클래스

private, protected 속성과 메서드

- Java 언어와 같이 따로 접근 제어자가 있는 것은 아니고 비슷한 용도로 활용할 수 있는 기법이 존재

속성과 메서드 보호하기

속성값 접근을 마음대로 허용하는 코드

```
class CoffeeMachine {
    // (직접 접근을 막고 싶은) 속성 정의
    waterAmount = 0;

    constructor(power) {
        this.power = power;
        alert( `전력량이 ${power}인 커피머신을 만듭니다.` );
    }
}

// 커피 머신 생성
let coffeeMachine = new CoffeeMachine(100);

// 직접적으로 속성값 변경 가능 (이러한 직접적인 대입을 피하고자 하는 상황)
coffeeMachine.waterAmount = 200;
```

- protected 제어자를 붙이고자 하는 속성과 메서드 앞에는 밑줄(_)을 붙여주기
    - (강제는 아니지만) 밑줄은 프로그래머들 사이에서 **외부 접근이 불가능한 속성이나 메서드**임을 알려주기 위해서 주로 사용됨

세터, 게터를 활용한 직접 접근 피하기

```
class CoffeeMachine {
    // 밑줄을 앞에 붙여서 속성 이름 정의
    _waterAmount = 0;

    // 원래 쓰고자 했던 속성 이름(waterAmount)으로 세터, 게터 함수 정의
    set waterAmount(value) {
        if (value < 0) throw new Error("물의 양은 음수가 될 수 없습니다.");
        this._waterAmount = value;
    }
    get waterAmount() {
        return this._waterAmount;
    }

    constructor(power) {
        this._power = power;
    }
}
```

```
// 커피 머신 생성
let coffeeMachine = new CoffeeMachine(100);

// 물 추가
// Error: 물의 양은 음수가 될 수 없습니다.
coffeeMachine.waterAmount = -10;
```

private 속성 및 메서드

- private 제어자를 붙이고자 하는 속성과 메서드 앞에는 샵(#)을 붙여주기
    - 이후 속성과 메서드는 클래스 내부에서만 접근할 수 있음 (이것은 문법적으로 강제되는 사항)

```
class CoffeeMachine {
    #waterLimit = 200;

    #checkWater(value) {
        if (value < 0) throw new Error("물의 양은 음수가 될 수 없습니다.");
        // private 프로퍼티는 클래스 내부에서만 접근 가능
        if (value > this.#waterLimit) throw new Error("물이 용량을 초과합니다.");
    }
}

let coffeeMachine = new CoffeeMachine()
// Uncaught SyntaxError: Private field '#waterLimit' must be declared in an
enclosing class
console.log( coffeeMachine.#waterLimit );
```

instanceof 연산자로 클래스 확인하기

- instanceof 연산자를 사용하면 **객체가 특정 클래스에 속하는지 아닌지를 확인할 수 있으며 이를 통해 상속 관계도 확인 가능**함

instanceof 연산자

- 특정 클래스의 객체인지 확인 가능

```
class Rabbit {}
let rabbit = new Rabbit();

// 왼쪽 객체가 클래스에 속하거나 해당 클래스를 상속받는 클래스에 속하면 true가 반환됨
alert( rabbit instanceof Rabbit ); // rabbit이 클래스 Rabbit의 객체? => true
```

- 객체의 상속 관계도 확인 가능

```
class Animal {}
class Rabbit extends Animal {}
let rabbit = new Rabbit();

console.log( rabbit instanceof Rabbit ); // true
console.log( rabbit instanceof Animal ); // true (상속 관계에 있는 모든 클래스에 대해서 true)
```

- instanceof 연산자는 **생성자 함수를 대상으로도 사용 가능** (클래스의 constructor => 생성자 함수)

```
// 클래스가 아닌 생성자 함수 정의
function Rabbit() {}

// 결과는 같음
let rabbit = new Rabbit();
alert( rabbit instanceof Rabbit ); // true
```

- 내장 클래스(Array, Object 등)에도 사용 가능

```
let arr = [1, 2, 3];
alert( arr instanceof Array );  // true
alert( arr instanceof Object ); // true (해당 문장은 모든 객체에 대해서 true를 반환)
```