CMSC335

Web Application Development with JavaScript

Maps, Form Validation, Classes

Department of Computer Science

University of MD, College Park

Slides material developed by Ilchul Yoon, Nelson Padua-Perez

Maps

- Collection of key - value pairs
- Keys and values can be primitive or references
- Can define a map via iterable over key-value pairs
- **keys()** - method returns iterable for keys in the map
- **values()** - method returns iterable for values in the map
- **entries()** - method returns iterable over (key, value) pairs
- **Example:** Map.html

WeakMaps

- A collection of key-value pairs
  - **In which primitive data types are not allowed as keys**
- The keys must be objects
- The values can be arbitrary values
- **Key is weakly held; if the object representing a key is the only reference to the object, the object will be garbage-collected**
- You cannot iterate over keys, values, or entries
- You need a key to get some content out of the map
- **Example:** WeakMap.html
  - As defined, the example relies on a Map
  - Run the example in Chrome, select **Inspect,** and under the **Memory** option, select "Heap Snapshot“, and select the "Take snapshot" button at the end
  - Search for " TerpObject" in the **Constructor** column
  - Repeat the above process, but using a WeakMap rather than a map
- Uses - to define private data in a “class”

Immediately Invoked Function Expression (IIFE)

- **IIFE** - approach to define a function so it gets invoked immediately
  - It does not have a name, and it is self-executing
- **Two parts**
  - Anonymous function with lexical scope enclosed within the grouping operator ()
- **IIFE** - Prevents accessing variables within the IIFE idiom as well as polluting the global scope
- Emulates block-scoped variables
- Not needed if “let” is used instead of “var”
- **Example:** IIFE.html

Form Validation

- Form data validation
  - We can validate the data associated with a form by recognizing the submit event
  - **window.onsubmit = validateData;**
    - » validateData is a function that will check for data validity
    - » It will return true or false
- Keep in mind that JavaScript can be disabled therefore, the server application must also validate the data
- Notice the organization code (HTML, CSS, JS separate) in the example
- **Example:** FormValidation

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 one constructor is allowed**

Static Properties and Methods

- **Private variables** - defined using #
- **Static properties**
  - Belongs to the class itself
  - Add **static** in front of the variable's name
- **Static methods**
  - Assign a method to the class itself
  - Add **static** in front of the method's name
- **Example:** ClassDeclaration.html

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 {}
```

Getters and Setters

- 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:** Car.html

Class Inheritance

- "**extends**" keyword used to define inheritance relationship
- **Syntax**

  class X extends Y { … }
- **Example:** SportsCar.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 the 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