

2025년 상반기 K-디지털 트레이닝

클래스(생성자 함수)

[KB] IT's Your Life

✓ 객체와 배열을 사용한 데이터 관리

- 추상화
→ 현실에 존재하는 객체의 필요한 속성을 추출하는 작업
- 학생의 성적 총점과 평균을 계산하는 예제 작성

```
var student0 = { 이름: '윤인성', 국어: 87, 수학: 98, 영어: 88, 과학: 95 };  
var student1 = { 이름: '연하진', 국어: 92, 수학: 98, 영어: 96, 과학: 98 };  
var student2 = { 이름: '구지연', 국어: 76, 수학: 96, 영어: 94, 과학: 90 };  
var student3 = { 이름: '나선주', 국어: 98, 수학: 92, 영어: 96, 과학: 92 };  
var student4 = { 이름: '윤아린', 국어: 95, 수학: 98, 영어: 98, 과학: 98 };  
var student5 = { 이름: '윤명월', 국어: 64, 수학: 88, 영어: 92, 과학: 92 };  
var student6 = { 이름: '김미화', 국어: 82, 수학: 86, 영어: 98, 과학: 88 };  
var student7 = { 이름: '김연화', 국어: 88, 수학: 74, 영어: 78, 과학: 92 };  
var student8 = { 이름: '박아현', 국어: 97, 수학: 92, 영어: 88, 과학: 95 };  
var student9 = { 이름: '서준서', 국어: 45, 수학: 52, 영어: 72, 과학: 78 };
```

✓ 객체와 배열을 사용한 데이터 관리

○ 배열에 데이터 추가

```
var students = [];  
students.push({ 이름: '윤인성', 국어: 87, 수학: 98, 영어: 88, 과학: 95 });  
students.push({ 이름: '연하진', 국어: 92, 수학: 98, 영어: 96, 과학: 98 });  
students.push({ 이름: '구지연', 국어: 76, 수학: 96, 영어: 94, 과학: 90 });  
students.push({ 이름: '나선주', 국어: 98, 수학: 92, 영어: 96, 과학: 92 });  
students.push({ 이름: '윤아린', 국어: 95, 수학: 98, 영어: 98, 과학: 98 });  
students.push({ 이름: '윤명월', 국어: 64, 수학: 88, 영어: 92, 과학: 92 });  
students.push({ 이름: '김미화', 국어: 82, 수학: 86, 영어: 98, 과학: 88 });  
students.push({ 이름: '김연화', 국어: 88, 수학: 74, 영어: 78, 과학: 92 });  
students.push({ 이름: '박아현', 국어: 97, 수학: 92, 영어: 88, 과학: 95 });  
students.push({ 이름: '서준서', 국어: 45, 수학: 52, 영어: 72, 과학: 78 });
```

✓ 객체와 배열을 사용한 데이터 관리

○ 메서드 추가

```
// 모든 students 배열 내의 객체에 메서드를 추가합니다.
for (var i in students) {
    // 총점을 구하는 메서드를 추가합니다.
    students[i].getSum = function () {
        return this.국어 + this.수학 + this.영어 + this.과학;
    };

    // 평균을 구하는 메서드를 추가합니다.
    students[i].getAverage = function () {
        return this.getSum() / 4;
    };
}
```

✓ 객체와 배열을 사용한 데이터 관리

○ 학생 성적 출력

```
var output = '이름\t총점\t평균\n';  
for (var i in students) {  
    with (students[i]) {  
        output += 이름 + '\t' + getSum() + '\t' + getAverage() + '\n';  
    }  
}
```

✓ 함수를 사용한 객체 생성

○ 객체의 반영

- 하나씩 만들어 배열에 사용 : 서로 다른 형태의 객체를 배열 안에 넣을 수 있는 장점
- 개별적 객체를 만드는 것이 객체의 특성을 정확히 반영

📌 17_makeobjfunction.js 함수를 사용한 객체 생성

```
function makeStudent(name, korean, math, english, science) {  
  let student = {  
    name : name,  
    korean : korean,  
    math : math,  
    english : english,  
    science : science,  
  
    getSum : function() {  
      return this.korean + this.math + this.english + this.science;  
    },  
  
    getAverage : function() {  
      return this.getSum() / 4;  
    },  
  
    toString : function() {  
      return `${this.name}\t${this.getSum()}\t${this.getAverage()}`;  
    }  
  };  
  return student;  
}
```

객체 생성 함수.

vue에서
vm=Vue.createApp(~~~) 함수도 마찬가지로

17_makeobjfunction.js 함수를 사용한 객체 생성

```
let students = [];  
students.push(makeStudent('윤인성', 90, 83, 76, 89));  
students.push(makeStudent('박찬호', 90, 83, 76, 89));  
students.push(makeStudent('류현진', 90, 83, 76, 89));  
students.push(makeStudent('이세돌', 90, 83, 76, 89));  
students.push(makeStudent('김세진', 90, 83, 76, 89));  
students.push(makeStudent('이하나', 90, 83, 76, 89));  
  
let output = 'name\t총점\t평균\n';  
for(let i in students) {  
    output += students[i].toString() + '\n';  
}  
  
console.log(output);
```


✓ 생성자 함수란?

- new 키워드를 사용해 객체 생성할 수 있는 함수

✓ student 생성자

- student 생성자 함수를 만드는 코드

📄 18_studentObject.js

```
function Student () {  
  
}
```

✓ new 키워드

○ new 키워드로 객체 생성

```
function Student () {  
  
}  
  
let student = new Student();
```

new 없이 사용하면
this가 달라지고
뭘 반환하는지
예상할 수 없다

✓ new 키워드

○ 생성자 호출시 사용

- new 생성자() 안에서 this는 {}
 - 새로운 인스턴스가 생성되고 this에 배정
 - this의 값이 리턴

○ new 없이 생성자 호출시

- 생성자() 보통 변환함수로 정의함
- 새로운 인스턴스가 생성되지 않고, this에 변화가 없음
- undefined 리턴

Number('123')
String(123)

✓ this 키워드

○ 생성자 함수로 생성될 객체의 속성 지정

```
function Student(name, korean, math, english, science) {  
  this.name = name;  
  this.korean = korean;  
  this.math = math;  
  this.english = english;  
  this.science = science;  
}
```

```
let student = new Student('김세진', 90, 83, 76, 89);
```

✔ 메서드 생성

```
function Student(name, korean, math, english, science) {  
    this.name = name;  
    this.korean = korean;  
    this.math = math;  
    this.english = english;  
    this.science = science;  
  
    this.getSum = function() {  
        return this.korean + this.math + this.english + this.science;  
    }  
  
    this.getAverage = function() {  
        return this.getSum() / 4;  
    }  
  
    this.toString = function() {  
        return `${this.name}\t${this.getSum()}\t${this.getAverage()}`;  
    }  
}  
  
let student = new Student('김세진', 90, 83, 76, 89);
```

19_objectArray.js 생성자 함수를 사용한 객체 배열 생성

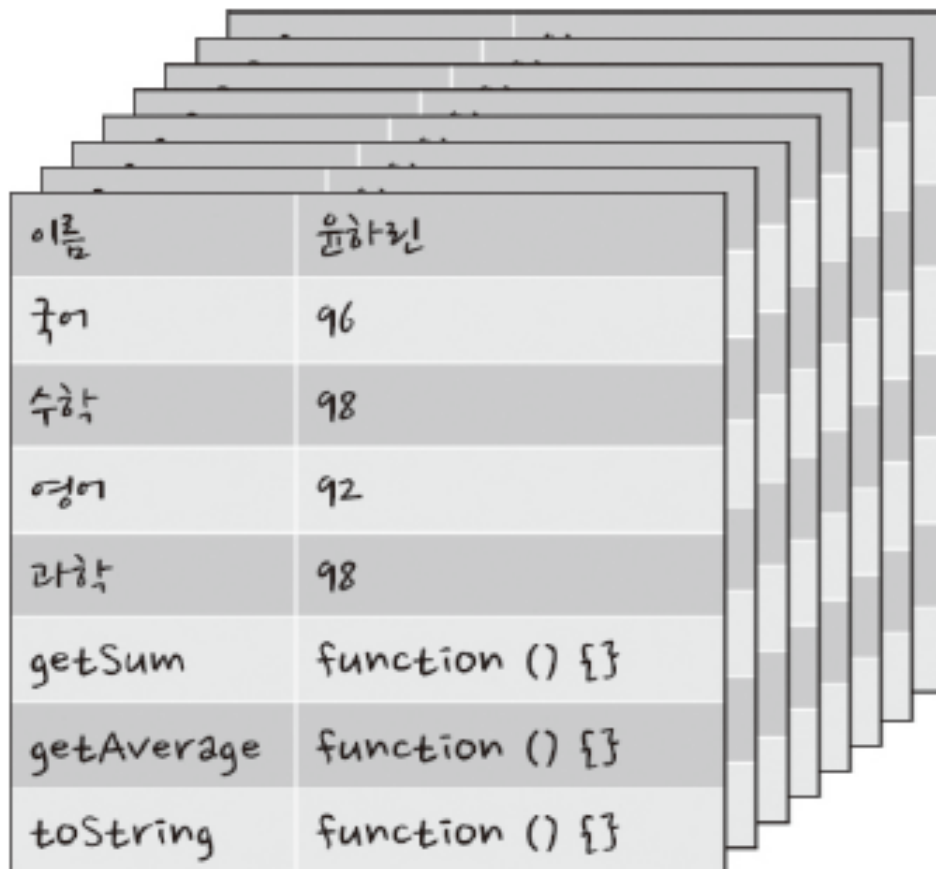
```
let students = [];  
  
students.push(new Student('윤인성', 90, 83, 76, 89));  
students.push(new Student('박찬호', 90, 83, 76, 89));  
students.push(new Student('류현진', 90, 83, 76, 89));  
students.push(new Student('이세돌', 90, 83, 76, 89));  
students.push(new Student('김세진', 90, 83, 76, 89));  
students.push(new Student('이하나', 90, 83, 76, 89));  
  
let output = 'name\t총점\t평균\n';  
for(let i in students) {  
    output += students[i].toString() + '\n';  
}  
  
console.log(output);
```

✓ 생성자 함수

○ 기존의 객체 구조

→ 이름, 국어, 수학, 영어, 과학 속성

→ getSum (), getAverage (), toString () 메서드

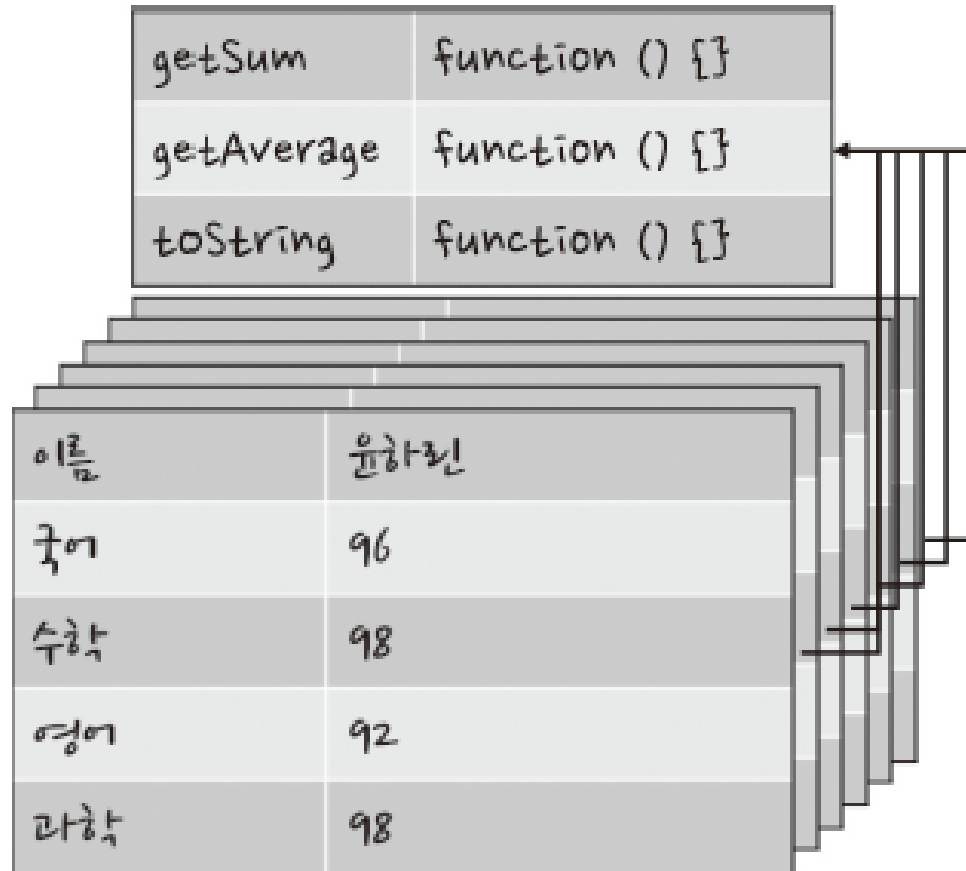


이름	윤하린
국어	96
수학	98
영어	92
과학	98
getSum	function () {}
getAverage	function () {}
toString	function () {}

✓ 메모리에 따른 문제 해결

- 프로토타입 역할로만 보면 super와 같다
 - 동일한 함수 생성에 따른 비효율적인 메모리 이용을 해결
 - 생성자 함수로 생성된 객체가 공통으로 가지는 공간
- 프로토타입을 사용한 객체 구조

js에서는 모든 객체가 원형을 의미하는 prototype 속성을 가지고 prototype속성값으로 원형모습을 나타내는 prototype객체를 가진다.



c++의 객체처럼
메서드는
공유하는 걸로

js에서는 prototype객체를 통해
객체들이 메서드를
공유함

객체의 메서드를
읽기(접근)(사용)할때는
객체에 없다면
prototype으로 올라가 찾아다니는데

쓸때는
객체.새로운함수=~~~;
일때는
prototype으로 올라가지 않고
해당 객체에만 있는
새로운 메소드가 추가된다,

prototype에 추가하고 싶다면
객체.prototype.새로운함수=~~~;
로 추가해야한다

객체와 prototype을 분리하고
메서드에 대한 읽기와 쓰기작업과정
이 다르다는 것을 기억하자

✓ 생성자 함수 구성

- 한 개의 메서드로 모든 객체가 사용
- 생성자 함수로 객체를 만들 때 → 생성자 함수 내부에 속성만 넣음

```
function Student(name, korean, math, english, science) {  
    this.name = name;  
    this.korean = korean;  
    this.math = math;  
    this.english = english;  
    this.science = science;  
}
```

✓ 프로토타입

- 자바스크립트의 모든 함수는 변수 `prototype`을 가짐
- `prototype`은 객체

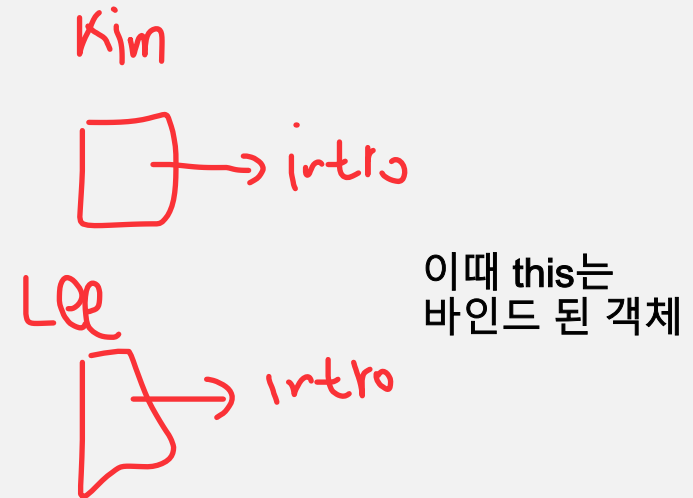
여기에서의 `this`는
프로토타입이 참조
하는 객체가 아닌
호출 시점에서의
객체다.

```
Student.prototype.getSum = function() {  
    return this.korean + this.math + this.english + this.science;  
}  
  
Student.prototype.getAverage = function() {  
    return this.getSum() / 4;  
}  
  
Student.prototype.toString = function() {  
    return `${this.name}\t${this.getSum()}\t${this.getAverage()}`;  
}
```

20_constructor.js

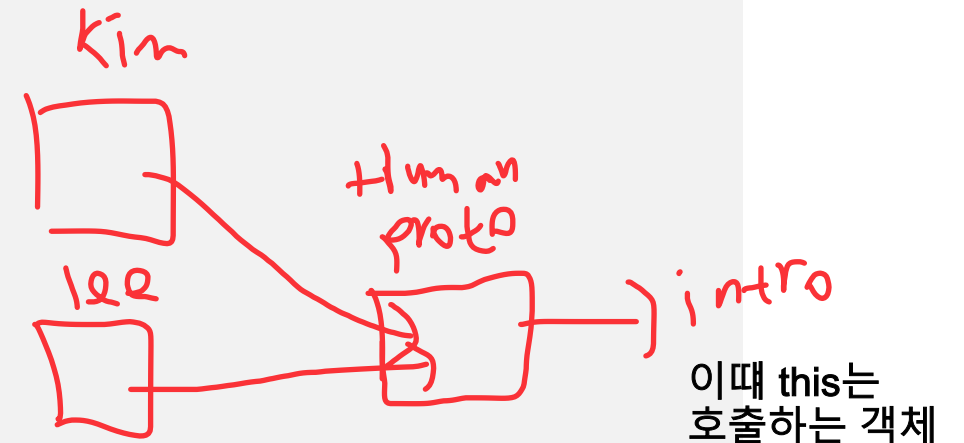
```
function Human(name, age) {  
  this.name = name;  
  this.age = age;  
  
  this.intro = function() {  
    console.log("name = " + this.name);  
    console.log("age = " + this.age);  
  };  
}
```

```
let kim = new Human("김상형", 29);  
let lee = new Human("이승우", 42);  
kim.intro();  
lee.intro();
```



21_prototype.js

```
function Human(name, age) {  
  this.name = name;  
  this.age = age;  
}  
  
Human.prototype.intro = function() {  
  console.log("name = " + this.name);  
  console.log("age = " + this.age);  
};  
  
let kim = new Human("김상형", 29);  
let lee = new Human("이승우", 40);  
kim.intro();  
lee.intro();
```



✓ 상속이란?

- 기존의 생성자 함수나 객체를 기반으로 새로운 생성자 함수나 객체를 쉽게 만드는 것
- 상속으로 만들어지는 객체는 기존 객체의 특성이 모두 있음
- 상속을 사용하면 이전에 만들었던 객체와 비슷한 객체를 쉽게 만들 수 있음

✓ 상속의 예

○ Rectangle

📄 22_inheritance.js

아래 지역변수인 width, height가
스코프에 지정되어있고
클로저로 내부 함수에서만 접근할 수 있다.

그러니 private 멤버지
멤버변수로 등록안하고 일부로 지역변수로
prototype으로도 접근 못함

```
function Rectangle(width, height) {  
  
    this.getWidth = function () { return width; };  
  
    this.getHeight = function () { return height; };  
  
    this.setWidth = function (w) {  
        width = w;  
    };  
  
    this.setHeight = function (h) {  
        height = h;  
    };  
}
```

✓ 상속의 예

- 생성자 함수 Square 내부에서 작성한 것
 - (1) base 속성에 생성자 함수 Rectangle을 넣고 실행한 것
 - (2) 생성자 함수 Square의 프로토타입에 Rectangle의 프로토 타입을 넣은 것
- (1)을 사용해 Rectangle 객체의 속성을 Square 객체에 추가
- (2)를 사용해 Rectangle 객체의 프로토타입이 가진 속성 또는 메서드를 Square 객체의 프로토타입에 복사

```
Rectangle.prototype.getArea = function () {  
    return this.getWidth() * this.getHeight();  
};
```

```
let rectangle = new Rectangle(5, 7);
```

```
console.log('AREA: ' + rectangle.getArea());
```

✓ 상속

```
function Square(length) {  
    let width = length;  
    let height = length;  
  
    this.getWidth = function () { return width; };  
  
    this.getHeight = function () { return height; };  
  
    this.setWidth = function (w) {  
        width = w;  
    };  
  
    this.setHeight = function (h) {  
        height = h;  
    };  
}  
Square.prototype.getArea = function () {  
    return this.getWidth() * this.getHeight();  
};
```

- Rectangle에 이미 구현되어 있는 코드임

✓ 상속

○ Rectangle을 상속받아 Square 정의

syntactic sugar로

우리가 아는
class extends
형태로
사용가능 예정

```
function Square(length) {  
    this.base = Rectangle;    // 부모 클래스 생성자 함수  
    this.base(length, length); // 부모 클래스의 생성자 함수 호출  
    // new 없이 호출했으므로 this가 Square의 인스턴스가 됨  
}  
  
Square.prototype = Rectangle.prototype;  
Square.prototype.constructor = Square;  
  
let rectangle = new Rectangle(5, 7);  
let square = new Square(5);  
console.log(rectangle.getArea() + ' : ' + square.getArea());  
  
console.log(square instanceof Rectangle);
```

✓ instanceof

- 객체가 특정 타입인지 검사

`O instanceof T`

- 객체 O가 T 생성자로부터 나왔으면 true
- 객체 O가 T의 prototype을 상속했는지 점검

2025년 상반기 K-디지털 트레이닝

클래스(class)

[KB] IT's Your Life

✓ class 키워드

- 객체의 원형을 정의하는 키워드
- ES6(ES2015)에서 추가

```
class 클래스명 {  
  
}
```

✓ 생성자

- constructor() 함수로 정의
 - this 객체가 새로 생성됨
 - 매개변수 지정 가능

```
class 클래스명 {  
    constructor([매개변수]) {  
        // this.를 이용하여 객체의 프로퍼티 정의  
    }  
}
```

student.js

```
class Student {  
  constructor(name) {  
    this.name = name;  
  }  
}
```

```
let s1 = new Student("홍길동");  
console.log(s1.name);
```

✔ 프로토타입 메서드

- class 블록안에 정의되는 함수
- function 키워드 없이 정의

```
class 클래스명 {  
    constructor([매개변수]) {  
        // this.를 이용하여 객체의 프로퍼티 정의  
    }  
  
    함수명([매개변수]) {    // 프로토타입 메서드  
    }  
}
```

student.js

```
class Student {  
  constructor(name, age) {  
    this.name = name;  
    this.age = age;  
  }  
  
  printProfile(){  
    console.log(`이름 : ${this.name}, 나이 : ${this.age}`);  
  }  
}  
  
let s1 = new Student("홍길동", 20);  
s1.printProfile();  
  
console.log("printProfile" in s1.__proto__);  
console.log("printProfile" in Student.prototype);
```

_프로퍼티명이면
private

✓ Getter/Setter 메서드

- get, set 키워드로 설정

```
class 클래스명 {  
    constructor([매개변수]) {  
        // this.를 이용하여 객체의 프로퍼티 정의 ✓  
    }  
  
    함수명([매개변수]) {    // 프로토타입 메서드  
    }  
  
    get 프로퍼티명() {  
    }  
    set 프로퍼티명(매개변수) {  
    }  
}
```

student.js

```
class Student {  
  constructor(name, age) {  
    this._name = name;  
    this.age = age;  
  }  
  
  printProfile(){  
    console.log(`이름 : ${this.name}, 나이 : ${this.age}`)  
  }  
  
  get name() {  
    return this._name;  
  }  
  
  set name(name) {  
    this._name = name;  
  }  
}
```

student.js

```
let s1 = new Student("홍길동", 20);
```

```
console.log(s1.name);  
s1.name = '고길동'  
console.log(s1.name);  
console.log(s1);
```

[실행결과]

홍길동

고길동

Student { _name: '고길동', age: 20 }

✓ 정적 메서드

- static 키워드로 메서드 정의
- 자체 인스턴스 (this) 없이 구현

```
class 클래스명 {  
    :  
  
    static 메서드명(매개변수) {  
  
    }  
}
```

✓ extends 키워드와 super 키워드 도입

```
class 자식클래스 extends 부모클래스 {  
  
    constructor([매개변수]) {  
        super([매개변수]);    // 부모의 생성자 호출, 이 후에 this 생성  
    }  
  
}
```

- 부모의 생성자는 반드시 호출

Inherit.js

```
class Parent {  
  constructor(name) {  
    this.name = name;  
  }  
  
  print() {  
    console.log("이름 : " + this.name);  
  }  
}
```

Inherit.js

```
class Child extends Parent {  
  constructor(name, age) {  
    super(name);  
    this.age = age;  
  }  
  
  print() {  
    super.print();  
    console.log("나|0| : " + this.age);  
  }  
  
  static sayHello() {  
    console.log('Hello~');  
  }  
}
```

override ~~~~~

Inherit.js

```
class GrandChild extends Child {  
  constructor(name, age, address) {  
    super(name, age);  
    this.address = address;  
  }  
  
  print() {  
    super.print();  
    console.log("주소 : " + this.address);  
  }  
}
```

```
let person = new GrandChild("홍길동", 20, "서울");  
person.print();  
GrandChild.sayHello();
```