JAVASCRIPT

Object-Oriented Design in JavaScript

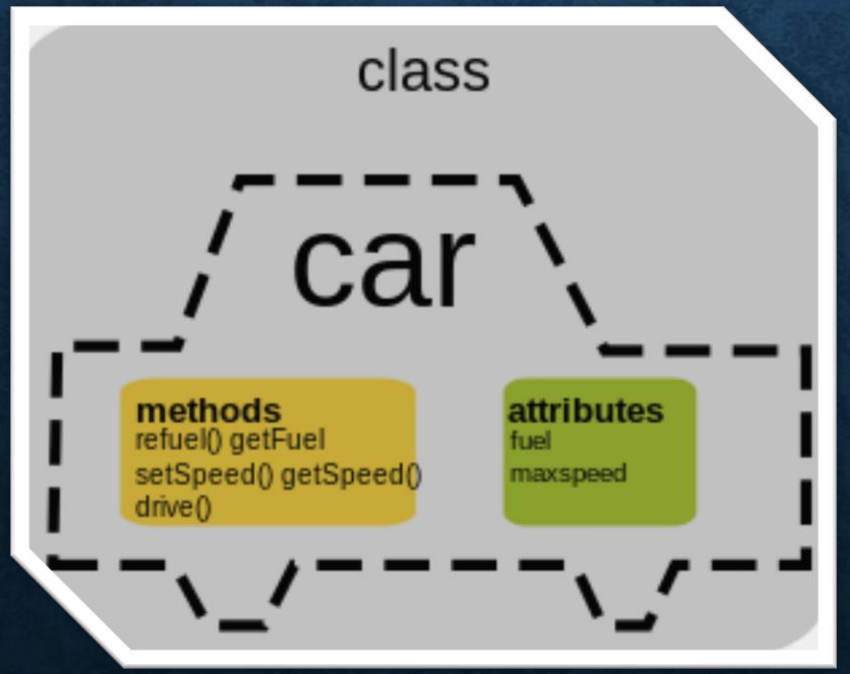
COS216
AVINASH SINGH
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF PRETORIA

OO JS – OVERVIEW

- JS does not natively support classes
  - There is no class declaration like in C++ and Java with **ES5**
- JS is a prototyped language
  - Allows classes to be "emulated" using prototypes
- JS can therefore be used for object-oriented design
  - Full support for polymorphism, due to JS's dynamically-typed nature
  - Limited class, inheritance, and static functionality

OO JS – OVERVIEW

- JS uses functions, prototypes, and variables for OO design
- JS supports function pointers
  - Allows functions to be stored in variables



OO JS – OVERVIEW

- There are many different ways to implement classes
- Some ways are more efficient than other
- Some ways are easier to understand than others
- Only two ways are discussed here
  - Prototypes
  - Nested functions

OO JS – DEFINITION

- Define a new class using a function

```
var Crypto = function() {};
```

- Or

```
function Crypto() {};
```

OO JS – INSTANTIATION

- Use the new keyword to instantiate an instance of the class

```
var coin1 = new Crypto();
var coin2 = new Crypto();
```

- Alternatively, use the built-in create function

```
var coin3 = Crypto.create();
```

OO JS – TYPE

- The typeof operator will always return "object", irrespective of the class

```
typeof coin1;
```

- Use instanceof to evaluate the class of an object

```
coin1 instanceof Crypto;
```

- Or use the constructor name, returned as a string

```
coin1.constructor.name;
```

OO JS – CONSTRUCTORS

- The constructor is called at the moment of instantiation
- In JavaScript the function serves as the constructor of the class
  - No need to explicitly define a constructor
- Every statement in the constructor gets sequentially executed at the time of instantiation
- The constructor is used to
  - Set the object's properties
  - Call methods to prepare the object for use

OO JS – CONSTRUCTORS

```
var Crypto = function()
{
        console.log("Coin created!");
};

var coin1 = new Crypto(); // Prints "Coin created!“
var coin2 = new Crypto(); // Prints "Coin created!"
```

OO JS – ATTRIBUTES

- Attributes are variables contained in the class
  - Typical member variables
  - Each instance has its own copy of those attributes
- Attributes are set in the constructor function of the class
  - Created for each instance
- Create using this keyword, which refers to the current object
- These attributes can be accessed outside the class declaration

OO JS – ATTRIBUTES

```
var Crypto = function()
{
        this.symbol = 'BTC';
        this.name = 'Bitcoin';
        this.price = {btc : 0, eth : 0};
};

var coin1 = new Crypto();
console.log(coin1.name);
console.log(coin1.symbol);
console.log(coin1.price.btc);
```

OO JS – CONSTRUCTORS

- Constructors can take function parameters

```
var Crypto = function(symbol, name)
{
        console.log('Crypto Constructor');
        this.symbol = symbol;
        this.name = name;
        this.price = {btc : 0, eth : 0};
};

var coin1 = new Crypto('BTC', 'Bitcoin');
```

OO JS – METHODS

- Methods follow the same logic as attributes
- Methods can be defined
  - By adding the method to the class prototype
  - By adding the method as a nested function inside the class/constructor

OO JS – METHODS

- Using the prototype

```
Crypto.prototype.print = function()
{
        console.log("Symbol: " + this.symbol);
        console.log("Name: " + this.name);
        console.log("Price: "
        + this.price.btc + " BTC" + ", " + this.price.eth + " ETH");
};
```

OO JS – METHODS

- Using the nested functions

```
var Crypto = function(symbol, name)
{
        console.log('Crypto Constructor');
        this.symbol = symbol;
        this.name = name;
        this.price = {btc : 0, eth : 0};

        this.print = function()
        {
                console.log("Symbol: " + this.symbol);
                console.log("Name: " + this.name);
                console.log("Price: "
                + this.price.btc + " BTC" + ", " + this.price.eth + " ETH");
        };
};
```

OO JS – METHODS

- JS class methods are normal functions
  - They can be "used out of context"
  - They can be assigned to a variable and executed later

```
var coin1 = new Crypto();
var function1 = coin1.print;
function1();
```

OO JS – METHODS

- The JS function call can be used to call a method with an explicit object
  - Basically takes the method and calls it with this as parameter, similar to C++

```
var coin1 = new Crypto();
var coin2 = new Crypto();

var function1 = coin1.print;

function1.call(coin1);
function1.call(coin2);
```

OO JS – INHERITANCE

- JS achieves inheritance by
  - Assigning an instance of the parent class to the child class
  - And then specializing it

OO JS – INHERITANCE

```
var Crypto = function(symbol, name) {
        this.symbol = symbol;
        this.name = name;
        this.price = {btc : 0, eth : 0};
};

Crypto.prototype.print = function() {
        console.log("Symbol: " + this.symbol);
        console.log("Name: " + this.name);
};

Crypto.prototype.setPrice = function(priceBtc, priceEth) {
        console.log(this.name + " Price Changed");
        this.price.btc = priceBtc;
        this.price.eth = priceEth;
};
```

OO JS – INHERITANCE

- Call the parent constructor

```
var Bitcoin = function()
{
        console.log('Bitcoin Constructor');
        Crypto.call(this, 'BTC', 'Bitcoin');
};
```

OO JS – INHERITANCE

- Inherit all the methods that were added to the parent's prototype
- Required since inheritance is not natively supported in JS

```
Bitcoin.prototype = Object.create(Crypto.prototype);
```

OO JS – INHERITANCE

- Since the child's prototype was set to the parent's prototype, the child's constructor is now overwritten by the parent's one
- Set the constructor back to the child's constructor
- Not required for the class to function, but without this certain things (like getting the constructor type) will returned wrong results

```
Bitcoin.prototype.constructor = Bitcoin;
```

OO JS – INHERITANCE

- Parent functions can be overwritten in child classes
- Since JS is dynamically-typed, parameters can differ, unlike C++

```
Bitcoin.prototype.setPrice = function(priceEth)
{
        // Bitcoin's price is always 1 BTC
        // Call the parent function
        Crypto.prototype.setPrice.call(this, 1, priceEth);
};
```

OO JS – INHERITANCE

- Create a second subclass

```
var Ethereum = function()
{
        console.log('Ethereum Constructor');
        Crypto.call(this, 'ETH', 'Ethereum');

        this.setPrice = function(priceBtc)
        {
                // Ethereum's price is always 1 ETH
                // Call the parent function
                Crypto.prototype.setPrice.call(this, priceBtc, 1);
        };
};

Ethereum.prototype = Object.create(Crypto.prototype);
Ethereum.prototype.constructor = Ethereum;
```

OO JS – INSTANCES

- Create instances and execute methods

```
var coinBitcoin = new Bitcoin();
coinBitcoin.setPrice(16.6);
coinBitcoin.print();

var coinEthereum = new Ethereum();
coinEthereum.setPrice(0.06);
coinEthereum.print();
```

OO JS – EXAMPLE

- Check out the example code
  - A few additional features and functions
  - Comments explaining everything

OOP

ES6+

- With ES5 OOP is not simple and is unnecessarily complicated
- With ES6+ introduced proper class syntax that we are used to in C++ and Java
- We not have reserved keywords:
  - class
  - constructor
  - static
  - extends
  - new
  - super

ES6+

Normal class declaration

```
class Crypto {
        constructor (symbol, name) {
                this.symbol = symbol;
                this.name = name;
                this.price = {btc : 0, eth : 0};
        }

        print() {
                console.log("Symbol: " + this.symbol);
                console.log("Name: " + this.name);
                console.log("Price: "
                + this.price.btc + " BTC" + ", " + this.price.eth + " ETH");
        }
}
```

ES6+

class declaration as an expression

```
var Crypto = class {
        constructor (symbol, name) {
                this.symbol = symbol;
                this.name = name;
                this.price = {btc : 0, eth : 0};
        }

        print() {
                console.log("Symbol: " + this.symbol);
                console.log("Name: " + this.name);
                console.log("Price: "
                + this.price.btc + " BTC" + ", " + this.price.eth + " ETH");
        }
}
```

ES6+

class declaration as a named expression

```
var Crypto = class Crypto {
        constructor (symbol, name) {
                this.symbol = symbol;
                this.name = name;
                this.price = {btc : 0, eth : 0};
        }

        print() {
                console.log("Symbol: " + this.symbol);
                console.log("Name: " + this.name);
                console.log("Price: "
                + this.price.btc + " BTC" + ", " + this.price.eth + " ETH");
        }
}
```

ES6+

Creating objects and calling methods

```
var coin1 = new Crypto('BTC', 'Bitcoin’);
coin1.print();
```

ES6+

Getters and Setters

```
class Crypto {
        constructor (symbol, name) {
                this.symbol = symbol;
                this.name = name;
                this.price = 0;
        }

        get getPrice() {
                return this.price;
        }

        set setName(n) {
                this.name = n;
        }
}
```

ES6+

Calling Getters and Setters

```
var coin1 = new Crypto('BTC', 'Sh1tcoin');
coin1.getPrice // 0
coin1.setName = 'Bitcoin’;
```

ES6+

Static Methods

```
class Crypto {
        constructor (symbol, name) {
                this.symbol = symbol;
                this.name = name;
                this.price = 0;
        }

        get getPrice() {
                return this.price;
        }

        set setName(n) {
                this.name = n;
        }
        static usdToZar(usd) {
                return usd*18.78
        }

}
```

ES6+

Calling static method

```
Crypto.usdToZar(100); // 1878
```

ES6+

Inheritance

```
class Crypto {
        constructor (symbol, name) {
                this.symbol = symbol;
                this.name = name;
                this.price = 0;
        }
}

class Bitcoin extends Crypto {
        print() {
                console.log("Symbol: " + this.symbol);
                console.log("Name: " + this.name);
                console.log("Price: " + this.price + " BTC");
        }
}

var btc = new Bitcoin('BTC', 'Bitcoin');
btc.print(); // Symbol: BTC Name: Bitcoin Price: 0 BTC
```

ES6+

Inheritance calling parent functions

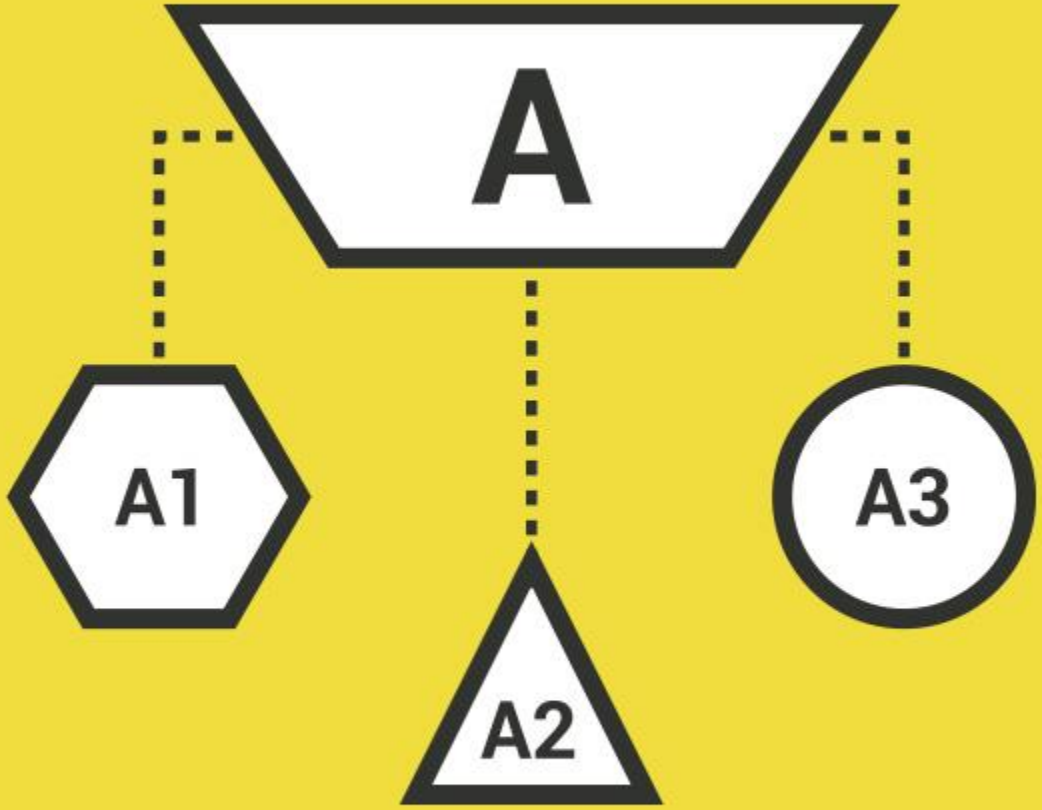
```
class Crypto {
        constructor (symbol, name) {
                this.symbol = symbol;
                this.name = name;
                this.price = 0;
        }
        print() {
                console.log("Symbol: " + this.symbol);
                console.log("Name: " + this.name);
                console.log("Price: " + this.price);
        }

}

class Bitcoin extends Crypto {
        print() {
                super.print();
        }
}

var btc = new Bitcoin('BTC', 'Bitcoin');
btc.print(); // Symbol: BTC Name: Bitcoin Price: 0
```


JS


A
A1
A2
A3