17. 생성자 함수에 의한 객체 생성

▼ 객체 리터럴과 생성자 함수

유사한 객체를 여러개 생성할 시, 같은 코드를 반복해서 작성해야 하는 문제가 생긴다.

이때 생성자 함수를 사용한다.

생성자 함수 convention

- 함수 이름의 첫 글자는 대문자로 시작한다.
- new 연산자를 붙여 실행한다.

```
// 예시
function Creator(name, age) {
    this.name = name;
    this.age = age;
}

const instance = new Creator('kim', 1000000)
```

▼ 생성자 함수의 동작 방식.

1. 빈 객체를 만들어 `this`에 할당한다.
2. 함수 본문을 실행하면서 `this`에 새로운 프로퍼티를 추가해 `this`를 수정한다.
3. 생성된 `this`를 반환한다.

▼ 재사용 필요가 없는 복잡한 객체 생성시

바로 변수에 할당하나, new 연산자와 익명함수를 같이 사용한다.

```
let user = new function() {
    this.name = 'name';
    this.isAdmin = true;
    isUser() {
        return !this.isAdmin;
```

```
    }
    //...생략
}
```

▼ new.target

new.target을 사용하면, 함수가 `new`와 함께 호출되었는지 확인할 수 있다.

`new`와 함께 호출되지 않았다면 new.target은 `undefined`가 된다.

이를 이용해 생성자 함수는 `new`와 함께 호출된 것처럼 동작하게 하여 human error를 방지할 수 있다.

```
function Person(name) {
    if(!new.target) {
        return new Person(name)
    }

    this.name = name;
}
```

▼ 생성자함수의 return문

보통 return문이 없다.

반환되어야 할 내용은 모두 this에 저장되고 자동으로 this를 반환하기 때문에 명시할 필요가 없다.

다만, return문이 있다면 다음과 같이 동작한다.

- 객체를 return 하면, this대신 객체가 반환된다.
- 원시값을 return 하면, return문이 무시되고 this를 return 한다.