**상속**

**질문**

* 객체리터럴을 변수에 직접 할당하여 컴퓨터를 표현 해보세요 . 3가지 프로퍼티와 2가지 메서드를 만들어보세요.
* 함수를 이용해서 User객체를 만들어 보세요. User객체는 3가지 전달인자를 받는데, 아이디, 비밀번호, 주소를 받아 프로퍼티에 할당 하세요. 또한 해당 객체의 비밀번호가 맞다면 주소를 출력하는 메서드를 만들어보세요. (캡슐화)x
* 위의 User함수를 클래스로 표현 해보세요.
* 캡슐화란 무엇인가 ?
* 함수로 만든 User클래스에서 모든 프로퍼티를 캡슐화하여, 비밀번호로 인자로 받아 만약 해당객체의 비밀번호와 인자로 받은 비밀번호가 일치한다면 주소를 return 하는 getMyaddress메서드를 만들어보세요.
* 인자로 기존 비밀번호와 새로운 비밀번호를 받는 메서드를 만들라 . 이 때 전달 받은 비밀번호가 캡슐화된 변수와 일치한다면 비밀번호변수를 전달 받은 인자로 변경 하고, 아니라면 비밀번호가 일치하지 않습니다. 라는 setMyPassword메서드를 만드시오
* 위의 캡슐화된 User 함수를 클래스로 표현 하시오
* 모듈패턴이란 무엇인가 ?

이전에도 말씀 드렸듯이 es6이전 자바스크립트 문법에는 클래스라는 문법이 없었습니다. 그말 인 즉슨 자바스크립트가 클래스기반의 객체지향언어가 아니라는 말입니다.

객체지향언어의 가장 큰 특징은 상속과 다형성입니다. 상속은 영어로 inheritance 라고 하는데, es5이전에는 클래스 문법이 없었기 때문에 프로토타입이라는 굉장한 녀석으로 상속을 진행 했습니다.

아 참 상속이 뭔지 설명을 안 드렸군요. 상속은 부모의 특징을 물려 받는 것을 상속이라고 합니다. 예를 들어 우리는 부모님의 특성을 일부 상속 받았습니다. 그중 저는 아버지의 성격을 닮았고, 어깨위에 있는 두 점을 물려 받았죠. 또 다른 거대한 예를 들어보죠. 톰과제리에 나오는 이 개를 기억 하시나요 ? 이 개는 아버지의 특성을 물려받았겠지만, 이 개는 불독의 특성을 물려받았죠. 그리고 이 불독은 포유류의 특성을 상속 받고, 더 나아가 포유류는 생명체의 특성을 물려받았을 겁니다. 이처럼 상속은 객체를 하나씩 정의 할 때 중복되는 코드라인을 없애주고 객체지향프로그래밍을 할 수 있도록 도와줍니다.

**클래스형 상속**

그럼 한번 자바스크립트의 상속을 한번 알아볼까요 ?

class Dalmasian{

constructor(*name*){

*// super();*

*this*.name = name;

*this*.beak = "wong wong";

*this*.birthYear = new Date().getFullYear;

*this*.skin = "polka dots"

}

bow(){

console.log(*this*.beak);

}

sayMyName(){

console.log(`hello my name is ${*this*.name}`)

}

}

먼저 달마시안이라는 클래스를 만들고 이 클래스에서는 달마시안의 특성으로 사용될 프로퍼티와 메서드를 정의하자

그리고 Dog이라는 클래스를 정의해보자

class Dog{

constructor(){

*this*.legs = 4;

*this*.eye = 2;

*this*.tail = true;

*this*.foot = 4;

*this*.asshole = 1;

}

poof(){

console.log("poofing Now");

}

eat(){

console.log("eat now");

}

}

이후 달마시안의 인스턴스를 만들고, 이 인스턴스에서 bow메서드를 호출해보지만 당연하게도 안됩니다.

그런데 여기서 달마시안 클래스에서 extends dog을 붙여줍니다. 이 단어는 상속할 클래스를 지정하게 해줍니다. 이렇게 하면 dog클래스를 상속 받을 준비가 거의 끝났습니다. 스크립트를 실행 해보면 super 생성자를 상속받은 클래스에서 this를 사용하기전에 꼭 호출 하라고 써있습니다. 인간이 무슨 권한이 있습니까 기계가 시키는 대로 해야지 . super함수를 달마시안 생성자 함수에서 호출합니다.

const dog = new Dog()

const anastasha = new Dalmasian("anastasha");

anastasha.poof()

다음 아나스타샤에서 없던 기능인 poof와 eat을 해보니 dog에 있던 기능들이 작동합니다. 그렇다면 이 super라는 녀석은 무엇일 까요? 바로 상속받는 클래스의 생성자 함수 입니다.

우리가 기존의 개 클래스의 생성자에서 tail은 true를 줬는데, 이를 생성자 함수로 어규먼트를 받아 true와 false를 지정해보도록 해보고 super에서 false를 강제적으로 줘보겠습니다.

달마시안의 생성자

constructor(*name*){

*super*(false);

*this*.name = name;

*this*.beak = "wong wong";

*this*.birthYear = new Date().getFullYear;

*this*.skin = "polka dots"

}

개의 생성자

constructor(*tail*){

*this*.legs = 4;

*this*.eye = 2;

*this*.tail = tail;

*this*.foot = 4;

*this*.asshole = 1;

}

아나스타샤의 꼬리의 상태가 변한것을 알 수 있습니다.

**다중 상속**

지금까지는 extends 키워드를 통하여 상위 클래스를 한 개씩만 상속 받았다. 예를 들어 달마시안클래스는 개라는 클래스를 상속받았고 작성하지는 않았지만 이 개라는 클래스도 포유류라는 클래스를 상속 받았을 것이다. 그러나 실제 현실 세계의 객체는 이렇게 단순하게 표현 되지 않는다. 예를 들어 우리는 인간이라는 클래스를 상속 받았겠다고 한개의 선으로 표현 할 수 있지만, 실제로는 인간이라는 클래스를 받은 여자 , 남자 그리고 서로 다른 부모님의 특징을 상속 받은 것이다. 이처럼 클래스가 동시에 상속 받은 형태를 다중 상속이라고 한다. 우리는 부모님의 특징 중에 성격, 혹은 신체적 특징 들을 물려 받았을 터이다. 좀 더 다른 예를 들어 라이거라는 종을 아는가 ? 라이거는 호랑이와 사자의 교배종이다. 이처럼 다중 상속은 특정 클래스에서 병렬적으로 두개 이상의 클래스를 상속 받는 것을 말한다.

그러나 이 다중 상속은 프로그래밍 언어로 표현 하기에 다소 난감한 부분이 있다. 바로 같은 메서드나 프로퍼티의 중복이다. 예를 들어 라이거는 호랑이와 사자를 동시에 상속 받는데, 호랑이와 사자의 공통적인 이름을 가진 shout나 skin이라는 것들 같은 메서드와 프로퍼티 중 라이거 클래스는 어떤 메서드를 상속 받아야 할지 모호해진다. 이런 점 때문에 여러 언어들이 다중 상속을 지원하지 않는 이유이다. 그러나 자바스크립트는 다중 상속을 믹스인이라는 기법을 통해 해결 하였다.

**믹스인**

믹스인 기법은 고대 신화에서 나오는 여러가지 동물이 섞인 괴수인 키메라를 만드는 것처럼 동시 상속 받는 클래스를 필요한 기능들만 가진 클래스를 만들어 다중 상속처럼 보이도록 하는 기법이다.

*//이 리턴된 클래스의 기능은 fly 메서드 하나다.*

const flyMixIn = (superClass) => {

return class extends superClass{

fly(){

console.log("it's fly");

}

}

}

*//이 리턴된 클래스의 기능은 드라이브 메서드 하나다.*

const driveMixIn = (superClass) =>{

return class extends superClass{

drive(){

console.log("it's driving");

}

}

}

*//조상님 클래스*

class Machine{

makeEnergyFromFuel(fuel){

console.log(`${fuel} 로써 엔진을 구동 중입니다.`);

}

}

*//플라잉카는 나는 것에 필요한 기능을 가진 클래스, 운전하는데 필요한 기능을 가진 클래스*

*//그 위 기계 클래스를 한번에 상속 받은 셈이 되었다.*

class FlingCar extends flyMixIn(driveMixIn(Machine)){

constructor(fuel){

super()

this.fuel = fuel;

}

turnOnSwitch(){

console.log("시동을 킵니다.");

this.makeEnergyFromFuel(this.fuel);

}

}

const zero = new FlingCar("gas");

zero.turnOnSwitch();

zero.fly();

zero.drive();