerful and flexible data structure to represent real world entities
  - Property with Data Type as State – Name, Value, ~~Type~~
  - Method as Behavior – Function
  - ID - unique
- Class / Instance
- Relationship / ID as Reference
- Inheritance – Single / Multiple
- OOP .vs. OOD
- In ES5, JavaScript Object, out of box, is NOT OOP

https://www.codeproject.com/Articles/22769/Introduction-to-Object-Oriented-Programming-Concep

OBJECTS

JS OBJECTS – SAME AS IN OOP

- Object is a container that can represent real world things with multiple or complex values, and can change over their lifetime
- It can contain properties with Dynamic Typing (or members)
  - consists of a name (or key) and value
- It can also contain functions (or methods)
  - Consists of a name and an anonymous function
- They are all instances of Object class

TYPES OF OBJECTS

- A native object / variable
  - defined by the ECMAScript specification
  - for example: arrays, functions, dates, and regular expressions
- A host object
  - defined by the host environment (such as a web browser) within which the JavaScript interpreter is embedded
  - For example: in a browser, window is the object for the hosting browser environment
- A user-defined object
  - created by the execution of JavaScript code

1-object-def.js

CREATING OBJECT (INSTANCE)

- object literal
- “new” keyword
- Object.create()

```
var empty = {};                                 // An object with no properties
var empty = new Object();
var point = { x:0, y:0 };                       // Two properties
var book = {
            "mainTitle": "JavaScript",          // Property names include spaces
            ‘subTitle': "The Definitive Guide", // and hyphens, so use string literals
            "for": "all audiences",             // for is a reserved word, so quote
            author: {                                   // The value of this property is
                        firstName: "David",     // itself an object. Note that
                        lastName: "Flanagan"    // these property names are unquoted.
            }
};
```

‘NEW’ OPERATOR

- Must be followed by a function invocation
- A function used in this way is called a constructor (or class)
- Built-in constructors for built-in data types (classes)

```
// In general, they should be avoided if possible
var o = new Object();      // Create an empty object: same as {}
var a = new Array();       // Create an empty array: same as []
var d = new Date();        // Create a Date object representing the current time
var r = new RegExp("js");  // Create a RegExp object for pattern matching

var n = new Number();    // should not be used
var s = new String();
var b = new Boolean();
```

GETTING AND SETTING PROPERTIES

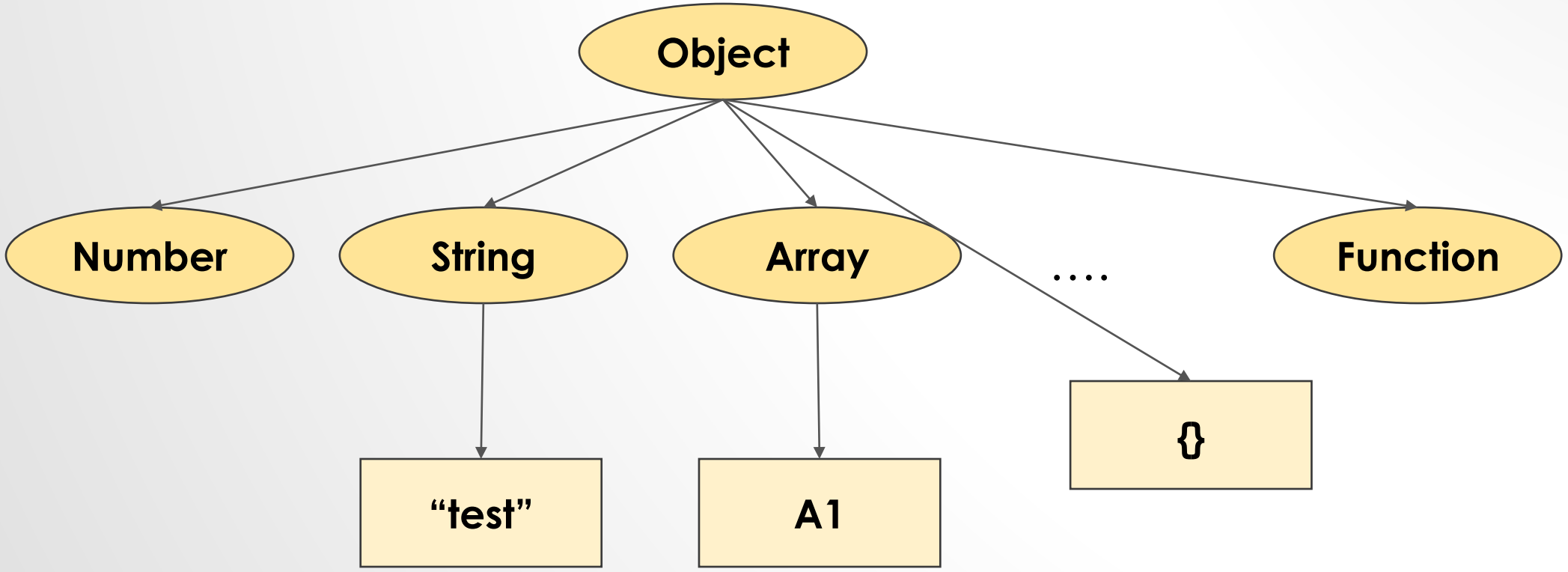
- To obtain the value of a property: dot (.) or square bracket ([ ]) operators on the right side of the assignment expression

```
var author = book.author;          // Get the "author" property of the book.
var name = author.lastName         // Get the “lastName" property of the author.
var title = book["mainTitle"]      // Get the "mainTitle" property of the book.
```

- To create or set a property: dot (.) or square bracket ([ ]) operators on the left side of the assignment expression

```
book.edition= 6;                          // Create an "edition" property of book.
book["mainTitle"] = "ECMAScript";         // Set the "mainTitle" property.
```

OBJECT HIERARCHY – 'OBJECT BASED'



Both variables and constants have inherited properties and methods
Even if they are empty.

SYSTEM OBJECT PROPERTIES

- prototype is a reference to another object from which properties are inherited
- class is a string that categorizes the type of an object
- extensible flag specifies (in ES6) whether new properties may be added to the object

2-object-def.js

EVERYTHING IS OBJECT, INTERNALLY

- Every variable is actually ‘an object’
- It inherits class properties and methods
  - For example, strings and arrays have methods/functions
- Even constants are objects!

```
var test = "some string";
console.log(test[7]);                  // shows letter 'r'
console.log(test.charAt(5));           // shows letter 's‘

console.log( "test".charAt(1) );       //shows letter 'e'
console.log( "test".substring(1,3) );  //shows 'es'
```

THIS KEYWORD

- this is a special read-only reference used inside a function
- When a function called, 'this' takes on the value of the specific object it was called on
- Bound '*where/how the function is called, not where the function is declared*'.

```
const o = {
        name: 'Wallace',
        speak: function() { return 'My name is ' + this.name + '!'; },
}

o.speak();                              // "My name is Wallace!
```

THIS KEYWORD

3-object-this.js

Left side of the function

- To use ‘this’, we need to know
  - “Where is this function being invoked?”
  - It is where the function using the this keyword was invoked / called
- There are 5 scenarios:
  - Implicit Binding – most common, equal to the object left to ‘.’
  - Explicit Binding – using call function to bind to an object
  - new Binding – using new to bind, see next
  - Lexical Binding – arrow functions don’t create context / this
  - window Binding – be default, this always references window object in Browser

3-operators.js

TWO ES2020 OPERATORS

- Optional Chaining (?.)
  - Used to get multi-level of object properties that could be non-existence
  - allows to access deeply nested object properties without having to worry if the property exists.
  - myObj.prop5.prop6 crashes code if prop5 doesn't exist
  - myObj.prop5?.prop7
- Nullish Coalescing Operator (??)
  - Used to assign (default) values whether a variable is null/undefined or not
  - This is a better choice than using 'or' ('||') operator
  - let foo = test ?? 'default value'

CLASS AND OOP

CLASSES

- ES6 introduced OOP syntax
  - ES5 uses function/prototype to implement OOP. Discuss later
- Defining and instantiating classes using 'class'
  - Class ClassName{ }
- Creating subclasses using 'extends'
  - Class subClassName extends superClassName{ }
- Calling super-classes using 'super'
  - Within subclass constructor calls
    - super(parameter1, parameter2, … , parameterN)
  - To refer to methods in the superclass
    - super.methodName()

4-oop-class.js

DEFINING A CLASS

- Use **class** keyword
- Define constructor and methods inside
  - one constructor function per class
- Really just syntax sugar over existing prototypal inheritance mechanism
  - creates a constructor function with same name as class
  - adds methods to prototype
  - discuss later

5-oop-subclass.js

DEFINING A SUBCLASS

- Inherit with **extends** keyword
- In subclasses, **constructor must** call **super(*args*)** and it must be **before this** is accessed
  - because the highest superclass creates an object too

6-oop-getset.js

'VIRTUAL' PROPERTIES WITH GET/SET

- Default behavior of get/set of behaviors can be changed.
- Via new get/set functions.
- The new implementation will override the default one.

Many libraries use this technique to expose only necessary properties

```
class o {
  constructor() {
    this.mx = 'initial';
  }
  get x() {
    console.log('x is being retrieved!');
    return this.mx;
  }
  set x(val) {
    console.log('x changed');
    this.mx = val;
  }
}

let e = new o();
console.log(e.x);
e.x = 5;
```

OOP IN ES5 – FOR UNDERSTANDING

7-es5-constructor.js

CONSTRUCTOR PATTERN

To use a 'Constructor' function, which extends system-defined constructors, such as Array and Object

```
function Person(name, age, job){
    this.name = name;
    this.age = age;
    this.job = job;
    this.sayName = function(){
                                console.log(this.name);
                                };
}

var person1 = new Person("Nicholas", 29, "Software Engineer");
var person2 = new Person("Greg", 27, "Doctor");
```

‘NEW’ OPERATOR

1. Create a new object.
2. Bind the ‘this’ value of the constructor to the new object (so this points to the new object).
   - this pointer’s ‘new’ binding
3. Execute the code inside the constructor (adds properties to the new object).
4. Return the new object.

CONSTRUCTOR

- They are normal functions
- The only difference between is the way in which they are called, with “new”

```
var person = new Person(“Nicholas”, 29, “Software Engineer”);
person.sayName();                    //”Nicholas”
```

- Constructor can be used in normally ways, without binding this:

```
//call as a function in brower
Person(“Greg”, 27, “Doctor”);          // window binding
window.sayName();                      // ”Greg”

//call in the scope of another object
var o = new Object();
Person.call(o, “Kristen”, 25, “Nurse”);
o.sayName();                           // ”Kristen”
```

PROBLEM WITH CONSTRUCTOR PATTERN

7-es5-constructor.js

- Methods (functions) are duplicated!

- One can see it from the constructor function

PROTOTYPE PATTERN

- Each function is created with a prototype property, which is an object containing properties and functions that should be available to all instances of a particular reference class (function).

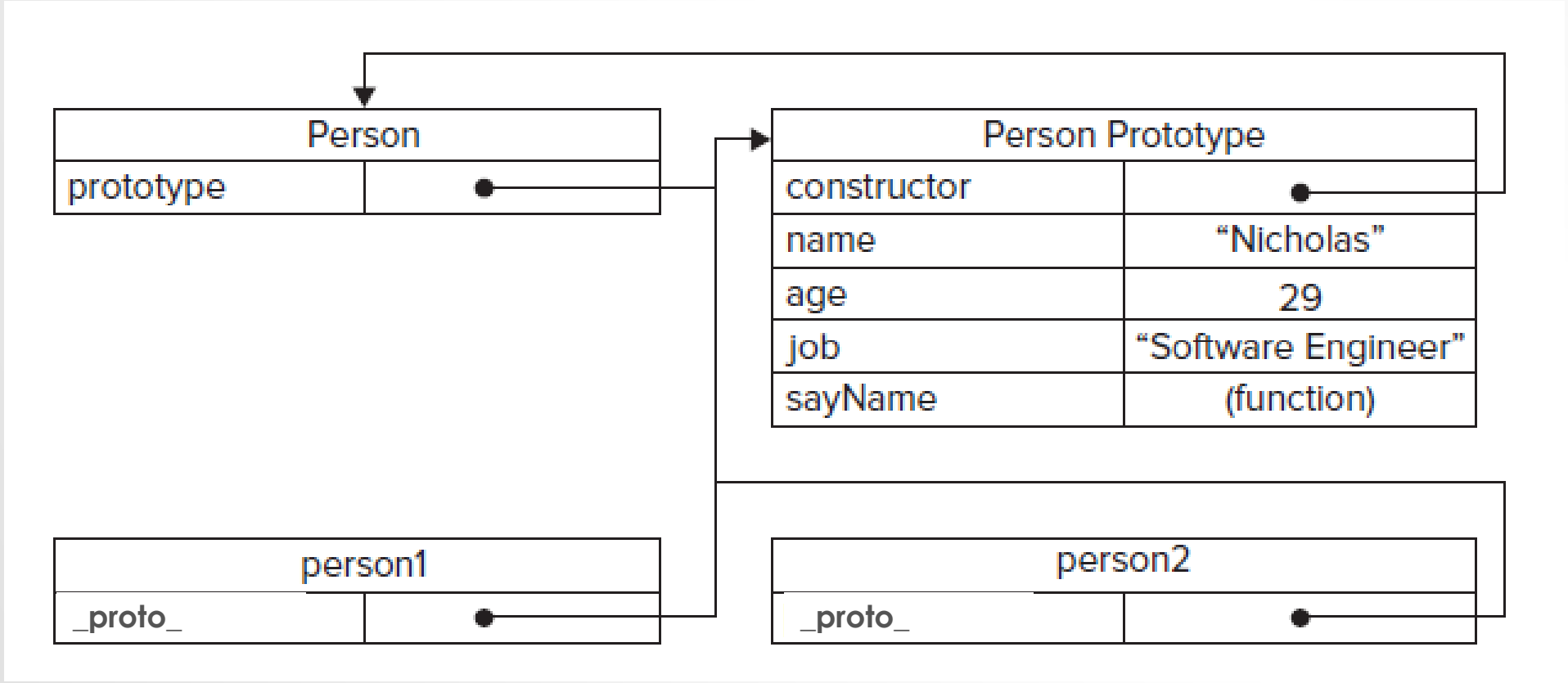
- Unlike the constructor pattern, the properties and functions are all shared among instances, so person1 and person2 are both accessing the same set of properties and the same sayName() function.

8-es5-prototype.js

DECLARATION

```
function Person(){}

Person.prototype.name = “Nick”;
Person.prototype.age = 29;
Person.prototype.job = “Software Engineer”;
Person.prototype.sayName = function(){
                                        console.log(this.name);
                                };

var person1 = new Person();
person1.sayName();                                              //” Nick”
var person2 = new Person();
person2.sayName();                                              //” Nick”
console.log(person1.sayName === person2.sayName);       //true
```

INSIDE PICTURE



8-es5-prototype.js

ACCESS VALUES

- Steps to retrieve a value from a property of an object (such as person1.Name):
  1. If a property with the given name exists on the object (instance), return its value;
  2. Otherwise, go to the object's prototype. If the property exists there, return its value;

8-es5-prototype.js

ASSIGN (=) VALUES

```
function Person(){}

Person.prototype.name = “Nick”;
Person.prototype.age = 29;
Person.prototype.job = “Software Engineer”;
Person.prototype.sayName = function(){
    alert(this.name);
};

var person1 = new Person();
var person2 = new Person();
person1.name = “Greg”;
console.log(person1.name);          //”Greg” - from instance,
                                    // shallow or block the value on the prototype
console.log(person2.name);          //”Nicholas” - from prototype
```

DELETE VALUES

```
function Person(){}

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

let person1 = new Person();
let person2 = new Person();
person1.name = "Greg";
console.log(person1.name); //"Greg" - from instance
console.log(person2.name); //" Nick" - from prototype
delete person1.name;
console.log(person1.name); //" Nick" - from the prototype
```

8-es5-prototype.js

UPDATE CLASS DEFINITION

```
var friend= new Person();

Person.prototype.sayHi = function(){
                                        console.log("hi");
};

friend.sayHi(); //"hi" - works!
```

8-es5-prototype.js

ANOTHER WAY TO DECLARE

```
function Person(){};

Person.prototype = {
    name : "Nick",
    age : 29,
    job : "Software Engineer",
    sayName : function () {
                    console.lg(this.name);
                     }
};

let friend = new Person();
```

Be really careful since this overwrites previous values!

COMPLETE CLASS DEF.

```
function Person(name, age, job){
    this.name = name;
    this.age = age;
    this.job = job;
    this.friends = ["Shelby", "Court"];
}
// --------------------
Person.prototype.sayName = function () {
                                                    console.log(this.name);
                                                    };
// OR
Person.prototype = {
        sayName : function () {
                console.log(this.name);
            }
        };
```

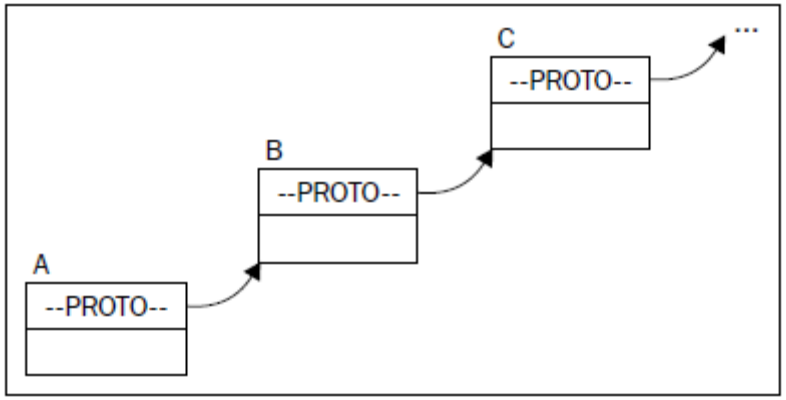
```
let person1 = new Person("Nick", 29, "Software Engineer");
let person2 = new Person("Greg", 27, "Doctor");

person1.friends.push("Van");
console.log(person1.friends);                                   //"Shelby,Court,Van"
console.log(person2.friends);                                   //"Shelby,Court"
console.log(person1.friends === person2.friends);               //false
console.log(person1.sayName === person2.sayName);               //true
```

INHERITANCE

9-es5-inheristance.js

- It is achieved via 'Prototype Chaining' because the properties and methods in a prototype is 'inherited' to its instances

- TwoDShape.prototype = Object.create(Shape.prototype);
- Triangle.prototype = Object.create(TwoDShape.prototype)
- (order matters)



CLASS INHERITANCE

1. Inherit local properties from the super class and set initial values, in TwoDShape's constructor:

```
Shape.call(this, <all initial values>)
```

2. Inherit shared methods from the super class

```
TwoDShape.prototype = Object.create(Shape.prototype)
```

3. Restore proper class / constructor reference

```
Object.defineProperty(TwoDShape.prototype, 'constructor', {
    value: TwoDShape,
    enumerable: false, // so that it does not appear in 'for in' loop
    writable: true
});
```

ES6 -> ES5

ts/class.js
execute it with tsc in console

```
class Dog {
        constructor(name) {
                this.name = name;
        }
        bark() {
                console.log("wuf! wuf!");
        }
}

var leo = new Dog("Leo");
leo.bark();
```

```
function Dog(name) {
        this.name = name;
}

Dog.prototype.bark = function() {
                        console.log("wuf! wuf!");
                        };

var leo = new Dog("Leo");
leo.bark();
```

- ES6 code is actually translated into ES5 code and then executed!
- The process is called ‘transpiling’
- Two popular transpilers: Babel and TypeScript (TS)

HOMEWORK

- #1 – Basic objects
- #2 – OOP. Function execution has to be a.shuffle()
- #3 – Subclasses
- #4 – Singleton class
- #5 – TwoSum class
- #6 – Concepts you learned today and should memorize for interview