LEARNING SPOONS

LEARNING SPOONS ONLINE

자바스크립트 기초

자바스크립트 객체지향 프로그래밍

LEARNING SPOONS

Unit

1. 객체지향 프로그래밍 시작하기
2. 상속 이해하기
3. ES6 클래스 활용하기
4. [실습] To-do 앱 클론코딩 2차

Unit 3-1

객체지향 프로그래밍 시작하기

LEARNING SPOONS

객체지향 프로그래밍Object-Oriented Programing

객체지향이란 시스템을 상호작용하는 자율적인 객체들의 공동체로 바라보고 객체를 이용해 시스템을 분할하는 방법이다.

- 객체란 식별 가능한 구체적인 사물 또는 추상적인 개념이다.
- 객체는 구별 가능한 식별자, 특징적인 행동, 변경 가능한 상태를 가진다.
- 소프트웨어 안에서 객체는 저장된 상태와 실행 가능한 코드를 통해 구현된다. (속성과 메소드)
- 자율적인 객체란 상태와 행위를 함께 지니며 스스로 자기 자신을 책임지는 객체를 의미
- 객체는 다른 객체와 협력하기 위해 메시지를 전송하고 수신된 메시지를 처리하는 방법을 메소드라고 한다.

참고: 객체지향의 사실과 오해, 조영호

LEARNING SPOONS

강예서 객체

아버지의 명석한 두뇌에 엄마의 야망을 유전자로 받아 공부에 있어서는 항상 에너지가 넘치는 우등생. 1등을 하지 않으면, 잠도 잘 못 자는 근성의 소유자다.
덕분에 '재수 없이 잘난 애'로 통해 친구들 사이에선 은근 시기의 눈총을 받지만 아랑곳하지 않는다. 명문 신아고등학교에 수석 입학해 엄마를 SKY 캐슬의 워너비맘으로 등극시킨, 엄마 마음에는 차고도 넘치는 자랑스러운 딸이니까.
서울의대를 졸업해 모교에서 수련의 과정을 거쳐 서울대병원 교수가 되는 것이 그녀의 목표. 그래야 불쌍한 엄마를 무시하는 콧대 높은 할머니의 코를 팍 꺾을 테니까.
…
…
…

출처:
https://namu.wiki/w/강예서

LEARNING SPOONS

강예서 객체

```
var yeoseo = {
 name: '강예서',
 nickname: '재수 없이 잘난 애',
 energy: 100,
 rank: 1,
 study: function(subject) {
   console.log(subject + " 공부중...");
   this.energy -= 1;
 },
 takeExam: function(exam) {
 },
 sleep: function() {
   if (this.rank != 1) {
     this.energy -= 50;
   }
 }
}
```

객체를 여러 그룹으로 분류classification

분류란 객체에 특정한 **타입을 적용하는 작업**이다. 객체에 특정한 타입을 적용하기로 결심했을 때 우리는 그 객체를 특정한 집합의 멤버로 분류하고 있는 것이다.

- 구체적이고 실제적인 객체가 무수히 존재하지만 공통적인 특성을 기준으로 객체를 여러 그룹으로 묶어 단순화한다. 즉, **추상화**를 하는 것이다.
- 객체는 특정한 타입이 적용될 수 있는 구체적인 사물을 의미하고 타입이 객체에 적용됐을 때 객체를 타입의 **인스턴스**라고 한다.
- 객체의 타입을 결정하는 것은 객체의 **행동** 뿐이다. 두 객체가 동일한 데이터를 가지고 있더라도 다른 행동을 한다면 서로 다른 타입이다.

참고: 객체지향의 사실과 오해, 조영호

LEARNING SPOONS

클래스 기반 / 프로토타입 기반 객체지향 프로그래밍

- 클래스 기반에서 객체는 클래스에 명시된 내용에 따라 생성된다.
- 프로토타입 기반에서는 특정 타입의 객체 하나를 생성하고 이와 똑같은 모양의 객체를 생성한다.

```java
public class Teacher {
 private String subject;
 private String name;

 public Teacher(String name, String subject) {
   this.name = name;
   this.subject = subject;
 }

 public void teach(Student student) {
   student.learn(this.subject);
 }
}
```

클래스 기반 - Java

```javascript
var teacher = {
 teach: function(student) {
   student.learn(this.subject);
 }
}

var jeado = Object.create(teacher);
jeado.name = 'jeado';
jeado.subject = 'JavaScript';
```

프로토타입 기반 - JavaScript

__proto__ 를 통한 타입 정의

- 모든 객체는 다른 객체의 원형(prototype)이 될 수 있습니다.
- 원형이 되는 객체를 __proto__ 속성으로 정의할 수 있습니다.
- __proto__ 속성은 enumerable는 false 이고 configurable은 true 입니다.

```
var studentProto = {
 study: function(subject) {
   console.log(subject + " 공부중...");
   this.energy -= 1;
   this.exp += 10;
 }
}

var yeoseo = {
 name: '강예서',
 energy: 100,
 exp: 100,
 __proto__: studentProto
};
```

팩토리 함수를 통한 객체 생성

```
var studentProto = {
 study: function(subject) {
   console.log(subject + " 공부중...");
   this.energy -= 1;
   this.exp += 10;
 }
};

function createStudent(name) {
 var student = Object.create(studentProto);
 student.name = name;
 student.energy = 100;
 student.exp = 0;
 return student;
}

var yeoseo = createStudent('김혜나');
var hyena = createStudent('강예서');
```

생성자 함수를 통한 타입 생성

- 함수를 통해 새로운 타입을 정의 할 수 있다.
- 함수에 new 키워드를 이용하여 호출하면 별도의 return이 없어도 객체가 반환된다.

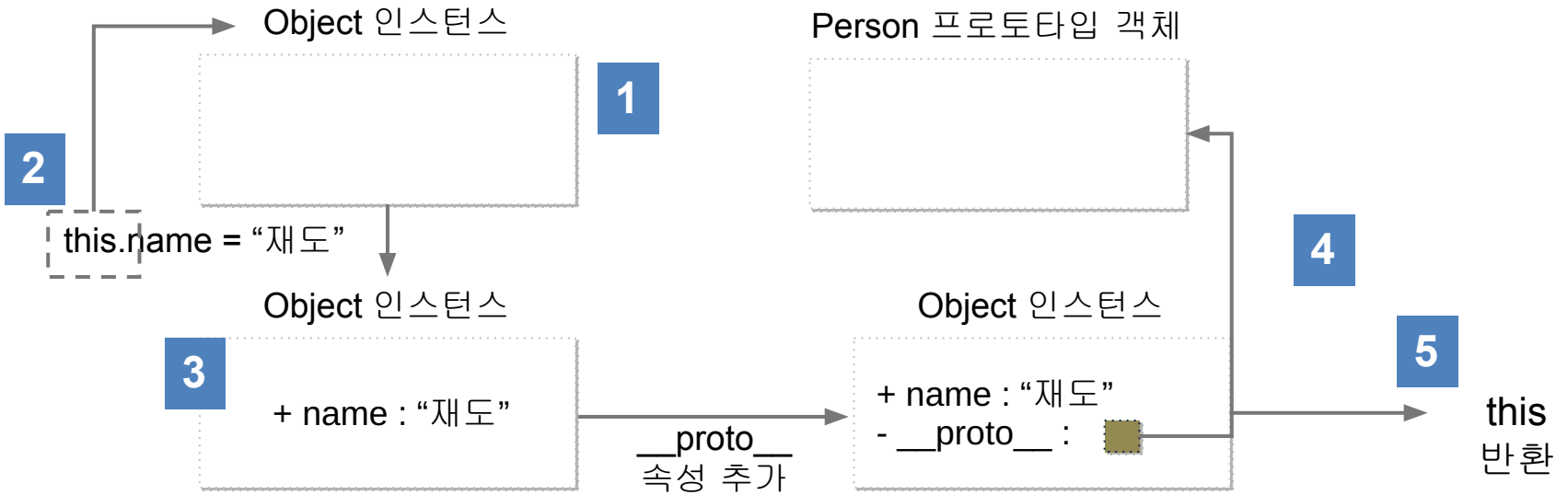
```
function Teacher(name, subject) {
 this.name = name;
 this.subject = subject;
 this.teach = function(student) {
   student.learn(this.subject)
 }
}

var jeado = new Teacher('jeado', 'JavsScript')
console.log(jeado instanceof Teacher)
console.log(jeado.constructor)
```

LEARNING SPOONS

생성자 함수를 통한 객체 생성

- 메모리에 인스턴스 멤버가 없는 비어있는 Object 인스턴스를 만듬
- 생성된 Object를 this에 할당
- 생성자 함수 호출 (this에 멤버 초기화)
- 비공개 __proto__를 추가하여 생성자의 프로토타입 객체와 인스턴스 연결
- this가 가리키는 최종 인스턴스를 생성자의 반환값으로 변환

```
function Person(name) {
 this.name = name;
}
var jeado = new Person('Jeado');
```



Unit 3-2

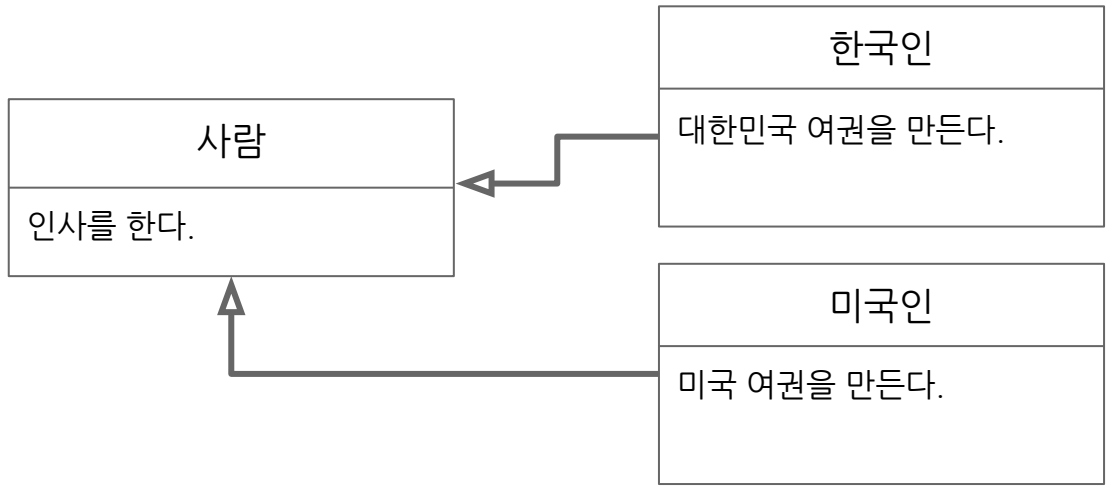
상속 이해하기

LEARNING SPOONS

일반화Generalization와 특수화Specialization

사람과 한국인의 관계처럼 타입과 타입 사이에는 일반화/특수화 관계가 존재할 수 있다.

- 두 타입 간에 일반화/특수화 관계가 성립하려면 한 타입이 다른 타입보다 더 특수하게 행동해야한다. 상태가 아닌 행동의 관점에서 특수한 타입을 정의해야 한다.
- 일반적인 타입을 슈퍼타입(Supertype)이라고 하고 좀 더 특수한 타입을 서브타입(Subtype)이라고 한다.
- 서브타입은 슈퍼타입의 행위에 추가적으로 툭수한 자신만의 행동을 추가하는 것이므로 슈퍼타입의 행동은 서브타입에게 자동으로 상속된다.



참고: 객체지향의 사실과 오해, 조영호

프로토타입 기반 언어에서의 상속

- **프로토타입 기반 언어**에서 상속은 객체와 객체 간의 관계로 이루어진다. 즉, 어떤 타입의 객체를 특정 객체 타입의 특수화로 만들거나 행동을 공유할 수 있게 만들고 싶은 경우 객체와 객체를 상속 관계를 통해 연결한다.

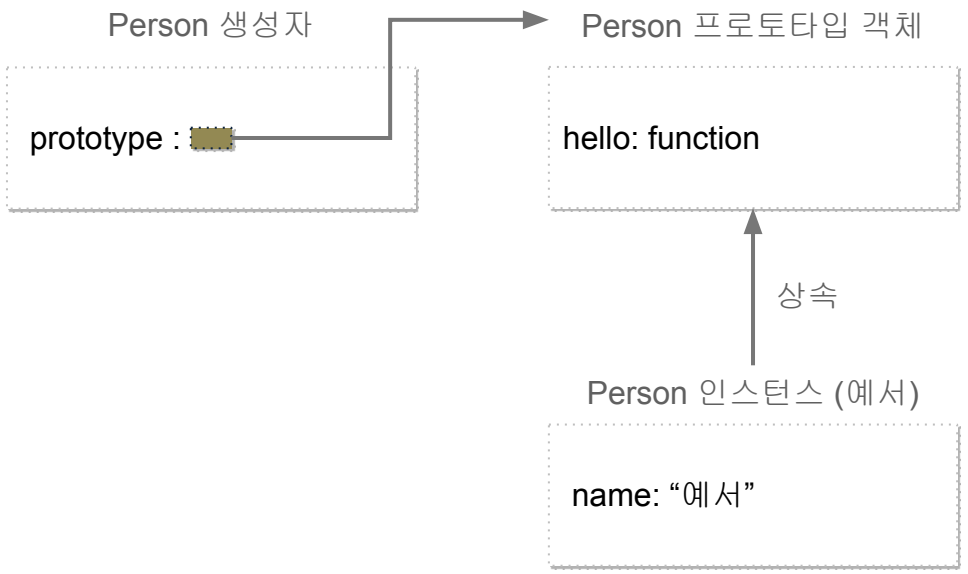
- 클래스 기반 언어와 프로토타입 기반 언어 모두 위임 메커니즘을 기반으로 메시지를 해석할 수 있는 대상을 선택한다. 메시지를 수신한 객체는 자신의 메시지를 이해할 수 없을 경우 부모 객체에게 위임한다. 클래스 기반과의 차이점은 단지 위임이 클래스를 기준으로 이루어게 된다.

참고: 객체지향의 사실과 오해, 조영호

LEARNING SPOONS

prototype 객체를 통한 상속

- 모든 인스턴스는 해당 생성자의 prototype 객체를 상속한다.

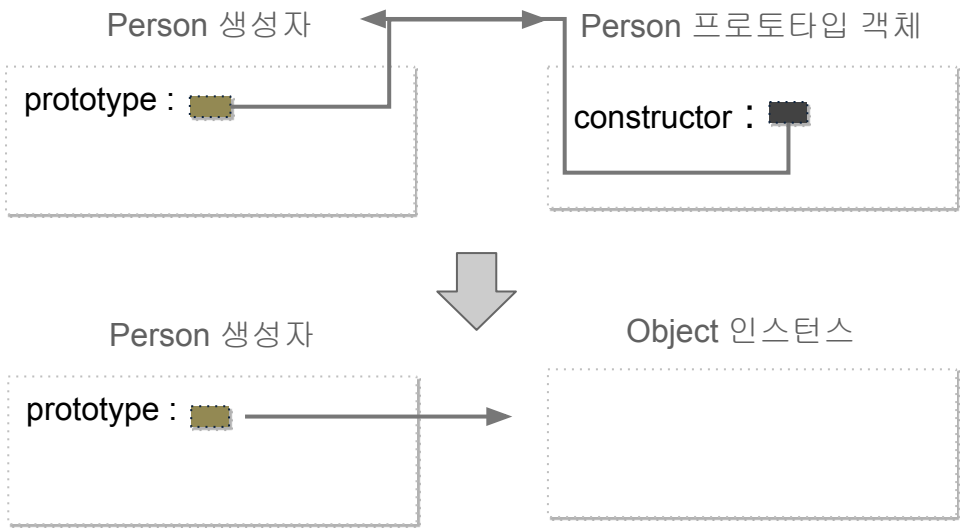



```
function Person(name) {
 this.name = name;
}
Person.prototype.hello = function() {
 console.log('hi! ' + this.name);
}

var jeado = new Person('예서');
jeado.hello();
```

LEARNING SPOONS

prototype 상속

- prototype 객체는 Object 타입의 인스턴스이다.

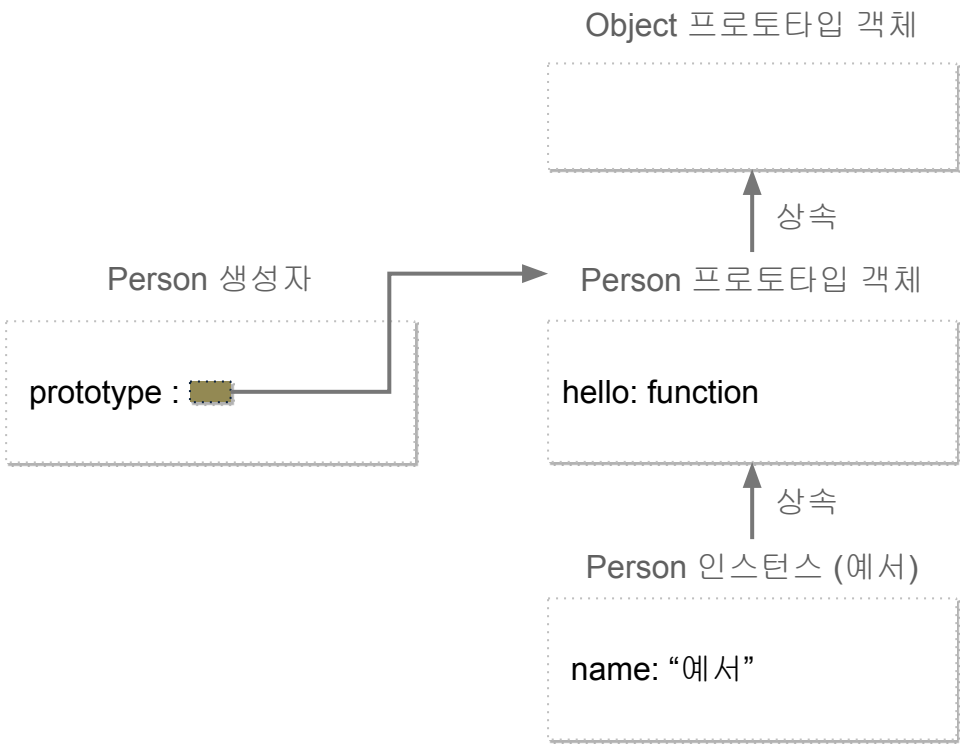


< 내부 >
Person.prototype = new Object();
Person.prototype.constructor = Person;

LEARNING SPOONS

prototype 상속

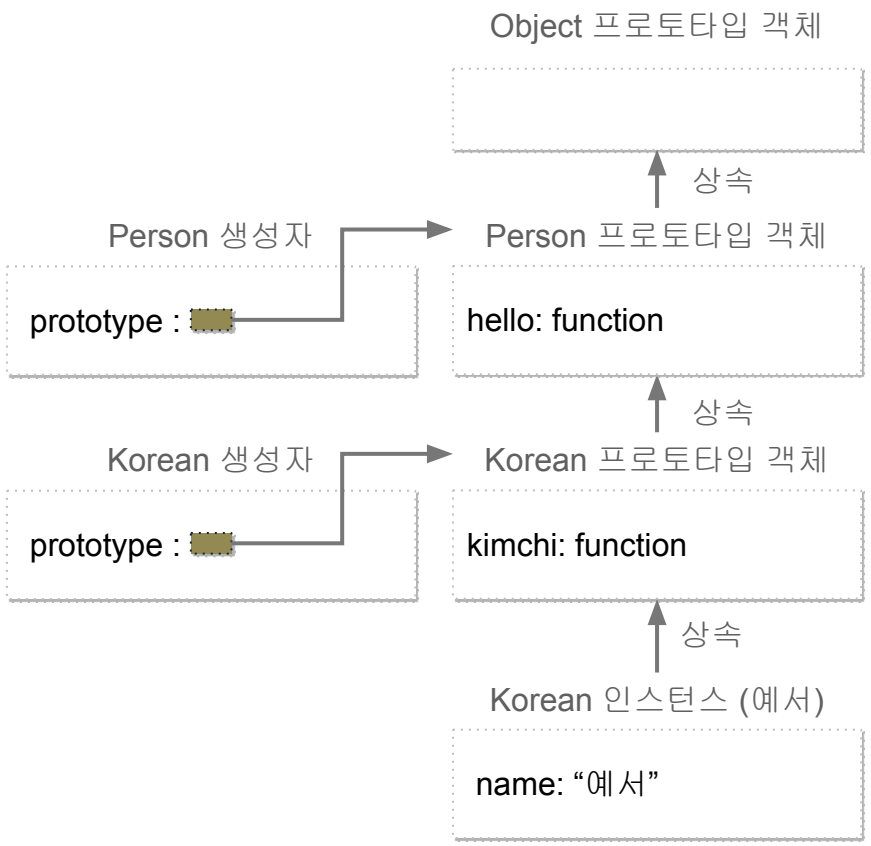
- 모든 인스턴스는 Object.prototype 객체를 상속




```
function Person(name) {
 this.name = name;
}
Person.prototype.hello = function() {
 console.log('hi! ' + this.name);
}

var jeado = new Person('jeado');
jeado.hello();
jeado.hasOwnProperty('age') // false
```

LEARNING SPOONS

prototype 상속




```
function Person(name) {
 this.name = name;
}
Person.prototype.hello = function() {
 console.log('hi! ' + this.name);
}

function Korean(name) {
 Person.call(this, name);
}
Korean.prototype =
 Object.create(Person.prototype);
Korean.prototype.kimchi = function() {
 console.log('kimchi~');
}

var jeado = new Korean('예서');
jeado.hello();
jeado.kimchi('age');
jeado.hasOwnProperty('age') // false
```

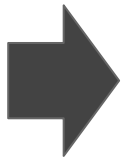
Unit 3

ES6의 클래스 시스템 활용하기

LEARNING SPOONS

class 정의

```
function Cart(user) {
    this.user =  user;
    this.store = { };
}

Cart.prototype.update = function(id, product) {
    this.store[id] = product;
}

Cart.prototype.getProudct = function(id) {
    return this.store[id];
}
```



```
class Cart {
    constructor(user) {
        this.user =  user;
        this.store = { };
    }
    update(id, product) {
        this.store[id] = product;
    }
    getProudct(id) {
        return this.store[id];
    }
}
```

getter & setter method

- 메소드 앞에 get과 set 키워드를 사용하여 정의

```
class LocalStorage {
 constructor(key) {
   this.key = key
 }
 get data() {
   return JSON.parse(localStorage.getItem(this.key))
 }
 set data(data) {
   localStorage.setItem(this.key, JSON.stringify(data))
 }
}
const ls = new LocalStorage('cart')
ls.data = [{ id: 1, price: 10}, { id: 2, price: 20 }]
```

static method

- 메소드 앞에 static 키워드를 이용하여 static 메소드 작성이 능하다.

```
function Util() { };
Util.getRandomInt = function() { };

class Util {
    static getRandomInt(min, max) {
        return Math.floor(Math.random() * (max - min + 1) + min);
    }
}

Util.getRandomInt(0, 9);
```

LEARNING SPOONS

Class 상속

```
function Person(name) { // 생성자 함수를 상속할 수도 있다.
    this.name = name;
}

class Korean extends Person {
    constructor(name, city) {
        super(name);
        this.city = city
    }
    kimchy() {
        console.log("^ _ ^");
    }
}
const john = new Korean("john", "seoul");
john.kimchy();
john.name; // john
```

실습

To-do 앱 클론코딩 2차