

* destructuring

```
var {body} = document;
```

K	V
:	
body	200
:	

~~200~~ <body> — </body>

document.getElementsByTagName("body")[0]

body == document.body ==

* function



function prototype (C/C++)
= method signature (Java)

function 함수명 (파라미터, 파라미터, ...)

Function body {
 `문장1;`
 :
 return 표현식;
}

리턴값 : "aaa", 20, true, {}-[], []
변수 : a, score, sum ...
식 : a + "hello", a * 2, 함수호출 ...

함수호출
(함수 선언시켜놓은 것)
⇒ 함수호출 (인자, 인자, ...);
 ↑ ↓
 아주먼드 (argument)

* 아규먼트와 파라미터

f(10);

function f(a, b) {
 arguments = [10]
}

• 모든 함수에 접근 가능한 Built-in 변수
• 아직 접근 가능한 대상이 없음

* 아규먼트와 파라미터

$f(10, 20);$

function $f(a, b) \{$

\equiv

}

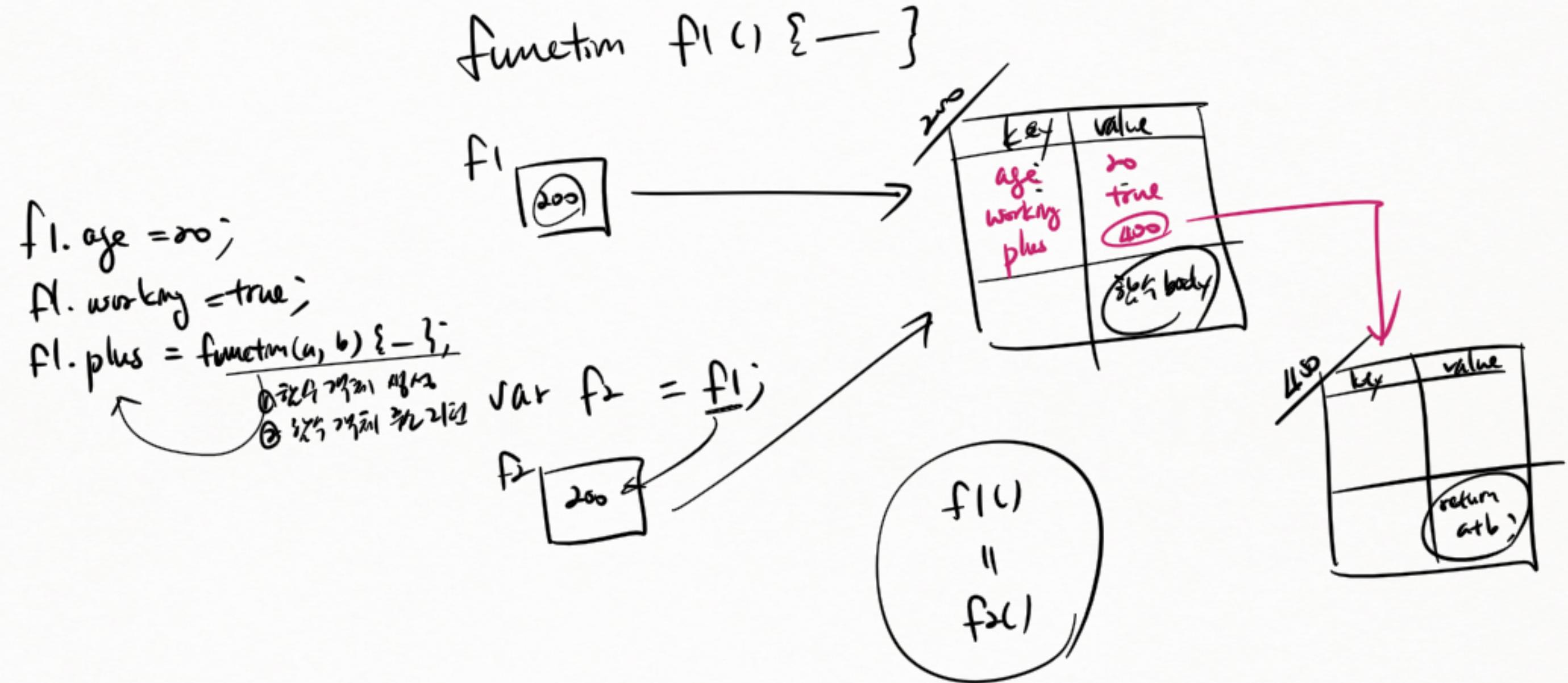
arguments = [10, 20]

* 아규먼트와 파라미터

$f(10, 20, 30, 40);$

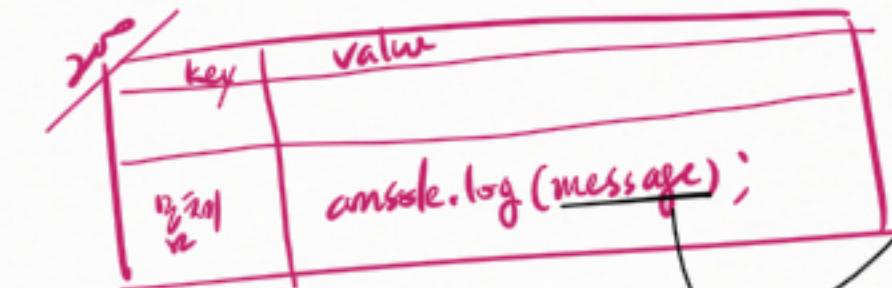
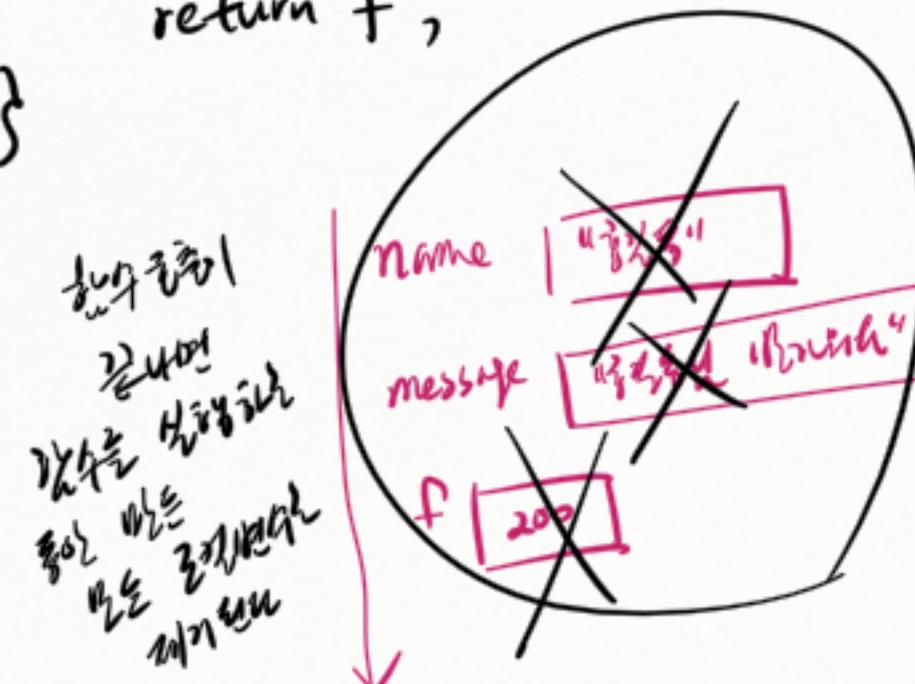
function $f(a, b)$ {
 \equiv arguments = [10, 20, 30, 40]
}

* 흡수와 레퍼런스



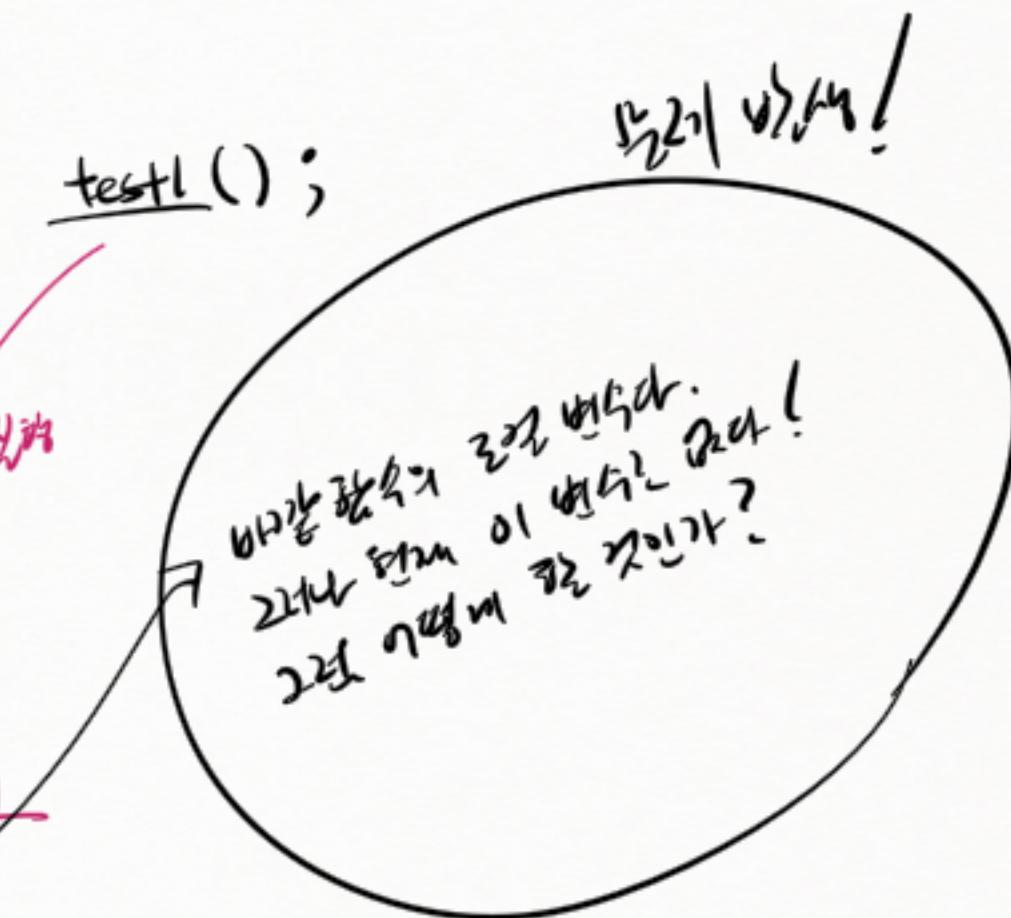
* closure

```
function createGreeting(name) {  
    var message = name + "님 반갑습니다";  
  
    var f = function() { console.log(message); };  
  
    return f;  
}
```



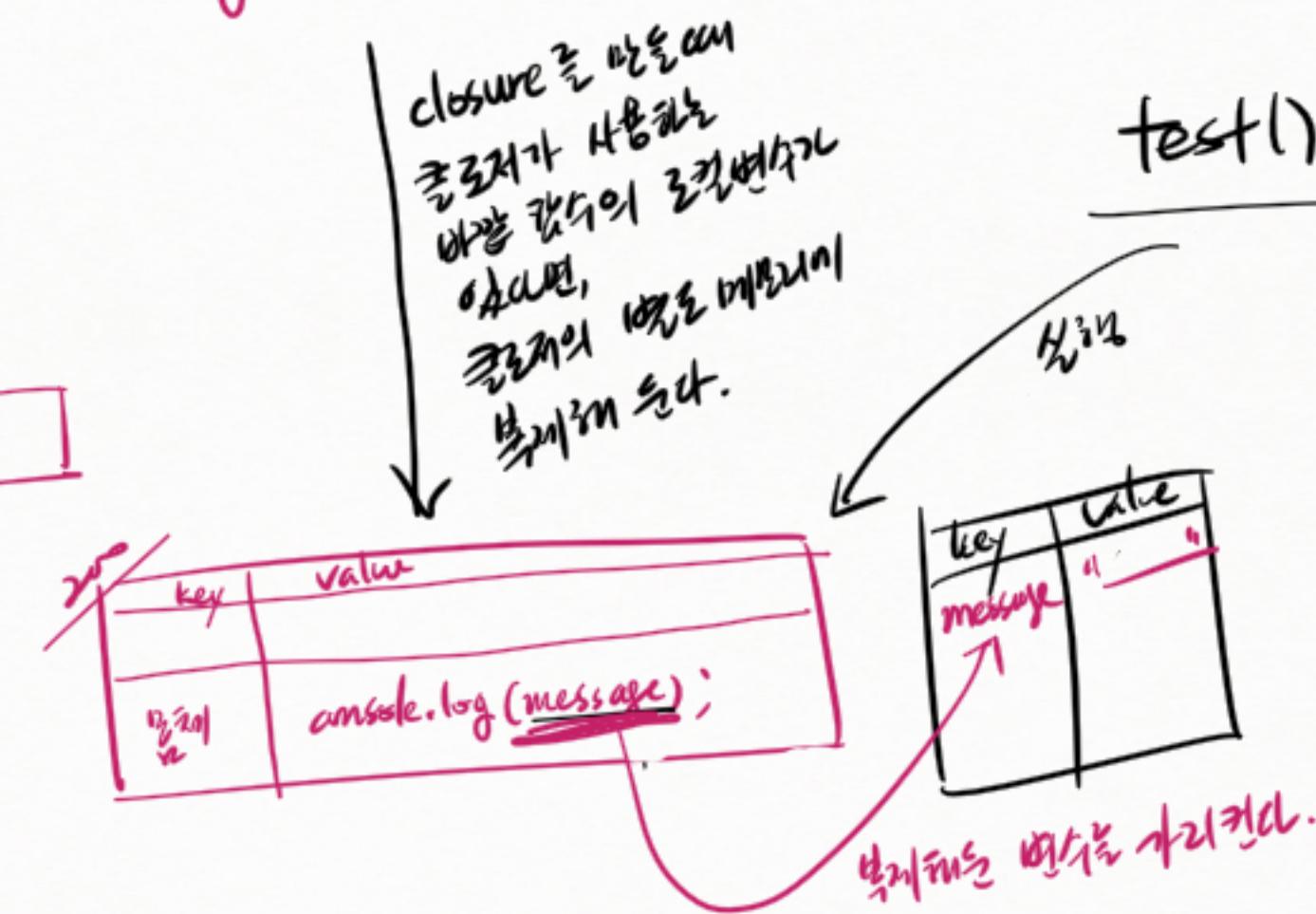
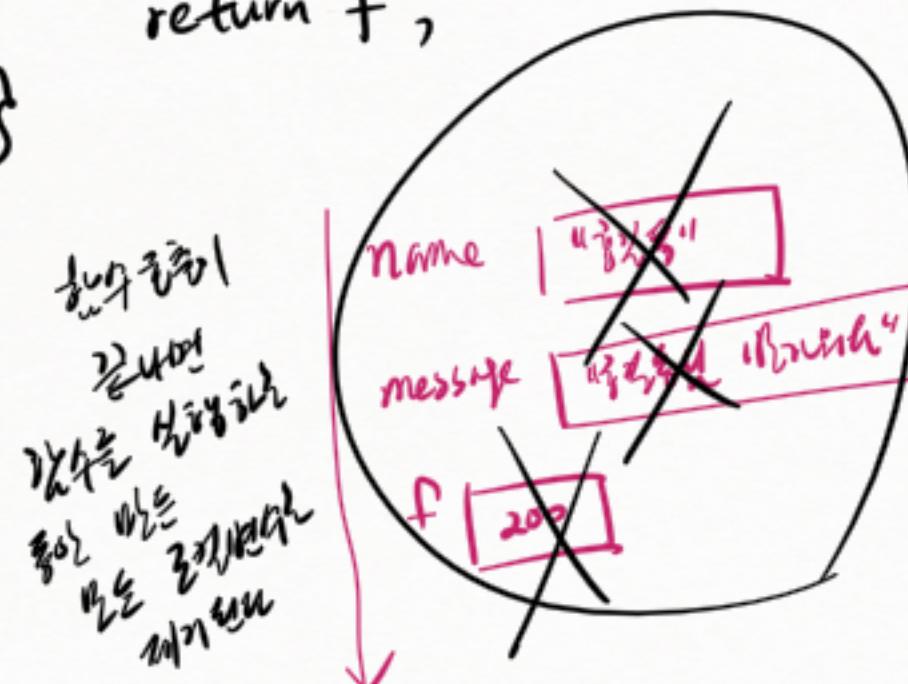
```
var test1 = createGreeting("김민석");  
test1  
200
```

- name
- message
- *



* closure : 1867년 3월 15일 헬기온
1871년 2월 3일 사망!

```
function createGreeting(name) {  
    var message = name + "님 환영합니다";  
  
    var f = function() { console.log(message); };  
  
    return f;  
}
```



var test1 = createGreeting("김현수");
test1
200

- name
- message
- *

test1();

1867년 3월 15일
생각
김현수
200
사망!

* closure 例

test(1)
test2()

var test1 = createGreeting ("Hello");

test1

200

key	value
	console.log(message);

closure 例
variable = 3210392 と
key value

key	value
	"Hello World"

var test2 = createGreeting ("Duck");

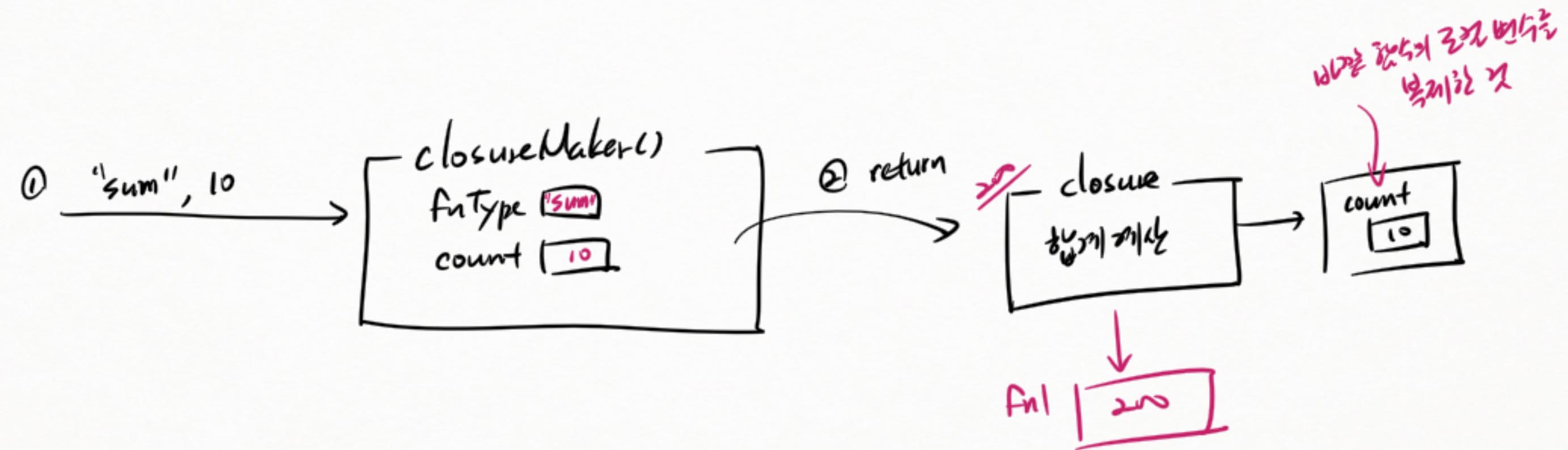
test2

300

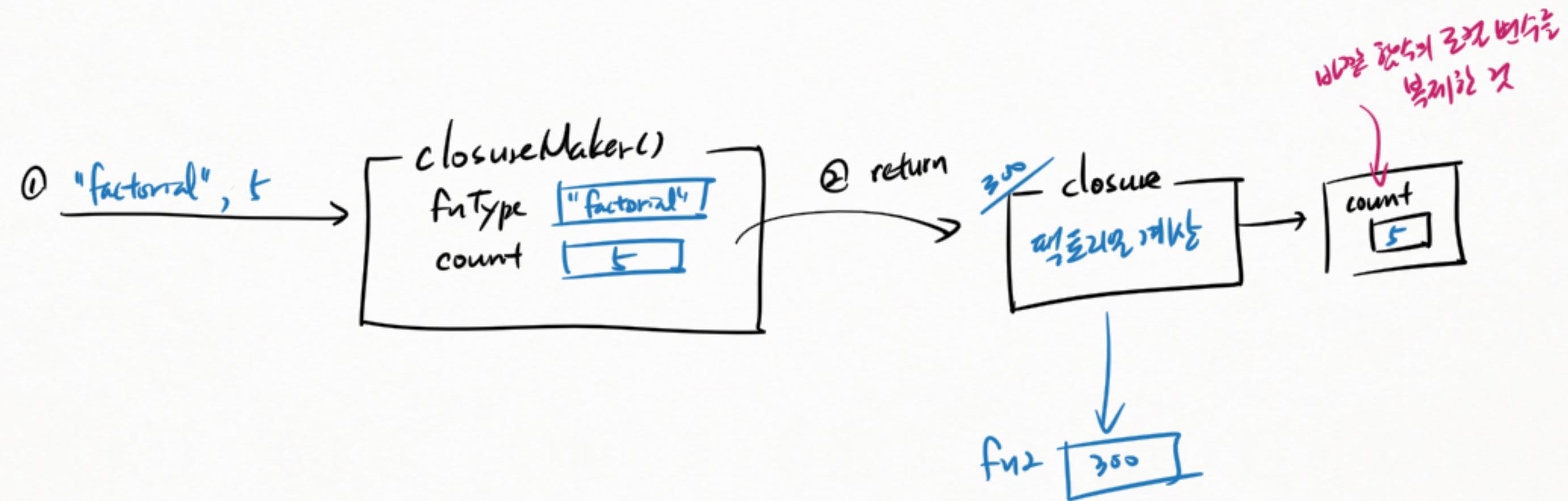
key	value
	console.log(message);

key	value
	"Duck is cute!"

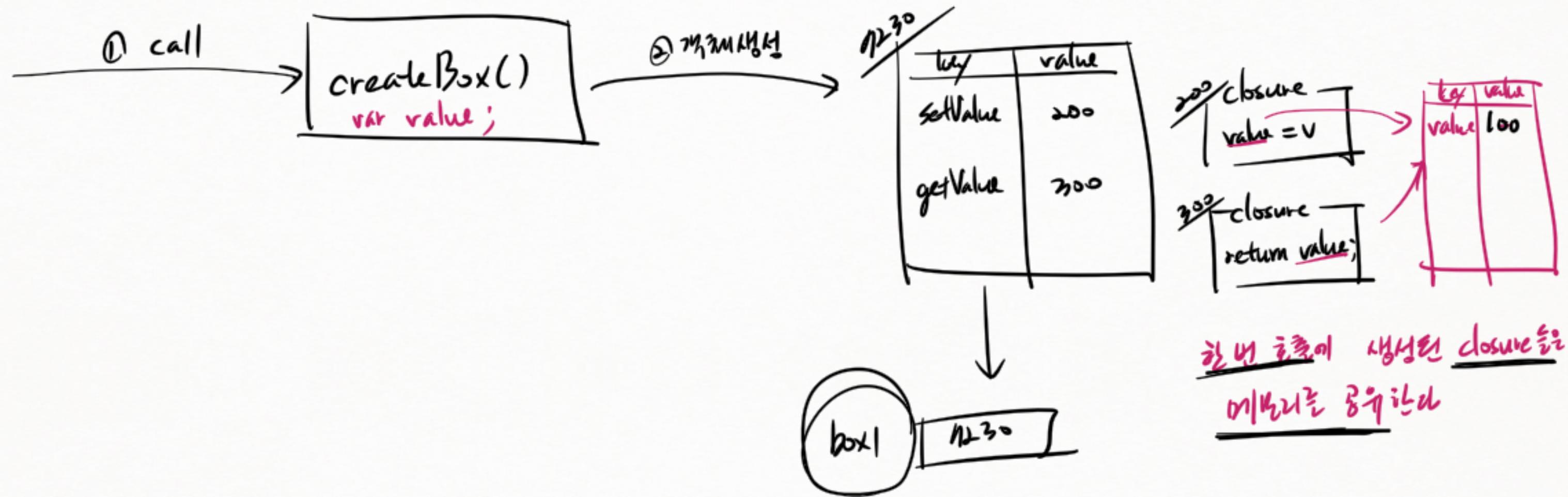
* closure № II



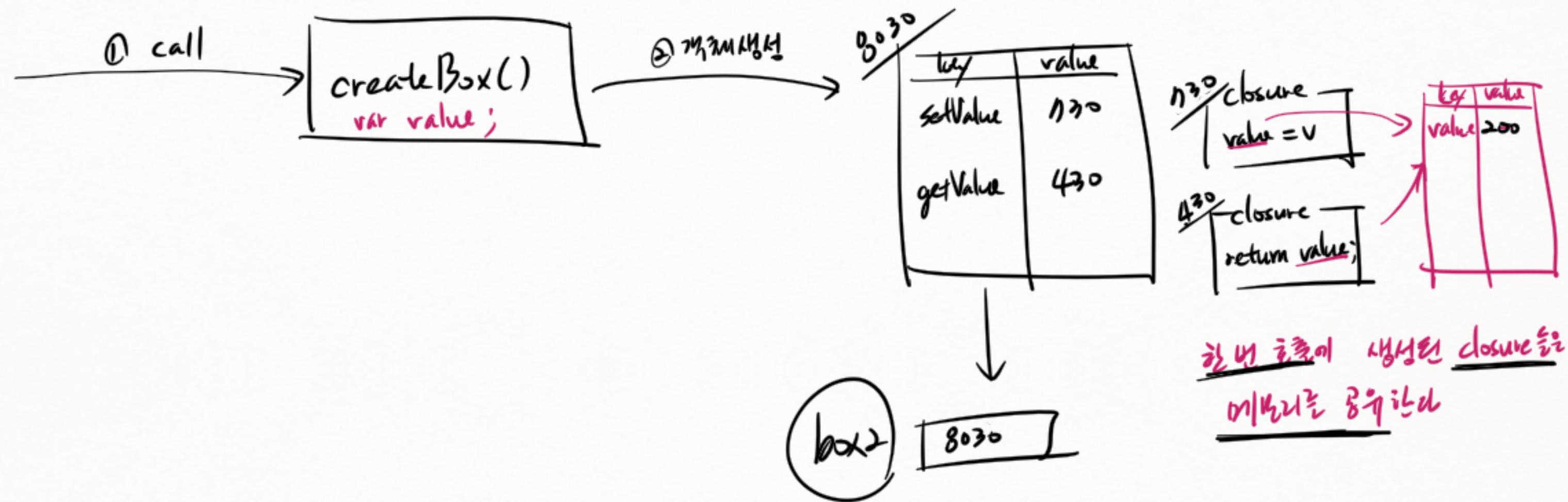
* closure № II



* 4th Ⅲ : box1

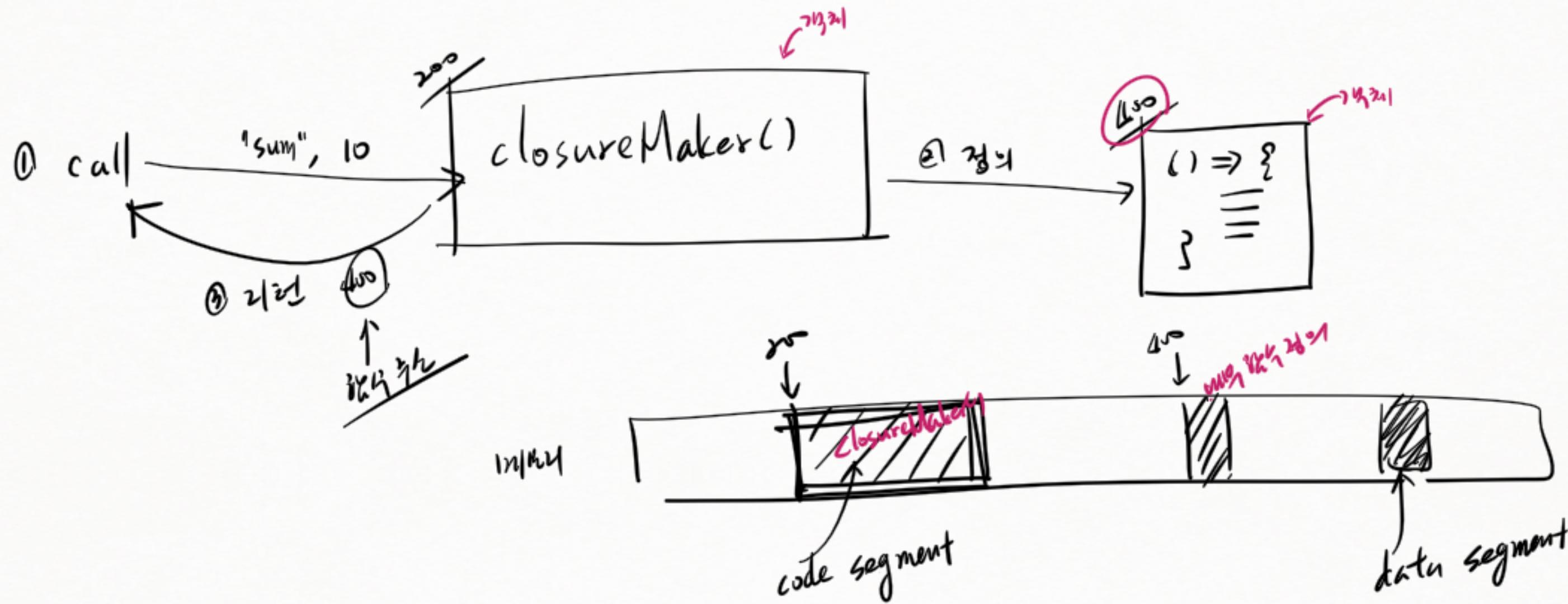


* 4th Ⅲ : box 2

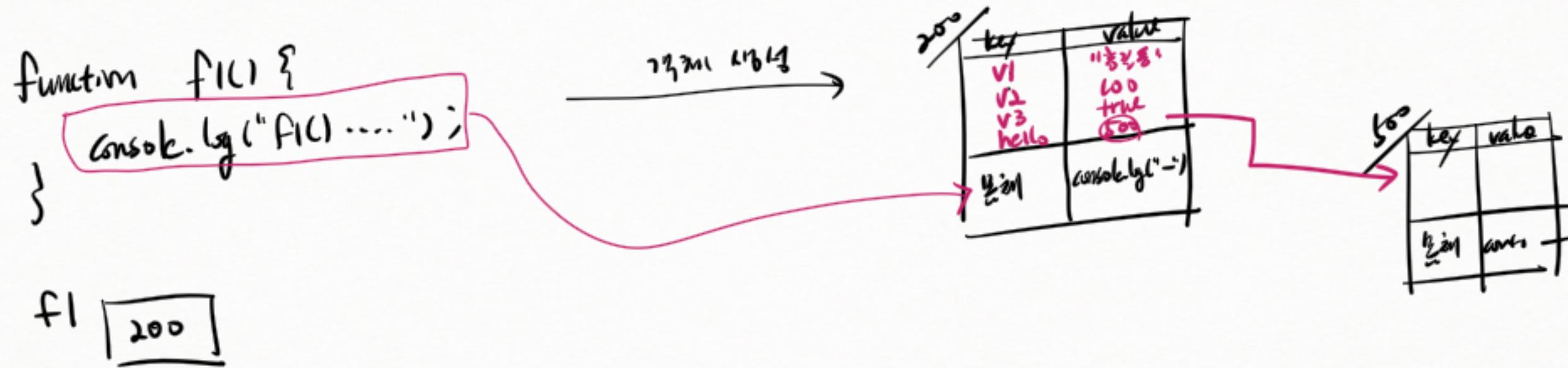


```
function() { return "안녕" }  
          ↓  
console.log( (값주기)() )  
          ↑  
값주기를 가지는 함수 호출  
  
          ( ) => "안녕"  
          ↓  
console.log( (값주기)() )
```

* closure 2단



* 值 (值)



f1 200

f1(); ↗
语句块 语句块! \Rightarrow “值” 值 (function call)"

f1.v1 = "Hello";

f1.v2 = 100;

f1.v3 = true;

f1.hello = function() {
 console.log("Hello!");
};

f1.hello(); ↗
语句块 语句块! \Rightarrow 值 (值)

* 동기화 와 비동기화
Synchronize Asynchronize

Synchronize(동기화)

기본적인 문법
변수 선언
함수 정의
변수 초기화
결과 출력
var a = 100;
var b = 200;
var result = plus(a, b);
console.log(result);

Asynchronize(비동기화)

var a = 100;
var b = 200;
var result = 0;
function calculate() {
 result = a + b;
}
window.setTimeout(plus, 5000);
console.log(result);

호이스팅 예제
변수
값 초기화

* eval()

"JavaScript is"
↓
eval() → this

* onclick 속성

HTML의 onclick = 콜백함수;

↑
호출되는 대상
클릭되었을 때
 onclick 이벤트
 ↓
 "콜백"
 "이벤트 핸들러"
 "이벤트 리스너"
Web Browser

* JSON.parse()

JSON 풀기

① JavaScript 객체를 문자열로 풀기

이후 브루트 채우기?
이후 브루트 채우기

② 문자열은 " "으로 풀기

③ 프로퍼티명은 문자열로 풀기

④ 헷갈리지 말기

- 같은 브루트 채우기 가능!
- 다른 객체 풀기 가능
- 배열 풀기 가능

가져온 JSON 형식으로 정의한 문자열

JSON.parse()

↑
JavaScript의
Built-in 객체

가져온
가져온

JavaScript
Object
Notation

+

JavaScript
Object Literal
풀기

* Data 포맷 : binary vs text

Data

{ 이름: ABC
나이: 20
재직: true

→
바이트 단위로 구조화 따라
저장

① binary

이름길이	이름	나이	재직여부
00	03	41	42

↳ 이를 때 바이트 규칙에 따라 읽는다

* 파일 크기가 가장 작은 편이다

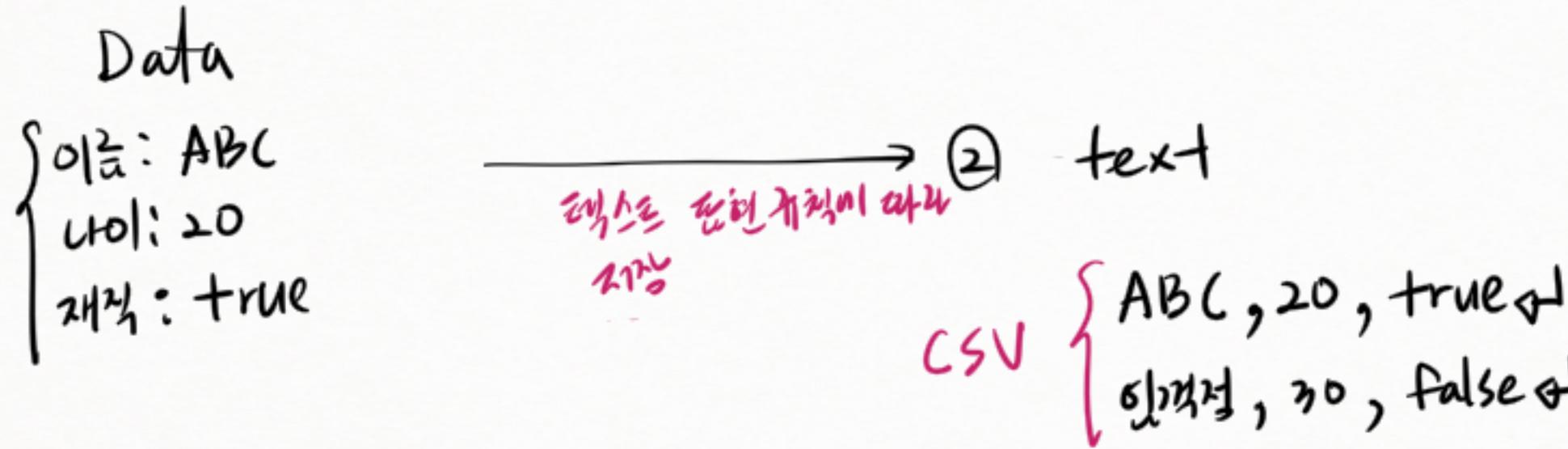
↳ 예? 데이터에 대한 목록이나 설명이 있다
(meta data)



바이트 저장 규칙을 몰라도
데이터를 차대로 읽을 수 있다

* 맥스로 편집기로 데이터를 차대로 볼 수 있다.

* Data 포맷 : binary vs text



- * 텍스트로 데이터를 쉽게 접근할 수 있다
- * binary 형식에 비해 파일 크기는 커진다.

* Text 파일 표기

① CSV

Comma-Separated Value

홍길동, 20, true ↪

임꺽정, 30, false ↪

• 간접적.

• 한 칸에 한 데이터

• 각 항목의 정보가 없다

↓
직접적으로 데이터가 흘러온다

• 매우 간단한 데이터를 다룰 때 좋다

② XML

extensible Markup Language

```
<members>
  <member>
    <name>홍길동</name>
    <age>20</age>
    <working>true</working>
    <schools>
      <school>
        <name>부드천중</name>
        <state>경기</state>
      </school>
      <school>
        <name>부드천고</name>
        <state>경기</state>
      </school>
    </schools>
  </member>
  :
</members>
```

data
meta data: 태그는 명명하는 것.
markup

- 개별 구조의 데이터 표현 가능
- 각 항목의 의미를 표현 가능
metadata

특정 항목의 항목을 찾기 쉬워

- data или metadata가
더 큼 수 있다.
↓
파일 크기가 크다.

* Text 파일 형식

① JSON

JavaScript Object Notation

```
[  
  { "name": "홍길동",  
    "age": 20,  
    "working": true,  
    "schools": [  
      { "name": "마르코폴로", "state": "경기" },  
      { "name": "비즈쿱", "state": "경기" }  
    ]  
  },  
  :  
]
```

- XML 보다 더 간결한 meta data
- JavaScript 와 의사 표로ini 형식이다

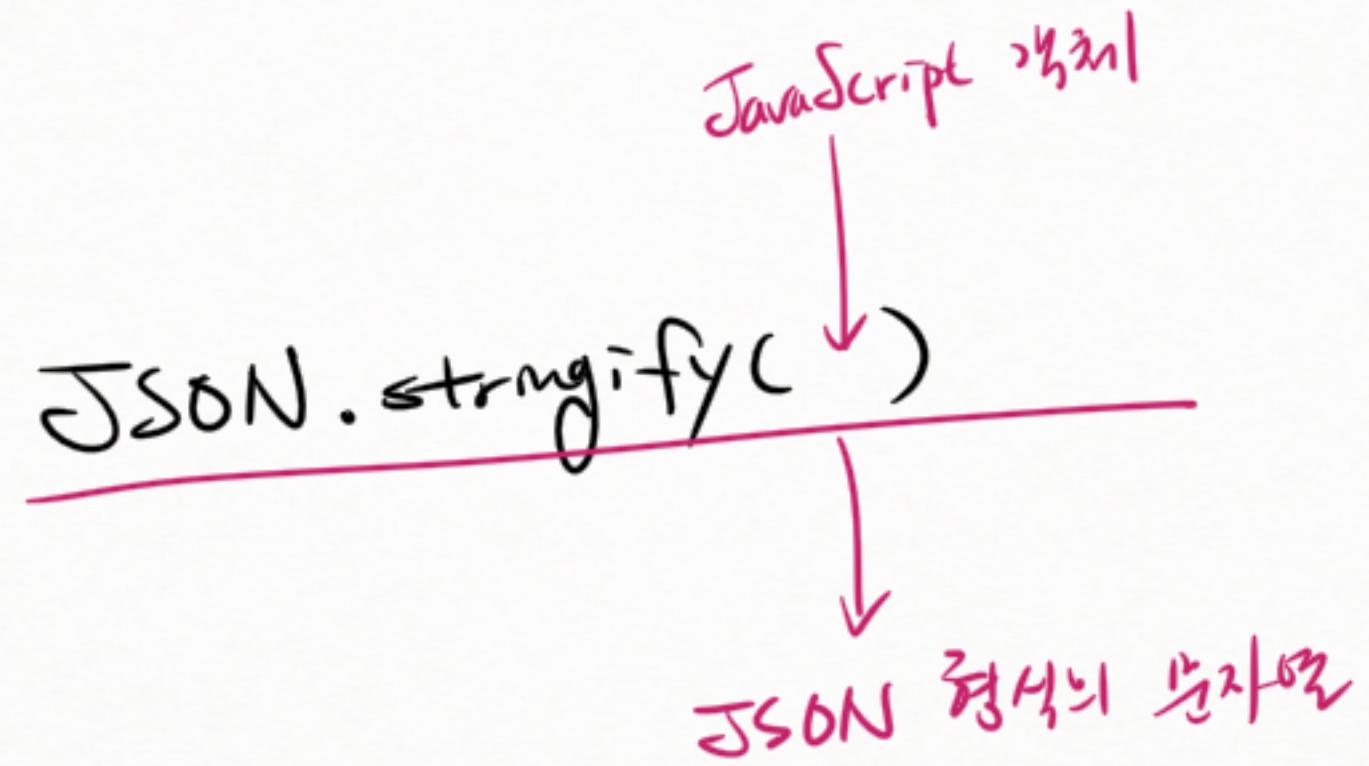
④ YAML

Yet Another Markup Language
↳ YAML ain't markup Language

```
name: 홍길동  
age: 20  
working: true  
schools:  
  - school:  
    name: 마르코폴로  
    state: 경기  
  - school:  
    name: 비즈쿱  
    state: 경기
```

- JSON 보다 더 간결
- 들여쓰기 (indent)로 세이증 구조를 표현

* JSON.stringify()



211 211

* 12월 105회 ~ 드로잉 106회

① Képalkotásokhoz gyűjtemények (pl: Java, C++, ...)

```
class Student {  
    String name;  
    int age;  
    boolean working;  
}
```

Diagram illustrating static type binding:

```

graph TD
    A["(all: Java, C++, ...)"] -- "Static type binding" --> B["Student object"]
    B -- "Static type binding" --> C["Student obj = new Student();"]
    C -- "Object creation" --> D["obj"]
    D -- "Value 200" --> E["200"]
    E -- "Reference to object" --> F["name: 'John' | age: 20 | working: true"]
    F -- "Object state" --> G["name: 'John' | age: 20 | working: true"]

```

The diagram shows the process of static type binding. It starts with a general context "(all: Java, C++, ...)" leading to a "Student object". This object then leads to the assignment statement "Student obj = new Student();". The creation of a new object leads to the variable "obj" containing the value "200". Finally, the variable "obj" is shown pointing to the actual object state, which includes fields "name: 'John'", "age: 20", and "working: true".

```
obj.name = "3218"
```

ob). age = 20;

6b). Working = true

obj. tel = "02-111-2222"

Complex $e^{2\pi i} = -1$

* JavaScript 와 프로토타이핑 비교

② 프로토타이핑 (prototyping) 방법으로 객체 만들기

함수로 접근!

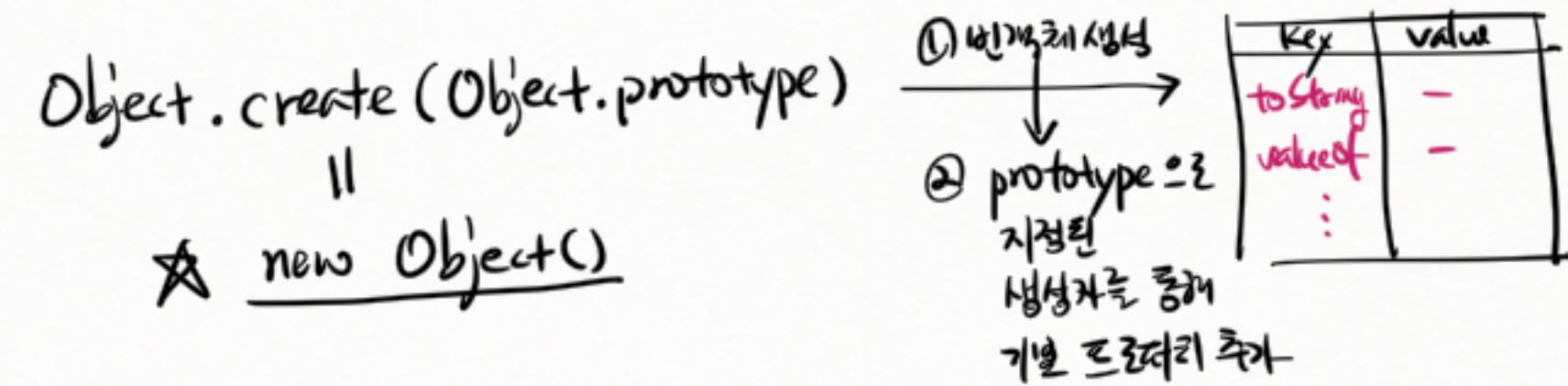
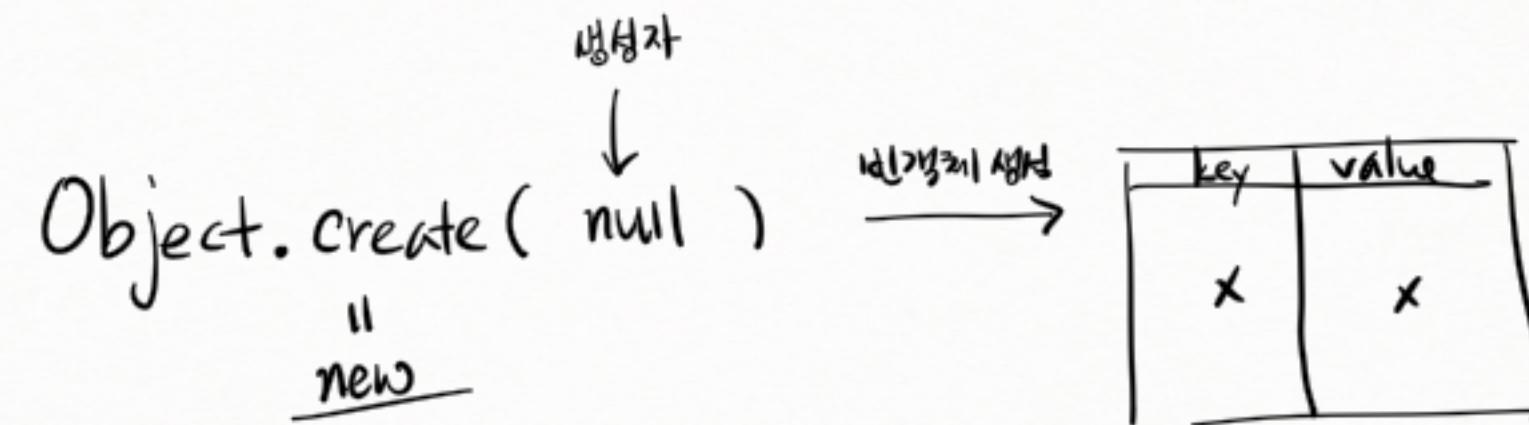


```
var obj = new Object();
```

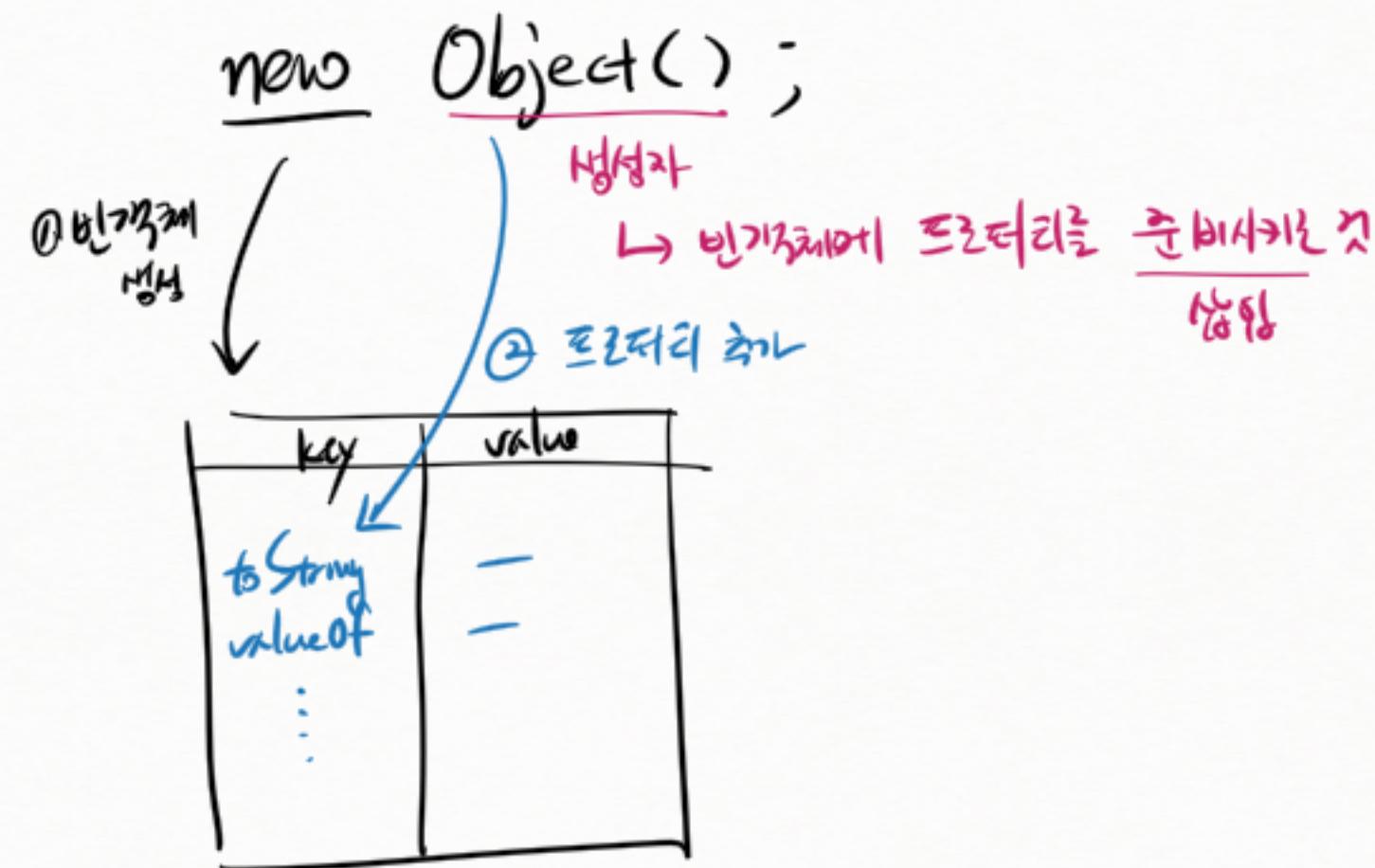
① 비주얼 형태	② 기본 프로퍼리 추가								
<pre>obj.name = "홍길동"; obj.age = 20; obj.working = true;</pre>	<p>기본 프로퍼리 추가 할 때마다 빌드 할 때마다 업데이트 된다.</p> <table border="1"><thead><tr><th>key</th><th>value</th></tr></thead><tbody><tr><td>name</td><td>"홍길동"</td></tr><tr><td>age</td><td>20</td></tr><tr><td>working</td><td>true</td></tr></tbody></table>	key	value	name	"홍길동"	age	20	working	true
key	value								
name	"홍길동"								
age	20								
working	true								

```
obj.name = "홍길동";  
obj.age = 20;  
obj.working = true;
```

* 객체 생성



* 생성자 (constructor)



* hasOwnProperty ("프로퍼티인가")

↓
기존에 추가시킨 프로퍼티인가 검사

var obj = new Object()

obj.hasOwnProperty("toString") → false
" " ("valueOf") → false

obj.plus3 = () => {};
↑
값이 없음

obj.title = " ";
obj["content"] = " ";
obj["viewCount"] = " ";
obj.plus1 = f1;
obj.plus2 = function() { };
↑
값이 없음

key	value
toString	-
valueOf	-
:	-
title	-
content	-
viewCount	-
plus1	-
plus2	-
plus3	-

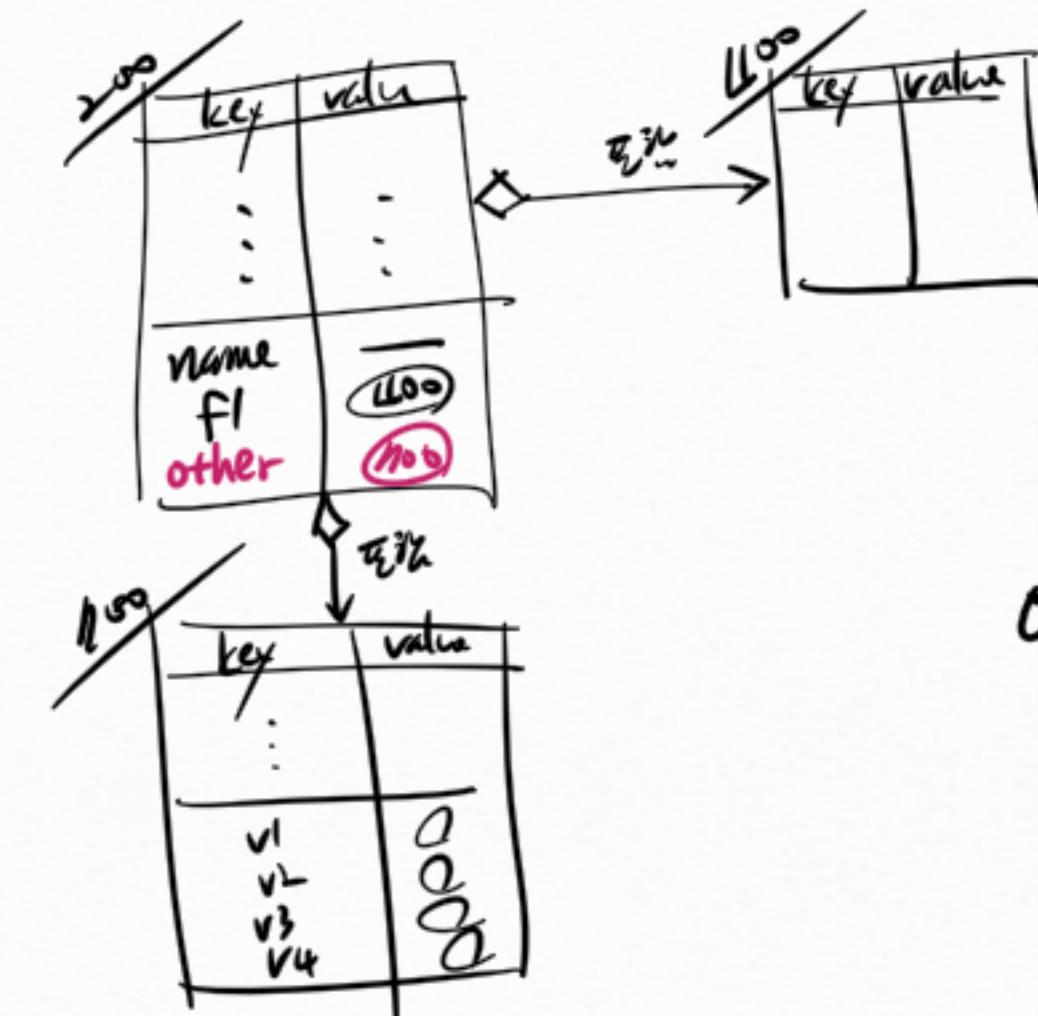
* گیگانی اتے چھپاں کیوں نہیں

let obj = new Object();
100

obj.name = "—";
obj.fl = () => {};

let obj2 = new Object();

obj2.v1 = 0;
obj2.v2 = 0;
obj2.v3 = 0;
obj2.v4 = 0;



obj.other = obj2;
100
ئىچىكى

* think of this

100

K	V
name	-
kor	-
eng	-
math	-
toString	200

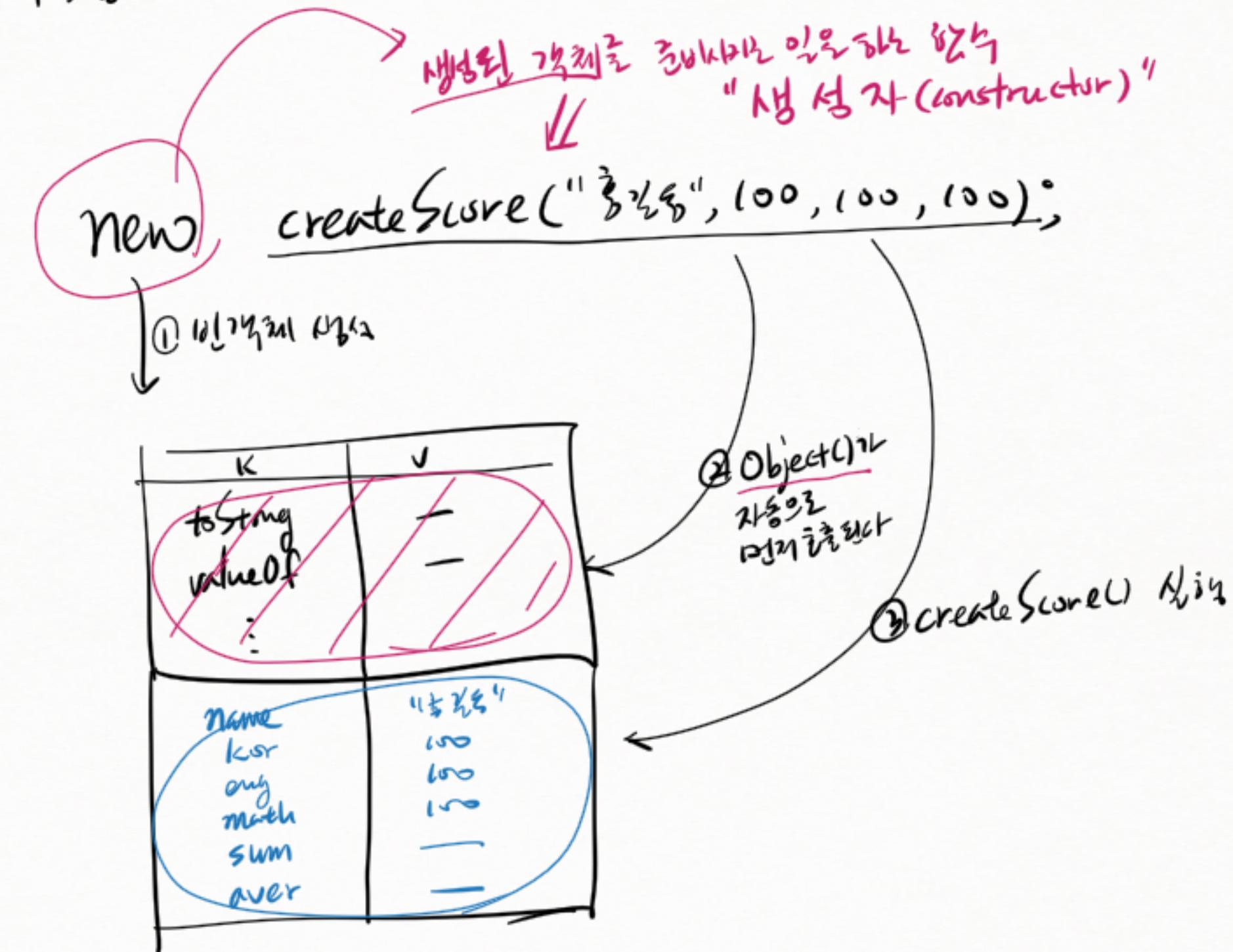
100

K	V
100	100

obj.toString = function() {
 return this.name + ...;
}

100 → return this.name + ...;

* new 484



* Higher prototype

`<생성자>`
Score()

prototype

K	V
sum	<code>f(){} -></code>
aver	<code>f(){} -></code>

Score()가 더 높은 prototype!
즉, 더 높은 prototype
보다 더 높은 prototype!

`new`

K	V
name	-
kor	-
eng	-
math	-

K	V
name	-
kor	-
eng	-
math	-

K	V
name	-
kor	-
eng	-
math	-

`var scores = [200, 300, 400]`

`scores[0].sum();`

↓ call

`Score.prototype.sum()`

`scores[0].aver();`

↓ call

`Score.prototype.aver()`

`scores[2].sum();`

↓ call

`Score.prototype.sum()`

* 생성자와 일상기능

function f() {}

var obj = new f();
 ↑
 new가 생성자임을 알기

① new ~~250~~ $\xrightarrow{\text{Object}}$ K V
 ↓
② Object $\xrightarrow{\text{toString}}$ toString value
 ↓
③ f() $\xrightarrow{\cdot}$. .

K	V
toString	—
value	—
:	:
.	.

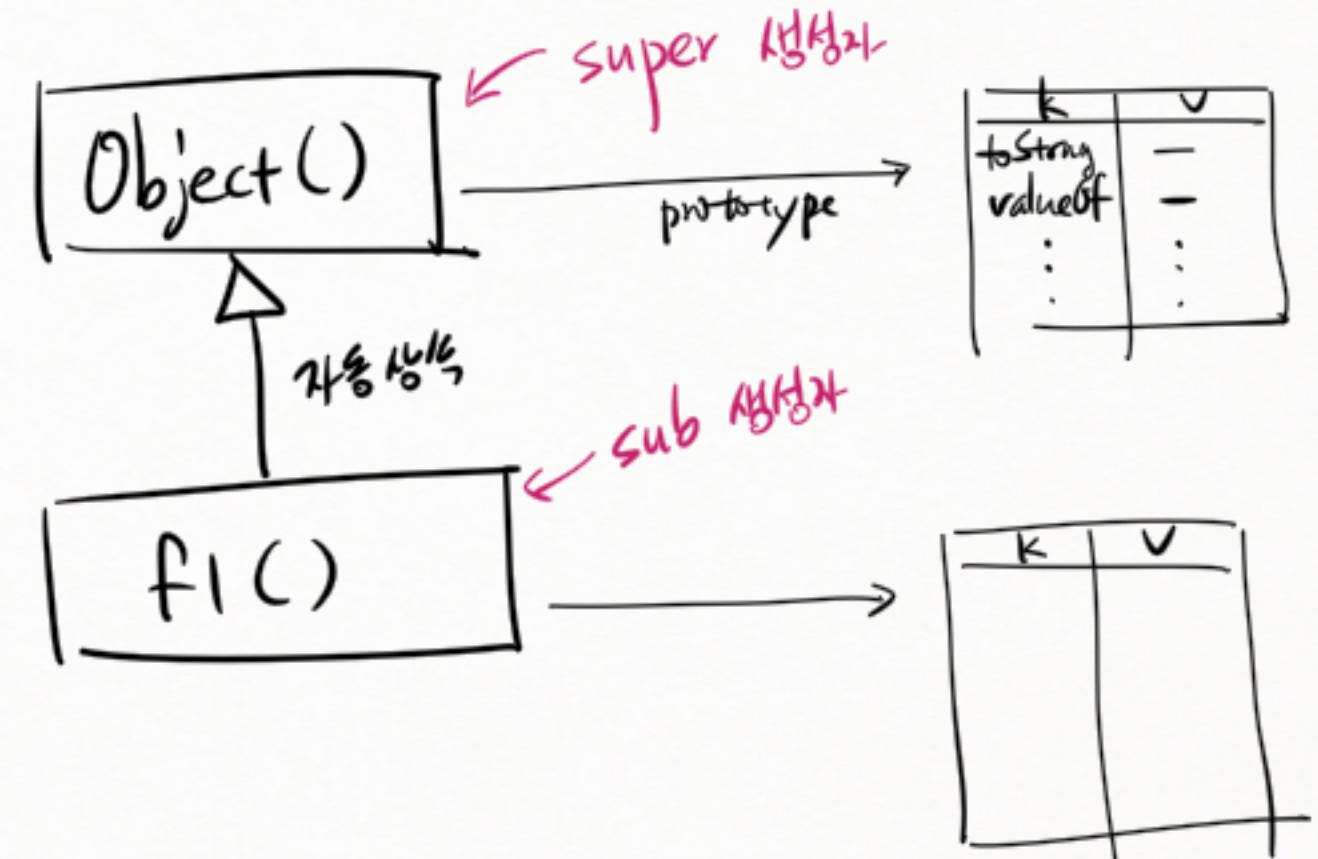
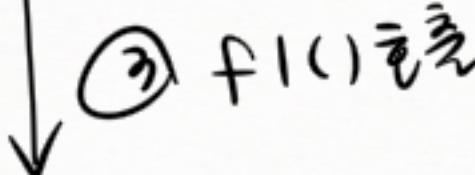
var obj = f();
 ↑
 undefined

* 생성자와 Object()

`var obj = new f1();`



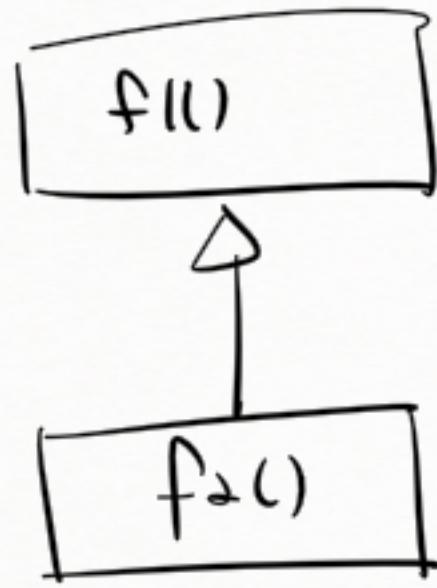
- ① new: 빈 객체 생성
- ② f1()의 super 생성자 호출
↳ Object()



`obj.toString()`

- ① 기본적인 출력.
- ② 생성자 `f1.prototype`에서 출력.
- ③ `super` 생성자 `Object.prototype`에서 출력.

* 생성자를 상속하기



```

f1(n) {
    this.name = n;
}
  
```

f2(n, k, e, m) ?

f1(n); ← 일반 함수 호출방법으로는 f2()가 this 라는

f1.call(this, n);

this.kor = k;

this.eng = e;

this.math = m;

}

변수에 받았을 때까지 주소를 f1()에 전달하고 그 다음에 f2()가 this 라는

var obj =

new f2("홍길동", 100, 90, 80);

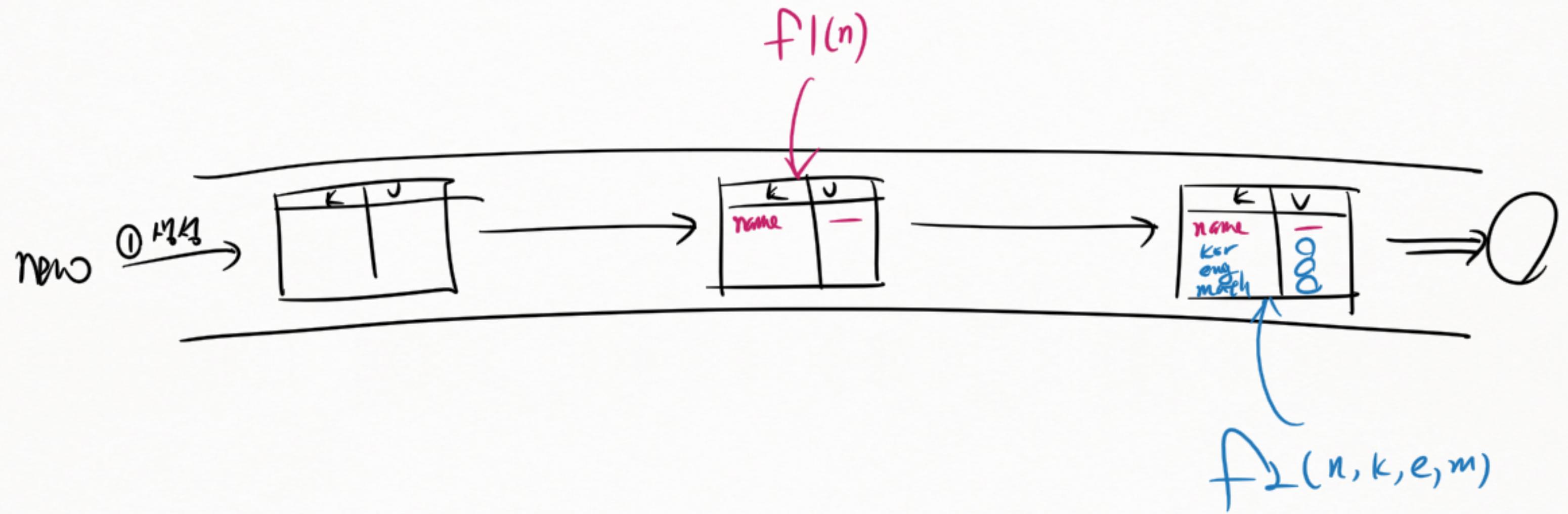
<u>200</u>	<u>K</u>	<u>V</u>

* 함수를 호출하는 방법

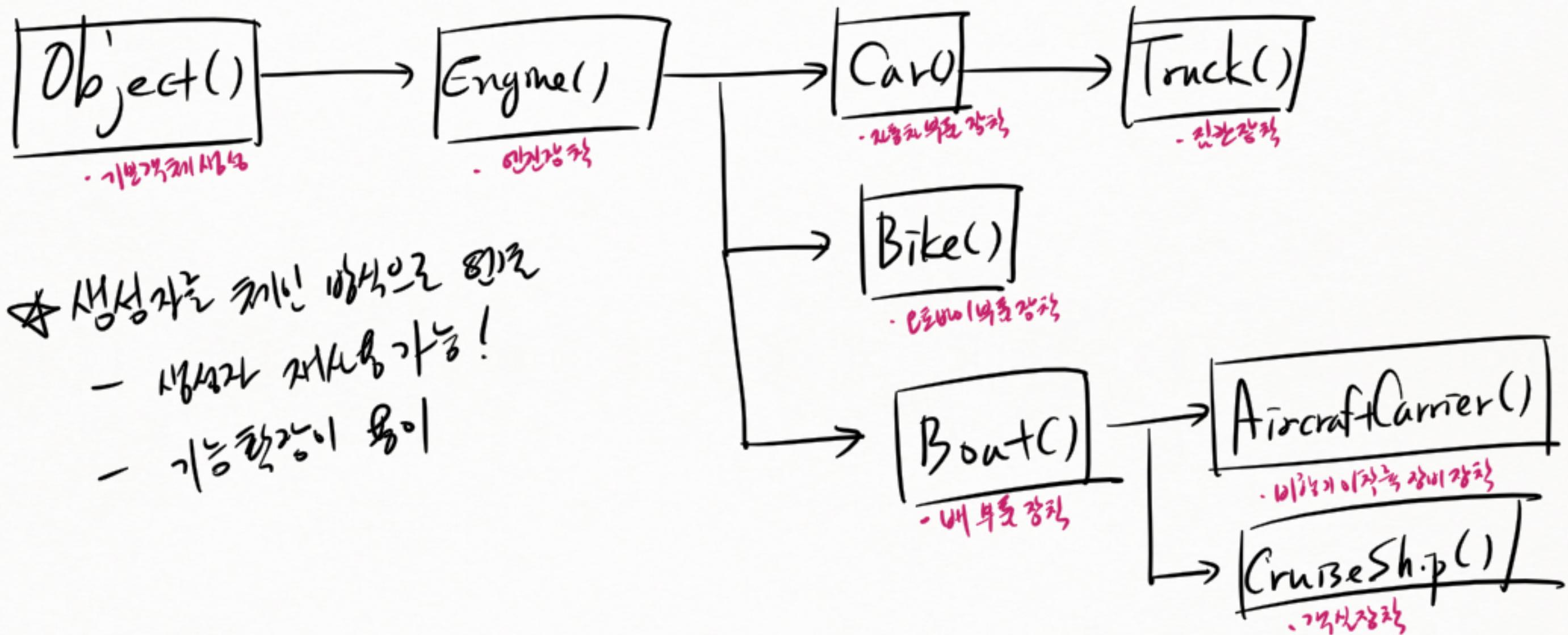
① 함수명(아구먼트, ...);
예) f1();

② 함수명.call(개체주소, 아구먼트, ...);
예) f1.call();

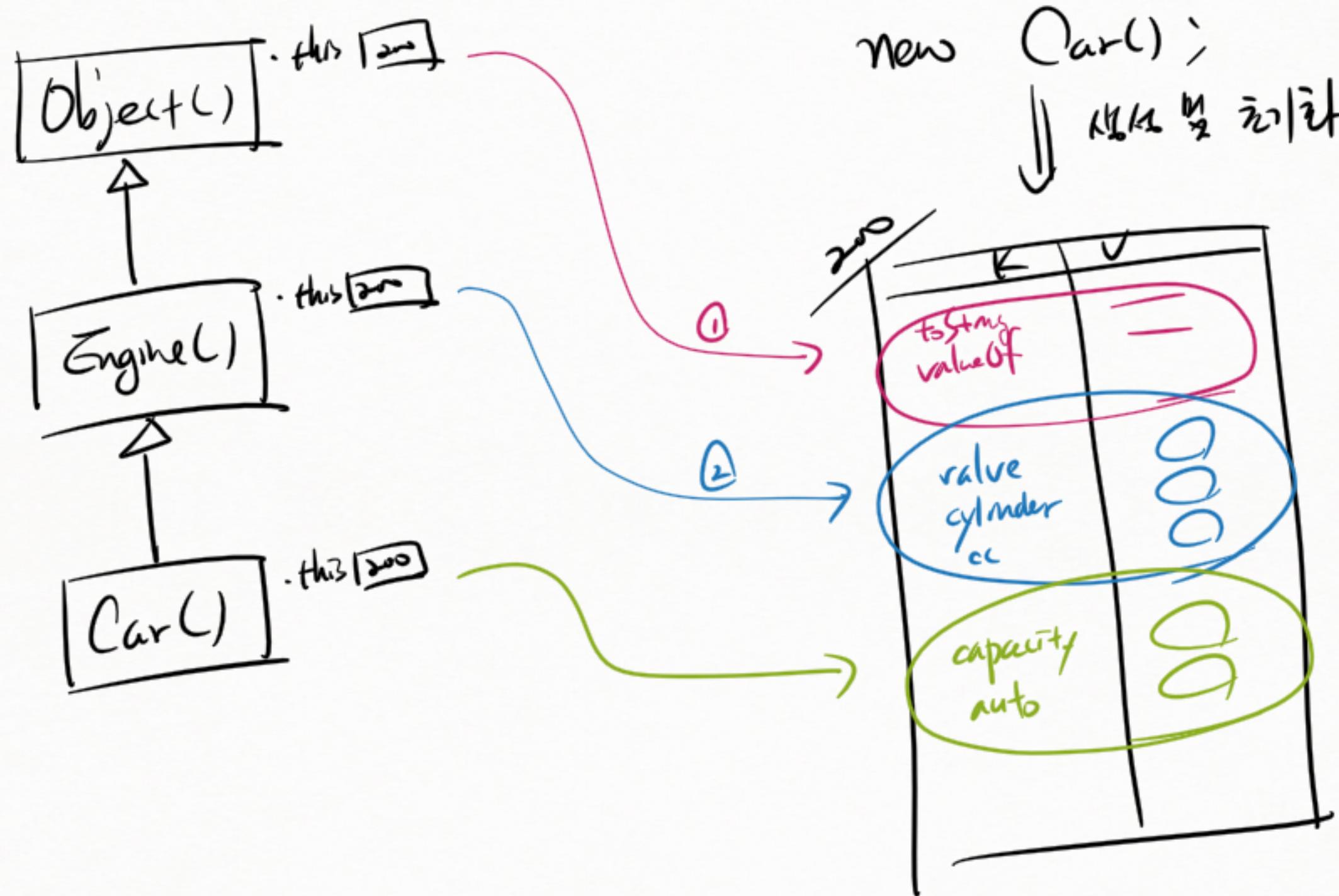
* 생성자와 전파하여 배운



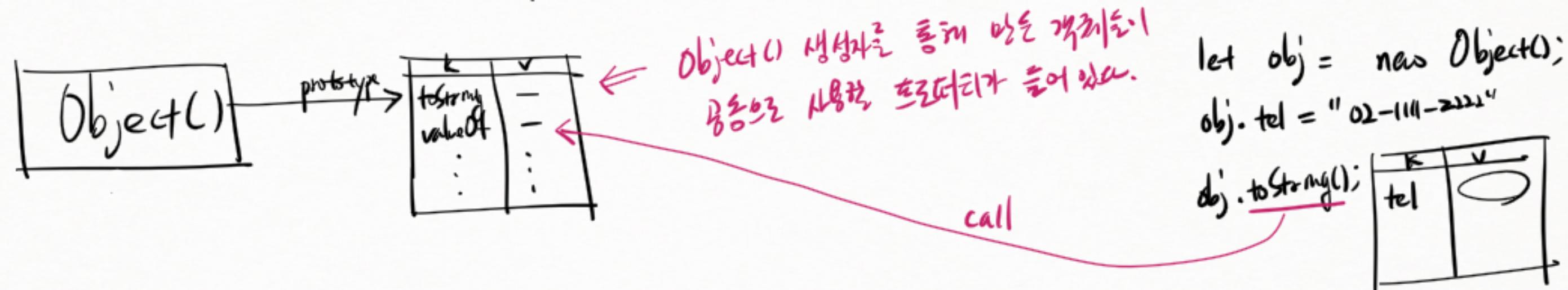
* 생성자를 차인으로 초기화되어 super-sub로 만드는 이유?



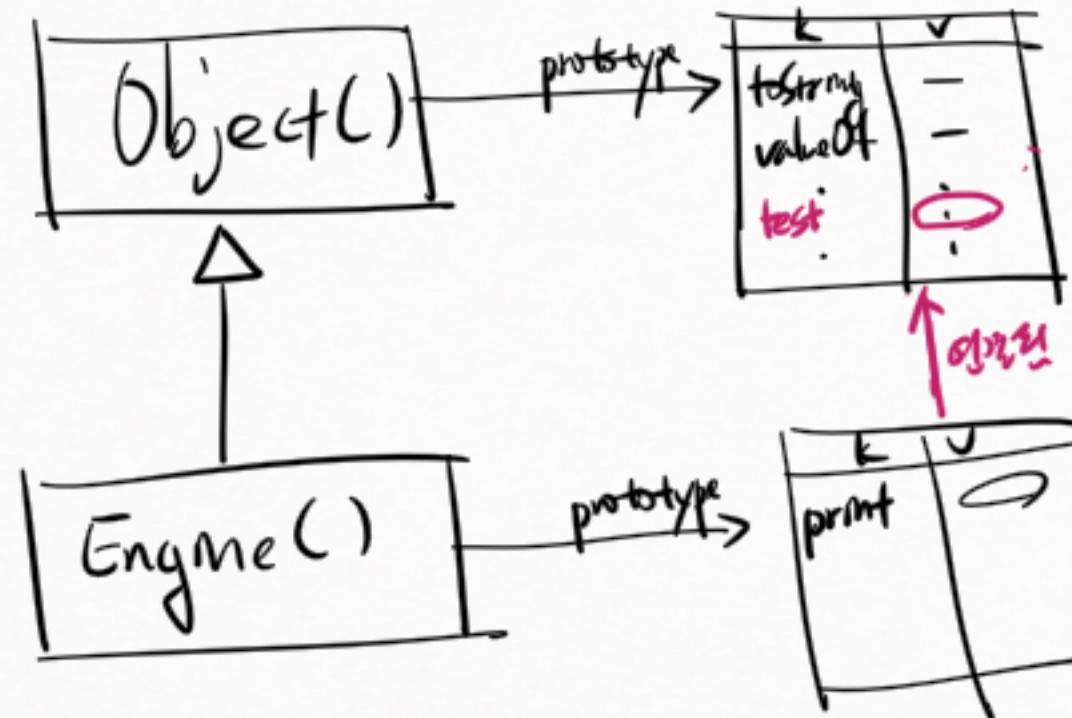
* 생성자 체인 예:



* 자식 생성자의 prototype 사용하기



* 자식 생성자의 prototype 사용하기



Engine() 생성자는 Object의
기반으로 만들어져서 Object의
프로토 타입을 상속받는다.

`let el = new Engine(...)`

K	V
value	---
cylinder	---
cc	---

`el. test();`

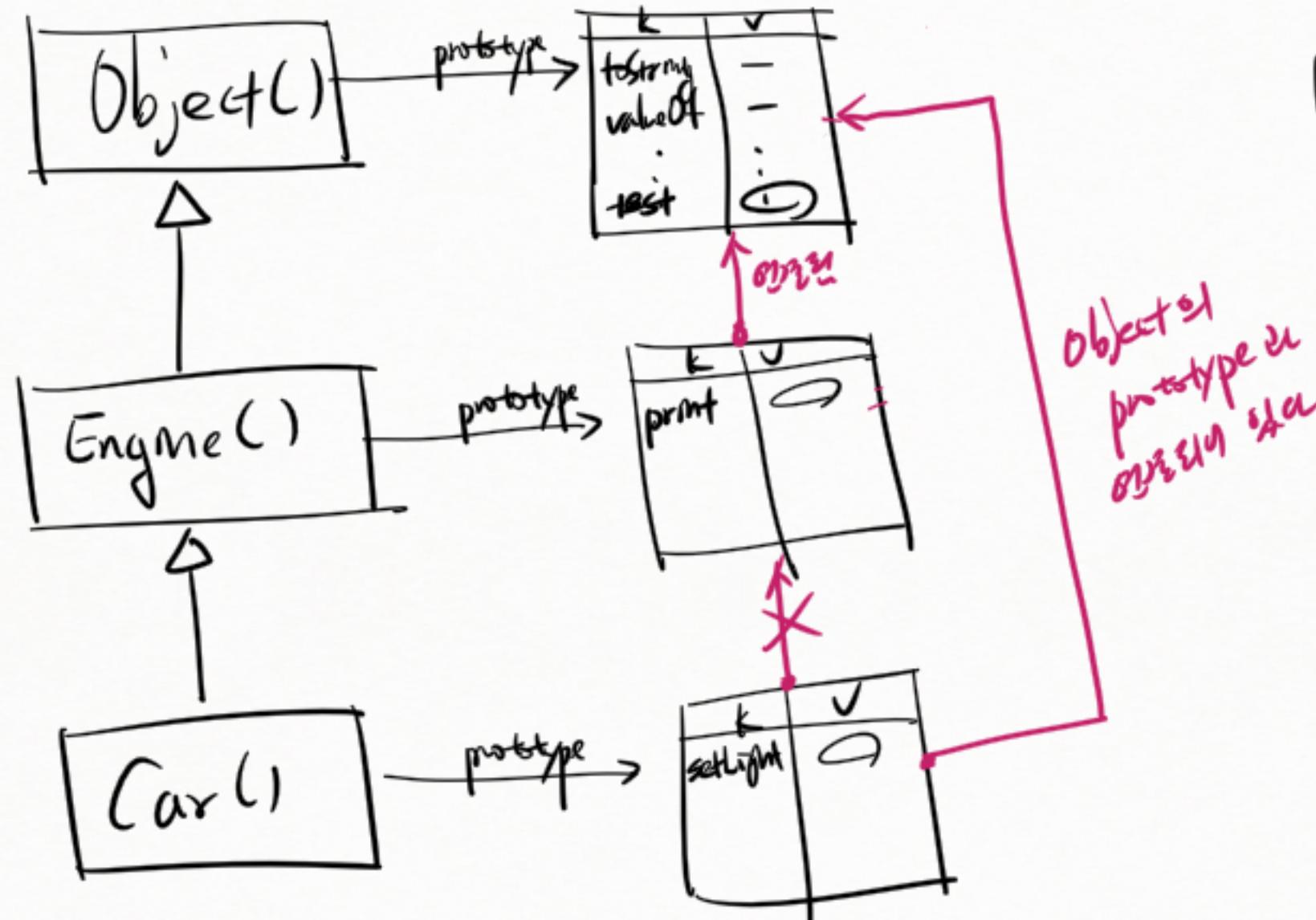
↳ Object.prototype.test()

`el. print();`

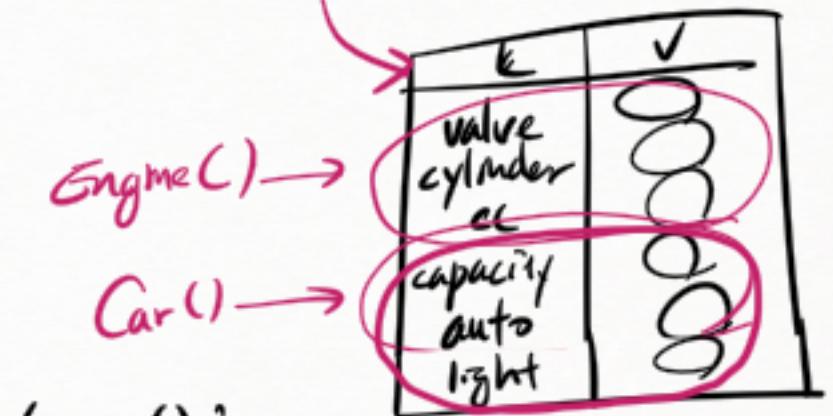
↳ Engine.prototype.print()

- ① 엔진 생성자인 `test()`는 없지만
- ② 엔진의 prototype에 있으므로
- ③ 상위 생성자의 prototype에 있으므로.

* 자식 생성자의 prototype 활용



`let c1 = new Car(...);`



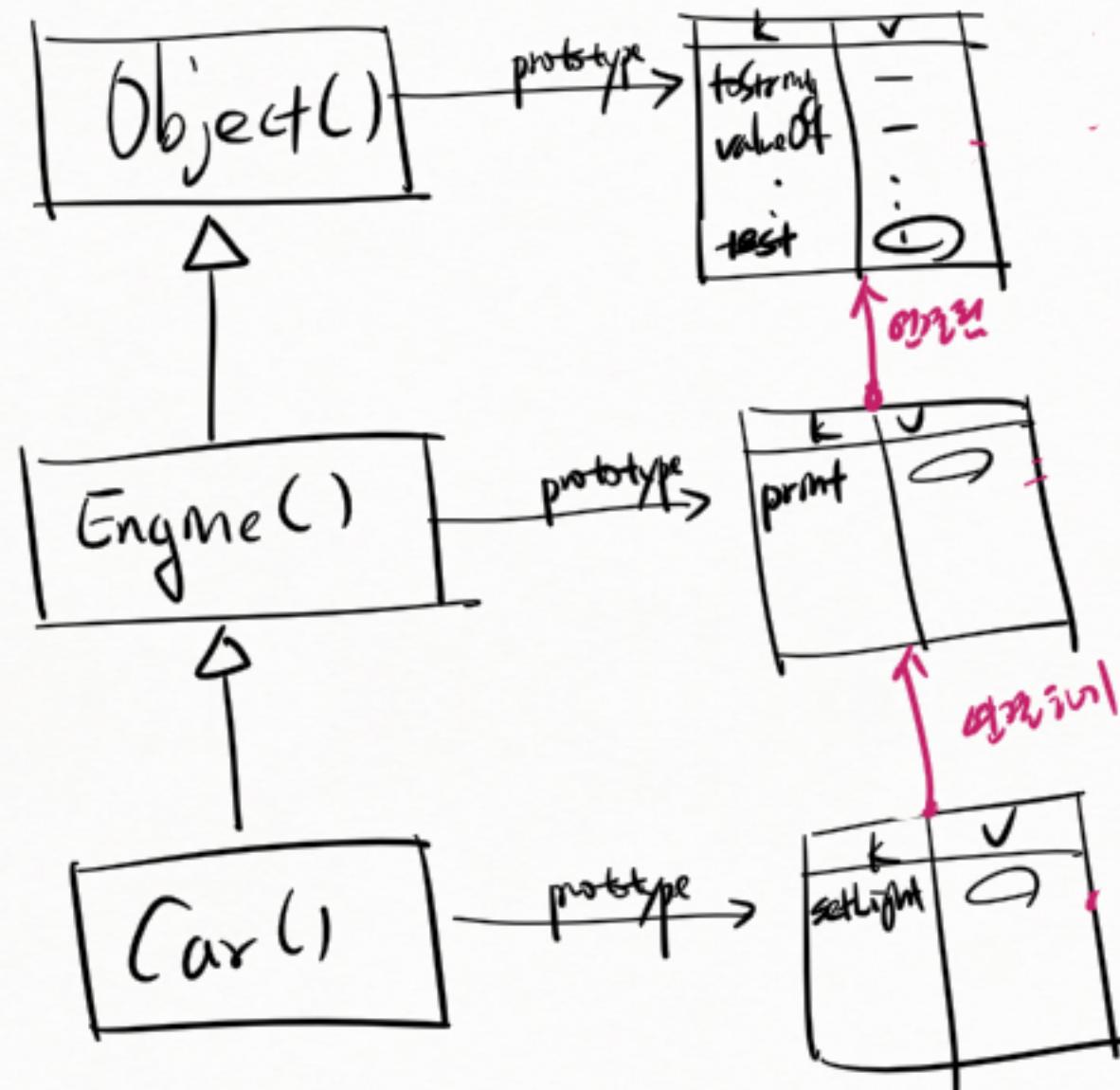
`c1.toString();`
 ↳ Object.prototype.toString()

`c1.setLight();`
 ↳ Car.prototype.setLight()

~~`c1.print();`~~
 ↳ Engine.prototype.print()

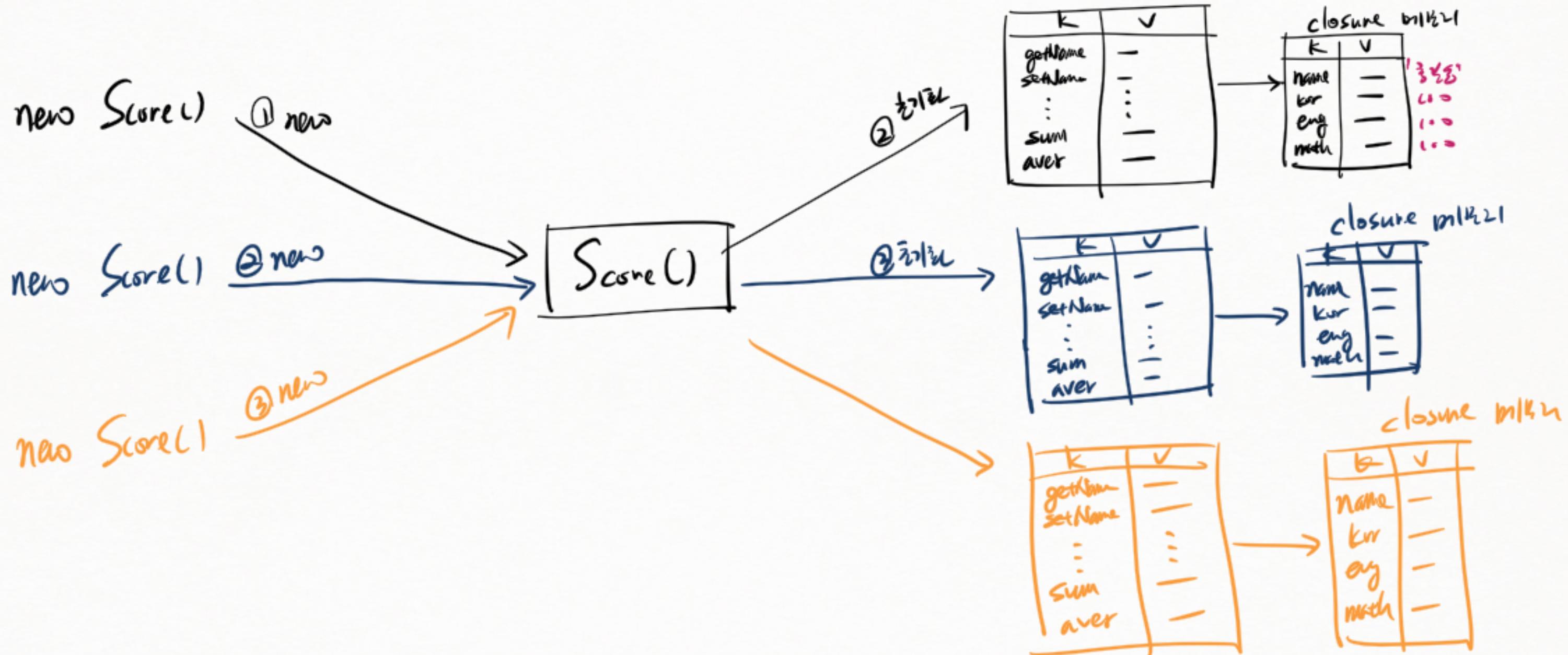
Car() 생성자 허무| 허무!
 Engine.prototype이 연결되어 있지 않아

* 자바스크립트의 prototype 개념



Object.setPrototypeOf(
Car.prototype,
Engine.prototype);
Object.setPrototypeOf(
Engine.prototype,
Object.prototype);

* closure et local var obj



* 함수 프로퍼티와 prototype 프로퍼티

★ 특정 객체에 대해 작업하지 않고 함수를 끌어올 때는

객체에 바로 저장한다.

(값수객체)

random()

min()

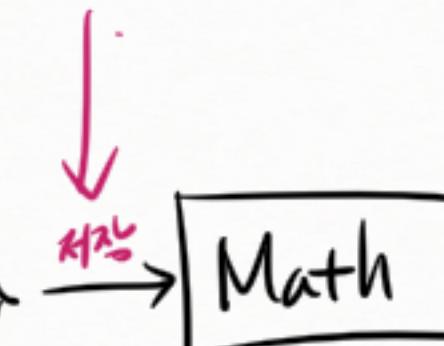
max()

sin()

round()

tan()

:



↓ 사용법

Math.random()

Math.min()

:

↑
값수객체

값수객체

★ 특정 객체의 값을 사용해서

작업을 수행하는 함수를 끌어올 때는

생성자의 prototype에 저장한다.



↓ 사용법

S1. sum()

S2. sum()

:

↑
값수객체

값수객체
기
대상값객체에 접근
하지 않는다

* 함수의 속성

① 기본적인 접근 허용

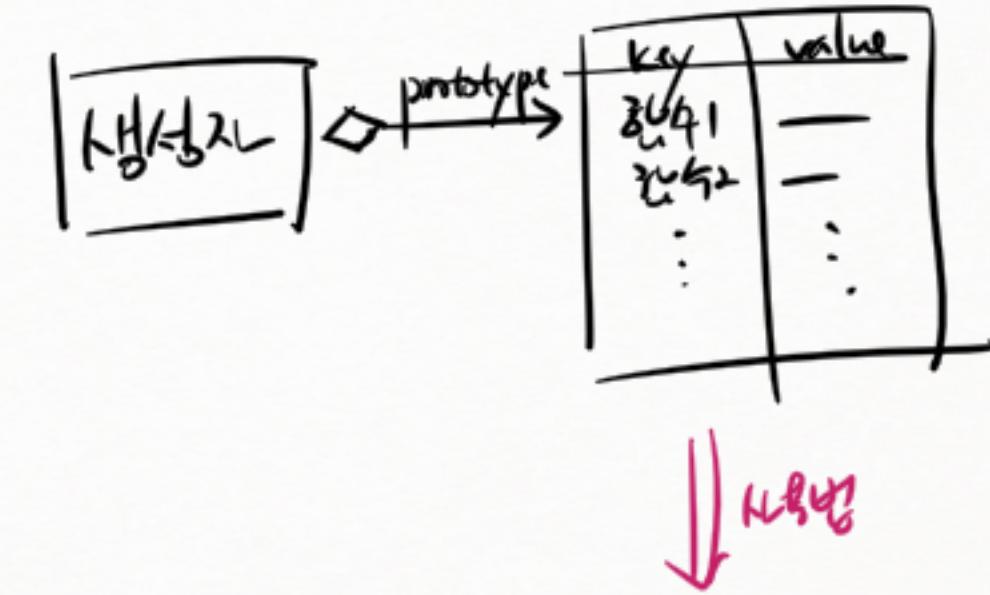
~~var m1 = new Math();
m1.random();~~

key	value
값1	-
값2	-
:	:

기본.값1();

값2.값3();
 ①) Math.random();
JSON.parse();

② 생성자의 prototype 속성



var obj = new 생성자();
obj.값1();

값2.값3();
 ↗
 내부링을 이용해 생성자를 찾는다.

②) s1.sum() str1.split(",")
s2.sum() str2.split(",")
s3.sum() arr1.forEach()

* 생성자와 인스턴스 (instance)

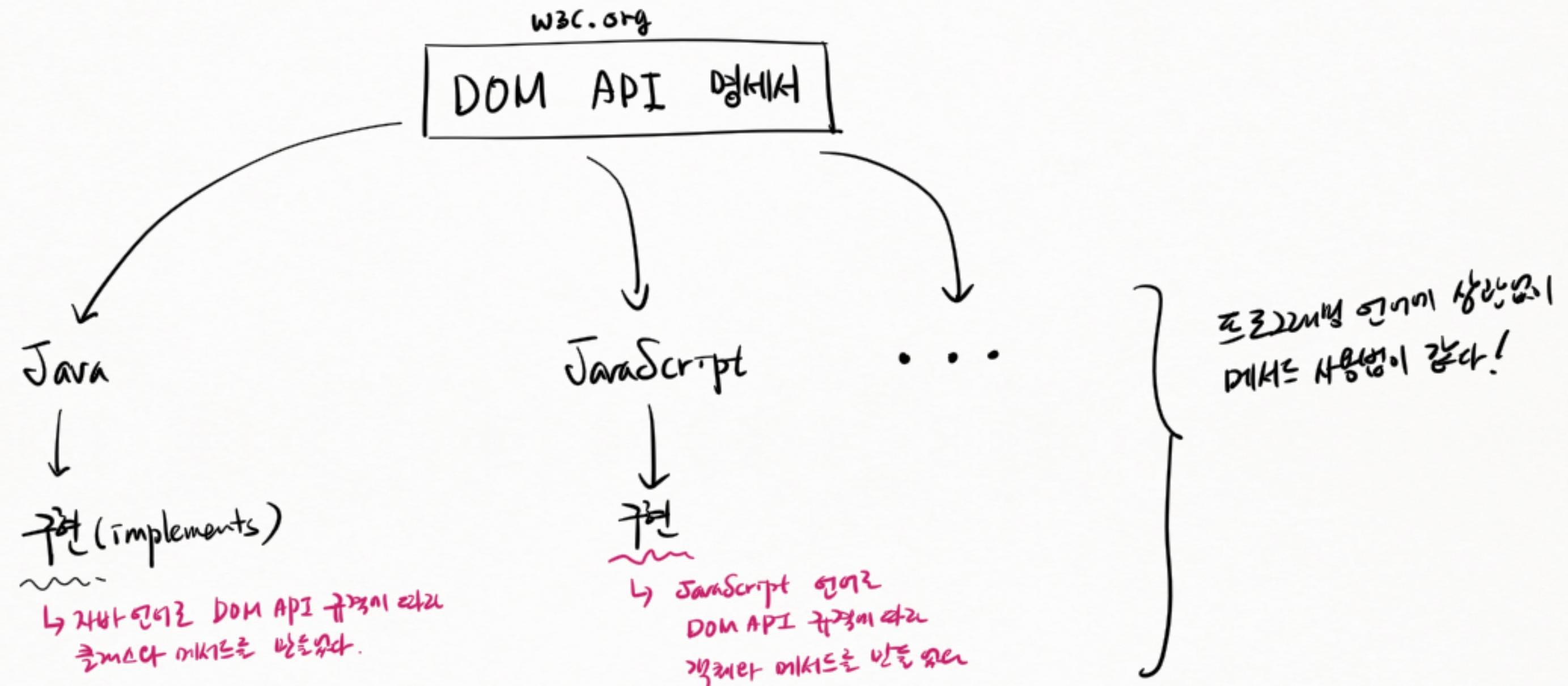
↳ 실제적인 예



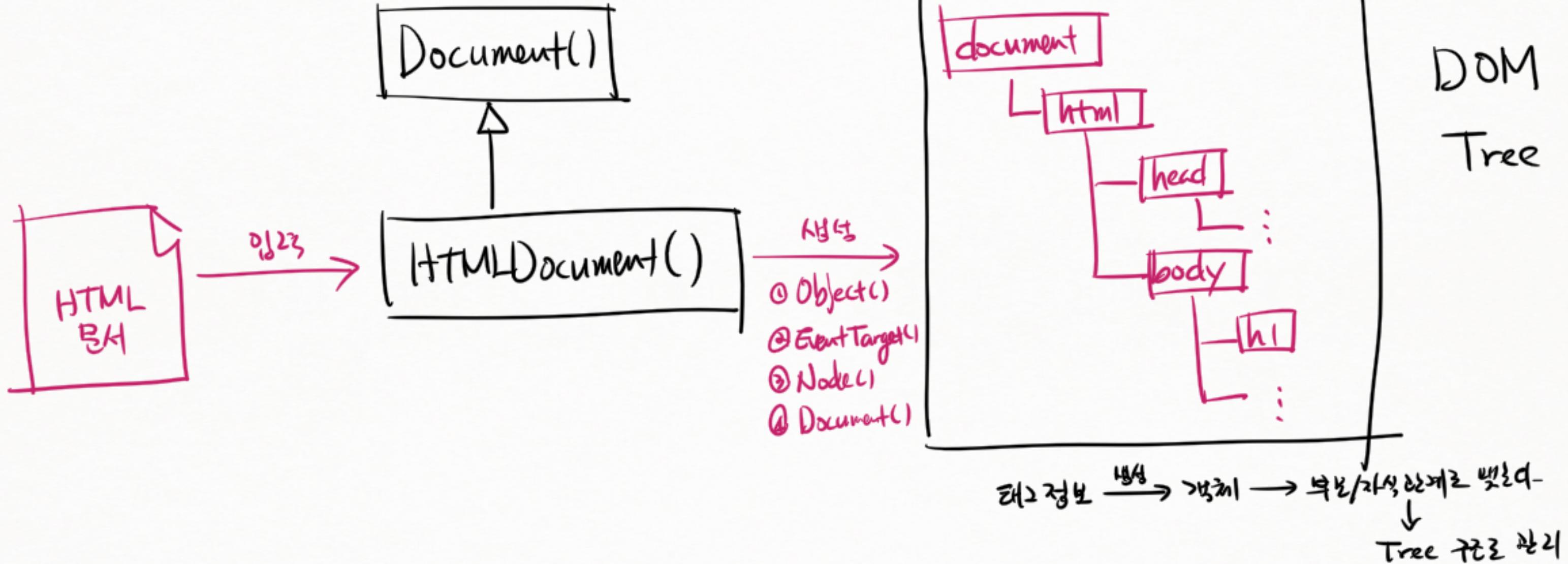
"`Score`의 인스턴스"
↓
`Score()` 생성자를 통한
기획된 객체
★ 자바
"Score 클래스(클래스)이
생성된 객체"

DOM API

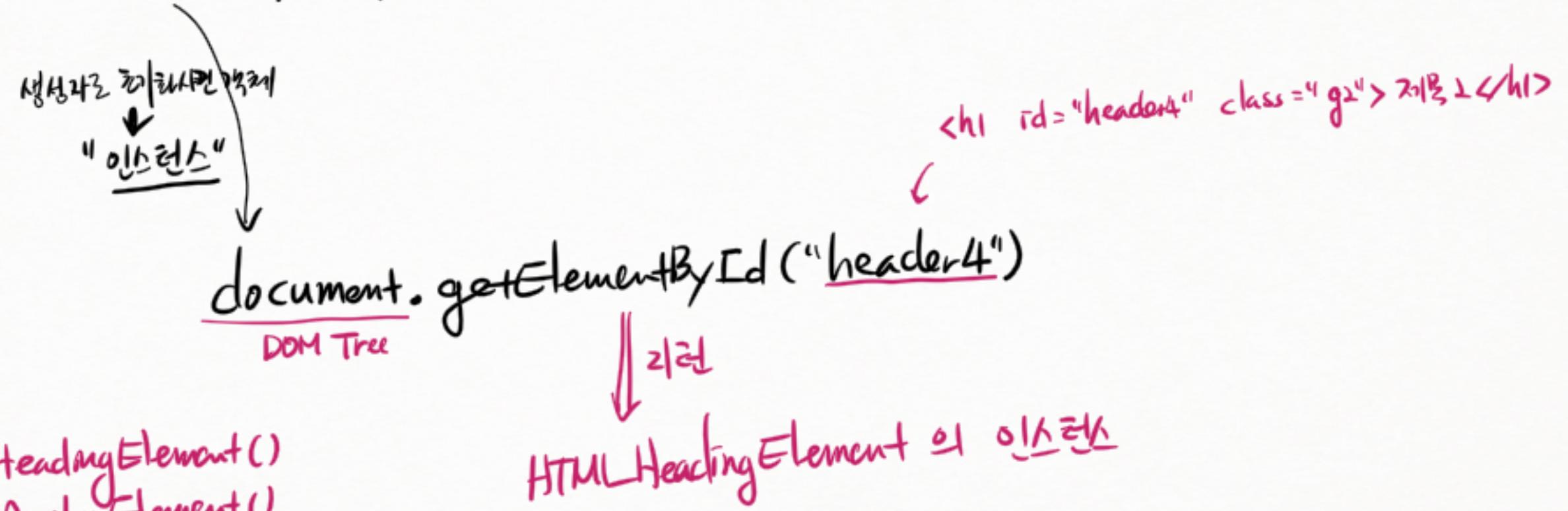
* DOM API



* HTMLDocument (Document()) et document



* Document.prototype.getElementById()



`<h1>` → `HTMLHeadingElement()`

`<a>` → `HTMLAnchorElement()`

`<p>` → `HTMLParagraphElement()`

:

:

* HTMLHeadingElement() 생성자 초기화의 과정

