객체지향기법

Javascript의 객체지향 특징

❖ 기존 방법의 특징

- 명시적인 class 정의가 없음
- 함수를 이용하여 객체 정의
- 프로토타입 기반의 상속
  - 프로토타입: 어떤 객체의 원본이 되는 객체

❖ ES2015 이후의 객체 지향 기법

- class 키워드 지원
- 상속을 지원

❖ 가장 간단한 클래스 정의

- 생성자 함수: new로 호출하는 함수

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

var mem = new Member();
```

```
function Member() {};

var mem = new Member();
```

❖ 생성자로 초기화하기

- 프로퍼티의 추가 : this를 이용하여 표시

```
function Member(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;

  this.getName = function() {
    return this.lastName + ' ' + this.firstName;
  }
};

var mem = new Member('철수', '강');
console.log(mem.getName());
```

❖ 동적으로 메서드 추가하기

- Javascript 객체의 프로퍼티는 동적으로 추가/삭제 가능

```
function Member(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
};

var mem = new Member('철수', '강');

mem.getName = function() {
  return this.lastName + ' ' + this.firstName;
}

console.log(mem.getName());
```

❖ this

- 문맥(호출하는 곳, 호출하는 방법)에 따라 내용이 변함

장소	this가 참조하는 곳
톱 레벨(함수의 바깥)	글로벌 객체
함수	글로벌 객체(Strict 모드에서는 undefined)
call/apply 메소드	인수로 지정된 객체
이벤트 리스너	이벤트의 발생처
생성자	생성한 인스턴스
메소드	호출원의 객체(=리시버 객체)

❖ this의 설정

```
func.call(that [,arg1 [,arg2 [,…]]])
func.apply(that [,args])
```

func: 함수 객체 — *that*: 함수 안에서 this 키워드가 가리키는 것

arg1, arg2 … : 함수에 건넬 인수 — *args*: 함수에 건넬 인수(배열)

```
var data = 'Global data';
var obj1 = { data: 'obj1 data' };
var obj2 = { data: 'obj2 data' };

function hoge() {
  console.log(this.data);
}

hoge.call(null);
hoge.call(obj1);
hoge.call(obj2);
```

❖ 생성자 함수 호출 주의점

- new를 사용하지 않으면 일반 함수 호출임
- this의 해석이 달라짐 → 전역 객체

```
function Member(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
};

var m = Member('인식', '정');
console.log(m);
console.log(firstName);
console.log(m.firstName);          // 에러
```

❖ 생성자 함수 호출 주의점

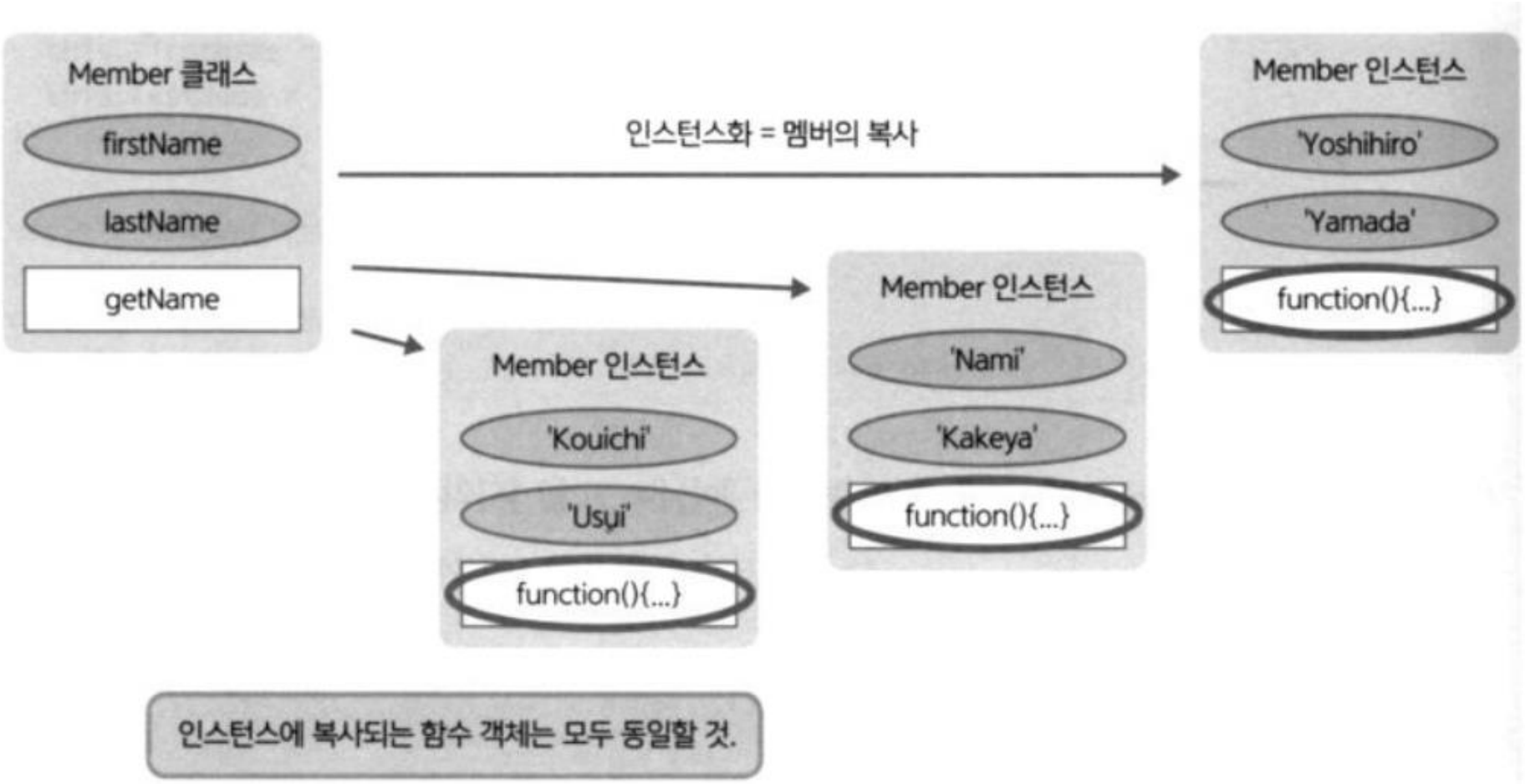
- new를 사용하지 않으면 일반 함수 호출임
- this의 해석이 달라짐 → 전역 객체

```
function Member(firstName, lastName) {
  if(!(this instanceof Member)) {
    return new Member(firstName, lastName);
  }
  this.firstName = firstName;
  this.lastName = lastName;
  this.getName = function(){
    return this.lastName + ' ' + this.firstName;
  }
};

var m = Member('인식', '정');

console.log(m);
console.log(m.firstName);
```

생성자의 문제점과 프로토타입

❖ 생성자의 문제점

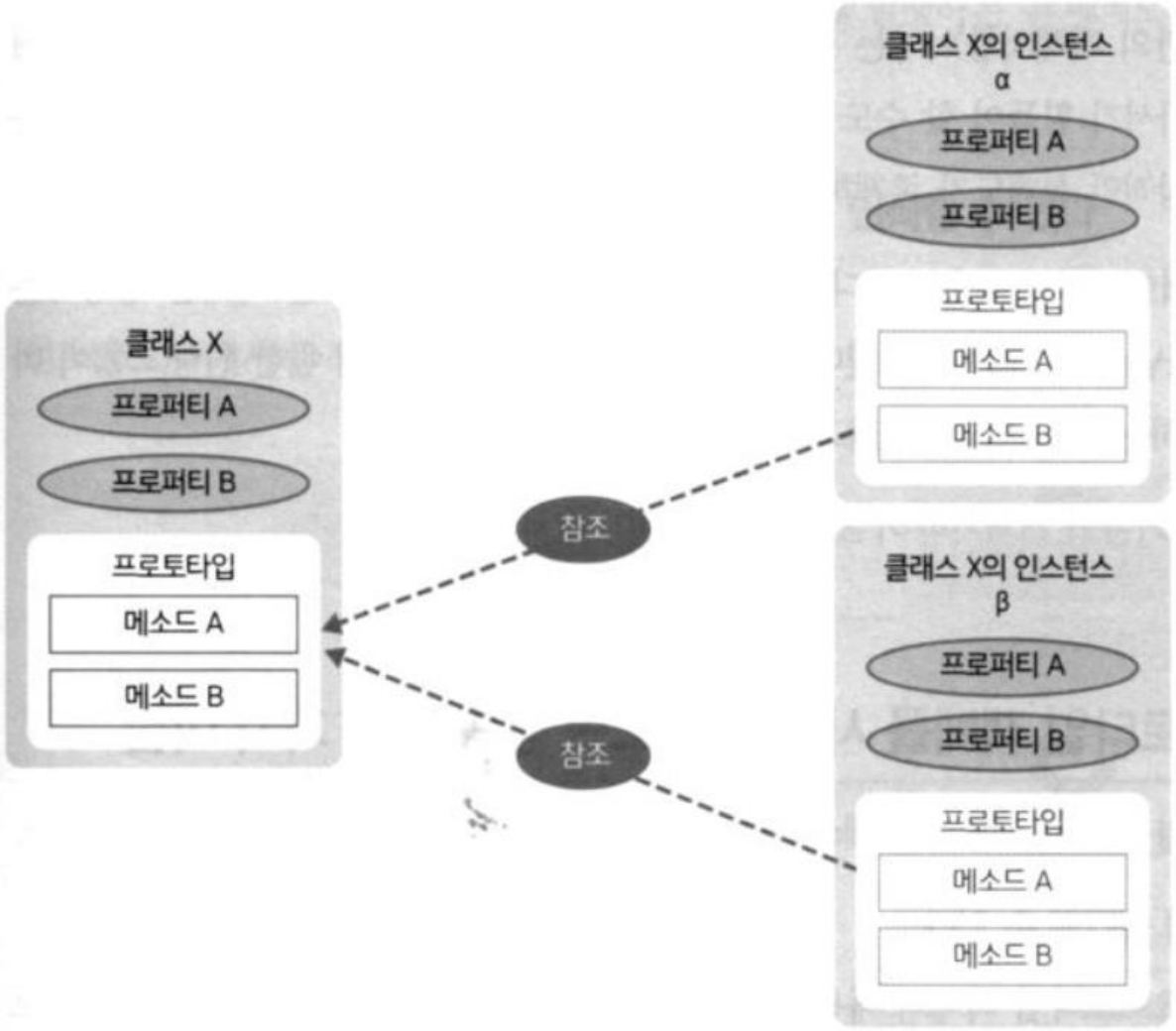
- 동일한 메서드를 인스턴스마다 가짐 → 메모리 효율이 나쁨



생성자의 문제점과 프로토타입

❖ 프로토타입 객체

- 메서드는 프로토타입 객체의 프로퍼티로 추가



생성자의 문제점과 프로토타입

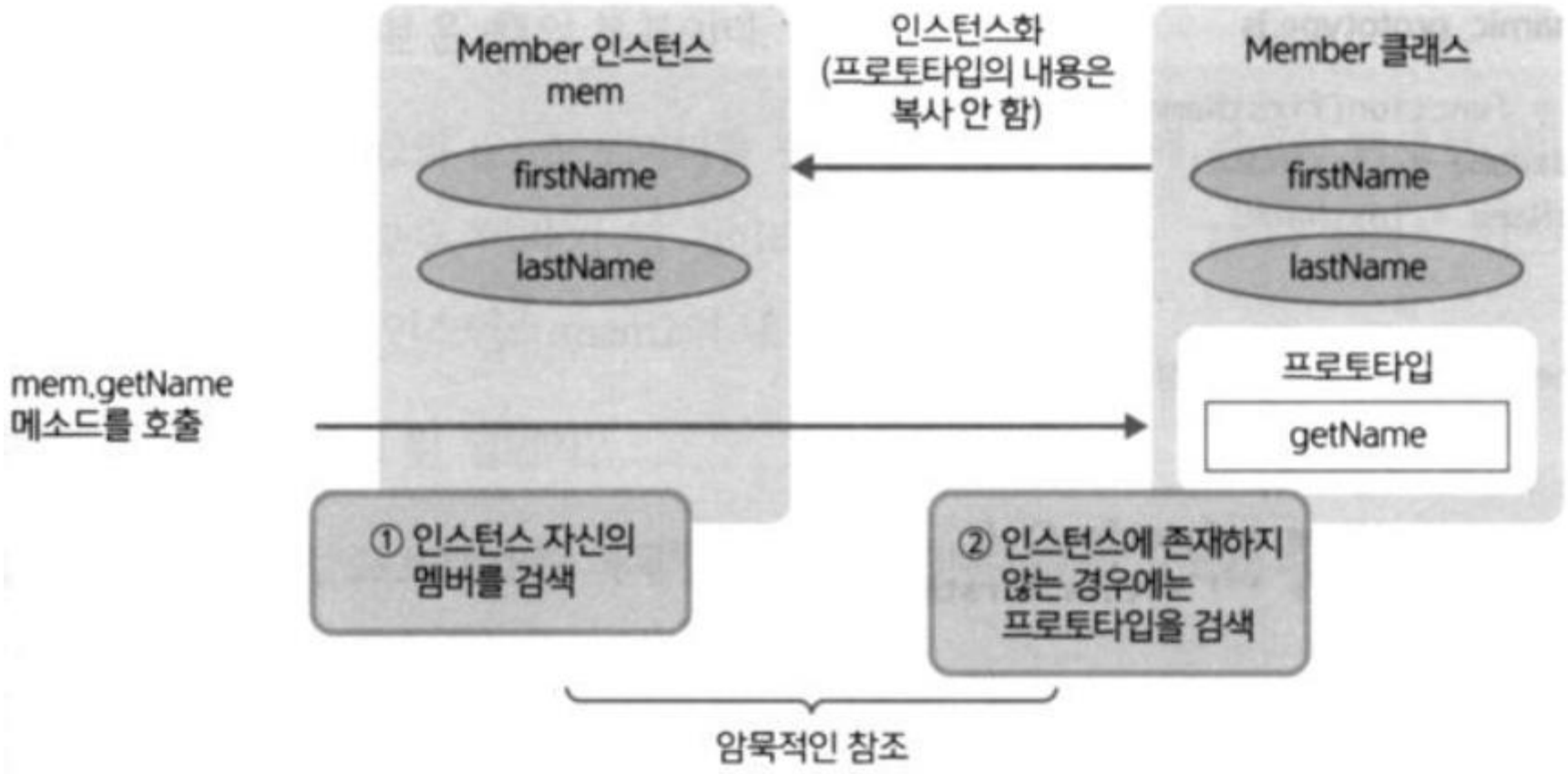
❖ 프로토타입 객체

- 일종의 부모 객체에 해당

```
function Member(firstName, lastName){
  this.firstName = firstName;
  this.lastName = lastName;
};

Member.prototype.getName = function() {
  return this.lastName + ' ' + this.firstName;
};

var mem = new Member('인식', '정');
console.log(mem.getName());
```

생성자의 문제점과 프로토타입

❖ 프로토타입 객체



생성자의 문제점과 프로토타입

❖ 프로토타입 객체

```
function Member(firstName, lastName){
  this.firstName = firstName;
  this.lastName = lastName;
};

var mem = new Member('인식', '정');

Member.prototype.getName = function() {
  return this.lastName + ' ' + this.firstName;
};

console.log(mem.getName());
```

생성자의 문제점과 프로토타입

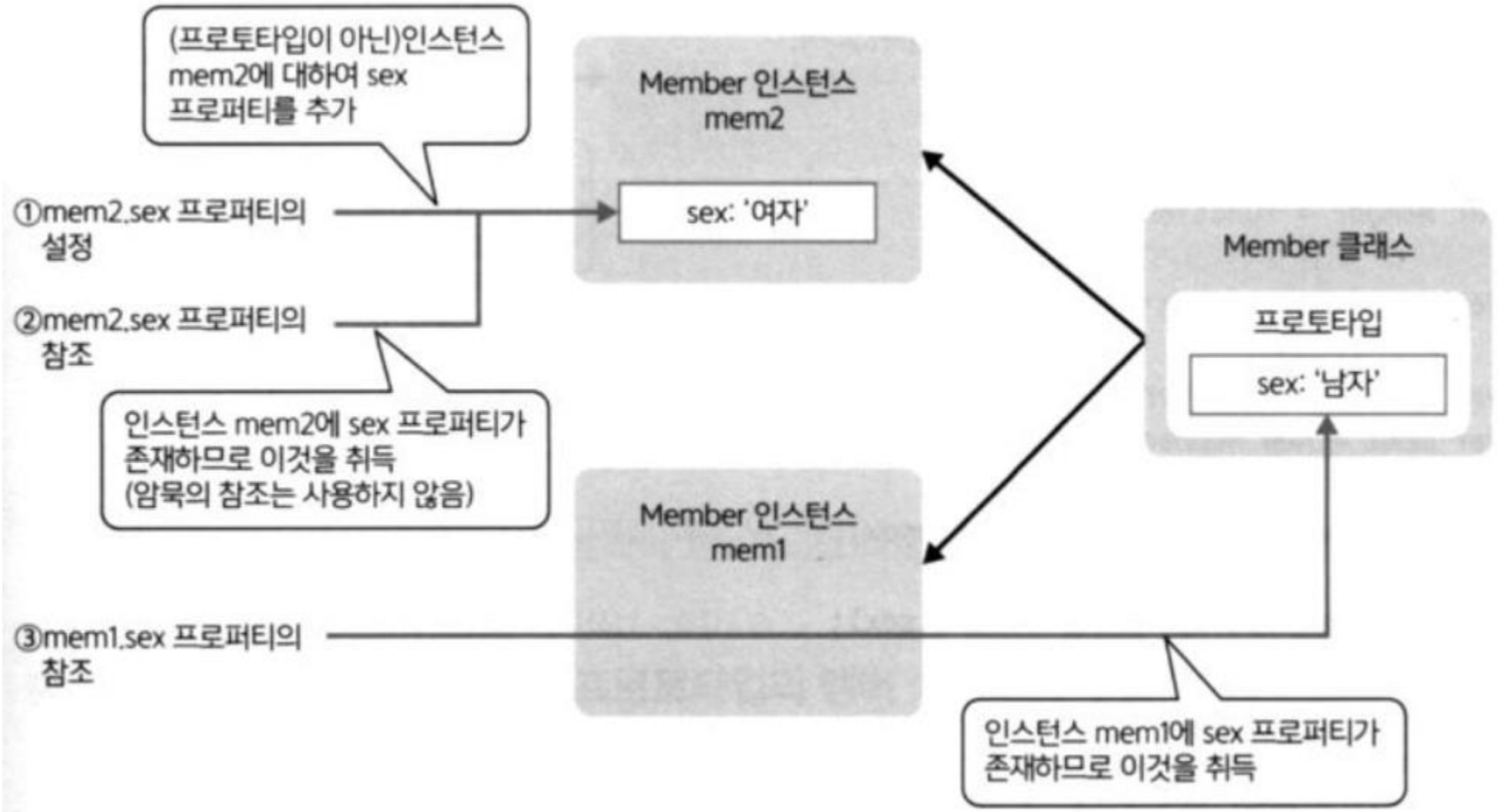
❖ 객체 프로퍼티 검색 순서

- 읽기, 쓰기가 다름

```
function Member() { };

Member.prototype.sex = '남자';
var mem1 = new Member();
var mem2 = new Member();

console.log(mem1.sex + '|' + mem2.sex);
mem2.sex = '여자';
console.log(mem1.sex + '|' + mem2.sex);
```

생성자의 문제점과 프로토타입

❖ 객체 프로퍼티 검색 순서



생성자의 문제점과 프로토타입

❖ 객체 리터럴로 프로토타입 정의하기

```
var Member = function(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
};

Member.prototype.getName = function() {
  return this.lastName + ' ' + this.firstName;
};

Member.prototype.toString = function() {
  return this.lastName + this.firstName;
};

var mem = new Member('성룡', '김');
console.log(mem.getName());
console.log(mem.toString());
```

생성자의 문제점과 프로토타입

❖ 객체 리터럴로 프로토타입 정의하기

```
var Member = function(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
};

Member.prototype = {
  getName : function() {
    return this.lastName + ' ' + this.firstName;
  },
  toString : function() {
    return this.lastName + this.firstName;
  }
};

var mem = new Member('성룡', '김');
console.log(mem.getName());
console.log(mem.toString());
```

생성자의 문제점과 프로토타입

❖ 정적 프로퍼티/정적 메서드 정의하기

- static은 인스턴스와 무관함을 의미
  - this 사용 불가
- 인스턴스를 통해 접근하는 것이 아니고, 객체명으로 접근

```
객체명.프로퍼티명 = 값
객체명.메소드명 = function() { /*메소드의 정의*/ }
```

생성자의 문제점과 프로토타입

❖ 정적 프로퍼티/정적 메서드 정의하기

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

Area.version = '1.0';

Area.triangle = function(base, height) {
  return base * height / 2;
};

Area.diamond = function(width, height) {
  return width * height / 2;
};

console.log('Area클래스의 버전:' + Area.version);
console.log('삼각형의 면적:' + Area.triangle(5, 3));

var a = new Area();
console.log('마름모의 면적:' + a.diamond(10, 2));    // 에러
```

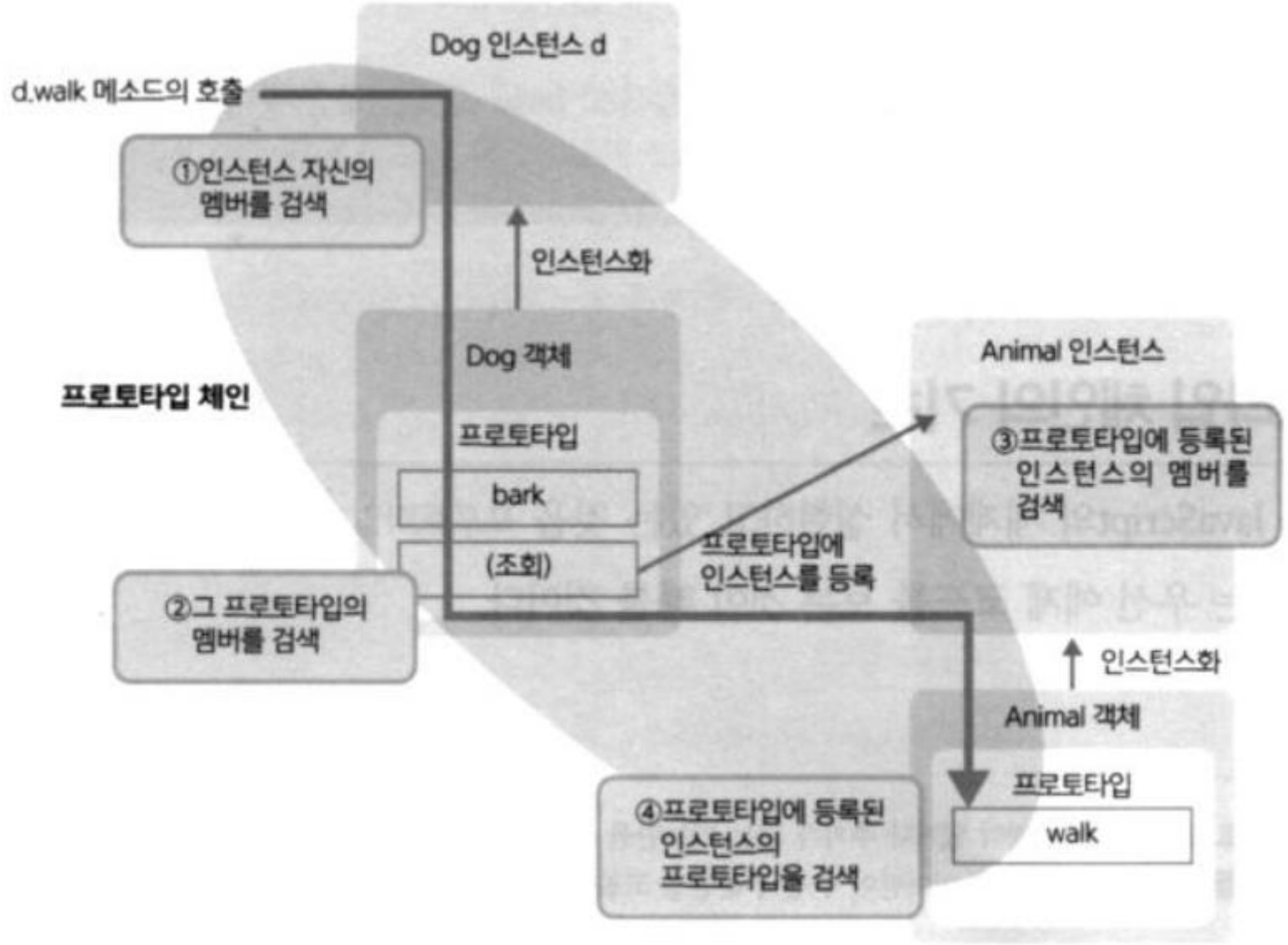
객체의 계승(상속)

❖ 프로토타입 체인

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

Animal.prototype = {
  walk : function() {
    console.log('종종...');
  }
};

var Dog = function() {
  Animal.call(this);
};
Dog.prototype = new Animal();
Dog.prototype.bark = function() {
  console.log('멍멍!! ');
}

var d = new Dog();
d.walk();
d.bark();
```

객체의 계승(상속)

❖ 프로토타입 체인



객체의 계승(상속)

❖ 객체의 타입 판정하기

- instanceof

```
var Animal = function() {};
var Hamster = function() {};
Hamster.prototype = new Animal();

var a = new Animal();
var h = new Hamster();
console.log(a.constructor === Animal);
console.log(h.constructor === Animal);
console.log(h.constructor === Hamster);

console.log(h instanceof Animal);
console.log(h instanceof Hamster);

console.log(Hamster.prototype.isPrototypeOf(h));
console.log(Animal.prototype.isPrototypeOf(h));
```

객체의 계승(상속)

❖ 멤버의 유무 판정하기

- in 연산자

```
var obj = { hoge: function(){}, foo: function(){} };

console.log('hoge' in obj);
console.log('piyo' in obj);
```

ES2015의 객체지향 구문

❖ class 키워드

- 생성자 : constructor()
  - 함수로서 호출할 수 없다. 반드시 new 객체명()을 통해서 호출
- 메서드 : 자동으로 프로토타입 메서드가 됨

```
class Member {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  getName() {
    return this.lastName + this.firstName;
  }
}

let m = new Member('시온', '정');
console.log(m.getName());
```

ES2015의 객체지향 구문

❖ 프로퍼티 정의

- 메서드 앞에 get/set 설정

```
class Member {
  constructor(firstName, lastName) {
    this._firstName = firstName;
    this._lastName = lastName;
  }

  get firstName() {
    return this._firstName;
  }

  set firstName(value) {
    this._firstName = value;
  }

  get lastName() {
    return this._lastName;
  }

  set lastName(value) {
    this._lastName = value;
  }
```

ES2015의 객체지향 구문

❖ 프로퍼티 정의

- 메서드 앞에 get/set 설정

```
  getName() {
    return this.lastName + this.firstName;
  }
}

let m = new Member('시온', '정');
console.log(m.getName());
```

ES2015의 객체지향 구문

❖ 정적 메서드 정의하기

```
class Area {
  static getTriangle(base, height) {
    return base * height / 2;
  }
}

console.log(Area.getTriangle(10, 5));
```

ES2015의 객체지향 구문

❖ 상속하기

```
class Member {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  getName() {
    return this.lastName + this.firstName;
  }
}

class BusinessMember extends Member {
  work() {
    return this.getName() + '은 공부하고 있습니다.';
  }
}

let bm = new BusinessMember('시온', '정');
console.log(bm.getName());
console.log(bm.work());
```

ES2015의 객체지향 구문

❖ 상속하기

```
class Member {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  getName() {
    return this.lastName + this.firstName;
  }
}

class BusinessMember extends Member {
  constructor(firstName, lastName, clazz) {
    super(firstName, lastName);
    this.clazz = clazz;
  }

  getName() {
    return super.getName() + ' / 직책:' + this.clazz;
  }
}

let bm = new BusinessMember('성룡', '김', '과장');
console.log(bm.getName());
```

ES2015의 객체지향 구문

❖ 객체 리터럴

```
let member = {
  name: '김성룡',
  birth: new Date(1970, 5, 25),
  toString() {
    return this.name + '/생일 : ' + this.birth.toLocaleDateString()
  }
};

console.log(member.toString());
```

ES2015의 객체지향 구문

❖ 변수를 동일 명칭의 프로퍼티에 할당하기

```
let name = '김성룡';
let birth = new Date(1970, 5, 25);
let member = { name, birth };

console.log(member);
```

ES2015의 객체지향 구문

❖ 프로퍼티를 동적으로 생성하기

```
let i = 0;
let member = {
  name: '김성룡',
  birth: new Date(1970, 5, 25),
  ['memo' + ++i]: '정규회원',
  ['memo' + ++i]: '지부회장',
  ['memo' + ++i]: '경기'
};

console.log(member);
```