06. Class Object-Oriented Language

1. class

- template
- declare once
- no data in

2. object

- instance of a class
- created many times
- data in

```
'use strict';
// Object-oriented programming
// class: template
// object: instance of a class
// JavaScript classes
// - introduced in ES6
// - syntactical sugar over prototype-based inheritance
```

(1) Class declarations

```
class Person {
  //constructor
  constructor(name, age) {
    //fields
    this.name = name;
    this.age = age;
  }

  // methods
  speak() {
    console.log(`${this.name}: hello!`);
  }
}

const smart = new Person('smart', 20)
console.log(smart.name);
console.log(smart.age);
smart.speak();
```

(2) getter ans setters

```
class User {
  constructor(firstNmae, lastName, age) {
    this.firstNameName = firstNmae;
    this.lastName = lastName;
    this.age = age;
  }
  get age() {
    return this._age;
  }

  set age(value) {
    // if (value < 0) {
    //   throw Error('age can not be negative!!!');
    // }
    //this._age = value;
    this._age = value < 0 ? 0 : value;
  }
}

const user1 = new User('Steve', 'Job', -1);
console.log(user1.age);
```

(3) Fields (public, private)

```
// Too soon!
// https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
class Experiment {
  publicField = 2;
  #privateField = 0;
}
const experiment = new Experiment();
console.log(experiment.publicField);
console.log(experiment.privateField);
```

(4) Static properties and methods

```
// Too soon!
// class 안에 있는 fields와 methods들은 새로운 object를 만들 때마다
// 그대로 복제되어서 값만 지정되어 만들어 진다.
// 그러나 object에 상관없이 class가 가지고 있는 고유한 값과 이런한 데이터와
// 상관없이 동일하게 반복적으로 사용되어지는 method가 있을 경우 static이라는
// 키워드를 붙이면 object와 상관없이 class 자체에 연결되어 있다.
class Article {
  static publisher = 'smart Coding';
  constructor(articleNumber) {
    this.articleNumber = articleNumber;
```

```
  }

  static printPublisher() {
    console.log(Article.publisher);
  }
}

const article1 = new Article(1);
const article2 = new Article(2);
// 만약 static을 사용하지 않았다면 아래 코드는 실행되었을 것이다.
// 그러나 실행하면 undefined 가 나온다.
// static은 object 마다 할당되어 지는 것이 아니라
// class Article에 붙어있기 때문이다.
//console.log(article1.publisher);
console.log(Article.publisher);
// 따라서 static 함수를 호출할 때에도 class 이름을 이용하여 호출하면 된다.
// static은 object에 들어오는 데이터에 상관없이 공통적으로 class에서
// 쓸수 있는 것이라면 static과 static method를 이용하여 작성하는 것이
// 메모리의 사용을 조금 더 줄여줄 수 있다.
Article.printPublisher();
```

(5) Inheritance & polymorphism

```
// a way for one class to extend another class.
class Shape {
  constructor(width, height, color) {
    this.width = width;
    this.height = height;
    this.color = color;
  }

  draw() {
    console.log(`drawing ${this.color} color of`)
    //console.log(`drawing ${this.color} color!!`)
  }

  getArea() {
    return this.width * this.height;
  }
}

// Inheritance
class Rectangle extends Shape { }
class Triangle extends Shape {
  // polymorphism
  draw() {
    super.draw();
    console.log('□');
  }
  // method overriding
```

```
  getArea() {
    return (this.width * this.height) / 2;
  }

  toString() {
    return `Triangle: color: ${this.color}`;
  }
}

const rectangle = new Rectangle(20, 20, 'blue');
rectangle.draw();
console.log(rectangle.getArea());
const triangle = new Triangle(20, 20, 'red');
triangle.draw();
console.log(triangle.getArea());
```

(6) Class checking: instanceof

```
console.log(rectangle instanceof Rectangle);  // true
console.log(triangle instanceof Rectangle);   // false
console.log(triangle instanceof Triangle);    // true
console.log(triangle instanceof Shape);       // true
console.log(triangle instanceof Object);      // true
console.log(triangle.toString());
```