Codingapple

Object는 {key: value} 형태로 자료를 저장하기위해 씁니다.

자료출력 : object.key1

자료입력 : object.key1 = ‘hello’

자료(key)제거 : delete object.key1

자료 iterate : for (var keys in object) { console.log(keys) }

단순한 string 자료 복사는 이렇게 합니다.

```
var name1 = 'Kim' ;
var name2 = name1 ; //복사
name2 = 'Park' ; //변경
```

name1의 값 : Kim

name2의 값 : Park

하지만 Object 자료를 복사하면 이상한 일이 일어남.

```
var object1 = {a : 1};
var object2 = object1 ; //복사
object2.a = 2 ;
```

object1 = {a : 2}
object2 = {a : 2}

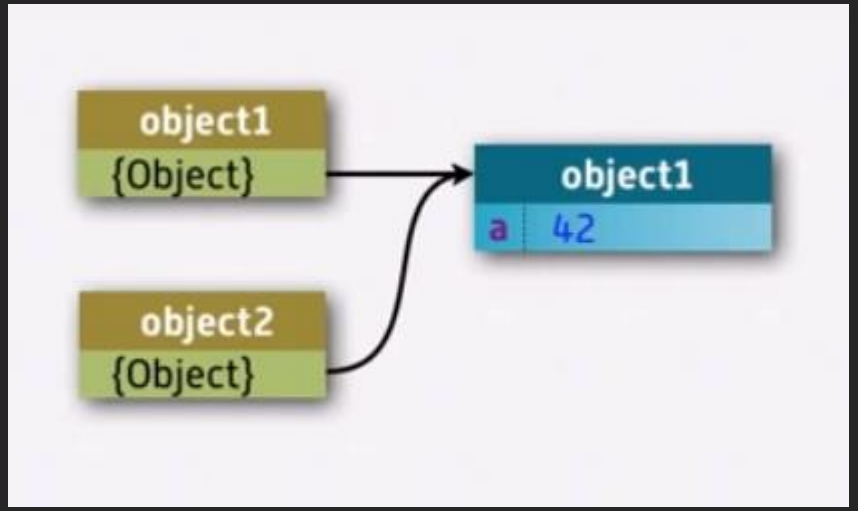
} 값이 둘다 바뀜 (충격적)

```
var object1 = {a : 1};
var object2 = object1 ; //복사
object2.b = 2 ;
```

object1 = {a : 1, b : 2}

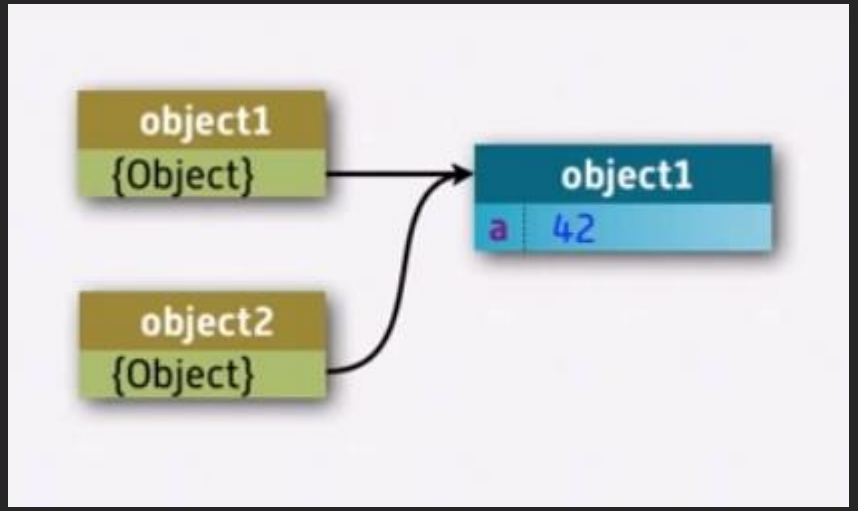
object2 = {a : 1, b : 2}

object, array 자료형은 숫자를 직접 저장하는게 아니라
숫자가 저장된 곳(메모리)을 알려주는 "화살표"를 저장함.
(그니까 복사하면 화살표가 복사됨)



함수도 Object라서

함수를 그대로 복사하면 같은 현상이 일어남



Primitive data type : 텍스트, 숫자, Boolean 등; 변수에 직접 저장됨

Reference data type : object, array 등; 화살표로 reference해서 사용

참고 :

Reference 자료형끼리 ==, === 비교시 화살표를 비교함.

(같은 화살표를 가지고 있어야 같다고 나옴)

why use object
http://javascriptissexy.com/oop-in-javascript-what-you-need-to-know/

자바스크립트에서도
객체지향프로그래밍을 하기 위해 Class를 만들어 사용합니다.

{}
{}
{}

Object{ }는
문자, 숫자같은 자료를 담아둘 수도 있고,
기능을 정의하는 function() 함수를 담을 수도 있다.

"Class는 비슷한 Object{ }를 반복적으로 많이 만들어내고 싶을 때 사용함"

어떨 때 쓰는지 예시로 알아봅시다.

Overwatch 게임
캐릭터 소개 페이지.html

다들 제각각의 특성을 가진 영웅이
30여명 정도 있습니다.

영웅마다 각각 특수능력이 2~3개 있습니다.

Shift

E

Q

‘안녕~’ 같은 인사기능도 있습니다.

Overwatch 영웅들을 object 자료형으로 정리해봅시다.

필요한 정보들 :

- Q 능력
- E 능력
- sayHi() 기능

목표 :
영웅이름.q 라고 적으면
Q에 저장된 스킬이름이 나와야함.

Overwatch 영웅을 object로 짜봅시다.

```
var tracer = {
        q : 'Pulsebomb',
        e : 'Recall',
        sayHi : function(){
                console.log('Hello')
        }
}
```

```
var hanzo = {
        q : 'DragonStrike',
        e : 'ScatterArrow',
        sayHi : function(){
                console.log('Hello')
        }
}
```

Overwatch 영웅 능력을 호출하려면

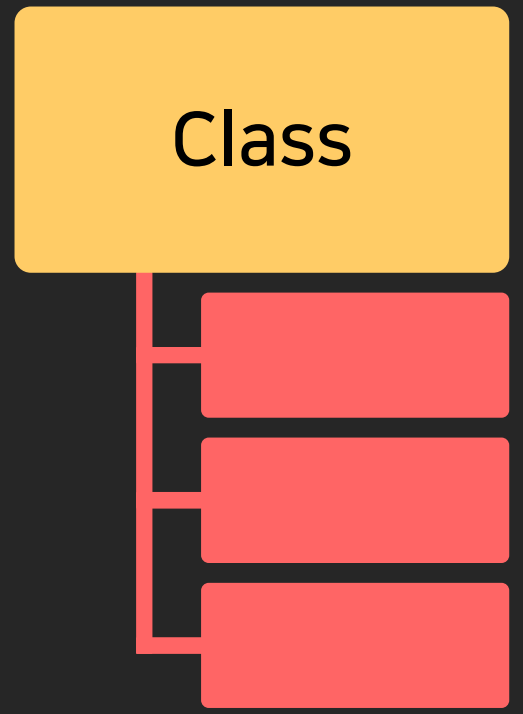
```
tracer.q;
tracer.sayHi();
```

```
hanzo.q;
hanzo.sayHi();
```

근데 보니까 영웅이 30개 가까이 되는거임..
object를 30개를 언제 다 만들지?

"Class는 비슷한 Object를 반복적으로 많이 만들어내고 싶을 때 사용함"

Class는 object를 반복적으로 찍어내는 기계를 담고 있습니다.
그 기계를 다른 말로 생성자(constructor)라고 하는데
생성자를 한번 만들어봅시다.



그래서 영웅 생성기를 만들었습니다.

```
function Hero( ){
    this.q = '스킬1' ;
    this.e = '스킬2' ;
    this.sayHi = function(){
        console.log( 'Hello' );
    }
}
```

- this라는 키워드는 새로 생성될 object를 말합니다.
(교양있는 말로 instance)

그럼 생성기로부터 새로운 object를 계속 뽑아봅시다.

```
function Hero( ){
    this.q = '스킬1';
    this.e = '스킬2';
    this.sayHi = function(){
        console.log( 'Hello' );
    }
}

var tracer = new Hero( );
var hanzo = new Hero( );
```

이제 새로운 영웅들은
q, e, sayHi() 기능을 가진
object가 되었습니다.

새로운 영웅들에게 각각 다른 스킬을 부여하려면? 함수의 인자!

```
function Hero(q, e){
    this.q = q;
    this.e = e;
    this.sayHi = function(){
        console.log(' says Hi' );
    }
}

var tracer = new Hero( 'PulseBomb' , 'Recall' );
var hanzo = new Hero( 'DragonStrike' , 'ScatterArrow' );
```

object 하드코딩으로 만드는 것보다 훨씬 코드가 짧음

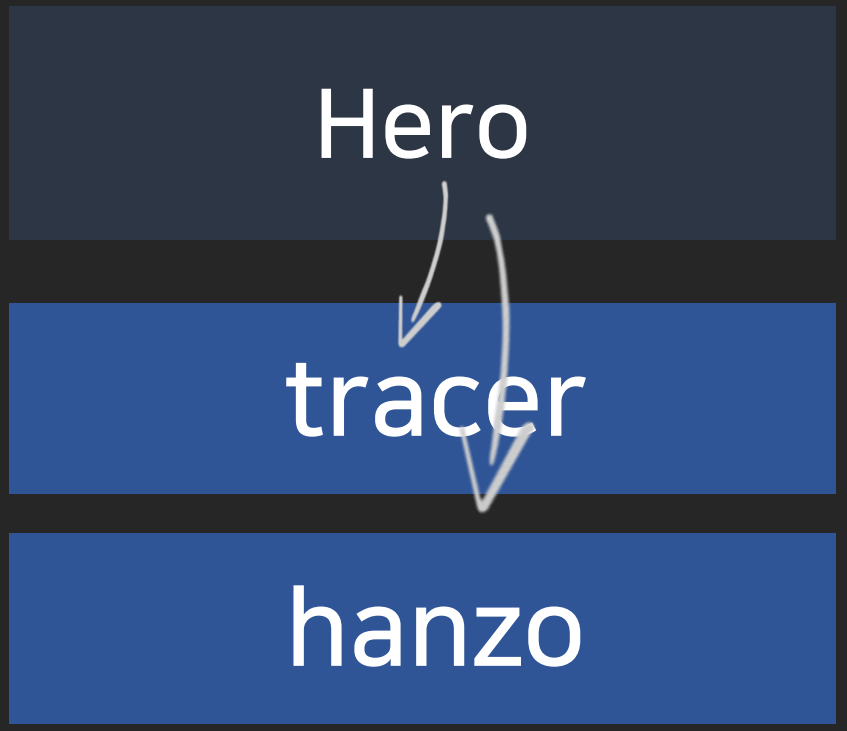
```
var tracer = {
        q : 'Pulsebomb',
        e : 'Recall',
        sayHi : function(){
                console.log('Hello')
        }
}
```

```
var hanzo = {
        q : 'DragonStrike',
        e : 'ScatterArrow',
        sayHi : function(){
                console.log('Hello')
        }
}
```

vs

```
var tracer = new Hero( 'PulseBomb' , 'Recall' );
var hanzo = new Hero( 'DragonStrike' , 'ScatterArrow' );
```

부모 & 자식관계

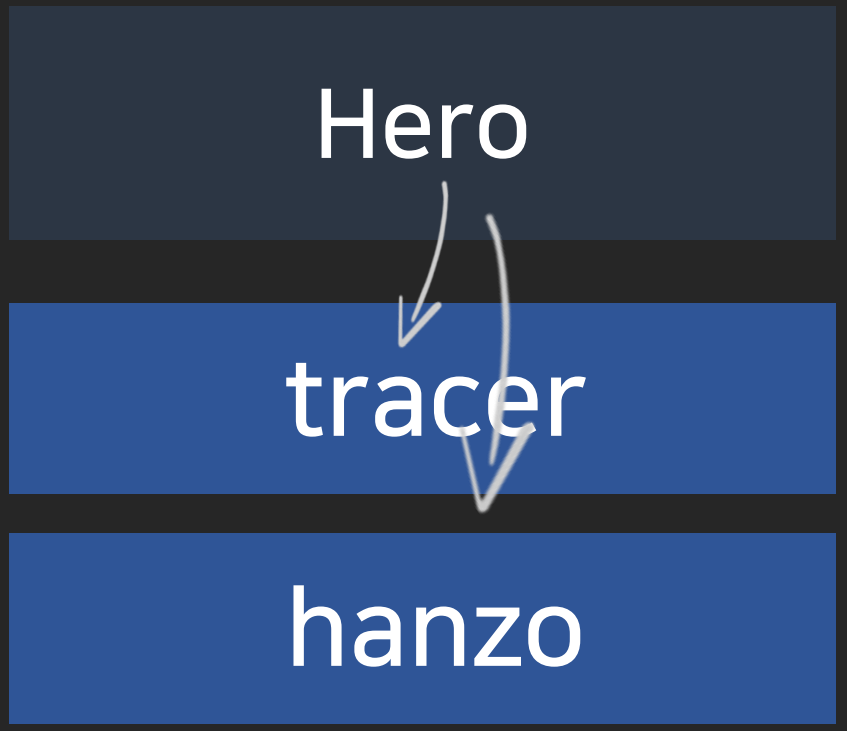


q,e,sayHi()

q,e,sayHi() 등의
속성을 물려받음

얘도 마찬가지

자손들에게 새로운 r 스킬을 추가하려면?

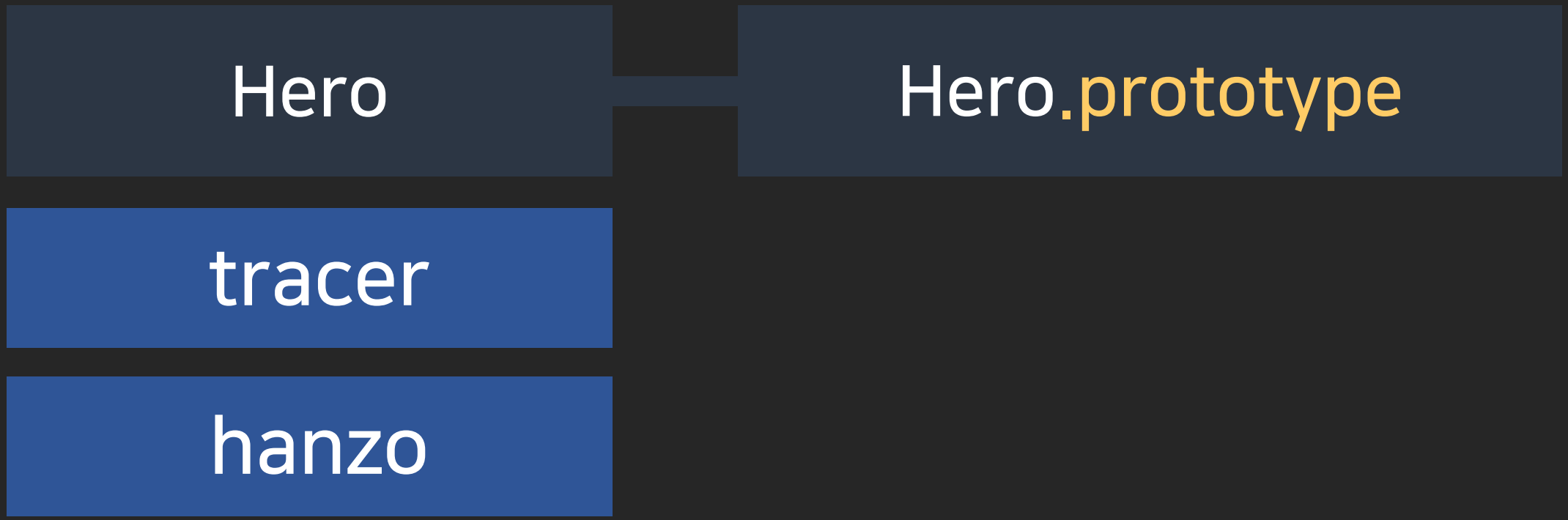


q,e,sayHi()

q,e,sayHi() 등의
속성을 물려받음

얘도 마찬가지

실은 function을 써서 class를 만들면
Hero.prototype 속성도 몰래 하나가 생성됩니다.



부모

Hero

tracer

hanzo

유전자

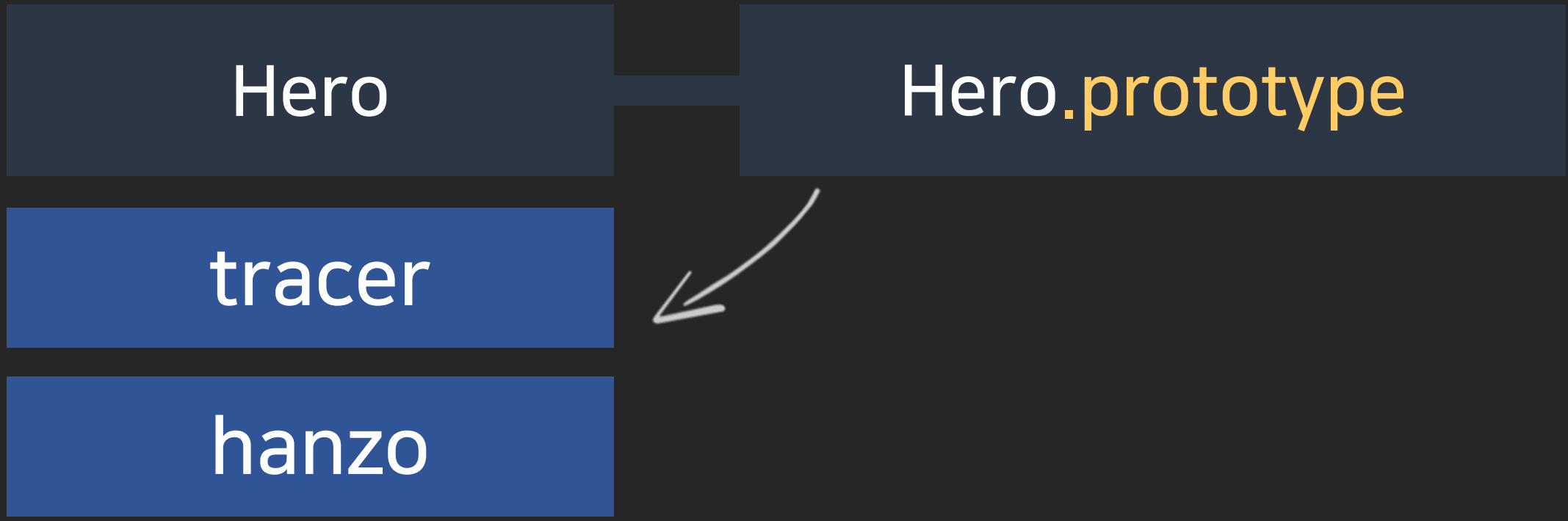
Hero.prototype

자식들이 부모 속성을 물려받을 수 있는
두번째 장치가
Hero.prototype

prototype :

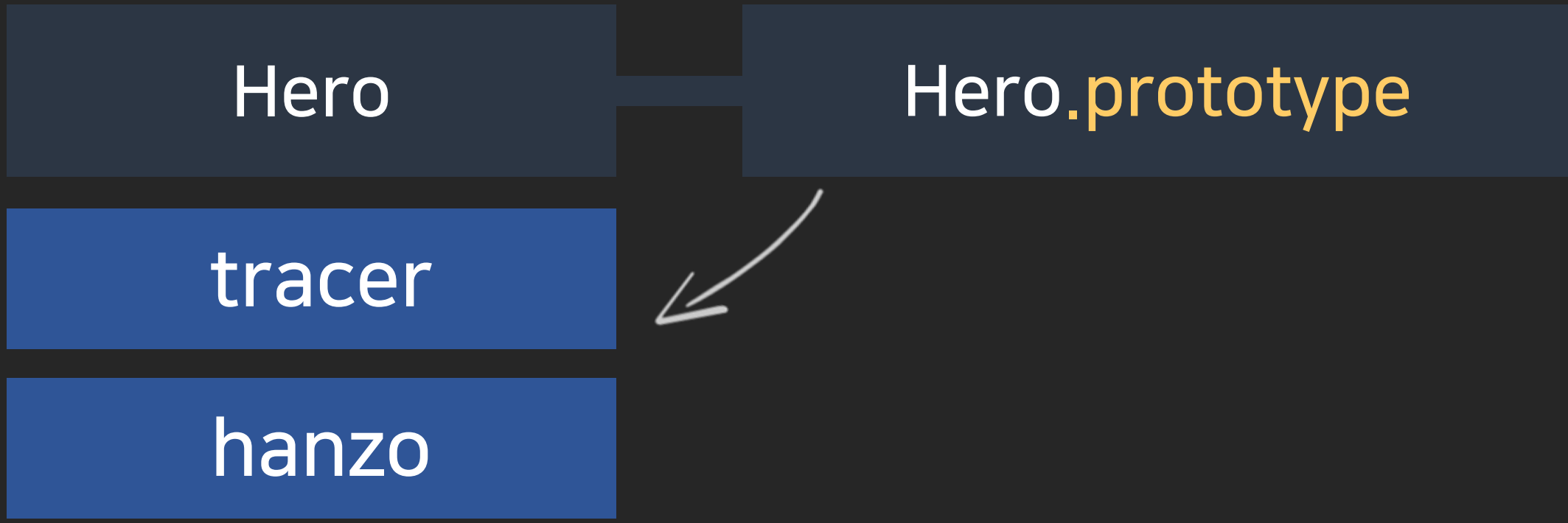
다른 객체지향 언어들처럼
'상속'기능을 구현하기 위해
만들어낸 장치

(Javascript가 원래 좀 이상함)



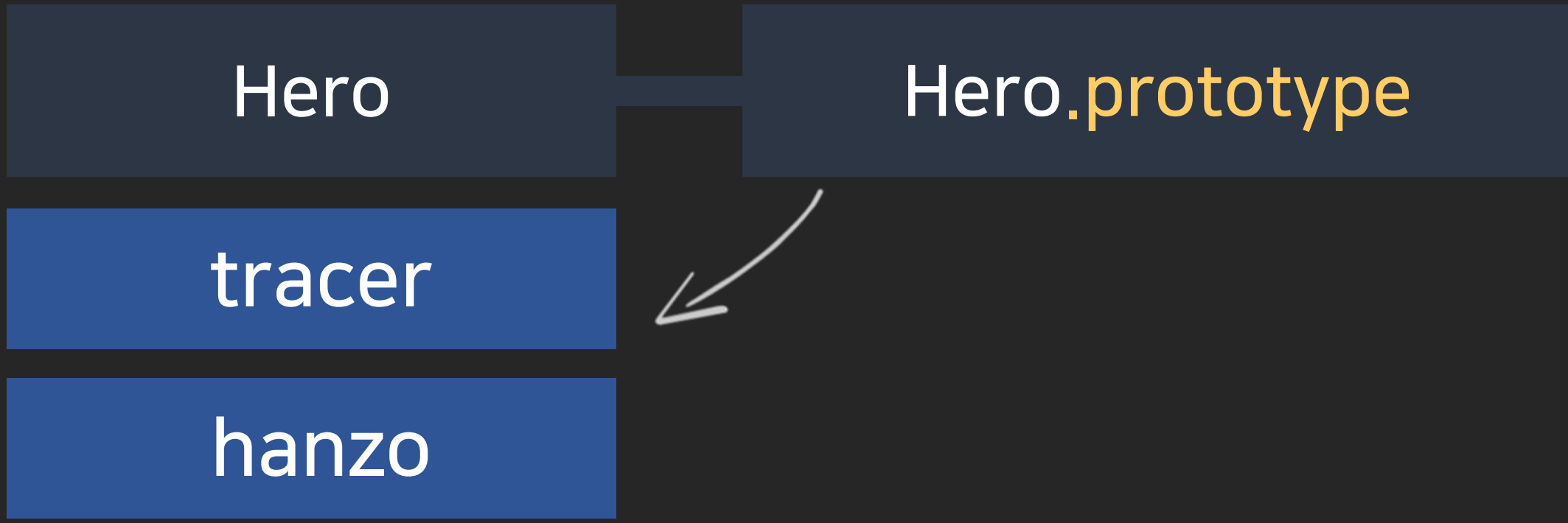
업데이트가 되어 영웅들이 r 스킬이 새로 생겼다고 합니다.
그러면 우리의 Hero 라는 생성기도 r 스킬을 추가해줘야겠죠?

Hero 클래스가 상속해줄 r 스킬을 추가하려면 어케하죠?
Hero.r = "스킬3" (?)



Hero 클래스가 상속해줄 새로운 능력인 r 스킬을 추가하려면:

Hero.prototype.r = '스킬3';
부모의 유전정보를 담은 prototype에다가 추가해서 사용해야합니다.



Hero.r = '어쩌구';

prototype이 아니라 원래 class 생성자에다가 추가하면
tracer와 hanzo는 r 스킬을 사용불가능합니다.
(Hero.prototype이 유전자니까요)

class를 만드는 이유1.

비슷한 Object를 반복적으로 많이 만들어내고 싶을 때

- 클래스로 Object를 무한히 찍어내고..
- Class를 상속해서 다른 클래스를 만들고..

class를 만드는 이유2.

기능(메소드)들을 묶어서 정리, 관리할 때

- 비슷한 기능끼리 묶고..
- 외부로부터 기능을 보호하고..

고양이와 강아지로 배우는 Encapsulation

class 고양이
- 발톱세우기()
- 야옹()

class 강아지
- 꼬리흔들기()
- 멍멍()

고양이 class로 부터 생성된 애들은
.멍멍() 하면 안됩니다.

마찬가지로
강아지들은 .야옹()하면 안됩니다.

Encapsulation : 기능들을 묶거나 외부로 부터 보호할 수 있다

```
function 강아지( ){
        this.짖기 = '왈왈';
}

var 쵸코 = new 강아지();

쵸코.짖기;   (가능!)
야옹이.짖기;   (불가능!)
```

짖기, 영역표시는
부모가 강아지인 것들만 할 수 있습니다.

고양이들은 하면 안돼요.

ES6 최신 문법으로 생성하는 class

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }

calcArea() {
    return this.height * this.width;
  }
}
```

Class를 이용하는 이유 두개 :

Inheritance – 상속으로 수많은 비슷한 Object를 만들어낼 수 있다.
Encapsulation – 비슷한 기능들을 한 곳에 묶어 보호/관리할 수 있다.