04

04-5. 클래스

JavaScript Object Oriented Programming

선언 및 생성

- 객체의 모형을 선언하고 그 모형을 이용해 객체를 만드는 방법은 우리가 이미 살펴본 생성자 함수를 이용하는 방법이 있고 클래스를 이용하는 방법이 있습니다.
- 생성자 함수를 이용하는 방법은 자바스크립트 초기부터 제공하던 전통적인 방법이며 클래스를 이용하는 방법은 ECMA2015부터 추가된 방법입니다.
- ECMA2015에서 대부분의 소프트웨어 언어에서 객체지향을 위해 제공하는 클래스 방식을 지원하기 시작한 것입니다.

선언 및 생성

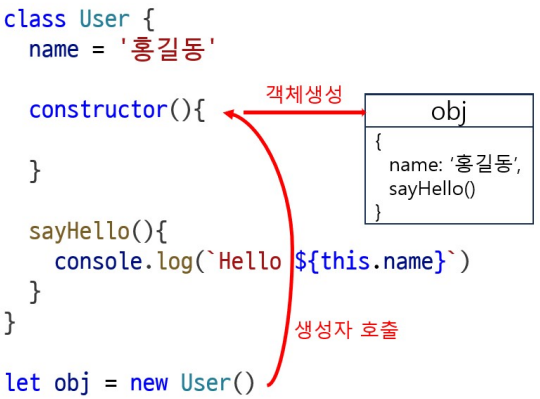
- 동일한 멤버로 구성되는 객체를 여러 개 만들기 위해서는 우선 객체의 모형을 선언해야 하는데 이 모형이 클래스입니다.
- 클래스는 class 라는 예약어로 선언되며 이 클래스내에 객체의 멤버인 프로퍼티와 함수를 선언합니다.
- 그리고 클래스를 바탕으로 객체를 생성하기 위해서는 new 연산자를 이용합니다.

클래스 선언 및 객체 생성

```
1   class User {
2     name = '홍길동'
3     sayHello(){
4       console.log(`Hello ${this.name}`)
5     }
6   }
7
8   let obj = new User()
```

생성자

- 클래스는 생성자, 프로퍼티, 메서드로 구성됩니다.
- 이중 생성자는 객체 생성시에 호출되어 클래스의 프로퍼티, 메서드를 메모리에 할당하는 역할을 합니다.
- 객체 생성을 위해 new 연산자를 이용할 때 호출되는 것은 생성자이며 클래스가 객체를 생성하기 위한 모형임으로 생성자를 가지지 않는 클래스는 존재할 수 없습니다.
- 그런데 만약 개발자가 코드에서 클래스를 선언하면서 명시적으로 생성자를 추가하지 않았다면 자동으로 기본 생성자(Default Constructor) 가 추가되게 됩니다.
- 기본 생성자는 매개변수를 가지지 않는 생성자입니다.



생성자

- 명시적으로 추가한다면 매개변수를 가지는 생성자를 추가할 수도 있습니다.
- 클래스내에 constructor() 로 생성자를 정의할 수 있는데 하나의 클래스내에 하나의 생성자만 추가할 수 있습니다. 즉 매개변수를 다르게 해서 여러 개의 생성자를 추가할 수는 없습니다.
-

매개변수를 가지는 생성자

```
1   class User {
2     name = '홍길동'
3
4     constructor(name){
5       this.name = name
6     }
7
8     sayHello(){
9       console.log(`Hello ${this.name}`)
10    }
11  }
12
13  let obj = new User('김길동')
14  obj.sayHello()//Hello 김길동
```

객체 멤버

- 객체가 가져야 하는 프로퍼티와 메서드가 클래스내에 선언되어 있어야 하고 실제 객체가 생성되면 선언된 프로퍼티와 메서드가 객체의 메모리에 할당되게 됩니다.

- 클래스에 멤버를 선언할 때는 프로퍼티의 경우 생성자내에서 "this.프로퍼티명" 형태로 선언하고 메서드의 경우 생성자 밖에 클래스 영역에 선언하는 것이 일반적입니다.
- 클래스 내에 멤버를 추가할 때는 일반적으로 변수를 선언하기 위한 let, var, const 예약어와 함수를 선언하기 위한 function 예약어는 사용할 수 없습니다.

클래스 멤버 선언

```
1   class User {
2     constructor(name, age) {
3       this.name = name
4       this.age = age
5     }
6     sayHello(){
7       console.log(`Hello ${this.name}, age = ${this.age}`)
8     }
9   }
10
11  let obj = new User('홍길동', 20)
```

객체 멤버

- 프로퍼티를 생성자에서 선언하지 않고 클래스 영역에서 선언할 수도 있습니다.
- 함수를 this.sayHello1 으로 생성자 내에서 선언할 수도 있습니다.
- 일반 메서드 내에서 클래스 멤버를 this.address 처럼 선언할 수도 있습니다.
- 객체 생성후에 클래스 외부에서 멤버를 추가할 수도 있습니다.

```
class User {
  name
  constructor(name, age) {
    this.name = name
    this.age = age
    this.sayHello1 = function(){
      console.log(`Hello1 ${this.name}, age = ${this.age}`)
    }
  }
  sayHello2(){
    this.address = 'seoul'
    console.log(`Hello2 ${this.name}, age = ${this.age}`)
  }
}

let obj = new User('홍길동', 20)
obj.phone = '0101111'
```

private 멤버

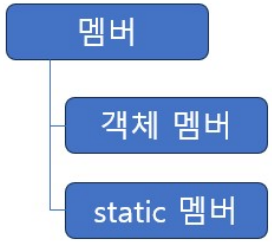
- 클래스 내부에서만 사용되는 멤버는 대부분의 소프트웨어 언어에서 private 예약어를 추가하는 방식으로 선언합니다.
- private 같은 예약어에 의해 멤버의 사용 범위가 제한된다는 의미에서 용어로 "접근제한자"라고 합니다.
- 하지만 자바스크립트에서는 일반 소프트웨어 언어에서 제공하는 접근제한자를 제공하지 않습니다.
- 대신 클래스 내부에 선언된 멤버가 외부에서 사용되지 않게 하려면 이름을 #으로 시작하는 이름을 지정해 주면 됩니다.
- #으로 시작하는 이름의 멤버는 클래스 영역에 멤버가 선언될 때만 사용할 수 있습니다. 생성자내에서 멤버를 선언할때는 사용할 수 없습니다.

private 멤버 선언

```
1   class User {
2     #name
3     age
4     constructor(name, age){
5       this.#name = name
6       this.age = age
7     }
8     #myFun(){
```

static 멤버

- 클래스에 선언된 멤버는 객체 멤버와 static 멤버로 구분됩니다.



- static 멤버는 static 예약어로 선언된 프로퍼티, 메서드이며 객체 멤버는 static 예약어로 선언되지 않은 프로퍼티, 메서드입니다.
- 클래스는 객체를 만들기 위한 모형이며 클래스를 이용해 객체를 생성한 후 객체를 이용해 멤버를 이용합니다.
- 그런데 static 멤버는 클래스내에 선언되기는 하지만 객체로 이용되기 위한 멤버가 아닙니다.
- 그럼으로 static 멤버는 객체 생성 없이 클래스명으로 이용이 가능합니다.

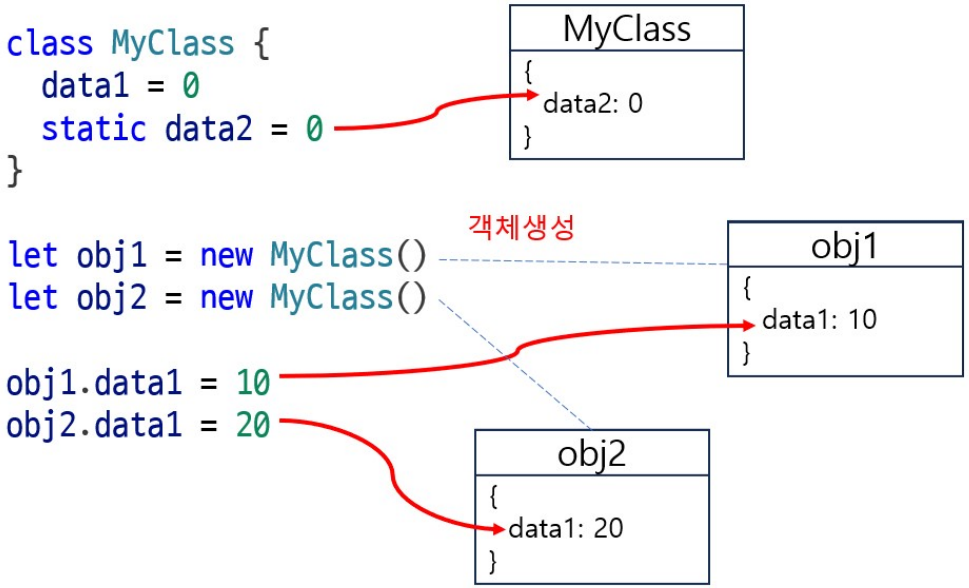
static 멤버

```
class MyClass {
  data1 = 10
  static data2 = 20

  myFun1() {
    console.log('myFun1 call..')
  }
  static myFun2() {
    console.log('myFun2 call..')
  }
}
```

```
console.log(MyClass.data2)//20

// MyClass.myFun1()//error
console.log(MyClass.data1)//undefined

let obj = new MyClass()
obj.myFun1()//myFun1 call..
console.log(obj.data1)//10

// obj.myFun2()//error
console.log(obj.data2)//undefined
```

static 멤버

- 객체 멤버는 클래스로 생성되는 객체를 위한 멤버임으로 객체 생성시마다 그 객체의 메모리에 할당되는 멤버입니다.
- 하지만 static 멤버는 객체를 위한 멤버가 아님으로 객체가 몇 개가 생성된다고 하더라도 하나의 단일 메모리에 할당되게 됩니다.



상속

- 객체지향에서 상속은 중요한 기법입니다.
- 공통적인 코드를 한곳에 작성하고 이를 상속받아 재사용하는 기법입니다.

- 생성자 함수를 이용해 객체지향 프로그래밍을 한다면 프로토타입을 이용해 상속을 구현합니다.

- 그런데 대부분의 소프트웨어 언어에서는 클래스로 객체지향 프로그램을 하며 extends 예약어로 상속관계를 명시해서 작성합니다.
- 자바스크립트에서 객체의 모형을 클래스로 만들었다면 다른 소프트웨어 언어와 마찬가지로 extends 예약어로 다른 클래스를 상속받아 작성할 수 있습니다.

```
class Rect extends Shape{
  width = 0
  height = 0
}
```

상속

private 멤버, static 멤버 상속

- 상위에 선언된 private 멤버, 즉 #으로 시작하는 이름의 멤버는 하위 클래스에 상속되지 않습니다.
- private 이라는 개념이 멤버를 자신의 클래스내에서만 사용하고자 하는 것임으로 상속관계로 명시했다고 하더라도 하위 클래스에서 상위 클래스의 #으로 시작하는 이름의 멤버는 사용할 수 없습니다.
- 그리고 상위의 멤버가 static 으로 선언되어 있다면 이 멤버는 하위에 상속되어 하위 클래스에 선언한 것처럼 사용할 수 있습니다.

```
class Super {
  data1 = 10
  #data2 = 20
  static data3 = 30
}

class Sub extends Super{
  static data4 = 40
  subFun(){
    console.log(this.data1)//10
    // console.log(this.#data2)//error
  }
}
```

```
let obj = new Sub()
obj.subFun()

console.log(Super.data3)//30
console.log(Sub.data3)//30
console.log(Sub.data4)//40
```

상속

super 로 상위 생성자 호출

- 자바스크립트 포함 모든 소프트웨어 언어에서 객체지향에 공통적으로 사용되는 예약어가 this, super 입니다.
- this 는 객체 자신을 지칭하는 예약어이며 super 는 상위 클래스를 지칭하는 예약어입니다.
- super의 사용은 하위 생성자에서 명시적으로 상위 생성자를 호출해야 하는 경우에 사용되거나 아니면 하위 클래스에서 상위에 선언된 멤버를 지칭할 때 사용됩니다.

- 객체를 생성하기 위해서 생성자를 호출하면 상위 클래스의 생성자까지 호출이 되어야 합니다.

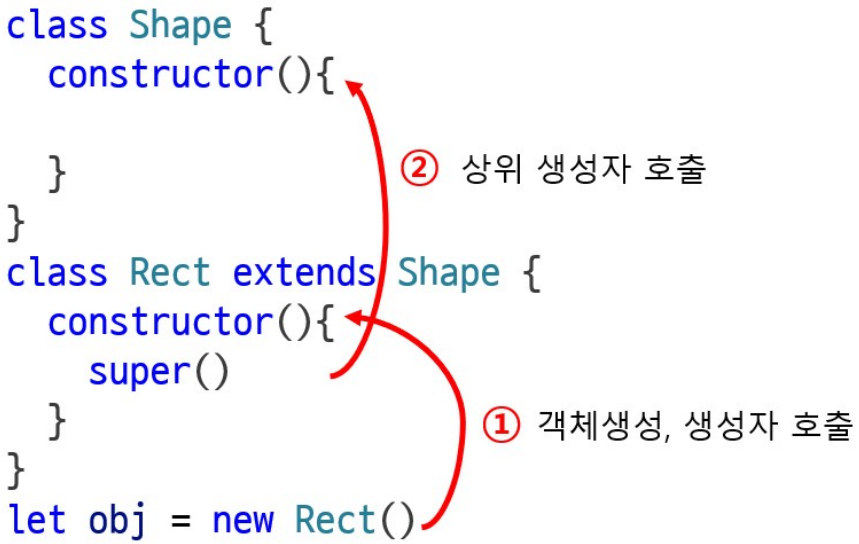
상속

```
class Shape {

}
class Rect extends Shape {

}
let obj = new Rect()
```

```
class Shape {
  constructor(){

  }
}
class Rect extends Shape {
  constructor(){

  }
}
let obj = new Rect()//error
```

```
class Shape {
  constructor(){

  }
}
class Rect extends Shape {
  constructor(){
    super()
  }
}
let obj = new Rect()
```

상속


class Shape {
constructor(){
}
}
class Rect extends Shape {
constructor(){
super()
}
}
let obj = new Rect()
② 상위 생성자 호출
① 객체생성, 생성자 호출

상속

super 로 상위 생성자 호출

- super() 로 상위의 생성자를 호출하는 구문은 생성자 내에서 가장 첫 줄에 한번만 작성할 수 있습니다.
- 상위 생성자를 super() 로 호출할 때 ( ) 에 상위 생성자에 전달해야 하는 매개변수 값을 명시할 수 있습니다.

```
class Rect extends Shape {
  constructor(name, x, y, width, height){
    super(name, x, y)
    this.width = width
    this.height = height
  }
}
```

상속

super 로 상위 멤버 이용

- 상위에 선언된 멤버는 하위에서 상속되어 이용됩니다.
- 즉 상위에 선언된 멤버를 자신 클래스에 선언된 멤버처럼 사용합니다.

- 그런데 경우에 따라 상위에 선언된 멤버를 하위에서 동일이름으로 다시 정의하는 경우가 있습니다.
- 이를 용어로 오버라이딩(Overriding) 이라고 합니다.

상속

```
class Shape {
  constructor(name, x, y){
    this.name = name
    this.x = x
    this.y = y
  }
  calcArea(){
    return `${this.name}의 면적을 계산합니다.`
  }
}
```

```
class Rect extends Shape {
  constructor(name, x, y, width, height){
    super(name, x, y)
    this.width = width
    this.height = height
  }
  calcArea(){
    return this.width * this.height
  }
}
```

상속

super 로 상위 멤버 이용

- 상위의 멤버를 오버라이딩으로 하위에서 재정의 해서 사용하는 경우 필요에 의해 상위에 선언된 동일이름의 멤버를 이용해야 하는 경우가 있습니다.
- 이때 super 라는 예약어로 명시적으로 상위의 멤버를 지칭하게 됩니다.

```
class Rect extends Shape {
  constructor(name, x, y, width, height){
    super(name, x, y)
    this.width = width
    this.height = height
  }
  calcArea(){
    super.calcArea()
    console.log(`면적은 ${this.width * this.height} 입니다.`)
  }
}
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare