**Ch07 생성자 함수**

**7.1 생성자 함수 개요**

생성자는 객체지향 프로그램에서 객체를 생성할 때 호출되는 단위 메서드를 말한다.

function 생성자함수명(속성명들) {

...

}

var 객체명 = new 생성자함수명(속성값들);

new 키워드로 객체를 생성하므로 생성자함수. 생성자 함수로 만든 객체를 객체 또는 인스턴스라 부른다. 해당 객체가 어떠한 생성자 함수로 생성됐는지 확인할 때 instanceof 키워드를 사용할 수 있다.

<script>

function Student(name, kor, eng, mat) {

this.name = name; // this는 현재 객체의 속성을 가리킬 때 씀, name은 파라미터로 받은 지역변수

this.kor = kor;

this.eng = eng;

this.mat = mat;

this.sum = kor+eng+mat;

this.getSum = function () { return this.kor + this.eng + this.mat; };

this.getAvg = function () { return Math.round(this.getSum() / 3\*10)/10; };

this.toString = function () { return this.name + '\t' + this.getSum() + '\t' + this.getAvg(); };

}

var student = new Student('윤하린', 100, 96, 98);

var students = [];

students.push(new Student('홍길동', 80, 70, 50));

students.push(new Student('김길동', 80, 70, 50));

students.push(new Student('김홍동', 100, 70, 50));

var output = '이름\t총점\t평균\n';

for (var i in students) {

output += students[i].toString() + '\n';

}

alert(output);

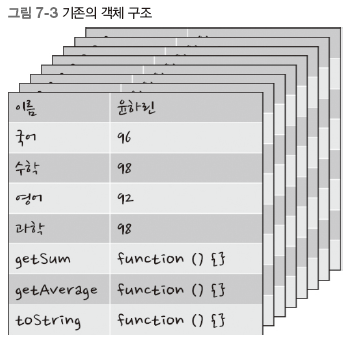
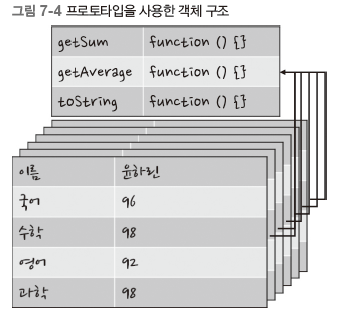
console.log ('student는 Student 생성자함수의 인스턴스인지 '+(student instanceof Student));

console.log ('student는 Array 생성자함수의 인스턴스인지 ' + (student instanceof Array));

</script>

**7.2 프로토타입**

일반함수를 사용하여 객체를 만든 것과 생성자함수를 사용하여 객체를 만드는 것의 차이점이 없습니다.

Student의 경우 속성은 모든 객체가 다른 속성을 가지지만 메소드는 모두 같습니다. 각 객체를 생성할 때마다 동일한 함수를 계속 생성하게 된다 이러한 문제를 해결하고자 포르토타입을 이용한다.

프로토타입이란, 생성자 함수로 생성된 객체가 공통으로 가지는 공간.

메서드를 모두 프로토타입 안에 넣어준다. 프로토타입은 우리가 만드는 것이 아니다. 함수안에 자동으로 만들어지는 배열 arguments와 마찬가지로 자바스크립트의 모든 함수는 prototype 객체를 갖는다.

<script>

function Student(name, kor, eng, mat) {

this.name = name;

this.kor = kor;

this.eng = eng;

this.mat = mat;

}

Student.prototype.getSum = function () { return this.kor + this.eng + this.mat; };

Student.prototype.getAvg = function () { return Math.round(this.getSum() / 3 \* 10) / 10; };

Student.prototype.toString = function () { return this.name + '\t' + this.getSum() + '\t' + this.getAvg(); };

var students = [];

students.push(new Student('홍길동', 80, 70, 50));

students.push(new Student('김길동', 80, 70, 50));

students.push(new Student('김홍동', 100, 70, 50));

var output = '이름\t총점\t평균\n';

for (var i in students) {

output += students[i].toString() + '\n';

}

alert(output);

</script>

**7.3 new 키워드**

<script>

function Constructor(value) {

this.value = value;

}

var constructor = new Constructor('Hello');

alert(constructor.value);

</script>

<script>

function Constructor(value) {

this.value = value; // this.value는 window.value

}

var constructor = Constructor('Hello'); //일반 함수가 되고

alert(constructor);

</script>

new 키워드로 함수를 호출하면 객체를 위한 공간을 만들고 this 키워드가 해당 공간을 의미한다. 그렇지 않는 경우, 일반적으로 this 키워드를 사용하면 window 객체를 나타낸다. Window 객체에 속성을 추가한 것이 된다.

**7.4 캡슐화 ;** 만일의 상황을 대비해서 특정 속성이나 메서드를 사용자가 바로 사용할 수 없게 막는 기술이다. 숨겨 놓은 특정 속성은 getter와 setter를 이용하여 간접적으로 접근할 수 있다.

<script>

function Rectanle(width, height) {

this.width = width;

this.height = height;

}

Rectanle.prototype.getArea = function () {

return this.width \* this.height;

};

var rectangle = new Rectanle(5, 7);

alert('AREA : ' + rectangle.getArea());

</script>

위의 예제를 width와 height를 사용자가 사용할 수 없게 숨겨 놓고 캡슐화 시켜 놓은 예제가 아래와 같다.

<script>

function Rectangle(w, h) {

var width = w;

var height = h;

this.getWidth = function () { return width; };

this.getHeight = function () { return height; };

this.setWidth = function (w) {

if (w < 0) {

throw '길이는 음수일 수 없습니다';

} else {

width = w;

}

};

this.setHeight = function (h) {

if (h < 0) {

throw '길이는 음수일 수 없습니다';

} else {

height = h;

}

};

}

Rectangle.prototype.getArea = function () {

return this.getWidth() \* this.getHeight();

};

var rectangle = new Rectangle(5, 7);

rectangle.setHeight(-1);

alert('AREA : ' + rectangle.getArea());

</script>

※

<script>

function Rectangle(w, h) {

if (w >= 0) { var width = w; } else { throw '길이는 음수일 수 없습니다'; }

if (h >= 0) { var height = w; } else { throw '길이는 음수일 수 없습니다'; }

this.getWidth = function () { return width; };

this.getHeight = function () { return height; };

this.setWidth = function (w) {

if (w < 0) {

throw '길이는 음수일 수 없습니다';

} else {

width = w;

}

};

this.setHeight = function (h) {

if (h < 0) {

throw '길이는 음수일 수 없습니다';

} else {

height = h;

}

};

}

Rectangle.prototype.getArea = function () {

return this.getWidth() \* this.getHeight();

};

var rectangle = new Rectangle(-5, 7);

//rectangle.setHeight(-1);

alert('AREA : ' + rectangle.getArea());

</script>

**7.5 상속 ;** 기존의 생성자 함수나 객체를 기반으로 새로운 생성자 함수나 객체를 쉽게 만드는 것 – 이 밑은 생략

<script>

function Rectangle(w, h) {

var width = w;

var height = w;

this.getWidth = function () { return w; };

this.getHeight = function () { return h; };

this.setWidth = function (w) {

if (w < 0) {

throw '길이는 음수일 수 없습니다';

} else {

width = w;

}

};

this.setHeight = function (h) {

if (h < 0) {

throw '길이는 음수일 수 없습니다';

} else {

height = h;

}

};

}

Rectangle.prototype.getArea = function () {

return this.getWidth() \* this.getHeight();

};

function Square(length) {

**this.base = Rectangle;**

**this.base(length, length);**

}

**Square.prototype = Rectangle.prototype;**

**Square.prototype.constructor=Square;**//Square.prototype.constructor이 Rectangle을 가르키지 않게 재정의

var rectangle = new Rectangle(5, 7);

var square = new Square(5);

alert(rectangle.getArea() + ':' + square.getArea());

alert(square instanceof Rectangle);

alert(square instanceof Square);

alert(Square.prototype.constructor); //

</script>

자바 스크립트에서 정확한 상속 방법은 없습니다. 상속이 됐다 안 됐다의 기준은 square instanceof Rectangle이 true이면 상속되었다고 판단한다.