객체:기본

메서드와 this

- 자바스크립트에선 객체의 **프로퍼티에 함수를 할당**해 객체에게 행동할 수 있는 능력을 부여
  - **객체 지향 프로그래밍 => 정보 + 행동**
  - 자바스크립트에서는 **정보(값)와 행동(함수) 모두 속성에 정의**

메서드 만들기

- **객체 속성에 할당된 함수를 메서드(method)**라고 부름

```
let user = {
    name: "John",
    age: 30
};

// sayHi 속성에 함수 할당 (=> 메소드 정의)
user.sayHi = function() {
    alert("안녕하세요!");
};

// 메소드 호출
user.sayHi();
```

- 다음과 같이 **기존에 정의된 함수를 객체의 메소드로 등록**하는 것도 가능
  - 다른 언어와 같이 **함수와 메소드가 서로 분리된 개념이 아님**

```
let user = {
    // ...
};

// 함수 선언
function sayHi() {
    alert("안녕하세요!");
};

// 선언된 함수를 메서드로 등록
user.sayHi = sayHi;
```

- 다음과 같이 객체 내부에 속성을 정의하는 동시에 곧바로 메소드 정의 가능

```
let user = {
    // 속성 정의하며 곧바로 메소드 정의
    sayHi: function() {
        alert("Hello");
    }
};
```

- 위와 똑같은 역할을 하는 **단축 구문을 사용 가능** (콜론, function 키워드 안 쓰고 메서드 정의)

```
let user = {
    // 단축 구문 이용하여 메소드 정의
    sayHi() {
        alert("Hello");
    }
};
```

메서드와 this

- (100%는 아니지만) 보통 메서드에는 **현재 객체의 속성값에 접근하거나 변경하는 코드**가 포함됨
    - 따라서 메서드 내부에서 객체를 참조할 방법이 필요함
    - **this 키워드를 사용하면 객체에 접근** 가능

```
let user = {
    name: "John",
    age: 30,
    sayHi() {
        // 이 예제에서 this는 "현재 객체"를 나타냄
        alert(this.name);
    }
};

// John 출력
user.sayHi();
```

자유로운 this (*)

- 다른 객체지향 언어(ex: Java, C++)와는 다르게 **this 값은 메소드가 호출되는 맥락에 따라 바뀔 수 있음**
    - 즉, **this 값은 코드를 실행하며 함수가 호출되는 시점에 유연하게 결정**될 수 있음
    - 기본적으로는 **메소드 호출 시점의 점(.) 앞 객체를 참조**함

```
let user = { name: "John" };
let admin = { name: "Admin" };

function sayHi() {
    alert( this.name );
}

user.f = sayHi;
admin.f = sayHi;
```

```
// this 값은 기본적으로 "메소드 호출 시점에 점(.) 앞 객체"를 참조
user.f(); // "John" 출력 (메소드 호출 시점에 점 앞의 객체는 user 따라서, this는 user)
admin.f(); // "Admin" 출력 (메소드 호출 시점에 점 앞의 객체는 admin 따라서, this는 admin)

// 대괄호를 통해서도 메소드 접근 가능
admin['f']();
```

- 만약 **특별한 맥락없이(=점 앞에 아무 객체도 없는 상태) 메소드를 호출할 경우, 엄격 모드 적용 여부에 따라 다음과 같이 다른 결과가 발생**함

엄격 모드인 경우

```
"use strict" // 엄격 모드를 적용하기 위해서 소스 코드 첫 줄에 "use strict" 문자열 사용

function sayHi() {
    alert(this);
}

// undefined (메소드를 호출하는 주체가 없다고 해석)
sayHi();
```

엄격 모드가 아닌 경우

```
function sayHi() {
    console.log(this);
}

// 어쨌든 this 값 바인딩이 필요하고 따로 주체가 없으므로 글로벌 객체로 설정 (브라우저에서는 window 객체)
sayHi();
```

- 자바스크립트에서 **this는 런타임에 결정**됨
    - 메서드가 **어디에 정의되었는지에 상관없이 this는 점 앞의 객체가 무엇인가에 따라 자유롭게 결정**됨
    - 다른 언어(ex: Java)에서는 this가 메서드가 정의된 객체를 가리키게 되는데 이러한 this를 **"bound this"**라고 함 (추후에 배울 bind 메서드를 사용하면 자바스크립트에서도 this 값을 고정 가능)

```
"use strict"

let obj = {
    x: 100,
    logThis() {
        console.log(this);
    }
};
obj.logThis(); // { x: 100 }

let another = { y: 200 };
another.logThis = obj.logThis;
another.logThis(); // { y: 200 }

let logThis = another.logThis;
```

```
logThis(); // Window 객체 (엄격 모드가 적용된 경우 undefined)

let o = {
    inner: {
        z: 300
    }
};

o.inner.logThis = obj.logThis;
// 여기서 점 앞의 객체는 o가 아닌 inner임을 유의
o.inner.logThis() // { z: 300 }
```

new 연산자와 생성자 함수

- **new 연산자와 생성자 함수를 사용**하면 유사한 객체 여러 개를 쉽게 생성 가능

생성자 함수 (*)

- 생성자 함수도 그냥 함수와 별반 다르지 않음, 그러나 보통 다음의 관례를 따름 (실무에서는 반드시 따라서 지켜야 함)

```
1. 함수 이름의 "첫 글자는 대문자"로 시작
2. 함수를 호출하며 "new 연산자"를 붙여서 호출
```

생성자 함수 정의 및 활용 코드

```
// 함수의 첫 글자는 대문자
function User(name) {
    // (new 연산자와 함께 함수 호출 시) 내부적으로 다음과 같은 코드가 동작
    // this = {}

    // 새로운 속성을 this에 추가 (생성자 역할 수행)
    this.name = name;
    this.isAdmin = false;

    // (new 연산자와 함께 함수 호출 시) 내부적으로 다음과 같은 코드가 동작
    // return this
}

// new 연산자와 함께 함수 호출
let user = new User("Jack");

alert(user.name); // Jack
alert(user.isAdmin); // false
```

- `new 생성자함수(인자값)` 호출 시 다음 순서로 동작

```
1. 빈 객체를 만들어 this 값에 할당
2. 함수 본문을 실행하며 this 객체에 새로운 속성을 추가하여 객체 구성
3. this 값을 반환
```

- **생성자 함수는 재사용할 수 있는, 객체 생성 및 구성 코드**가 담긴 함수
- 생성자 함수에서는 **자동으로 this 값이 반환되므로 return 문이 불필요**
- 전달할 인수값이 없는 생성자 함수는 괄호를 생략해서 호출 가능

```
// 생성자 함수로 전달할 인수가 없으므로 괄호 없이 호출 가능
let user = new User;
// 아래 코드는 위 코드와 똑같이 동작
let user = new User();
```

- 생성자에서 메서드 추가 가능

```
function User(name) {
    this.name = name;

    // 생성자에서 메서드 추가
    this.sayHi = function() {
        alert( "My name is: " + this.name );
    };
}

let john = new User("John");

// My name is: John
john.sayHi();

let sam = new User("Sam");

alert( john.sayHi === sam.sayHi ); // false, 서로 다른 참조를 가진 함수 객체
```

- **class 키워드**를 이용해서 생성자 함수를 이용한 객체 생성을 대체할 수 있음 (단, class 키워드는 ES6에서 추가된 새로운 문법)