**캡슐화**

**사전 질문**

* 객체란 무엇인가
* 객체란 왜 사용하는 것 인가 ?
* 객체의 프로퍼티는 무엇과 무엇의 쌍으로 이루어져 있나
* 객체의 메서드란 무엇인가 ? 또한 이 메서드가 일반적인 함수와 다른 점은 무엇인가
* 클래스를 아는 만큼 설명해보라
* 변수에 객체 리터럴을 할당해서 본인을 표현해보라 . 프로퍼티 3개, 메서드 1개
* 만든 변수의 프로퍼티 중 한가지를 출력하고, 메서드를 호출해서 활용하라
* 함수를 이용해서 스마트폰 클래스를 만들어 보라. 이 때 모델명, 통신사, 전화번호를 매개변수로 받아 각 프로퍼티에 할당하고, 전달인자로 전화번호를 받는 메서드를 만들어 `${나의 번호}가 ${전화 번호}에 전화를 거는 중입니다.` 라는 메시지를 출력하도록 해보라
* 클래스문법으로 위의 함수로 만든 클래스의 내용을 똑같이 만들고, showInfo라는 메서드를 하나 더 만들어 해당 인스턴스의 전달인자, 모델명, 통신사를 출력하는 메서드를 만들어보라
* 전화번호 프로퍼티를 변경하라 .

**문제제기**

현재 처럼 객체를 만들면 문제가 하나 있다. 바로 외부에서 속을 들여다 볼 수 있다는 점이다.

스마트폰 클래스를 하나 만들어서 내용을 확인해보자 .

class SmartPhone{

constructor(*name*,*modelId*,*whoMade*,*display*,*adminKey*){

*this*.name = name;

*this*.modelId = modelId;

*this*.whoMade = whoMade;

*this*.display = display;

*this*.adminKey = adminKey

}

call(*phoneNumber*){

console.log(`ring ring calling to ${phoneNumber}`);

}

adminMode(*adminKey*){

if(adminKey == *this*.adminKey){

console.log('activate admin mode');

}else{

console.log('admin key does not matched');

}

}

}

const iphonePro = new SmartPhone('Iphone Pro',123123,'apple','super retina xdr',"qwe123");

console.log(iphonePro);

이렇게 하고 아이폰프로를 콘솔 로그로 찍어 보니 보여선 안되는 adminKey에 대한 정보가 보인다. 이게 의미 하는 것은 보여선 안되는 것들이 노출되고, 쉽게 해당 프로퍼티에 접근하여 정보를 변경할 수 있다는 의미이다. 쉽게 말해서 보안이 전혀 안된 객체라는 의미다.

이런 성격의 프로퍼티와 메서드를 퍼블릭(공개된) 성격의 프로퍼티, 메서드라고 한다. C#이나, 자바 같은 oop언어들은 외부에서 해당 프로퍼티에 접근을 할 수 있게 퍼블릭 설정과 객체 외부에서 접근 하지 못하고, 객체 내부에서만 접근 할 수 있도록 설정하는 프라이베이트 설정을 지원한다. 그러나 자바스크립트는 객체지향언어가 아니기 때문에 private설정을 할 수 없고, 객체의 모든 정보가 들어난다.

그렇지만 자바스크립트의 객체가 이런 성격이라고, 그냥 두면 큰일난다. 자바스크립트는 웹의 프론트에서 주로 쓰이는 언어인 만큼 객체들이 민감한 성격의 정보를 가지고 있다. 그렇기에, 보안을 철저히 할 필요가 있다.

이 문제를 해결 할 수 있는 방법은 캡슐화이다. 캡슐화는 객체 내부에 있는 프로퍼티를 숨기고, 접근을 제한 하는 방법이다. 어려워 보이지만 지역 스코프와 클로저를 이해한 자스민이라면 어렵지 않을 것이다.

**함수선언 캡슐화**

수업은 먼저 객체를 함수선언으로 만들어서 캡슐화를 하는 방법부터 알려드리겠습니다.

User객체를 하나 만들 도록 하겠습니다. 3개의 파라미터를 받고 비밀번호 변경과 전화번호 변경 메서드를 만들어 보도록 하겠습니다.

function User(*id*,*password*,*phoneNumber*){

*//id는 노출 되어도 괜찮은 레벨이지만 password와 phonenumber는 아니다.*

let password = password;

let phoneNumber = phoneNumber;

*//객체의 인스턴스를 만들면 return 내부에 있는 객체를 받는다.*

*//그러나 password, phonenumber는 객체보다 상위스코프에 있다.*

return {

id : id,

setPassword : function(*password\_c*,*setPassword*){

if(password\_c === password){

password = setPassword;

}else{

console.log('password does not matched');

}

},

getPhoneNumber : function(*password\_c*){

if(password\_c === password){

return phoneNumber;

}else{

return "does not matched";

}

}

}

}

phoneNumber,password는 함수스코프를 가진다. 선언 부에서 클로저를 활용하여, setPassword와 getPhoneNumber가 두 변수에 접근 할 수 있게 되었다.

이렇게 하면 두개의 메서드를 제외하고는 두 변수를 제어할 방법이 없어진다.

이 처럼 객체 내부에 프라이베이트 한 변수를 get,set메서드를 이용해서 접근 하는 방식을 모듈 패턴이라고 한다.

만약 이때 myArticle이라는 내 게시글 정보를 가진 배열을 두었는데, 개발자가 실수로 배열의 내용을 반환하는 getMyArticle에서 myArticle 배열 전체를 반환하게 하였다면 어떤 결과가 이루어 질까 ?

문제는 굉장히 심각해진다.

function User(*id*,*myPassword*,*myPhoneNumber*){

*//id는 노출 되어도 괜찮은 레벨이지만 password와 phonenumber는 아니다.*

let password = myPassword;

let phoneNumber = myPhoneNumber;

let myArticle = ['a'];

*//객체의 인스턴스를 만들면 return 내부에 있는 객체를 받는다.*

*//그러나 password, phonenumber는 객체보다 상위스코프에 있다.*

return {

id : id,

setPassword : function(*password\_c*,*setPassword*){

if(password\_c === password){

password = setPassword;

}else{

console.log('password does not matched');

}

},

getPhoneNumber : function(*password\_c*){

if(password\_c === password){

return phoneNumber;

}else{

return "does not matched";

}

},

write : function(*article*){

myArticle.push(article);

},

getMyWrite : function(){

return myArticle;

}

}

}

const lee = User('admin','gozld','01099099278');

let hackingRoot = lee.getMyWrite();

console.log(hackingRoot);

hackingRoot[2] = "hacked";

console.log(hackingRoot);

console.log(lee.getMyWrite());

콜바이 레퍼런스와 비슷한 현상이 일어나는데, 이를 깊은 복사라고 한다. 메모리 셀안에 있는 주소를 반환하다보니, 할당을 한다면 접근 할 수 있는 루트가 생긴다.

getMyWrite함수에 파라메터로 숫자타입을 받아 myArticle의 지정된 번호를 출력하는 것을 만들어 보자,

getMyWrite : function(*number*){

if(number < 0 || number > myArticle.length){

return "실패"

}else{

return myArticle[number];

}

}

이렇게 한다면 해당 인덱스의 내용을 반환하는 메서드가 될 것이다. 이때 메모리 주소를 반환하는 것이 아니기 때문에, 이 메서드는 온전히 문자열만 반환하게 된다. 이때 메모리 주소를 반환하는 것이 아닌 메모리 내부에 값을 전달하는 행위를 얕은 복사라고 한다.

이번에는 클래스를 이용해서 캡슐화를 진행해보자.

class User{

constructor(*id*,*myPassword*,*myPhoneNumber*){

*// let id = id;*

let password = myPassword;

let phoneNumber = myPhoneNumber;

*this*.getMyPhoneNumber = (*password\_c*)=>{

if(password === password\_c){

return phoneNumber;

}else{

return "it does not match";

}

}

}

}

const lee = new User("admin","gozld","01099099278");

console.log(lee);

console.log(lee.getMyPhoneNumber("gozld"));

이 생성자 함수에서 가장 중요한 점은 this를 붙여 프로퍼티를 사용하지 않았다는 점이다. 중요한 점은 변수를 사용했다는 점이고, 이 변수에 접근 하기 위해서는 생성자 함수 내부에서 메서드를 정의해야만 한다. 생성자 외부에서 메서드를 선언하게 되면 생성자 함수 내부에 있는 변수와 스코프가 달라 접근 할 수 없게 된다.

**심볼을 이용한 클로저**

다음은 심볼이라는 최근에 나온 데이터 타입을 이용한 클로저 입니다.

let SimpleDate = (function() {

let \_yearKey = Symbol();

let \_monthKey = Symbol();

let \_dayKey = Symbol();

class SimpleDate {

constructor(year, month, day) {

// Check that (year, month, day) is a valid date

// ...

// If it is, use it to initialize "this" date

this[\_yearKey] = year;

this[\_monthKey] = month;

this[\_dayKey] = day;

}

addDays(nDays) {

// Increase "this" date by n days

// ...

}

getDay() {

return this[\_dayKey];

}

}

return SimpleDate;

}());

이 심볼이라는 것의 특징은 우리가 볼 수 없는 유니크한 값을 반환합니다. 심볼을 콘솔로그로 찍어보면 그냥 의미 없는 symbol() 이라는 것들만 나옵니다. 이 심볼의 특징은 symbol 값을 구하면 symbol로 생성한 값을 반환하지 않고, symbol값을 생성했던 형태를 반환합니다. 브라우저의 개발자 도구에서도 값을 볼 수 없습니다. 이것이 심볼의 특징입니다.

현재 보이는 this[\_yearKey] 같은 형태는 해쉬테이블과 같은 형태로 문자열 혹은 다른 값을 특정한 해쉬함수라는 것을 이용하여, 특정 값을 특정 숫자로 변경해줍니다. 이렇기 때문에 더욱 안전하게 데이터를 보관 할 수 있습니다.