

CMSC335

Web Application Development with JavaScript



JavaScript8

Department of Computer Science

University of MD, College Park

Slides material developed by Ilchul Yoon, Nelson Padua-Perez

JavaScript Class

- Prototype-based inheritance using **constructor** function and **new** keyword was discussed
- JavaScript class introduced in ES6 (ECMAScript 2015)
 - Primarily syntactical sugar over prototype-based inheritance
 - Does not introduce a new object-oriented inheritance model
- "special functions" in the form of **class expressions** and **class declarations**
- A class constructor, unlike a function, cannot be invoked unless you use **new**
 - Student() call (**assuming Student is a class will not work**)

Class Declaration

- **Basic syntax**

```
class MyClass {  
    constructor(args) { ... }  
    method1() { ... } // methods are non-enumerable  
    method2() { ... }  
}
```

- **Usage**

```
let c = new MyClass(args);  
c.method1();
```

- **new MyClass()** to create a new object
- **constructor** method is automatically called by 'new'
 - **Used to create and initialize a new object**
 - **Only 1 constructor is allowed**
- **Example:** ClassDeclaration.html

In Fact, class Is Not Really New!

- Creates a function named **StoreBranch**, that becomes the result of the class declaration
 - The function code is taken from the constructor method
- Stores **displayInfo** in **StoreBranch.prototype**

```
class StoreBranch {  
    constructor(name, location) {  
        this.name = name;  
        this.location = location;  
    }  
  
    displayInfo() {  
        document.writeln("<br><strong>displayInfo(): </strong>");  
        document.writeln(this.name + " located in " + this.location + "<br>");  
    }  
}
```

Class Declarations are NOT Hoisted

- **function declarations** are hoisted
- **class declarations** are NOT hoisted
 - Declare your class first and then access it
 - Otherwise code like the following will throw a ReferenceError

```
const p = new Rectangle(); // ReferenceError
```

...

```
class Rectangle {}
```

Class Expression

- Can be **named** or **anonymous**
 - The name given to a named class expression is local to the class's body – similar to function name given in the function expression
 - Class name can be retrieved through the class's (not an instance's) **name** property
- **Basic syntax**

```
let X = class {  
    constructor(args) { ... }  
    method1() { ... }  
}
```
- **Usage**

```
let c = new X(args);  
c.method1();
```
- **Example:** ClassExpression.html

Getters, Setters, and Object Property

- Declaring property of object in class
 - `propertyName = value;` in class declaration (above constructor)
 - The property name is not placed in the prototype
 - It is a property of the object itself
- Getters/setters can be used as wrappers over “real” property values
 - **get** binds an object property to a function that will be called when that property is looked up
 - **set** binds an object property to a function to be called when there is an attempt to set that property
- `getPropName` and `setPropName` vs. `getter/setter` syntax
 - Your preference
- **Example:** `Accessors.html`

Class Inheritance

- “**extends**” keyword
- **Syntax**
 class X extends Y { ... }
- The prototype and constructor function setting is done behind the scene
 - As we saw before, prototype chain will be followed when a function is called
- **Example:** Inheritance.html

Static Properties and Methods

- **Static methods**
 - Assign a method to the class itself
 - **this** in a static method is the class constructor itself
 - Used for **Factory** methods
- **Static properties**
 - Belongs to the class itself
 - Simply, add **static** in front of the variable name
- **Example:** Inheritance.html

Method Overriding

- **Overriding constructor**

- constructors in a child class must call **super(...)** before **using this** in order to override constructor
- `super(...)` is only allowed in constructor
- If not overridden, the following will be created for you

```
constructor(...args) {  
    super(...args);  
}
```

- **Overriding non-constructor methods**

- Simply define a method with the same name in a child class. It will shadow the parent's method
- **`super.method(...)`** to call a parent method

Private Variables

- We can use # to define private variables
- **Example:** Car.html
- **Example:** SportsCar.html