# Ch.10 객체 리터럴

▼ 🌟🌟객체 타입

타양한 타입의 값(원시값 또는 다른 객체)을 하나의 단위로 구성한 복합적인 자료구조.

원시 타입의 값은 변경 불가능한 값이지만 객체타입의 값은 변경 가능한 값이다.

**객체는 프로퍼티와 메서드로 구성된 집합체다.**

객체는 0개 이상의 프로퍼티로 구성된 집합이며, 프로퍼티는 키와 값으로 구성된다.

프로퍼티 값이 함수일 경우 일반함수와 구분하기 위해 메서드라 부른다. 그림 10-2 p125(e159)

- **프로퍼티**: 객체의 상태를 나타내는 값(data)
- **메서드**: 프로퍼티(상태 데이터)를 참조하고 조작할 수 있는 동작(behavior)

객체는 상태와 동작을 하나의 단위로 구조화 할 수 있어 유용하다.

객체의 집합으로 프로그램을 표현하려는 프로그래밍 패러다임을 객체지향 프로그래밍이라 한다.

▼ 🌟🌟인스턴스

**클래스에 의해 생성되어 메모리에 저장된 실체를 말한다.**

객체지향 프로그래밍에서 객체는 클래스와 인스턴스를 포함한 개념이다.

클래스는 인스턴스를 생성하기 위한 탬플릿의 역할을 한다.

인스턴스는 객체가 메모리에 저장되어 실제로 존재하는 것에 초점을 맞춘 용어다.

▼ 🌟🌟자바스크립트의 객체 생성 방법

자바스크립트는 프로토타입 기반 객체지향 언어로서 클래스 기반 객체지향 언어와는 달리 다양한 객체 생성 방법을 지원한다.

- **객체 리터럴**

    - { }내에 0개 이상의 프로퍼티를 정의하니다. 변수에 할당되는 시점에 자바스크립트 엔진은 객체 리터럴을 해석해 객체를 생성한다.
    - 객체 리터럴의 중괄호는 코드블록을 의미하지 않는다. 객체 리터럴은 값으로 평가되는 표현식이다. 따라서 객체 리터럴의 닫는 중괄호 뒤에는 세미콜론을 붙인다.
    - 객체를 생성한 이후에 프로퍼티를 동적으로 추가할 수도 있다.
    - 객체 리터럴 외의 객체 생성 방식은 모두 함수를 사용해 객체를 생성한다.
- Object 생성자 함수
- 생성자 함수
- Object.create 메서드
- 클래스(ES6)

▼ 🌟🌟프로퍼티

프로퍼티를 나열할 때는 쉼표로 구분한다.

**프로퍼티 키**: 빈 문자열을 포함하는 모든 문자열 또는 심벌 값. 일반적으로 문자열 사용.

- 프로퍼티 키는 프로퍼티 값에 접근할 수 있는 이름으로서 식별자 역할을 한다.
- 프로퍼티 키는 문자열이므로 ' ', " " 로 묶어야 하지만 식별자 네이밍 규칙을 준수하는 이름, 즉 자바스크립트에서 사용 가능한 유효한 이름의 경우 따옴표를 생략할 수 있다. 식별자 네이밍 규칙을 따르지 않는 이름에는 반드시 따옴표를 사용해야 한다.
- 문자열 또는 문자열로 평가할 수 있는 표현식을 사용해 프로퍼티 키를 동적으로 생성할 수도 있다. 이 경우에는 프로퍼티 키로 사용할 표현식을 대괄호[ ]로 묶어야 한다. p128(e162)

  예제 10-06

```
var obj = { };
var key = 'hello';

// ES5: 프로퍼티 키 동적 생성
obj[key] = 'world';
// ES6: 계산된 프로퍼티 이름
```

```
var obj = {[key]: 'world'};

console.log(obj); // {hello: "world"}
```

- 프로퍼티 키에 문자열이나 심벌 값 외의 값을 사용하면 암묵적 타입 변환을 통해 문자열이 된다.
- 이미 존재하는 프로퍼티 키를 중복 선언하면 나중에 선언한 프로퍼티가 먼저 선언한 프로퍼티를 덮어쓴다.

**프로퍼티 값**: 자바스크립트에서 사용할 수 있는 모든 값

- 프로퍼티 값이 함수일 경우 일반 함수와 구분하기 위해 메서드라 부른다. 즉, 메서드는 객체에 묶에 였는 함수를 의미한다.

+**메서드 내부에서 사용한 this키워드는 객체자신을 가리키는 참조변수다.** (circle 객체)

p130(e164) 예제 10-11

```
var circle = {
    radius: 5,
    getDiameter: function (){
        return 2 * this.radius;
    }
};
```

▼ 🌟🌟프로퍼티에 접근하는 방법

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

**객체에 존재하지 않는 프로퍼티에 접근하면 undefined를 반환한다.** (ReferenceError가 발생하진 않는다. )

**마침표 표기법** (마침표 프로퍼티 접근 연산자 . )

**대괄호 표기법** (대괄호 프로퍼티 접근 연산자 [ ] )

- 대괄호 프로퍼티 접근 연산자 **내부에 지정하는 프로퍼티 키는 반드시 따옴표로 감싼 문자열**이어야 한다. p131(e165) 예제 10-12

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

console.log(person.name); // Lee

console.log(person['name']); // Lee
```

  대괄오 프로퍼티 접근 연산자 내의 따옴표로 감싸지 않은 이름은 자바스크립트 엔진이 식별자로 해석한다.

- 프로퍼티 키가 네이밍 규칙을 준수하지 않는 이름은 반드시 대괄호 표기법을 사용해야 한다.
- 프로퍼티 키가 숫자로 이뤄진 문자열인 경우 따옴표를 생략할 수 있다.

▼ 프로퍼티 동적 생성

존재하지 않는 프로퍼티에 값을 할당하면 프로퍼티가 동적으로 생성되어 추가되고 프로퍼티 값이 할당된다. p133(e167) 예제 10-17

▼ 프로퍼티 삭제

delete 연산자는 객체의 프로퍼티를 삭제한다.

▼ 🌟🌟ES6에서 추가된 객체 리터럴의 확장 기능

- 프로퍼티 축약 표현 p134(e168) 예제 10-19, 10-20
  - 프로퍼티 값은 변수에 할당된 값, 즉 식별자 표현식일 수도 있다. ES6에서는 프로퍼티 값으로 변수를 사용하는 경우 **변수 이름과 프로퍼티 키가 동일한 이름일 때 프로퍼티 키를 생략할 수 있다.** **이때 프로퍼티 키는 변수 이름으로 자동생성된다.**

```
var x =1, y =2;

var obj = {
    x: x,
    y: y
};

console.log(obj);  // {x:1, y:2}
```

```
let x =1, y =2;

const obj = {x,y};   // 키가 생략되고 프로퍼티 값만 있는 싱

console.log(obj);  // {x:1, y:2}
```

- 계산된 프로퍼티 이름 p135(e169)
  - 문자열 또는 문자열로 타입 변환할 수 있는 값으로 평가되는 표현식을 사용해 프로퍼티 키를 동적으로 생성 할 수도 있다. 단, 프로퍼티 키로 사용할 표현식을 대괄호로 묶어야 한다. 이를 계산된 프로퍼티 이름이라 한다.

    예제 10-22

```
const prefix = 'prop';
let i = 0;

const obj = {
    [`${prefix}-${++i}`]: i,
    [`${prefix}-${++i}`]: i,
    [`${prefix}-${++i}`]: i
}

console.log(obj);  // {prop-1: 1, prop-2: 2, prop-3:
```

- 메서드 축약 표현 p136(e170)

- ES6에서는 메서드를 정의할 때 function키워드를 생략한 축약 표현을 사용할 수 있다.

  예제 10-24

  ```
  const obj = {
      name: 'Lee',

      sayHi() {
          console.log('Hi! ' + this.name);
      }
  };

  obj.sayHi();
  ```

- 메서드 축약 표현으로 정의한 메서드는 프로퍼티에 할당한 함수와 다르게 동작한다.