코어 자바스크립트

7강 클래스

참고문헌

- 코어자바스크립트
- 모던자바스크립트 DeepDive

클래스(class) 클래스와 인스턴스의 개념 이해

class [klæs]

(3인칭 단수 현재 : classes)

명사

1. (공통적 성질을 가진) 종류(kind), 부류
2. [CU] (학교의) 클래스, 학급, 반(cf. FORM); (클래스의) 학습 시간, 수업; (편물·요리 등의) 강습

in[after] **class**
수업 중[방과 후]에

instance ['ɪnstəns]

(3인칭 단수 현재 : instances)

명사

1. 보기(example), 사례, 경우, 실례, 실증

in most instances
대개의 경우에는

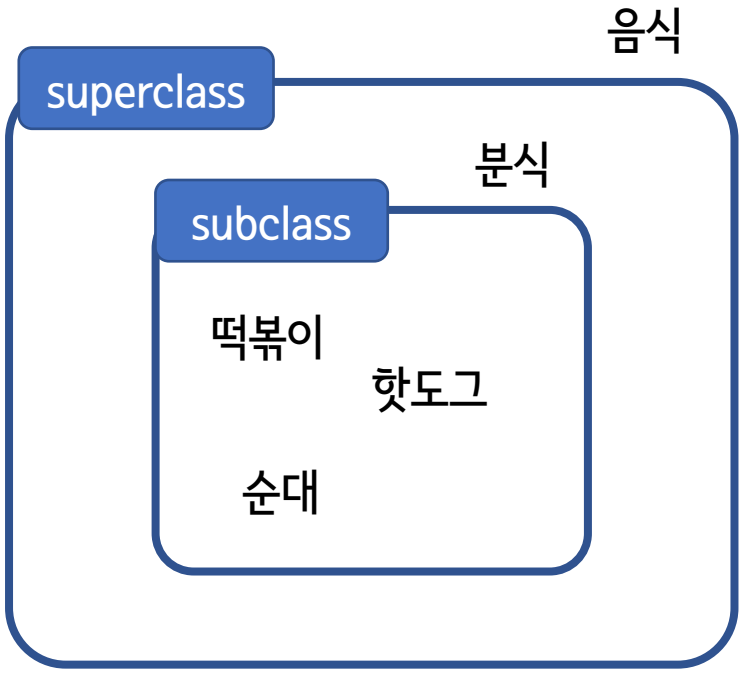
2. 단계, 경우(case)

class는 단어의 뜻과 동일하게 '계급, 집단, 집합'의 개념으로 접근!

상위클래스(superclass)와 하위클래스(subclass)의 개념이 있음

instance는 어떤 클래스의 속성을 지니는 실존하는 개체, 즉 어떤 조건에 부합하는 구체적인 예시

클래스(class) 클래스와 인스턴스의 개념 이해



하나의 개체가 같은 레벨이 있는 서로 다른 여러 클래스의 인스턴스일 수 있음.

ex) 사람 -> (국적, 성별, 직업...)

현실

인스턴스들로부터 공통점을 발견해 클래스 정의

클래스는 추상적인 개념

프로그래밍 언어

클래스가 먼저 정의되어야 인스턴스 생성 가능

클래스는 추상적인 대상일 수도 있고 구체적인 개체가 될 수도

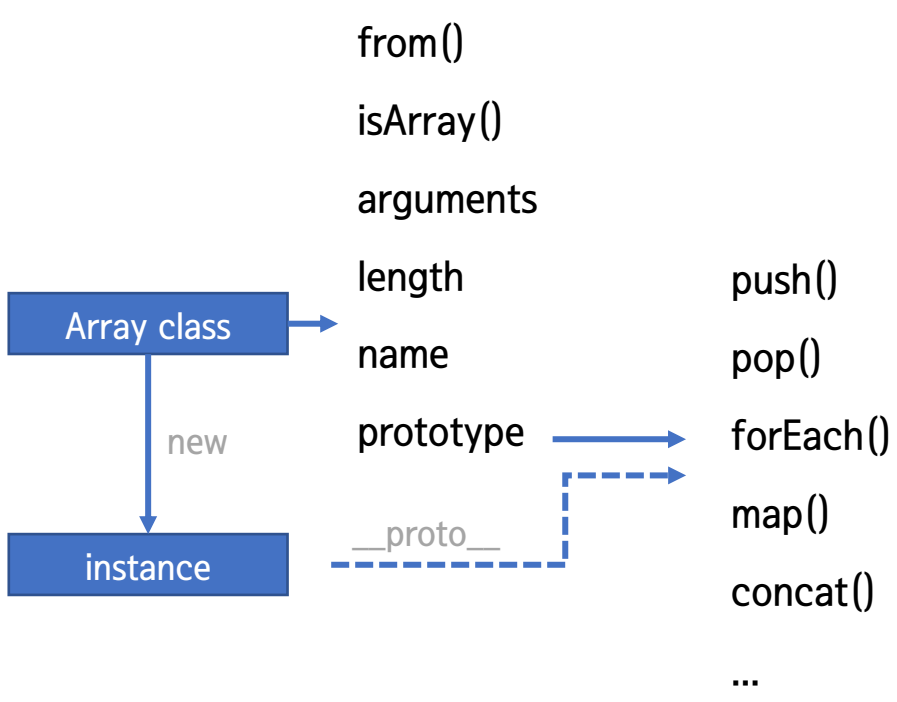
*추상: 여러 가지 사물이나 개념에서 공통되는 특성이나 속상 따위를 추출하여 파악하는 작용

클래스(class)

생성자 함수에 new연산자와 함께 호출하면 인스턴스가 생성됨. 이때 생성자 함수를 일종의 클래스라 하면 prototype 객체 내부 요소들이 인스턴스에 상속된다고 볼 수 있음!

일반적으로 인스턴스에 상속되는지 여부에 따라 스태틱 멤버, 인스턴스 멤버로 표현.

자바스크립트에서는 인스턴스에서 메서드 정의 가능하기 때문에, 인스턴스 멤버 대신 프로토타입 메서드라고 사용함. (* MDN에서는 인스턴스 메서드)




```
var Rectangle = function (width, height) {    생성자 함수
  this.width = width;
  this.height = height;
};
Rectangle.prototype.getArea = function () {   프로토타입 메서드
  return this.width * this.height;
};
Rectangle.isRectangle = function (instance) { 스태틱 메서드
  return instance instanceof Rectangle && instance.width > 0
&& instance.height > 0;
};

var rect1 = new Rectangle(3, 4);
console.log(rect1.getArea());// 12
console.log(rect1.isRectangle(rect1));// Error
console.log(Rectangle.isRectangle(rect1));// true클래스가 하나의 개체
```

클래스 상속 프로토타입 체이닝을 이용한 기본 구현(ES5)

* 유사배열객체에서 prototype을 이용해 Array의 인스턴스 메서드 사용

```
var Grade = function () {
  var args = Array.prototype.slice.call(arguments);
  for (var i =0; i < args.length; i++) {
    this[i] = args[i];
  }
  this.length = args.length;
};
Grade.prototype = [];
var g =new Grade(100, 80);
g.push(90);
console.log(g); // Grade { 0: 100, 1: 80, 2: 90,
length: 3}

delete g.length;
g.push(70);* length프로퍼티가 configurable(삭제가능)
console.log(g); // Grade { 0: 70, 1: 80, 2: 90,
length : 1}
```

* length프로퍼티를 삭제했어도 프로토타입 체이닝에 의해 빈배열의 length:0에 접근하여 push를 수행

클래스 상속 프로토타입 체이닝을 이용한 기본 구현(ES5)

* 유사배열객체에서 prototype에 요소가 있는 배열을 매칭

```
var Grade = function () {
  var args = Array.prototype.slice.call(arguments);
  for (var i =0; i < args.length; i++) {
    this[i] = args[i];
  }
  this.length = args.length;
};
Grade.prototype = ['a','b','c','d'];
var g =new Grade(100, 80);
g.push(90);
console.log(g); // Grade { 0: 100, 1: 80, 2: 90,
length: 3}

delete g.length;
g.push(70);* length프로퍼티가 configurable(삭제가능)
console.log(g); // Grade { 0: 100, 1: 80, 2: 90, 4:
70, length : 5}
```

* length프로퍼티를 삭제했어도 프로토타입 체이닝에 의해 요소가 있는 배열의 length:4에 접근하여 push를 수행

클래스에 있는 값이 인스턴스 동작에 영향을 주는 것은 클래스의 추상성을 해침!

클래스는 인스턴스가 사용할 메서드만을 지닌 추상적인 틀로만 작용하게 해야함

클래스 상속 프로토타입 체이닝을 이용한 기본 구현(ES5)

* Rectangle, Square 클래스

```
var Rectangle = function (width, height) {
  this.width = width;
  this.height = height;
};
Rectangle.prototype.getArea = function () {
  return this.width * this.height;
};
var rect = new Rectangle(3,4);
console.log(rect.getArea()); // 12;

var Square = function(width) {
  this.width = width;
};
Square.prototype.getArea = function () {
  return this.width * this.width;
};
var sq = new Square(5);
console.log(sq.getArea()); // 25
```

* Square 클래스 변형

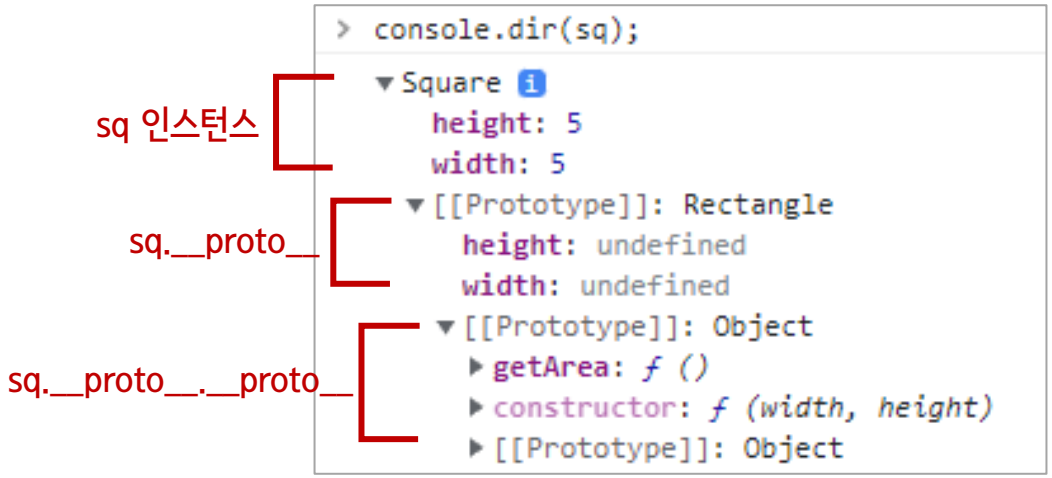
```
var Rectangle = function (width, height) {
  this.width = width;
  this.height = height;
};
Rectangle.prototype.getArea = function () {
  return this.width * this.height;
};
var rect = new Rectangle(3,4);
console.log(rect.getArea()); // 12;
var Square = function(width) {
  this.width = width;
  this.height = width;
};
Square.prototype.getArea = function () {
  return this.width * this.height;
};
var sq = new Square(5);
console.log(sq.getArea()); // 25
```

클래스 상속 프로토타입 체이닝을 이용한 기본 구현(ES5)

* Rectangle을 상속하는 Square 클래스

```
var Rectangle = function (width, height) {
  this.width = width;
  this.height = height;
};
Rectangle.prototype.getArea = function () {
  return this.width * this.height;
};
var rect = new Rectangle(3,4);
console.log(rect.getArea()); // 12;

var Square = function(width) {
  Rectangle.call(this, width, width);
};
Square.prototype = new Rectangle();
var sq = new Square(5);
console.log(sq.getArea()); // 25
```



sq의 width나 height 프로퍼티를 임의로 지울 수 있고, 이를 지우게 되면 sq.__proto__의 width나 height 프로퍼티에 접근하게 되므로 원치 않는 결과를 얻게 됨!

또한 constructor도 Rectangle의 constructor여서 sq.constructor(2,3)로 width가 2고 height가 3인 Rectangle을 만들 수 있어서 문제가 될 수 있다

클래스 상속 클래스가 구체적인 데이터를 지니지 않게 하는 방법(1)

프로퍼티를 지우고 새로운 프로퍼티를 추가할 수 없게 하기

```
var Rectangle = function (width, height) {
  this.width = width;
  this.height = height;
};
Rectangle.prototype.getArea = function () {
  return this.width * this.height;
};
var rect = new Rectangle(3,4);
console.log(rect.getArea()); // 12;
var Square = function(width) {
  Rectangle.call(this, width, width);
};
Square.prototype = new Rectangle();

delete Square.prototype.width;
delete Square.prototype.height;
Object.freeze(Square.prototype);

var sq = new Square(5);
console.log(sq.getArea()); // 25
```

```
> console.dir(sq)
  ▼ Square i
      height: 5
      width: 5
    ▼ [[Prototype]]: Rectangle
      ▼ [[Prototype]]: Object
        ▶ getArea: ƒ ()
        ▶ constructor: ƒ (width, height)
        ▶ [[Prototype]]: Object
```

```
> delete sq.width
< true
> console.dir(sq)
  ▼ Square i
      height: 5
    ▶ [[Prototype]]: Rectangle
```

```
> sq.prototype.width = 5;
✖ ▶ Uncaught TypeError: Cannot set properties of
  undefined (setting 'width')
      at <anonymous>:1:20
```

클래스 상속 클래스가 구체적인 데이터를 지니지 않게 하는 방법(1)+ constructor복구

범용적인 함수

```
var extendClass1 = function (SuperClass, SubClass, subMethods) {
  SubClass.prototype = new SuperClass();
  for (var prop in SubClass.prototype) {
    if(SubClass.prototype.hasOwnProperty(prop)){
      delete SubClass.prototype[prop];
    }
  }
  SubClass.prototype.constructor = SubClass;
  if(subMethods) {
    for (var method in subMethods) {
      SubClass.prototype[method] = subMethods[method];
    }
  }
  Object.freeze(SubClass.prototype);
  return SubClass;
};
```

superClass, subClass, subClass에 추가할 메서드들이 정의된 객체를 인자로 받아서

subClass의 prototype 내용을 정리하고 freeze하고,

SubClass.prototype.constructor가 원래의 SubClass를 바라보도록 함.

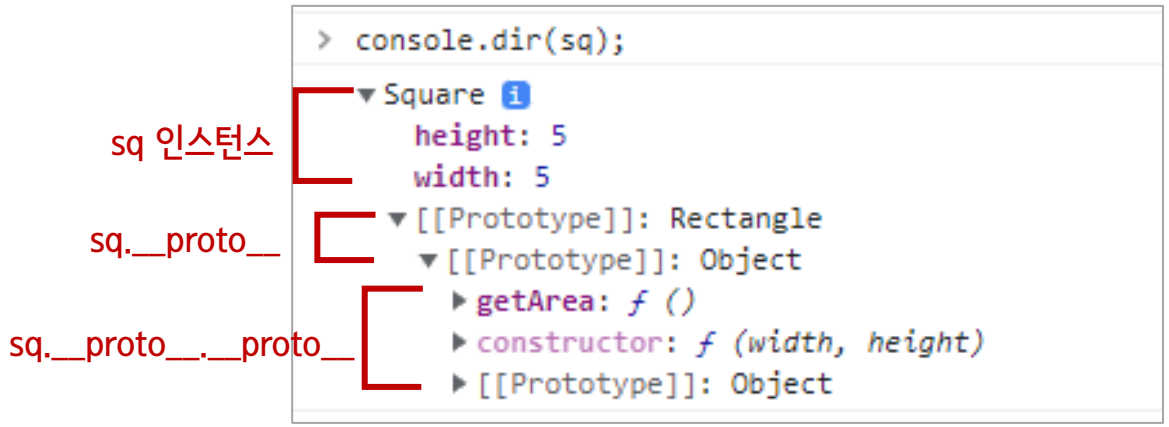
```
var Square = extendClass1(Rectangle, function (width) {
  Rectangle.call(this, width, width);
});
```

클래스 상속 클래스가 구체적인 데이터를 지니지 않게 하는 방법(2)

SubClass의 prototype에 어떤 프로퍼티도 생성하지 않는 빈 생성자함수(Bridge)를 만듦 -> Bridge의 prototype이 SuperClass의 prototype을 바라봄 -> SubClass의 prototype에 Bridge의 인스턴스 할당 *더글라스 크락포드 제안

```
var Rectangle = function (width, height) {
    this.width = width;
    this.height = height;
};
Rectangle.prototype.getArea = function () {
  return this.width * this.height;
};
var rect = new Rectangle(3,4);
console.log(rect.getArea()); // 12;
var Square = function(width) {
  Rectangle.call(this, width, width);
};
var Bridge = function () {};
Bridge.prototype = Rectangle.prototype;
Square.prototype = new Bridge();
Object.freeze(Square.prototype);

var sq = new Square(5);
```



sq는 Square class이기에 sq.__proto__는 Bridge, Bridge의 prototype객체는 Rectangle의 prototype객체임!

클래스 상속 클래스가 구체적인 데이터를 지니지 않게 하는 방법(2) + constructor복구

범용적인 함수

```
var extendClass2 = (function () {
  var Bridge = function() {};
  return function (SuperClass, SubClass, subMethods) {
    Bridge.prototype = SuperClass.prototype;
    SubClass.prototype = new Bridge();
    SubClass.prototype.constructor = SubClass;
    if(subMethods) {
      for(var method in subMethods) {
        SubClass.prototype[method] = subMethods[method];
      }
    }
    Object.freeze(SubClass.prototype);
    return SubClass;
  };
})();
```

superClass, subClass, subClass에 추가할 메서드들이 정의된 객체를 인자로 받아서

subClass의 prototype 내용을 정리하고 freeze하고,

SubClass.prototype.constructor가 원래의 SubClass를 바라보도록 함.

```
var Square = extendClass2(Rectangle, function (width) {
  Rectangle.call(this, width, width);
});
```

클래스 상속 클래스가 구체적인 데이터를 지니지 않게 하는 방법(3)

Object.create 사용

```
Object.create(proto[, propertiesObject])
```

proto: 새로 만든 객체의 프로토 타입이어야할 객체

propertiesObejct: 새로운 속성 정의 (configurable, enumerable, value, writable, get, set)

반환값: 생성된 새로운 객체

```
var Rectangle = function (width, height) {
  this.width = width;
  this.height = height;
};
Rectangle.prototype.getArea = function () {
  return this.width * this.height;
};
var rect = new Rectangle(3,4);
console.log(rect.getArea()); // 12;
var Square = function(width) {
  Rectangle.call(this, width, width);
};
Square.prototype = Object.create(Rectangle.prototype);
Object.freeze(Square.prototype);
var sq = new Square(5);
```

클래스 상속 클래스가 구체적인 데이터를 지니지 않게 하는 방법(3) + constructor복구

범용적인 함수

```
var extendClass3 = function (SuperClass, SubClass, subMethods) {
  SubClass.prototype = Object.create(SuperClass.prototype);
  SubClass.prototype.constructor = SubClass;
  if (subMethods) {
    for (var method in subMethods) {
      SubClass.prototype[method] = subMethods[method];
    }
  }
  Object.freeze(SubClass.prototype);
  return SubClass;
};
```

superClass, subClass, subClass에 추가할 메서드들이 정의된 객체를 인자로 받아서

subClass의 prototype 내용을 정리하고 freeze하고,

SubClass.prototype.constructor가 원래의 SubClass를 바라보도록 함.

```
var Square = extendClass3(Rectangle, function (width) {
  Rectangle.call(this, width, width);
});
```

클래스 상속 상위 클래스에의 접근 수단 제공

하위 클래스의 메서드에서 상위 클래스의 메서드 실행 결과를 바탕으로 추가적인 작업을 수행할 때, SuperClass.prototype.mehod.apply(this, arguments)로 하지 않고 다른 객체지향 언어들처럼 super 메서드를 구현

```
var extendClass = function (SuperClass, SubClass, subMethods) {
  SubClass.prototype = Object.create(SuperClass.prototype);
  SubClass.prototype.constructor = SubClass;
  SubClass.prototype.super = function (propName) {
    var self = this;
    if(!propName) return function () {
      SuperClass.apply(self, arguments);
    }
    var prop = SuperClass.prototype[propName];
    if (typeof prop !== 'function') return prop;
    return function () {
      return prop.apply(self, arguments);
    }
  };
  if (subMethods) {
    for (var method in subMethods) {
    SubClass.prototype[method] = subMethods[method];
    }
  }
  Object.freeze(SubClass.prototype);
  return SubClass;
};
```

클래스 상속 상위 클래스에의 접근 수단 제공

```
// .. 생략 ...
SubClass.prototype.super = function (propName) {
  var self = this;
  if(!propName) return function () {
    SuperClass.apply(self, arguments);
  }
  var prop = SuperClass.prototype[propName];
  if (typeof prop !== 'function') return prop;
  return function () {
    return prop.apply(self, arguments);
  }
};
// .. 생략 ...
```

인자가 비어있을 때, SuperClass의 생성자 함수에 접근

클로저를 활용해 this가 바뀌지 않도록 함.

인자가 함수가 아니면 해당 값을 그대로 반환하고,

함수이면 클로저를 활용해 메서드에 접근

call메서드로 this를 바인딩하지 않고,

this.super()(width, width)로 표현할 수 있음!

(클로저를 썼기 때문에)

sq.super('getArea')()처럼 super메서드를 통해

상위 메서드에 접근 가능!

```
var Square = extendClass(Rectangle,
  function (width) {
    this.super()(width, width);
  }, {
    getArea: function () {
      console.log('size is:',
this.super('getArea')());
    }
  }
);
var sq = new Square(10);
sq.getArea(); //size is: 100
console.log(sq.super('getArea')()) // 100
```

ES6의 클래스 및 클래스 상속

ES5 클래스

```
var ES5 = function (name) {
  this.name = name;
};
ES5.staticMethod = function () {
  return this.name + ' staticEMethod';
};
ES5.prototype.method = function () {
  return this.name + ' method';
};
var es5Instance = new ES5('es5');
console.log(ES5.staticMethod()); //es5 staticMethod
console.log(es5Instance.method()); //es5 method
```

ES6 클래스

```
const ES6 = class {
  constructor (name) {
    this.name = name;
  }
  static staticMethod () {
    return this.name + ' staticMethod';
  }
  method() {
    return this.name + ' method';
  }
};
var es6Instance = new ES6('es6');
console.log(ES6.staticMethod()); //es6 staticMethod
console.log(es6Instance.method()); //es6 method
```

ES6의 클래스 및 클래스 상속

ES6 클래스 상속

```
const Rectangle = class {
  constructor (width, height) {
    this.width = width;
    this.height = height;
  }
  getArea () {
    return this.width * this.height;
  }
};
const Square = class extends Rectangle {
  constructor (width) {
    super(width, width);
  }
  getArea () {
    console.log('size is :', super.getArea());
  }
};
```

extends로 상속관계 표현 끝!

super로 상위클래스 메서드 접근가능!

클래스 호이스팅

클래스 선언문으로 정의한 클래스는 함수 선언문과 같이 런타임 이전에 먼저 평가되어 함수 객체를 생성함.

이 때 생성한 함수 객체는 생성자 함수로서 호출할 수 있는 함수, constructor 임! 이 시점에 프로토타입도 같이 생성됨.

함수 선언문과 달리 클래스 선언문은 호이스팅이 일어나지 않는 것처럼 보이지만 let키워드처럼 일시적 사각지대(TDZ)에 ReferenceError 결과가 나옴

```
console.log(a); //[Function: a]
console.log(b); // ReferenceError: Cannot access 'b' before initialization
function a(){};
class b{};
```

QnA 지수님 스터디 참고

- 자바스크립트에서 클래스라는 개념이 있나요?
- 클래스에 있는 값이 인스턴스에 영향을 주면 안되는 이유가 뭔가요?
- ES5 이전에 클래스가 구체적인 데이터를 지니지 않게 하는 방법에는 무엇이 있나요?
- 프로토타입 메서드와 스태틱 메서드에 대해서 설명해주세요