객체

오늘부터 배울 객체는 프로그래밍에서 매우 중요한 개념이며, 여지 것 배운 내용을 모두 포함하는 데이터 타입입니다. 단번에 이해하기 복잡하고 양이 많습니다. 그렇기에 자주 만들어보고 사용해야 합니다.

러닝 자바스크립트의 이선 브라운은 객체를 다음과 같이 정의하였습니다. 객체는 데이터와 기능을 논리적으로 묶어 놓은 것 이라고 말이죠. 객체는 메서드라는 것과, 프로퍼티라는 녀석을 가지고 있습니다. 메서드는 어떤 객체가 수행할 수 있는 기능이며, 프로퍼티는 이 객체가 가지고 있는 정보입니다.

객체를 사람으로 비유해볼까요 ? (사람을 그린다.)

사람은 크게 머리, 가슴, 다리, 팔, 배등의 프로퍼티로 나눌 수 있습니다. 그리고 이 머리, 가슴, 배 등도 하나의 객체로써 머리는 뇌와 눈, 코 등으로 나뉘고, 이 또한 머리의 프로퍼티입니다. 이 객체를 쪼개서 뇌까지 도달한다면 뇌에 담긴 기억들 까지 있겠죠.

그리고 사람의 기능들이 무엇이 있을까요?

[숨을 쉰다, 밥을 먹는다, 공부한다, 기지게를 편다. …] 등이 있겠죠. 이런 기능을 메서드라고 합니다.

이 객체라는 녀석을 만들기 위해서는 클래스라는 것이 필요하고, 이 클래스로 인하여 생성된 것이 바로 객체(인스턴스)입니다.

아까는 사람을 빗대서 객체를 만들었는데, 설명하기 너무 거대하니 작은 붕어빵에 빗대서 말씀드리도록 하겠습니다. 클래스라는 것은 붕어빵틀 입니다. 이 붕어빵 틀은 특정한 모양의 빵을 만들기 위한 것인데, 이 붕어빵 틀에 반죽을 넣고 구우면 생김새가 똑같은 맛있는 붕어빵들이 만들어 집니다. 이때 만들어진 붕어빵이 객체라고 부르고 인스턴스라고 불립니다.

이 때 중요한게 있습니다. 바로 국화빵인데요 ! 우리가 붕어빵 틀을 가지고 있는데, 국화빵이 먹고 싶어서 붕어빵틀에 반죽을 넣는다 한들 국화빵의 모양이 나올까요? 당연히 그렇지 않죠. 붕어빵을 먹고 싶으면 붕어빵 클래스를 이용하고, 국화빵을 만들고 싶으면 국화빵 클래스를 만들면 됩니다. 이 말인 즉 클래스를 이용하면, 어떤 동일한 특징을 가진 객체를 복사 하여 만들 수 있다. 입니다.

이 객체를 만드는 방법은 2가지 방법이 있습니다. 바로 아까처럼 붕어빵틀을 이용해서 붕어빵을 대량 생산 하는 방식과 직접 붕어빵 하나를 손으로 오밀 조밀 빚어 만드는 방식이 있습니다.

처음은 삽질부터 해봐야 하니 객체를 직접 생성하는 방식을 이용해보도록 하겠습니다.

let leejongho = {

eye : "two",

age : 27,

face : "handsom in the world",

hair : "풍성",

tall : 180,

weight : 65,

sayMyName : function(){

console.log("hello my name is leejongho");

},

}

객체를 직접 생성할 때는 변수에 중괄호를 할당해줍니다. 이때의 중괄호를 객체 리터럴이라고 합니다. 이 객체 리터럴 안에서 이 객체의 정보인 프로퍼티와 이 객체만 사용할 수 있는 기능인 메소드를 정의합니다.

프로퍼티는 key와 value로 이루어져 있는데, key는 이 프로퍼티에 접근 하기 위한 변수명이라고 생각하셔도 됩니다. 다만 변수와는 다르게 할당식을 사용하지 않고, 콜론을 이용하여 해당 key에 값을 할당합니다. 그리고 value의 설정이 끝났다면 세미콜론 대신 쉼표를 찍어 프로퍼티의 할당을 종료합니다.

그리고 메서드는 프로퍼티와 마찬가지로 메서드 명을 적어주고, 콜론으로 익명 함수를 작성합니다. 기명함수를 적어도 괜찮지만 지난번 함수표현식을 설명할 때, 설명 드렸듯이 함수가 여러 이름을 가지는 것은 바람직한 일이 아닙니다. 그리고 메서드명으로 해당 함수를 부를 수 도 있기 때문에 기명함수를 작성할 이유가 없습니다.

사용 예

console.log(leejongho.eye); *//프로퍼티에 접근하려면 변수명에 .을 붙여서 해당 프로퍼티 명을 적으면 프로퍼티의 값이 반환된다.*

leejongho.sayMyName(); *//메서드도 마찬가지로 변수명에 .을 붙여 메서드명을 적고 호출하면 함수를 사용할 수 있다.*

(이때 객체의 프로퍼티에 할당식을 사용하여, 값을 할당하는 행위도 가능하다. )

클래스 활용 예

이번에는 붕어빵을 만드는 클래스를 만들어 보도록 하겠다. 이번에는 생성자 함수를 사용해볼건데, 생성자 함수는 쉽게 설명하면, 다음과 같다.

붕어빵틀 안에 팥 대신 피자소를 넣어 피자 빵이 될 수도 있고, 크림을 넣어 크림 붕어빵을 만들 수도 있다. 그런데 붕어빵 안에 피자소와 크림을 넣어 각각 피자붕어빵과 크림 붕어빵이 되었지만 여전히 붕어빵 특성을 담고 있는 붕어빵이다. 이처럼 객체를 생성할 때 필요에 의해서 내부 프로퍼티에 값을 전달인자의 값으로 변경할 수 있게 하는 것을 constructor 생성자 함수라고 하는데, 이는 함수의 파라메터, 어규먼트를 이해한 자스민 사람이라면 누구나 쉽게 사용 할 수 있다.

이번에 클래스로 만들어 볼 것은 tv이다.

class Tv{

constructor(*display*,*speker*,*who*){

*this*.display = display; *// 디스플레이*

*this*.speker = speker; *// 스피커*

*this*.whoMade = who; *// 제조사*

*this*.Chanel = 7; *//디폴트 채널*

}

presentChanel(){return *this*.Chanel}; *// 현재 채널을 알려주는 메서드*

changeChanel(*move*){*this*.Chanel = move}; *//채널을 원하는 채널로 옮겨주는 메서드*

up(){*this*.Chanel++; return *this*.Chanel;} *// 현재 채널을 올리는 메서드*

down(){*this*.Chanel--; return *this*.Chanel;} *//현재 채널을 옮기는 메서드*

}

const samsungTv = new Tv("led","j&b","samsung"); *//생성자 인스턴스 생성*

samsungTv.remote = "lg";

보통 클래스의 첫 글자는 대문자로 작성하는 것이 관례이다. 클래스 내부에는 생성자 함수가 있다. 근데 특이하게도 this라는 것이 나왔다. 이것은 굉장히 중요한 단어인데, 이 this는 객체 인스턴스 스스로를 가리킨다. 그래서 삼성티비.display라는 것과 this.display는 같다. 그래서 생성자 함수 내부에서 초기 프로퍼티 설정을 다해주고 이 클래스의 메서드로 넘어가자

객체를 직접 생성하여 메서드를 만들 때는 키 콜론 익명함수의 형태를 취했다. 그러나 클래스 문법에서는 함수선언 방식에서 function키워드만 제거하여 메서드를 정의한다.

4개의 메서드를 만들어 보자

이제 이 클래스로 tv를 만들어 보자, 이 클래스를 활용하기 위해서는 생성자 함수를 호출해야 하는데, tv클래스를 호출하고 클래스 명앞에 new라는 키워드를 붙여주면 객체가 생성이 된다.

[ 사용 방법을 보여주자 ]

이후 본인의 스마트폰 클래스를 만들어 연습 시키자, 객체를 직접생성해보고, 클래스를 이용해보라 하자

이 클래스 문법은 사실 2015년도 이후에나 나온 최신 문법입니다. 그래서 그 전에는 함수를 사용해서 객체를 반환했습니다.

그래서 이번에는 함수를 이용해서 객체를 만들 수 있는 클래스를 만들어보도록 하겠습니다.

function Tv(*display*,*speker*,*who*){

return {

display : display,

speker : speker,

whoMade : who,

chanel : 7,

presentChanel : function(){return *this*.Chanel}, *// 현재 채널을 알려주는 메서드*

changeChanel : function(*move*){*this*.Chanel = move}, *//채널을 원하는 채널로 옮겨주는 메서드*

up : function(){*this*.Chanel++; return *this*.Chanel}, *// 현재 채널을 올리는 메서드*

down: function(){*this*.Chanel--; return *this*.Chanel;} *//현재 채널을 옮기는 메서드*

}

}

이 함수 선언으로 클래스 문법과 동일하게 만들 수 있습니다. 다만 클래스 문법보다는 조금 어설퍼 보이는 건 어쩔 수 없는 듯 해 보입니다.

다음은 static 키워드인데, 이 static 키워드는 인스턴스의 메서드가 아닌 클래스 자체의 메서드이다. 그래서 클래스.메서드명()으로 호출하며, 어플리케이션의 유틸리티 함수로 사용된다.

Math.random()과 같은 것들이 정적인 메서드들이다.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<style>

.tvContainer{

width: 200px;

height: 200px;

border: 1px solid black;

display: flex;

}

.chanel{

flex-basis: 150px;

}

.buttonContainer{

flex-grow: 1;

border-left : 1px solid black;

display : flex;

flex-direction: column;

}

.buttonDiv{

flex-grow: 1;

}

.buttonDiv:nth-child(2){

border-top: 1px solid black;

}

</style>

<body>

*<!-- <button></button> -->*

<div id="container">

</div>

</body>

<script>

let appleTv;

(function(*display*,*speaker*,*who*){

class Tv{

constructor(*display*,*speker*,*who*){

*this*.display = display; *// 디스플레이*

*this*.speker = speker; *// 스피커*

*this*.whoMade = who; *// 제조사*

*this*.Chanel = 7; *//디폴트 채널*

}

presentChanel(){return *this*.Chanel}; *// 현재 채널을 알려주는 메서드*

changeChanel(*move*){*this*.Chanel = move}; *//채널을 원하는 채널로 옮겨주는 메서드*

up(){*this*.Chanel++; return *this*.Chanel;} *// 현재 채널을 올리는 메서드*

down(){*this*.Chanel--; return *this*.Chanel;} *//현재 채널을 옮기는 메서드*

}

appleTv = new Tv(display,speaker,who);

document.getElementById("container").innerHTML = `<div class="tvContainer">

<div id="chanel" class="chanel">

<h3>${appleTv.whoMade} TV </h3>

채널 ${appleTv.Chanel}

</div>

<div class="buttonContainer">

<div class="buttonDiv">

<button onClick=changeChanel('up')>up</button>

</div>

<div class="buttonDiv">

<button onClick=changeChanel('down')>down</button>

</div>

</div>

</div>`;

})("애플","하만카돈","애플");

function changeChanel(*order*){

if(order == 'up'){

document.getElementById("chanel").innerHTML = `

<h3>${appleTv.whoMade} TV </h3>

채널 ${appleTv.up()}

`;

}else{

document.getElementById("chanel").innerHTML = `

<h3>${appleTv.whoMade} TV </h3>

채널 ${appleTv.down()}

`;

}

}

</script>

</html>