25. 클래스

#1. 클래스는 프로토타입의 문법적 설탕인가?

자바스크립트는 프로토타입 기반(prototype-based) 객체지향 언어다. 비록 다른 객체지향 언어들과의 차이점에 대한 논쟁이 있긴 하지만, 자바스크립트는 강력한 객체지향 프로그래밍 능력을 지니고 있다.

프로토타입 기반 객체지향 언어는 클래스가 필요 없는(class-free) 객체지향 프로그래밍 언어이다. ES5에서는 클래스 없이도 아래와 같이 생성자 함수와 프로토타입 체인, 클로저를 사용하여 객체 지향 언어의 상속, 캡슐화(정보 은닉) 등의 개념을 구현할 수 있다.

JAVASCRIPT

```
// ES5 생성자 함수
var Person = (function () {
  // 자유 변수이며 private하다
  var _name = '';

  // 생성자 함수(클로저)
  function Person(name) { _name = name; }

  // 프로토타입 메소드(클로저)
  Person.prototype.sayHi = function () {
    console.log('Hi! My name is ' + _name);
  };

  // 생성자 함수 반환
  return Person;
```

25. 클래스

#1. 클래스는 프로토타입의 문법적 설탕인가?

자바스크립트는 프로토타입 기반(prototype-based) 객체지향 언어다. 비록 다른 객체지향 언어들과의 차이점에 대한 논쟁이 있긴 하지만, 자바스크립트는 강력한 객체지향 프로그래밍 능력을 지니고 있다.

프로토타입 기반 객체지향 언어는 클래스가 필요 없는(class-free) 객체지향 프로그래밍 언어이다. ES5에서는 클래스 없이도 아래와 같이 생성자 함수와 프로토타입 체인, 클로저를 사용하여 객체 지향 언어의 상속, 캡슐화(정보 은닉) 등의 개념을 구현할 수 있다.

JAVASCRIPT

```
// ES5 생성자 함수
var Person = (function () {
  // 자유 변수이며 private하다
  var _name = '';

  // 생성자 함수(클로저)
  function Person(name) { _name = name; }

  // 프로토타입 메소드(클로저)
  Person.prototype.sayHi = function () {
    console.log('Hi! My name is ' + _name);
  };

  // 생성자 함수 반환
  return Person;
```

```
// ES5 생성자 함수
var Person = (function () {
  // 자유 변수이며 private하다
  var _name = '';

  // 생성자 함수(클로저)
  function Person(name) { _name = name; }

  // 프로토타입 메소드(클로저)
  Person.prototype.sayHi = function () {
    console.log('Hi! My name is ' + _name);
  };

  // 생성자 함수 반환
  return Person;

}());

// 인스턴스 생성
var me = new Person('Lee');

// _name은 지역 변수이므로 외부에서 접근하여 변경할 수 없다. 즉, private하다.
// me 객체에는 _name 프로퍼티가 존재하지 않기 때문에 me._name 프로퍼티를 동적 추가할 뿐이
다.
me._name = 'Kim';
me.sayHi(); // Hi! My name is Lee
```

```
}());

// 인스턴스 생성
var me = new Person('Lee');

// _name은 지역 변수이므로 외부에서 접근하여 변경할 수 없다. 즉, private하다.
// me 객체에는 _name 프로퍼티가 존재하지 않기 때문에 me._name 프로퍼티를 동적 추가할 뿐이다.
me._name = 'Kim';
me.sayHi(); // Hi! My name is Lee
```

클래스와 생성자 함수의 공통점
- 프로토타입 기반의 인스턴스 생성
- 프로토타입 기반의 객체지향을 구현

하지만 클래스 기반 언어에 익숙한 프로그래머들은 프로토타입 기반 프로그래밍 방식에 혼란을 느낄 수 있으며 자바스크립트를 어렵게 느끼게 하는 하나의 장벽처럼 인식되었다.

ES6에서 새롭게 도입된 클래스는 기존 프로토타입 기반 객체지향 프로그래밍보다 Java나 C#과 같은 클래스 기반 객체지향 프로그래밍에 익숙한 프로그래머가 보다 빠르게 학습할 수 있도록 클래스 기반 객체지향 프로그래밍 언어와 매우 흡사한 새로운 객체 생성 메커니즘을 제시하고 있다.

그렇다고 ES6의 클래스가 기존의 프로토타입 기반 객체지향 모델을 폐지하고 새롭게 클래스 기반 객체지향 모델을 제공하는 것은 아니다. 사실 클래스는 함수이며 기존 프로토타입 기반 패턴을 클래스 기반 패턴처럼 사용할 수 있도록 하는 문법적 설탕(Syntactic sugar)이라고 볼 수도 있다.

사람이 이해하기 쉽고, 표현하기 쉽게 컴퓨터언어를 디자인해 놓은 문맥

사람이 사용하기에 단순하다는것이 유래되었다. 간결하고 명확성에 표현이 가능하다

단, 클래스와 생성자 함수는 모두 프로토타입 기반의 인스턴스를 생성하지만 정확히 동일하게 동작하지는 않는다. 클래스는 생성자 함수보다 엄격하며 생성자 함수에서는 제공하지 않는 기능도 제공한다.

클래스는 생성자 함수와 매우 유사하게 동작하지만 아래와 같이 몇가지 차이가 있다.

상속관계 구현을 보다 간결하고 명료하게 한다.

클래스와 생성자 함수의 차이점

1. 클래스는 new 연산자를 사용하지 않고 호출하면 에러가 발생한다. 하지만 생성자 함수는 new 연산자를 사용하지 않고 호출하면 일반 함수로서 호출된다.
2. 클래스는 상속을 지원하는 extends와 super 키워드를 제공한다. 하지만 생성자 함수는 extends와 super 키워드를 지원하지 않는다.
3. 클래스는 호이스팅이 발생하지 않는 것처럼 동작한다. 하지만 함수 선언문으로 정의된 생성자 함수는 함수 호이스팅이, 함수 표현식으로 정의한 생성자 함수는 변수 호이스팅이 발생한다.
4. 클래스 내의 모든 코드에는 암묵적으로 strict 모드가 지정되어 실행되며 strict 모드를 해지할 수 없다. 하지만 생성자 함수는 암묵적으로 strict 모드가 지정되지 않는다.
5. 클래스의 constructor, 프로토타입 메소드, 정적 메소드는 모두 프로퍼티 어트리뷰트 [[Enumerable]]의 값이 false이다. 다시 말해, 열거되지 않는다.

생성자 함수와 클래스는 프로토타입 기반의 객체지향을 구현했다는 점에서 매우 유사하다. 하지만 클래스는 생성자 함수를 기반으로 한 객체 생성 방식보다 견고하고 명료하다.(그렇다고 클래스가 자바스크립트의 다른 객체 생성 방식보다 우월하다고 생각하지는 않는다.) 특히 클래스의 extends와 super 키워드는 상속 관계 구현을 보다 간결하고 명료하게 한다.

따라서 클래스를 프로토타입 기반 객체 생성 패턴의 단순한 문법적 설탕이라고 보기 보다는 **새로운 객체 생성 메커니즘**으로 보는 것이 보다 합당하다.

2. 클래스 정의

클래스는 class 키워드를 사용하여 정의한다. 클래스 이름은 생성자 함수와 마찬가지로 파스칼 케이스를 사용하는 것이 일반적이다. 파스칼 케이스를 사용하지 않아도 에러가 발생하지는 않는다.

JAVASCRIPT

```
// 클래스 선언문
class Person {}
```

일반적이지는 않지만, 함수와 마찬가지로 표현식으로 클래스를 정의할 수도 있다. 이때 클래스는 함수와 마찬가지로 이름을 가질 수도 있고, 갖지 않을 수도 있다.

JAVASCRIPT

```
// 익명 클래스 표현식
const Person = class {};

// 기명 클래스 표현식
const Person = class MyClass {};
```

클래스를 표현식으로 정의할 수 있다는 것은 클래스가 값으로 사용할 수 있는 일급 객체이라는 것을 의미한다. 즉, 클래스는 일급 객체로서 아래와 같은 특징을 갖는다.

- 무명의 리터럴로 생성할 수 있다. 즉, 런타임에 생성이 가능하다.
- 변수나 자료 구조(객체, 배열 등)에 저장할 수 있다.
- 함수의 매개 변수에게 전달할 수 있다.
- 함수의 반환값으로 사용할 수 있다.

좀 더 자세히 말하자면 클래스는 함수이다. 따라서 클래스는 값처럼 사용할 수 있는 일급 객체이다. 이에 대해서는 차차 알아보도록 하자.

클래스 몸체에는 0개 이상의 메소드만을 정의할 수 있다. 클래스 몸체에서 정의할 수 있는 메소드는 construnctor(생성자), 프로토타입 메소드, 정적 메소드 3가지가 있다.

JAVASCRIPT

```
// 클래스 선언문
class Person {
  // 생성자
  constructor(name) {
    // 인스턴스 생성 및 초기화
    this.name = name; // r
  }

  // 프로토타입 메소드
  sayHi() {
    console.log(`Hi! My na
  }

  // 정적 메소드
  static sayHello() {
    console.log('Hello!');
  }
}

// 인스턴스 생성
const me = new Person('Lee');

// 인스턴스의 프로퍼티 참조
console.log(me.name); // Lee
// 프로토타입 메소드 호출
me.sayHi(); // Hi! My name is Lee
// 정적 메소드 호출
Person.sayHello(); // Hello!
```

```
class Person {
    constructor(name) {
        this.name = name;
    }

    sayHi() {
        console.log(`Hi! My name is ${this.name}`);
    }

    static sayHello() {
        console.log('Hello!');
    }
}

const me = new Person('Lee');

console.log(me.name);
me.sayHi();
Person.sayHello();
```

Person의 name에 name을 대입하겠다. 결과 name이 'Lee'다.

`Hi! My name is ${this.name}`을 출력해라.

여기서 말하는 this는 Person이다.

대입

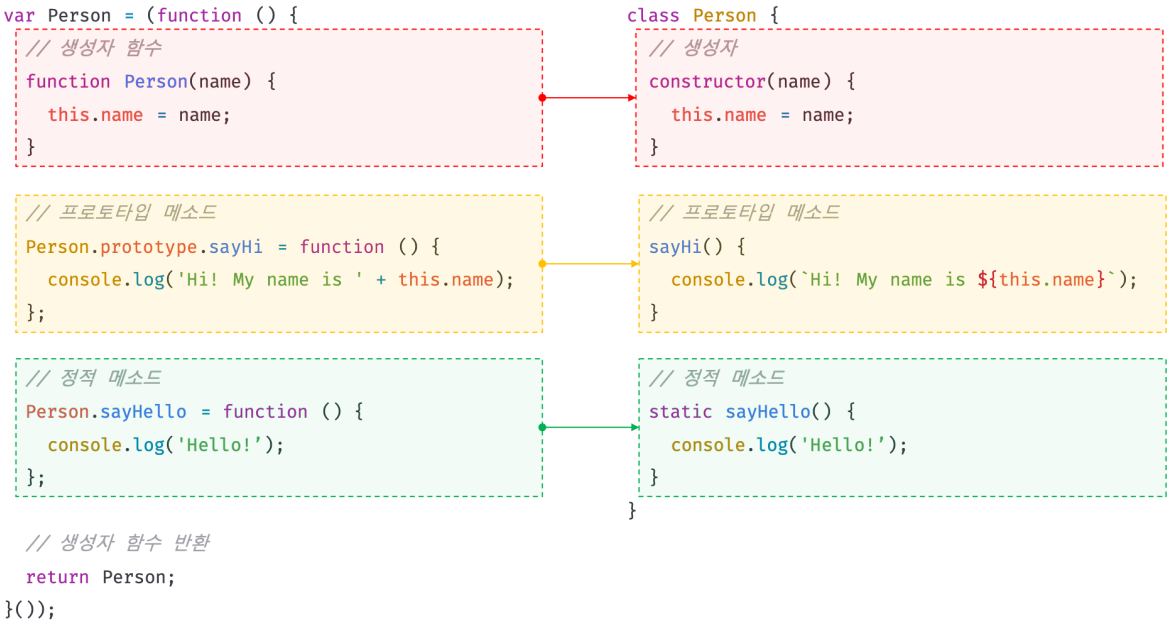
즉, 'Hi! My name is Person의 name을 찍어라 = 'Hi! My name is 'Lee'

Person.sayHi(); //ERROR

→ ∵ 이렇게하려면 sayHi앞에 static을 붙여야한다.

static을 전역으로 쓰겠다는건데, 보통 new 연산자 써서 객체만들고, 그객체에 안에있는 함수를 호출을 쓰는건데, static을 붙이면 굳이 new 안써도된다.

클래스와 생성자 함수의 정의 방식을 비교해 보면 아래와 같다.



클래스와 생성자 함수의 정의 방식 비교

이처럼 클래스와 생성자 함수의 정의 방식은 형태적인 면에서 매우 유사하다.

3. 클래스 호이스팅

클래스는 클래스 정의 이전에 참조할 수 없다.

JAVASCRIPT

```
console.log(Person);
// ReferenceError: Cannot access 'Person' before initialization

// 클래스 선언문
class Person {}
```

클래스 선언문은 마치 호이스팅이 발생하지 않는 것처럼 보이나 그렇지 않다. 아래 예제를 살펴보자.

JAVASCRIPT

```
const Person = '';

{
  // 호이스팅이 발생하지 않는다면 ''이 출력되어야 한다.
  console.log(Person);
  // ReferenceError: Cannot access 'Person' before initialization
```

```
  // 클래스 선언문
  class Person {}
}
```

클래스 선언문도 변수 선언, 함수 정의와 마찬가지로 호이스팅이 발생한다. 단, 클래스는 let, const 키워드로 선언한 변수처럼 호이스팅된다. 따라서 클래스 선언문 이전에 일시적 사각지대(Temporal Dead Zone; TDZ)에 빠지기 때문에 호이스팅이 발생하지 않는 것처럼 동작한다.

var, let, const, function, function*, class 키워드를 사용하여 선언된 모든 식별자는 호이스팅된다. 모든 선언문은 런타임 이전에 먼저 실행되기 때문이다.

4. 인스턴스 생성

클래스는 함수로 평가된다.

JAVASCRIPT

```
class Person {}

console.log(typeof Person); // function
```

즉, 클래스는 생성자 함수이며 new 연산자와 함께 호출되어 인스턴스를 생성한다.

JAVASCRIPT

```
class Person {}

// 인스턴스 생성
const me = new Person();
console.log(me); // Person {}
```

함수는 new 연산자의 사용 여부에 따라 일반 함수로 호출되거나 인스턴스 생성을 위한 생성자 함수로 호출되지만 클래스는 인스턴스를 생성하는 것이 유일한 존재 이유이므로 반드시 new 연산자와 함께 호출하여야 한다.