Ch10 객체 리터럴

객체란?

객체란 **0개 이상의 프로퍼티로 구성된 집합이며 프로퍼티는 key 와 value 로 구성된다.**

원시 값을 제외한 나머지 값 (함수, 배열, 정규 표현식 등)은 모두 객체로 원시 값은 변경 불가능한 값 , 객체는 변경 가능한 값이다.

함수는 일급 객체이며 값으로 취급할 수 있으므로 함수도 프로퍼티 값으로 사용할 수 있다.

프로퍼티 값이 함수일 경우, 일반 함수와 구분하기 위해 method 라고 한다.

```
var counter = {
    num: 0, // -> property
    increase: function () { // -> method
        this.num++;
    }
}
```

- property: 객체의 상태를 나타내는 값
- method: 프로퍼티를 참조하고 조작할 수 있는 동작

객체 리터럴에 의한 객체 생성

클래스 기반 객체지향 언어(C++, 자바) 객체 생성 방법

클래스를 사전에 정의하고, 필요한 시점에 `new 연산자` 와 함께 생성자를 호출하여 인스턴스를 생성한다.

> **※ 인스턴스**
>
> **인스턴스**란 클래스에 의해 생성되어 메모리에 저장된 실체를 말한다. 객체지향 프로그래밍에서 객체는 클래스와 인스턴스를 포함한 개념으로 클래스는 인스턴스를 생성하기 위한 템플릿의 역할을 한다.

프로토타입 기반 객체지향 언어(자바스크립트) 객체 생성 방법

- 객체 리터럴
- Object 생성자 함수
- 생성자 함수
- Object.create 메서드
- 클래스(ES6)

객체 리터럴

- 중괄호 `({…})` 내에 n개 이상의 프로퍼티를 정의
- 변수에 할당되는 시점에 자바스크립트 엔진은 객체 리터럴을 해석해 객체를 생성
- 중괄호 내에 프로퍼티를 정의하지 않으면 빈 객체가 생성됨
- 코드 블록의 닫는 중괄호 뒤에는 세미콜론을 붙이지 않지만 **객체 리터럴**은 값으로 평가되는 표현식으로 코드 블록을 의미하지 않기 때문에 **닫는 중괄호 뒤에 세미콜론을 붙인다**.

프로퍼티

객체는 프로퍼티의 집합이며, **프로퍼티**는 **키(key)**와 **값(value)**으로 구성된다.

- key: **빈 문자열을 포함하는** 모든 문자열 또는 Symbol 값
    - 식별자 네이밍 규칙을 따르지 않는 이름에는 **반드시 따옴표를 사용해야 한다.**

    - 문자열 또는 문자열로 평가할 수 있는 표현식을 사용해 key 를 동적으로 생성할 수 있다. (**대괄호([...])**로 묶음)
    - 빈 문자열을 사용해도 에러가 발생하지 않지만, 의미를 갖지 못하므로 권장하지 않음
    - 프로퍼티 키에 문자열이나 Symbol 값 외의 값을 사용하면 **암묵적 타입 변환**을 통해 문자열이 된다.
    - var, function 같은 예약어를 사용해도 에러가 발생하지 않지만 권장하지 않는다.
    - 존재하는 프로퍼티 key 를 중복 선언하면 나중에 선언한 프로퍼티가 먼저 선언한 프로퍼티를 덮어쓰고 에러는 발생하지 않는다.
- value: 자바스크립트에서 사용할 수 있는 모든 값

메서드

자바스크립트의 함수는 객체 (일급 객체) 로 값으로 취급할 수 있어서 프로퍼티 값으로 사용할 수 있다.

프로퍼티 값이 함수일 경우 **메서드**라 부른다.

프로퍼티 접근

- **마침표 표기법 (dot notation)**
- **대괄호 표기법 (bracket notation)**

```
var person = {
  name: 'Lee'
};

// 마침표 표기법에 의한 프로퍼티 접근
console.log(person.name); // Lee

// 대괄호 표기법에 의한 프로퍼티 접근
console.log(person['name']); // Lee
```

프로퍼티 키가 식별자 네이밍 규칙을 준수하는 이름이면 마침표 표기법과 대괄호 표기법을 모두 사용할 수 있다.

대괄호 프로퍼티 접근 연산자 내부에 지정하는 프로퍼티 key 는 반드시 따옴표로 감싼 문자열이어야 한다.

객체에 존재하지 않는 프로퍼티에 접근하면 undefined 를 반환하기 때문에 ReferenceError 는 발생하지 않는다.

```
// 대괄호 프로퍼티 접근 연산자 내에 따옴표로 감싸지 않음
var person = {
  name: 'Lee'
};

console.log(person[name]); // ReferenceError: name is not defined

// 객체에 존재하지 않는 프로퍼티
var person = {
  name: 'Lee'
};

console.log(person.age); // undefined
```

프로퍼티 키가 식별자 네이밍 규칙을 준수하지 않는 이름이면 반드시 **대괄호 표기법**을 사용해야 한다.

단, 프로퍼티 키가 숫자로 이뤄진 문자열인 경우 따옴표를 생략할 수 있다.

```
var person = {
  'last-name': 'Lee',
```

```
  1: 10
};

person.'last-name';  // -> SyntaxError: Unexpected string
person.last-name;    // -> 브라우저 환경: NaN
                     // -> Node.js 환경: ReferenceError: name is not defined
person[last-name];   // -> ReferenceError: last is not defined
person['last-name']; // -> Lee

// 프로퍼티 키가 숫자로 이뤄진 문자열인 경우 따옴표를 생략할 수 있다.
person.1;     // -> SyntaxError: Unexpected number
person.'1';   // -> SyntaxError: Unexpected string
person[1];    // -> 10 : person[1] -> person['1']
person['1'];  // -> 10
```

프로퍼티 값 갱신

이미 존재하는 프로퍼티에 값을 할당하면 갱신된다.

프로퍼티 동적 생성

존재하지 않는 프로퍼티에 값을 할당하면 동적으로 생성되고 값이 할당된다.

프로퍼티 삭제

`delete` 연산자는 객체의 프로퍼티를 삭제한다. 이때 delete 연산자의 피연산자는 프로퍼티 값에 접근할 수 있는 표현식이어야 한다. 만약 존재하지 않는 프로퍼티를 삭제하면 아무런 에러 없이 무시된다.

ES6 에서 추가된 객체 리터럴의 확장 기능

프로퍼티 축약 표현

프로퍼티 값으로 변수를 사용하는 경우 변수 이름과 프로퍼티 키가 동일한 이름일 때 key 를 생략할 수 있다.

계산된 프로퍼티 이름

ES5 에서는 객체 리터럴 외부에서 대괄호 `([…])` 표기법을 사용해야 한다.

ES6는 객체 리터럴 내부에서도 계산된 프로퍼티 이름으로 동적 생성할 수 있다.

메서드 축약 표현

ES6 는 메서드를 정의할 때 `function` keyword 를 생략한 축약 표현을 사용할 수 있다.