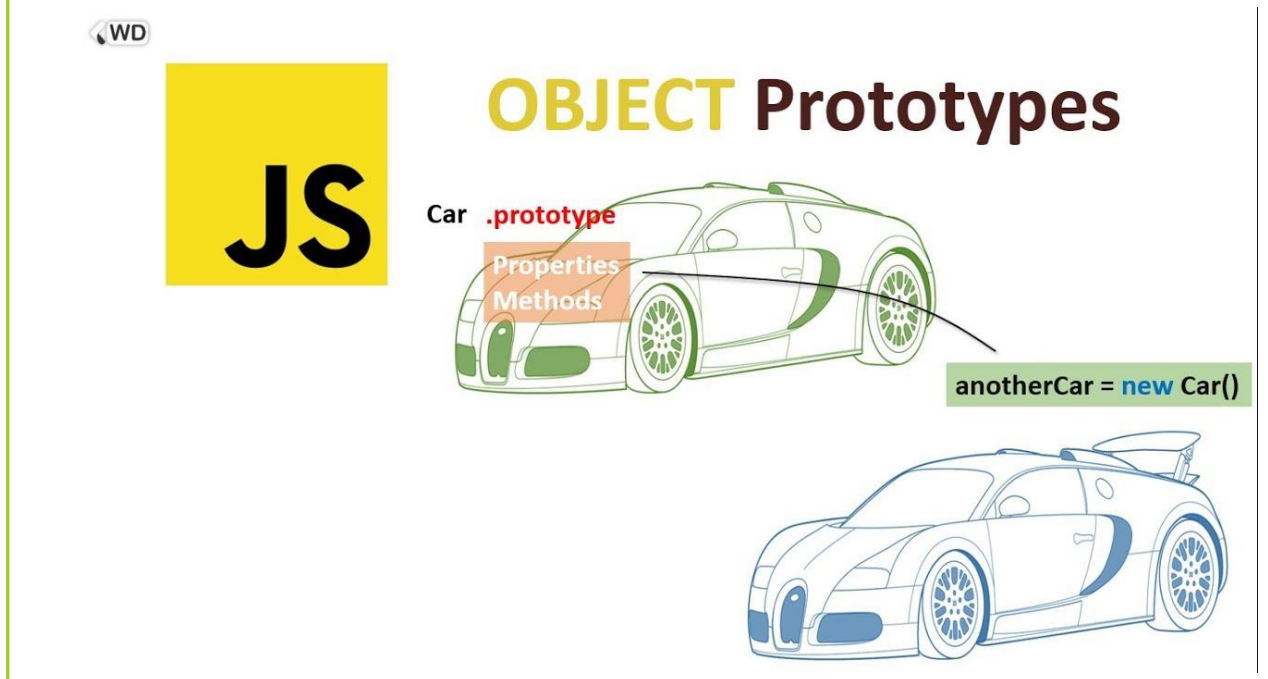
JavaScript

Prototype과 생성자 함수 – 김근형 강사

Prototype

Prototype



Prototype

- Prototype이란?
  - **이미 해당 참조형에 필요한 모든 것을 제공하는 프로퍼티**
  - Prototype은 그 객체의 성질을 정의하며 모든 참조형은 Prototype 을 가진다.
  - 일반적인 참조형 들은 내부에 Prototype으로 [__proto__] 라는 프로퍼티를 가지고 있다.
  - 기본형은 Prototype이 존재하지 않으며 모든 참조형의 Prototype은 최종적으로 Object를 가진다.

```
var a = 1;
console.dir(a);

var b = {
    aa : 1,
    bb : 'hello'
}
console.dir(b);
```

Prototype

- Prototype에 접근 방법
  - 보통 Prototype에 접근하기 위해서는 [instance].__proto__ 라는 명칭을 통해 접근할 수 있다.
  - 쓰거나 읽을 경우 위의 방법을 쓸 수는 있으나 읽을 시 굳이 "__proto__" 라는 프로퍼티를 통해서 접근할 필요가 없다.
  - 즉 "__proto__" 는 생략이 가능하며 이런 성질을 이용해 __proto__에 함수나 변수를 선언하여 마치 그 객체의 프로퍼티와 변수를 쓰는 것 처럼 사용이 가능하다.

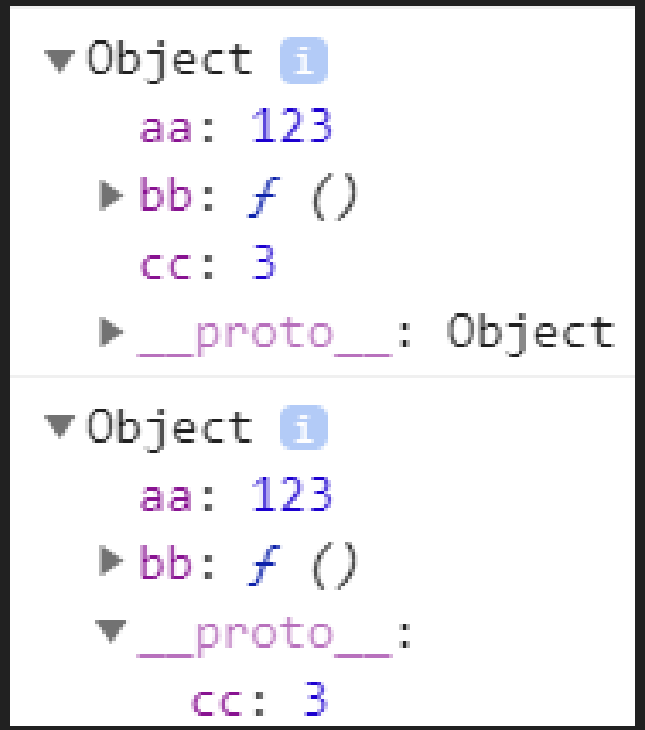
```
var a = {
    aa : 123,
    bb : 'hello'
};

a.__proto__.funcA = function(){
    console.log('funcA 의 기능');
};

a.__proto__.paramB = 'hi';

// __proto__ 를 넣어도 상관은 없지만 생략 가능

a.__proto__.funcA();
a.funcA();
console.log(a.__proto__.paramB);
console.log(a.paramB);
```

```
funcA 의 기능
funcA 의 기능
hi
hi
```

Prototype

- __proto__ 에 프로퍼티 생성 시 위치 비교

```
var a = {
    aa : 123,
    bb : function(){
        console.log('a 객체 내 bb function');
    }
}

a.cc = 3;

console.dir(a);

var b = {
    aa : 123,
    bb : function(){
        console.log('b 객체 내 bb function');
    }
}

b.__proto__cc = 3;
console.dir(b);
```



Prototype

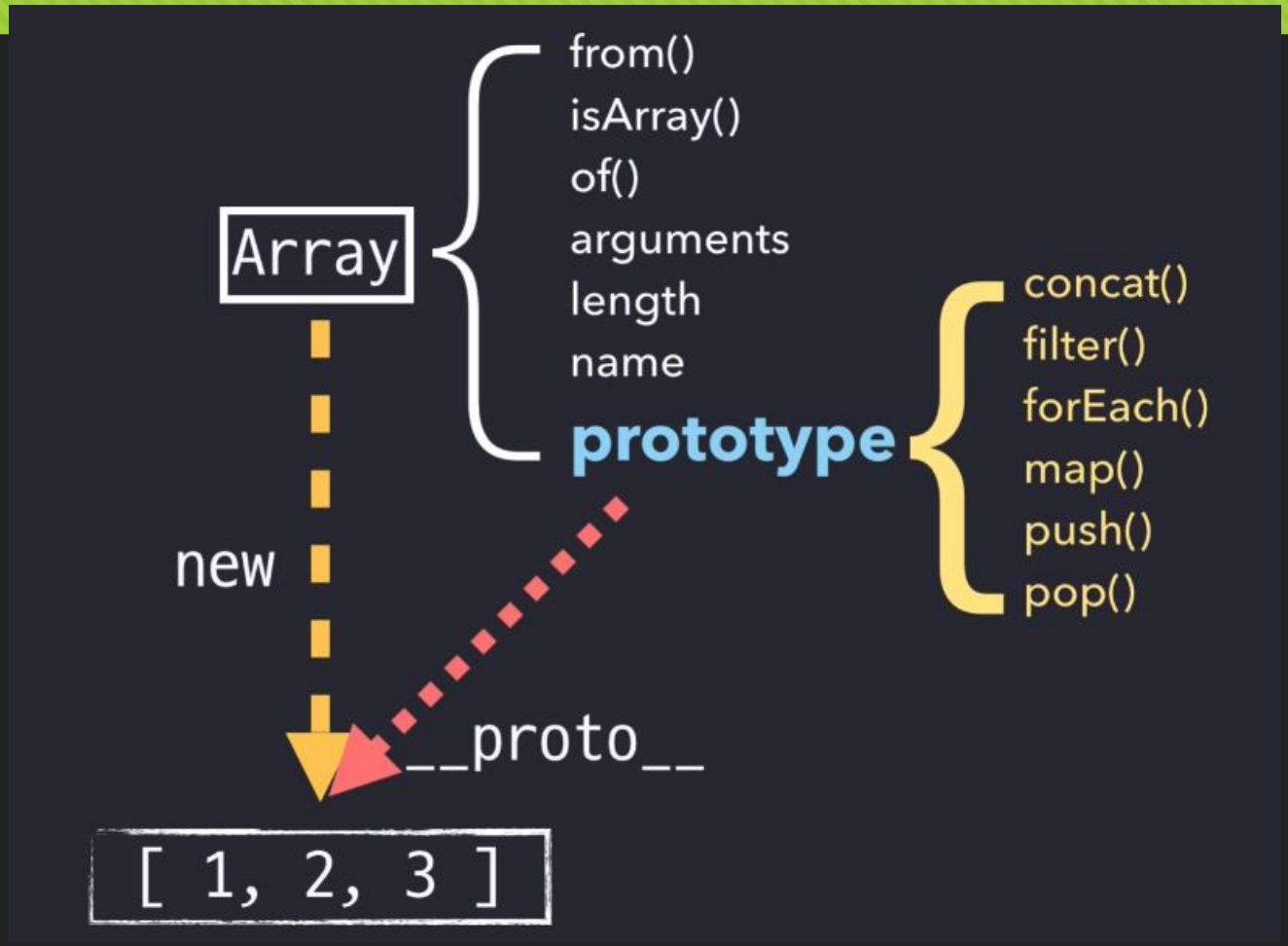
- 배열 객체에 대한 Prototype 고찰
  - 모든 참조형은 Object가 가장 최상위 Prototype 객체이며 Prototype 안에는 다른 Prototype이 존재할 수 있다.


```
var a = [1,2,3];
console.dir(a.__proto__);
console.dir(a.__proto__.__proto__);
```

```
▶ Array(0)
▶ Object
```

Prototype

- 배열 객체에 대한 Prototype 고찰



Prototype

- 배열 객체에 대한 Prototype 고찰
  - 같은 prototype을 서로 다른 인스턴스에서 프로퍼티로 참조할 경우 해당 prototype은 공유된다.
  - 즉 한 객체의 prototype에 프로퍼티를 넣을 경우 다른 객체에서 동일 prototype을 참조할 시 해당 프로퍼티 사용이 가능하다.

```
var a = [1,2,3];
var b = [4,5,6];
a.__proto__.funcA = function(){
    console.log('hello');
}
b.funcA();
```

```
hello
```

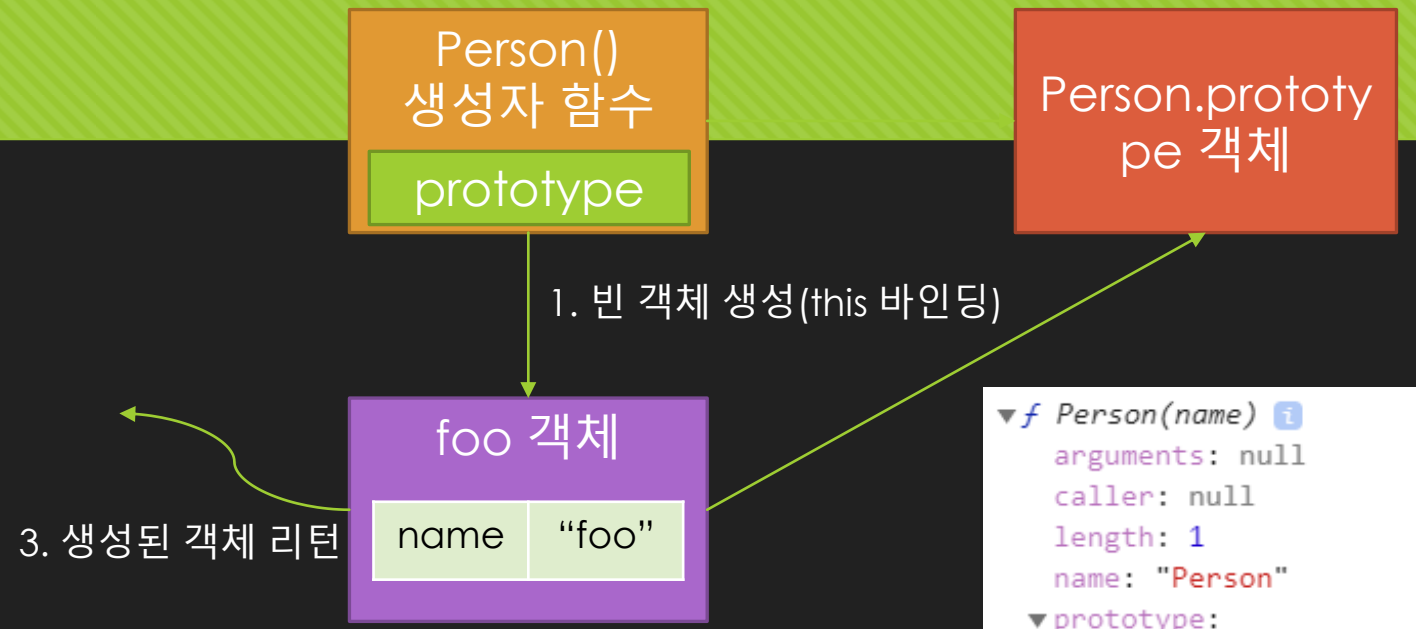
생성자 함수

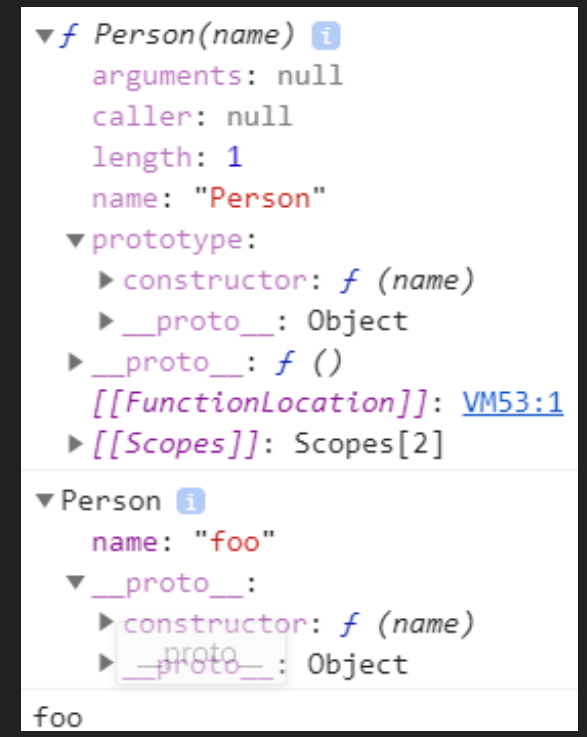
- 생성자 함수
  - "객체"를 생성할 때 사용하는 함수
  - 기존 함수에 new 연산자를 붙여서 호출하면 해당 함수는 생성자 함수로 동작한다.
  - 자바 스크립트에서는 특정 함수가 생성자 함수로 정의되어 있음을 알리기 위해 함수 이름의 첫 문자를 대문자로 쓰기를 권하고 있다.
  - 생성자 함수 호출 시, 생성자 함수 내부에 존재하는 this는 일반 함수 호출 방식에서의 this와 다르게 해당 함수로 생성한 객체 자기자신을 가리킨다.
  - 생성자 함수를 통해 인스턴스가 생성되면 해당 인스턴스는 기존 함수에서 Prototype으로 참조하고 있던 객체를 그대로 __proto__로 참조한다.

생성자 함수

- 생성자 함수

```
var Person = function(name){
    this.name = name;
}

var foo = new Person('foo');

console.dir(Person);
console.dir(foo);
console.log(foo.name);
```





생성자 함수

- 객체 리터럴 방식과 생성자 함수를 통한 객체 생성 방식의 차이

```
// 리터럴 방식의 객체 생성
var foo = {
    name : 'foo',
    age : 35,
    gender : 'man'
}
console.dir(foo);

// 생성자 함수 생성
function Person(name, age, gender, position){
    this.name = name;
    this.age = age;
    this.gender = gender;
}

// Person 생성자 함수를 이용해 bar 객체, baz 객체 생성
var bar = new Person('bar', 33, 'woman');
console.dir(bar);

var baz = new Person('baz', 25, 'man');
console.dir(baz);
```

생성자 함수

- 객체 리터럴 방식과 생성자 함수를 통한 객체 생성 방식의 차이

```
// 리터럴 방식의 객체 생성
var foo = {
    name : 'foo',
    age : 35,
    gender : 'man'
}
console.dir(foo);

// 생성자 함수 생성
function Person(name, age, gender, position){
    this.name = name;
    this.age = age;
    this.gender = gender;
}

// Person 생성자 함수를 이용해 bar 객체, baz 객체 생성
var bar = new Person('bar', 33, 'woman');
console.dir(bar);

var baz = new Person('baz', 25, 'man');
console.dir(baz);
```

```
▼Person i
    age: 33
    gender: "woman"
    name: "bar"
  ▼__proto__:
    ▶constructor: ƒ Person(name, age, gender, position)
    ▶__proto__: Object
▼Person i
    age: 25
    gender: "man"
    name: "baz"
  ▼__proto__:
    ▶constructor: ƒ Person(name, age, gender, position)
    ▶__proto__: Object
```

생성자 함수

- 객체 리터럴 방식과 생성자 함수를 통한 객체 생성 방식의 차이
  - 객체 리터럴 방식과 생성자 함수 방식의 차이는 프로토타입 객체에 있다.
  - 객체 리터럴 방식의 경우 Object를 프로토타입 객체로, 생성자 함수 방식의 경우에는 생성자 함수의 prototype의 프로퍼티가 가리키는 객체를 자신의 프로토타입 객체로 설정한다.

생성자 함수

- 호출 패턴과 this 바인딩
  - 생성자 함수를 new를 붙이지 않고 호출할 경우 일반 함수 호출의 경우 this는 window 전역 객체에 바인딩 된다.

```
function Person(name, age, gender, position){
    this.name = name;
    this.age = age;
    this.gender = gender;
}

var qux = Person('qux', 20, 'man');
console.log(qux);

console.log(window.name);
console.log(window.age);
console.log(window.gender);
```

```
undefined
qux
20
man
```

생성자 함수

- 호출 패턴과 this 바인딩
  - 생성자 함수를 new를 붙이지 않고 호출할 경우 일반 함수 호출을 통해 강제로 인스턴스 생성할 경우에는 다음과 같이 할 수 있다.

```
function Person(name, age, gender){
    if(!(this instanceof Person)){
        return new Person(name, age, gender);
    }
    this.name = name;
    this.age = age;
    this.gender = gender;
}

var qux = Person('qux', 20, 'man');
console.log(qux);
```

```
▶ Person {name: "qux", age: 20, gender: "man"}
```

생성자 함수

- 함수 리턴
  - 자바 스크립트 함수는 항상 리턴값을 반환한다.
  - 특히 return 문을 사용하지 않았더라도 다음의 규칙으로 항상 리턴값을 전달한다.
    1. 일반 함수나 메서드는 리턴값을 지정하지 않을 경우, undefined 값이 리턴된다.
    2. 생성자 함수에서 리턴 값을 지정하지 않을 경우 생성된 객체가 리턴된다.
    3. 생성자 함수에서 리턴 값으로 넘긴 값이 객체가 아닌 불린, 숫자, 문자열의 경우 이러한 리턴 값을 무시하고 this로 바인딩된 객체가 리턴된다.

생성자 함수

- 함수 리턴 예제

```
var NoReturn = function(){
    console.log('리턴 값 없음');
}

var result = NoReturn();
console.log(result)  // undefined
```

```
function Person(name, age, gender, position){
    this.name = name;
    this.age = age;
    this.gender = gender;

    //return {name:'bar', age:25, gender:'woman'};
}

var foo = new Person('foo',30,'woman');
console.dir(foo);
```

생성자 함수

- 함수 리턴 예제

```
function Person(name, age, gender){
    this.name = name;
    this.age = age;
    this.gender = gender;

    return 100;
}

var foo = new Person('foo', 30, 'man');
console.log(foo);
```

```
▶ Person {name: "foo", age: 30, gender: "man"}
```

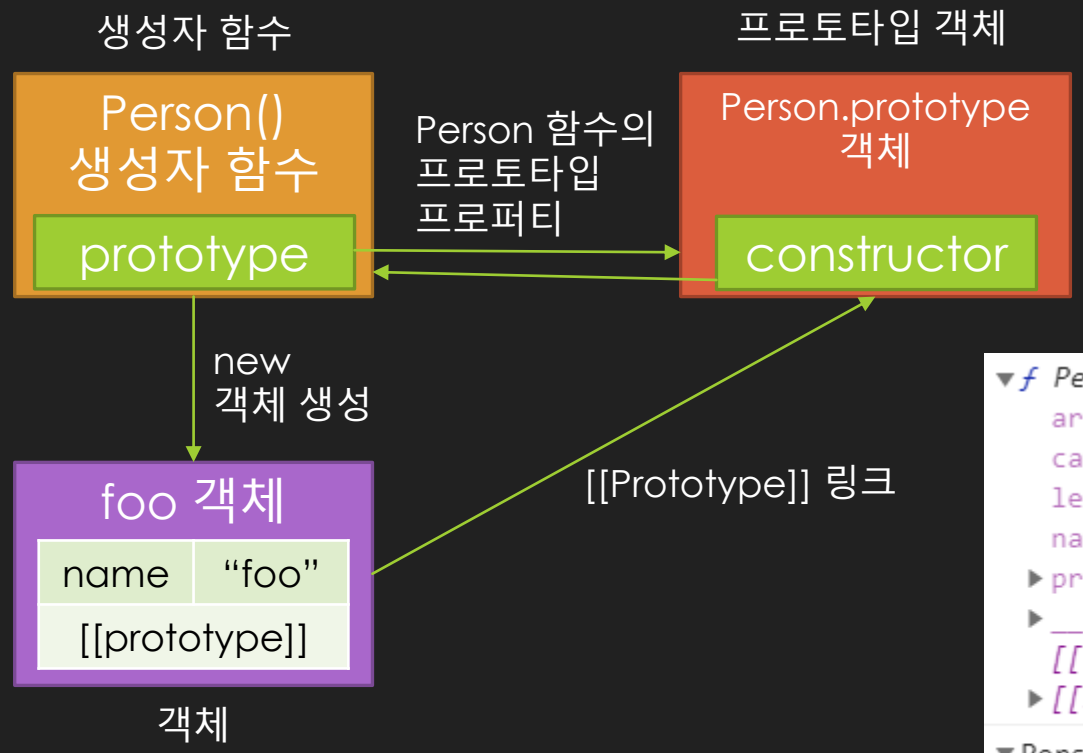
프로토타입 체이닝

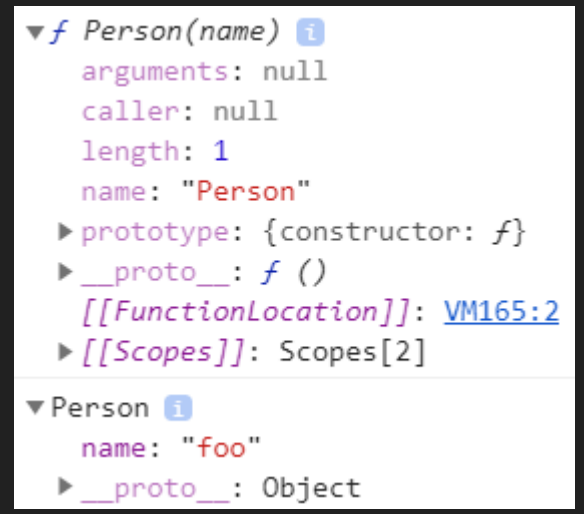
- 프로토타입 두 가지 의미
  - 자바 스크립트는 객체지향 프로그래밍 언어와는 달리 프로토타입 기반의 객체지향 프로그래밍을 지원한다.
  - 자바 스크립트에는 클래스 개념이 없으며 기존의 객체 리터럴이나 생성자 함수로 객체를 생성한다.
  - 이렇게 생성된 객체의 부모 클래스가 바로 '프로토타입' 객체이다.
  - 프로토타입은 [[prototype]] 링크와 prototype 프로퍼티를 구분해야 한다.
    - prototype 프로퍼티 : 생성자 함수가 가리키는 프로토타입 객체를 가리킨다. 보통 이 프로토타입 객체는 생성자 함수 자신을 가리킨다.
    - [[prototype]] : 생성된 객체가 가리키는 프로토타입 객체, 보통 생성자 함수로 정의되는 객체의 경우 해당 생성자 함수가 가리키는 prototype 프로퍼티 객체를 가리킨다.

프로토타입 체이닝

- 프로토타입 두 가지 의미

```
// Person 생성자 함수
function Person(name) {
    this.name = name;
}

// foo 객체 생성
var foo = new Person('foo');

console.dir(Person);
console.dir(foo);
```





프로토타입 체이닝

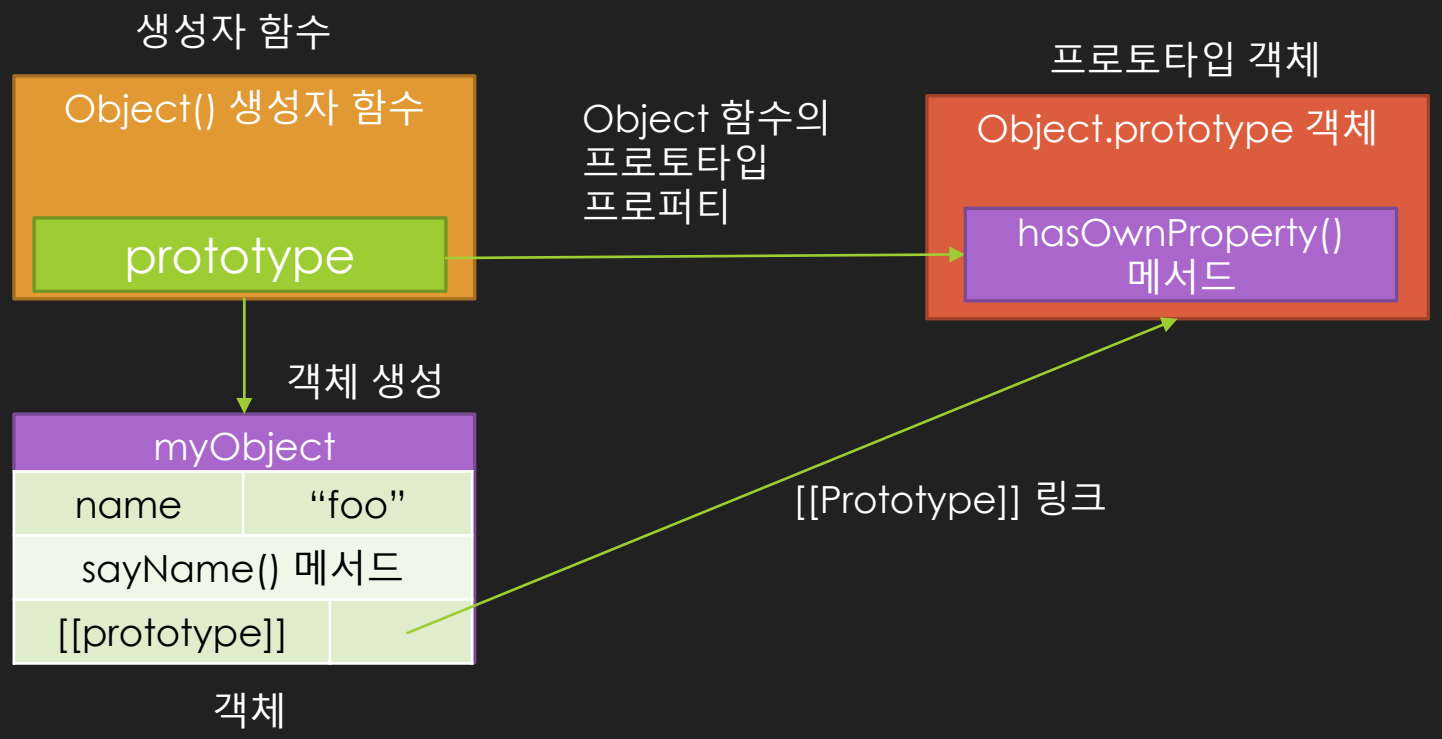
- 객체 리터럴 방식으로 생성된 객체의 프로토타입 체이닝
  - 자바 스크립트에서 객체는 자기 자신의 프로퍼티 뿐만 아니라, 자신의 부모 역할을 하는 프로토타입 객체의 프로퍼티 또한 마치 자신의 것 처럼 접근하는게 가능하다.
  - 이것을 가능하게 하는 기술이 바로 프로토타입 체이닝이다.

```
var myObject = {
    name: 'foo',
    sayName: function () {
        console.log('My Name is ' + this.name);
    }
};

myObject.sayName();
console.log(myObject.hasOwnProperty('name'));
console.log(myObject.hasOwnProperty('nickName'));
myObject.sayNickName();
```

```
My Name is foo
true
false
```

프로토타입 체이닝

- 객체 리터럴 방식으로 생성된 객체의 프로토타입 체이닝



프로토타입 체이닝

- 프로토타입 체이닝
  - 자바스크립트에서 특정 객체의 프로퍼티나 메서드에 접근하려고 할 때, 해당 객체에 접근하려는 프로퍼티 혹은 메서드가 없다면 [[prototype]] 링크를 따라 자신의 부모 역할을 하는 프로토타입 객체의 프로퍼티를 차례대로 검색한다
  - 위와 같은 것을 프로토타입 체이닝이라고 한다.
  - 자바 스크립트에서 모든 객체는 자신을 생성한 생성자 함수의 prototype 프로퍼티가 가리키는 객체를 자신의 프로토타입 객체로 취급한다.

프로토타입 체이닝

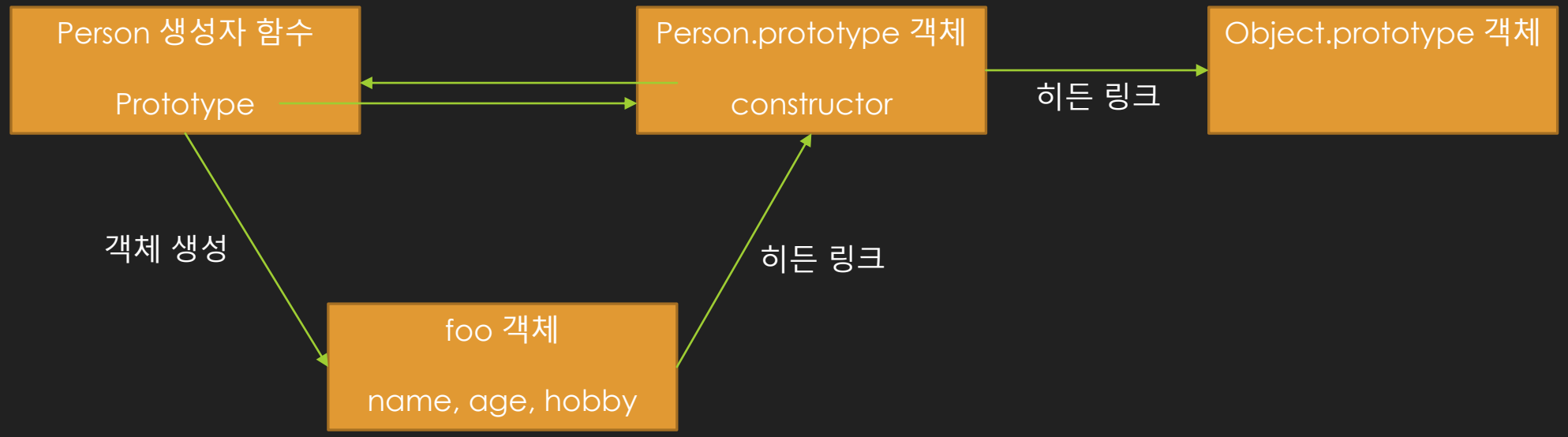
- 프로토타입 체이닝

```
// Person 생성자 함수
function Person(name, age, hobby) {
    this.name = name;
    this.age = age;
    this.hobby = hobby;
}

// foo 객체 생성
var foo = new Person('foo', 30, 'tennis');

// 프로토타입 체이닝
console.log(foo.hasOwnProperty('name')); // true

// Person.prototype 객체 출력
console.dir(Person.prototype);
```

```
true
▼Object
  ▶constructor: ƒ Person(name, age, hobby)
  ▶__proto__: Object
```

프로토타입 체이닝

- 프로토타입 체이닝
  - Object.prototype 객체는 프로토타입 체이닝의 종점이다.



프로토타입 체이닝

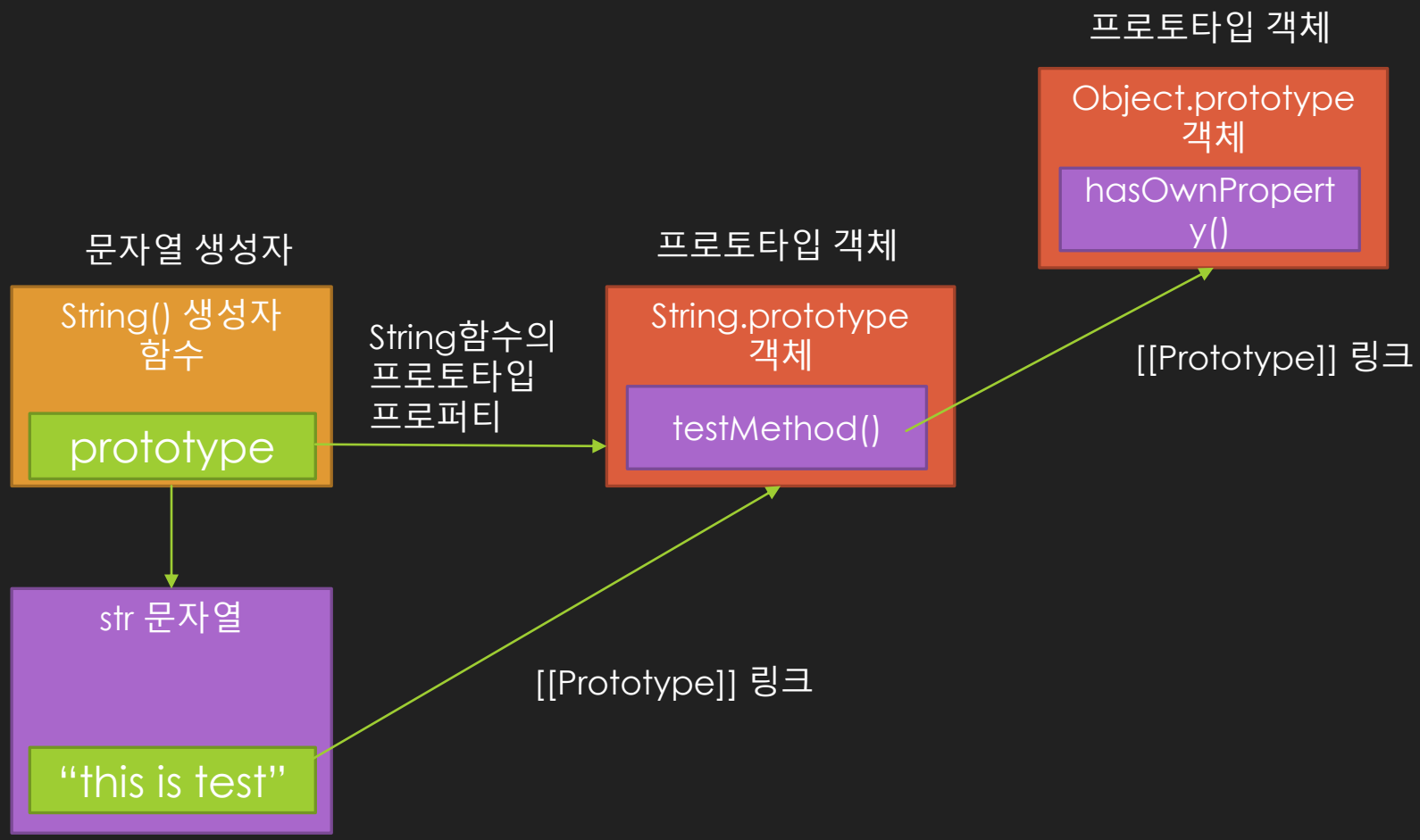
- 기본 데이터 타입의 확장
  - 앞의 방식처럼 자바스크립트의 숫자, 문자열, 배열 등에서 사용되는 표준 메서드들의 경우 이들의 프로토타입인 Number.prototype, string.prototype, array.prototype 등에 정의되어 있다.
  - 위와 같은 기본 내장 프로토타입 객체 또한 Object.prototype을 자신의 프로토타입으로 가지고 있어서 프로토타입 체이닝으로 사용이 가능하다.
  - 자바 스크립트는 Object.prototype, string.prototype과 같이 표준 빌트인 프로토타입 객체에도 사용자가 직접 정의한 메서드들을 추가하는 것을 허용한다.

```
String.prototype.testMethod = function () {
    console.log('This is the String.prototype.testMethod()');
};


var str = "this is test";
str.testMethod();


console.dir(String.prototype);
```

```
This is the String.prototype.testMethod()
▼ String
  ▶ testMethod: ƒ ()
  ▶ anchor: ƒ anchor()
  ▶ big: ƒ big()
  ▶ blink: ƒ blink()
  ▶ bold: ƒ bold()
```

프로토타입 체이닝

- 기본 데이터 타입의 확장



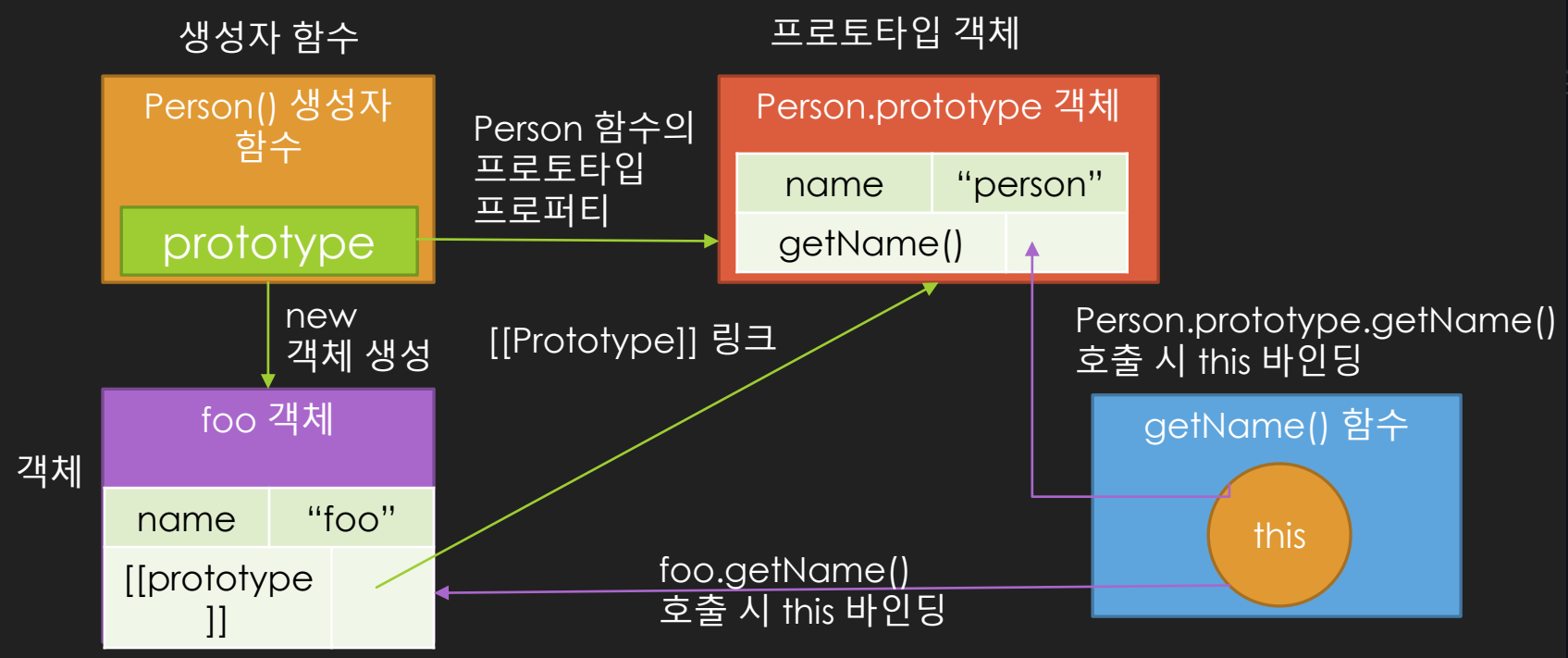
프로토타입 체이닝

- 프로토타입도 자바 스크립트 객체다.
  - 프로토타입 객체 역시 자바스크립트 객체이므로 일반 객체처럼 동적으로 프로퍼티를 추가/삭제하는 것이 가능하다.

```
// Person() 생성자 함수
function Person(name) {
    this.name = name;
}

// foo 객체 생성
var foo = new Person('foo');

//foo.sayHello();

// 프로토타입 객체에 sayHello() 메서드 정의
Person.prototype.sayHello = function () {
    console.log('Hello');
};

foo.sayHello(); // Hello
```

프로토타입 체이닝

- 프로토타입 메서드와 this 바인딩




```
// Person() 생성자 함수
function Person(name) {
    this.name = name;
}

// getName() 프로토타입 메서드
Person.prototype.getName = function () {
    return this.name;
};

// foo 객체 생성
var foo = new Person('foo');

console.log(foo.getName()); // foo

//Person.prototype 객체에 name 프로퍼티 동적 추가
Person.prototype.name = 'person';

console.log(Person.prototype.getName()); // person
```

프로토타입 체이닝

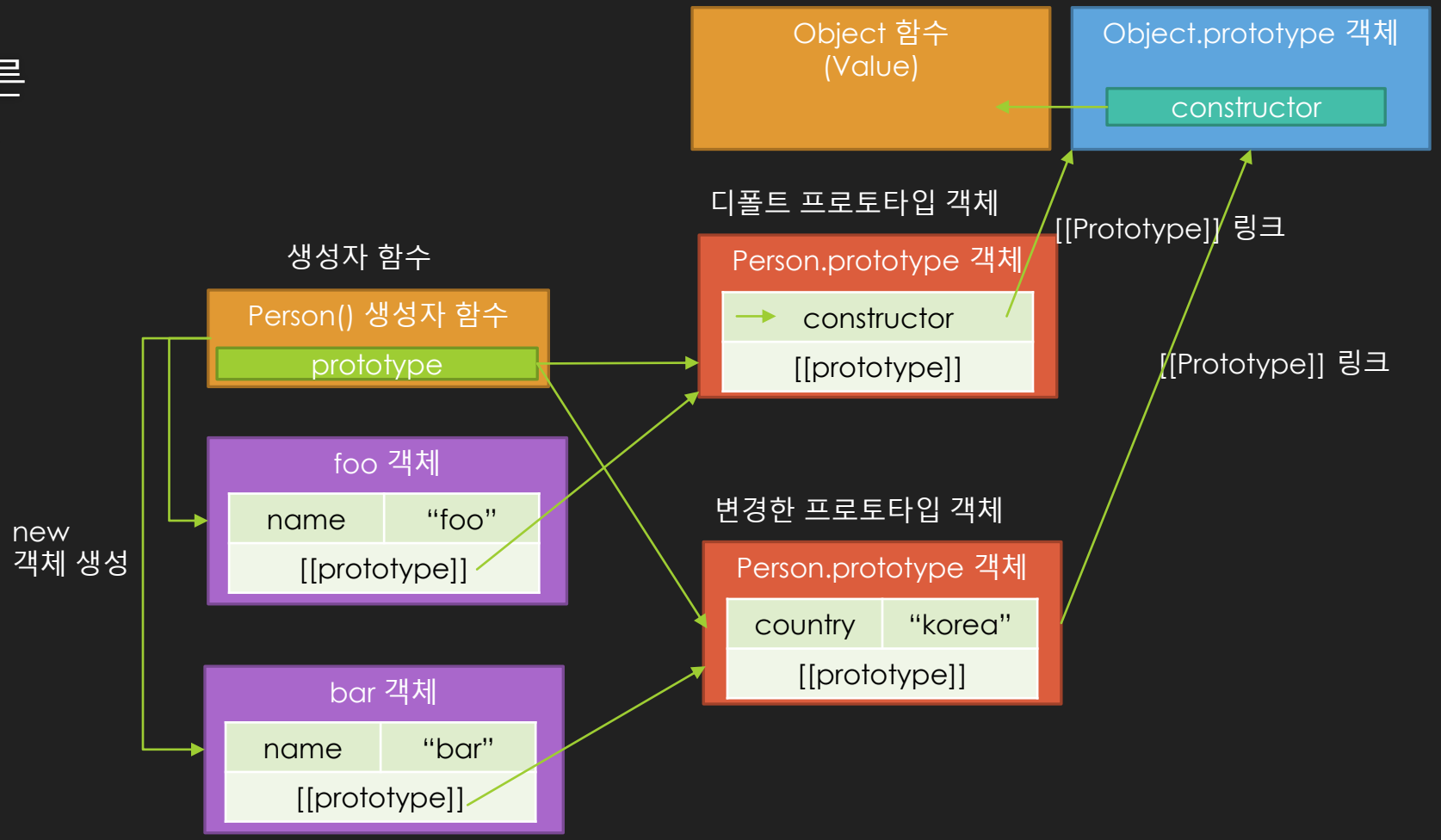
- 디폴트 프로토타입은 다른 객체로 변경이 가능하다.
  - 디폴트 프로토타입 객체를 다른 일반 객체로 변경하는 것이 가능하다.
  - 여기서 주의할 점은 생성자 함수의 프로토타입 객체가 변경되면, 변경된 시점 이후에 생성된 객체들은 변경된 프로토타입 객체로 [[prototype]] 링크를 연결한다.
  - 생성자 함수의 프로토타입이 변경되기 이전에 생성된 객체들은 기존 프로토타입 객체로의 [[prototype]] 링크를 그대로 유지한다.

```
▶ ƒ Person(name)
undefined
▶ ƒ Object()
undefined
korea
▶ ƒ Person(name)
▶ ƒ Object()
```

```
// Person() 생성자 함수
function Person(name) {
    this.name = name;
}
console.dir(Person.prototype.constructor);

// foo 객체 생성
var foo = new Person('foo');
console.log(foo.country);

// 디폴트 프로토타입 객체 변경
Person.prototype = {
    country: 'korea'
};
console.dir(Person.prototype.constructor);

// bar 객체 생성
var bar = new Person('bar');
console.log(foo.country);
console.log(bar.country);
console.dir(foo.constructor);
console.dir(bar.constructor);
```

프로토타입 체이닝

- 디폴트 프로토타입은 다른 객체로 변경이 가능하다.



프로토타입 체이닝

- 객체의 프로퍼티 읽기나 메서드를 실행할 때만 프로토타입 체이닝이 동작한다.
  - 객체에 있는 특정 파라미터에 값을 쓰려고 할 경우에는 프로토타입 체이닝이 일어나지 않지만 읽거나 메서드를 실행할 경우 프로토타입 체이닝이 동작한다.

```
// Person() 생성자 함수
function Person(name) {
    this.name = name;
}

Person.prototype.country = 'Korea';

var foo = new Person('foo');
var bar = new Person('bar');
console.log(foo.country);
console.log(bar.country);
foo.country = 'USA';

console.log(foo.country);
console.log(bar.country);
```

```
Korea
Korea
USA
Korea
```

프로토타입 체이닝

- 객체의 프로퍼티 읽기나 메서드를 실행할 때만 프로토타입 체이닝이 동작한다.