객체 리터럴 프로퍼티 기능 확장

ES6에서는 객체 리터럴 프로퍼티 기능을 확장하여 더욱 간편하고 동적인 객체 생성 기능을 제공한다.

프로퍼티 축약 표현

ES5에서 객체 리터럴의 프로퍼티는 프로퍼티 이름과 프로퍼티 값으로 구성된다. 프로퍼티의 값은 변수에 할당된 값일 수도 있다.

```
// ES5
var x = 1, y = 2;

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

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

ES6에서는 프로퍼티 값으로 변수를 사용하는 경우, 프로퍼티 이름을 생략(Property shorthand)할 수 있다. 이때 프로퍼티 이름은 변수의 이름으로 자동 생성된다.

```
// ES6
let x = 1, y = 2;

const obj = { x, y };

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

프로퍼티 키 동적 생성

문자열 또는 문자열로 변환 가능한 값을 반환하는 표현식을 사용해 프로퍼티 키를 동적으로 생성할 수 있다. 단, 프로퍼티 키로 사용할 표현식을 대괄호([...])로 묶어야 한다. 이를 계산된 프로퍼티 이름(Computed property name)이라 한다.

ES5에서 프로퍼티 키를 동적으로 생성하려면 객체 리터럴 외부에서 대괄호([...]) 표기법을 사용해야 한다.

```
// ES5
var prefix = 'prop';
var i = 0;

var obj = {};

obj[prefix + '-' + ++i] = i;
obj[prefix + '-' + ++i] = i;
obj[prefix + '-' + ++i] = i;

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

ES6에서는 객체 리터럴 내부에서도 프로퍼티 키를 동적으로 생성할 수 있다.

```
// ES6
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: 3}
```

메소드 축약 표현

ES5에서 메소드를 선언하려면 프로퍼티 값으로 함수 선언식을 할당한다.

```
// ES5
var obj = {
  name: 'Lee',
  sayHi: function() {
    console.log('Hi! ' + this.name);
  }
};

obj.sayHi(); // Hi! Lee
```

ES6에서는 메소드를 선언할 때, function 키워드를 생략한 축약 표현을 사용할 수 있다.

```
// ES6
const obj = {
  name: 'Lee',
  // 메소드 축약 표현
  sayHi() {
    console.log('Hi! ' + this.name);
  }
};

obj.sayHi(); // Hi! Lee
```

__proto__ 프로퍼티에 의한 상속

ES5에서 객체 리터럴을 상속하기 위해서는 Object.create() 함수를 사용한다. 이를 프로토타입 패턴 상속이라 한다.

```
// ES5
var parent = {
  name: 'parent',
  sayHi: function() {
    console.log('Hi! ' + this.name);
```

```
  }
};

// 프로토타입 패턴 상속
var child = Object.create(parent);
child.name = 'child';

parent.sayHi(); // Hi! parent
child.sayHi();  // Hi! child
```

ES6에서는 객체 리터럴 내부에서 __proto__ 프로퍼티를 직접 설정할 수 있다. 이것은 객체 리터럴에 의해 생성된 객체의 __proto__ 프로퍼티에 다른 객체를 직접 바인딩하여 상속을 표현할 수 있음을 의미한다.

```
// ES6
const parent = {
  name: 'parent',
  sayHi() {
    console.log('Hi! ' + this.name);
  }
};

const child = {
  // child 객체의 프로토타입 객체에 parent 객체를 바인딩하여 상속을
구현한다.
  __proto__: parent,
  name: 'child'
};

parent.sayHi(); // Hi! parent
child.sayHi();  // Hi! child
```