



6 객체 지향 프로그래밍

모던 웹을 위한 Javascript jQuery 입문

❖ 배열 선언과 접근

코드 6-1 배열의 선언

```
<script>  
    // 변수를 선언합니다.  
    var array = ['사과', '바나나', '망고', '딸기'];  
</script>
```

■ 배열 접근

- array[0] ⇨ '사과'
- array[2] ⇨ '망고'

그림 6-1 표로 나타낸 배열

인덱스	요소
0	사과
1	바나나
2	망고
3	딸기



6.1 객체 개요

❖ 객체 생성과 배열

코드 6-2 객체의 생성

```
<script>
  // 변수를 선언합니다.
  var product = {
    제품명: '7D 건조 망고',
    유형: '당절임',
    성분: '망고, 설탕, 메타중아황산나트륨, 치자황색소',
    원산지: '필리핀'
  };
</script>
```

그림 6-2 표로 나타난 객체

키	요소
제품명	'7D 건조 망고'
유형	'당절임'
성분	'망고, 설탕, 메타중아황산나트륨, 치자황색소'
원산지	'필리핀'



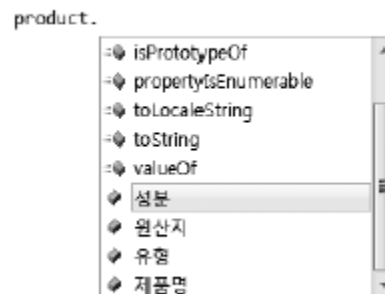
❖ 객체 생성과 배열

■ 배열과의 비교

- product['제품명'] ⇨ '7D 건조 망고'
- product['유형'] ⇨ '당절임'
- product['성분'] ⇨ '망고, 설탕, 메타중아황산나트륨, 치자황색소'
- product['원산지'] ⇨ '필리핀'

- product.제품명 ⇨ '7D 건조 망고'
- product.유형 ⇨ '당절임'
- product.성분 ⇨ '망고, 설탕, 메타중아황산나트륨, 치자황색소'
- product.원산지 ⇨ '필리핀'

그림 6-3 두 번째 방법을 사용하면 보조 기능을 이용할 수 있습니다.



❖ 객체의 키

- 대부분 개발자가 식별자를 키로 사용
- 식별자로 사용할 수 없는 단어를 키로 사용할 때는 문자열 사용
 - 대괄호 사용해야 객체 요소 접근 가능
 - **object['widht space']** ⇨ 273
 - **object['with ~!@#\$\$%^&*()_+']** ⇨ 52

코드 6-3 문자열을 키로 사용

```
<script>
  // 변수를 선언합니다.
  var object = {
    'with space': 273,
    'with ~!@#$$%^&*()_+': 52
  };
</script>
```



6.2 객체의 속성과 메서드

❖ 요소와 속성

- 배열 내부에 있는 값 – 요소 (element)
- 객체 내부에 있는 값 – 속성 (property)

코드 6-4 객체의 속성이 가질 수 있는 자료형

```
var object = {  
  number: 273,  
  string: 'RintIanTta',  
  boolean: true,  
  array: [52, 273, 103, 32],  
  method: function () {  
  
  }  
};
```



6.2 객체의 속성과 메서드

❖ 메서드 (method)

■ 객체의 속성 중 함수 자료형

코드 6-5 속성과 메서드의 구분

```
<script>
// 변수를 선언합니다.
var person = {
  name: '윤인성',
  eat: function (food) { }
};

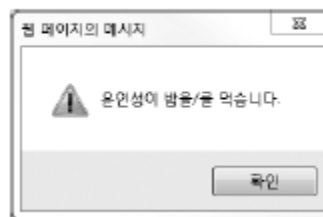
// 메서드를 호출합니다.
person.eat();
</script>
```

코드 6-6 this 키워드

```
<script>
// 변수를 선언합니다.
var person = {
  name: '윤인성',
  eat: function (food) {
    alert(this.name + '이 ' + food + '을/를 먹습니다. ');
  }
};

// 메서드를 호출합니다.
person.eat('밥');
</script>
```

그림 6-4 메서드 사용



6.3 객체와 반복문

❖ 객체와 반복문

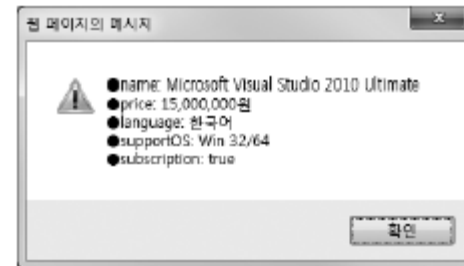
- 배열 접근 - 단순 for 반복문과 for in 반복문 사용 가능
- 객체 속성 - for in 반복문을 사용해야 함

코드 6-7 객체와 반복문

```
<script>
// 변수를 선언합니다.
var product = {
    name: 'Microsoft Visual Studio 2010 Ultimate',
    price: '15,000,000원',
    language: '한국어',
    supportOS: 'Win 32/64',
    subscription: true
};

// 출력합니다.
var output = '';
for (var key in product) {
    output += '●' + key + ': ' + product[key] + '\n';
}
alert(output);
</script>
```

그림 6-5 반복문을 사용한 객체의 속성 출력



6.4 객체와 관련된 키워드

❖ 객체 관련 키워드

■ 객체 생성

코드 6-8 객체 생성

```
<script>
    // 변수를 선언합니다.
    var student = {
        이름: '연하진',
        국어: 92, 수학: 98,
        영어: 96, 과학: 98
    };
</script>
```



6.4 객체와 관련된 키워드

❖ 객체 관련 키워드

- in 키워드 - 해당 키가 객체 안에 있는지 확인

코드 6-9 in 키워드

```
<script>
    // 변수를 선언합니다.
    var output = '';
    var student = {
        이름: '연하진',
        국어: 92, 수학: 98,
        영어: 96, 과학: 98
    };

    // in 키워드를 사용합니다.
    output += "'이름' in student: " + ('이름' in student) + '\n';
    output += "'성별' in student: " + ('성별' in student);

    // 출력합니다.
    alert(output);
</script>
```

그림 6-6 in 키워드의 사용



6.4 객체와 관련된 키워드

❖ 객체 관련 키워드

- with 키워드 - 복잡하게 사용해야 하는 코드를 짧게 줄여주는 키워드
- 예제 1) with 키워드 사용하지 않은 경우

코드 6-10 with 키워드를 사용하지 않은 경우

```
<script>
    // 변수를 선언합니다.
    var student = {
        이름: '연하진',
        국어: 92, 수학: 98,
        영어: 96, 과학: 98
    };

    // 출력합니다.
    var output = '';
    output += '이름: ' + student.이름 + '\n';
    output += '국어: ' + student.국어 + '\n';
    output += '수학: ' + student.수학 + '\n';
    output += '영어: ' + student.영어 + '\n';
    output += '과학: ' + student.과학 + '\n';
    output += '총점: ' + (student.국어 + student.수학 + student.영어 + student.과학);
    alert(output);
</script>
```



6.4 객체와 관련된 키워드

❖ 객체 관련 키워드

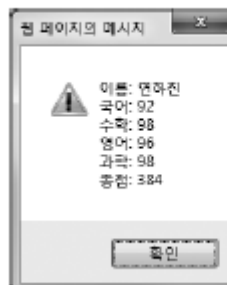
- 예제 2) with 키워드 사용한 경우

코드 6-11 with 키워드를 사용한 경우

```
<script>
// 변수를 선언합니다.
var student = {
    이름: '연하진',
    국어: 92, 수학: 98,
    영어: 96, 과학: 98
};

// 출력합니다.
var output = '';
with (student) {
    output += '이름: ' + 이름 + '\n';
    output += '국어: ' + 국어 + '\n';
    output += '수학: ' + 수학 + '\n';
    output += '영어: ' + 영어 + '\n';
    output += '과학: ' + 과학 + '\n';
    output += '총점: ' + (국어 + 수학 + 영어 + 과학);
}
alert(output);
</script>
```

그림 6-7 with 키워드의 사용



6.5 객체의 속성 추가와 제거

❖ 동적인 객체 생성과 추가

- 처음 객체를 생성하는 시점 이후에 객체의 속성 추가하거나 제거
- 빈 객체 생성해 동적으로 속성 추가하는 예제

코드 6-12 빈 객체 생성

```
<script>
  // 변수를 선언합니다.
  var student = {};
</script>
```

코드 6-13 동적으로 속성 추가

```
<script>
  // 변수를 선언합니다.
  var student = {};

  // 객체에 속성을 추가합니다.
  student.이름 = '윤인성';
  student.취미 = '악기';
  student.특기 = '프로그래밍';
  student.장래희망 = '생명공학자';
</script>
```



6.5 객체의 속성 추가와 제거

❖ 동적인 객체 생성과 추가

■ 동적 메서드 추가 예제

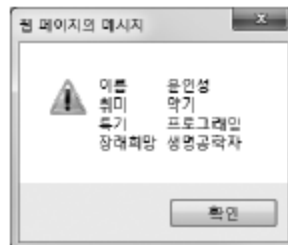
코드 6-14 동적으로 메서드 추가

```
<script>
    // 변수를 선언합니다.
    var student = {};
    student.이름 = '윤인성';
    student.취미 = '악기';
    student.특기 = '프로그래밍';
    student.장래희망 = '생명공학자';

    // toString() 메서드를 만듭니다.
    student.toString = function () {
        var output = '';
        for (var key in this) {
            // toString() 메서드는 출력하지 않게 합니다.
            if (key != 'toString') {
                output += key + '\t' + this[key] + '\n';
            }
        }
        return output;
    };

    // 출력합니다.
    alert(student.toString());
</script>
```

그림 6-8 동적으로 생성된 객체의 속성



6.5 객체의 속성 추가와 제거

❖ 동적인 객체 속성 제거 - delete 키워드 사용

코드 6-15 객체의 속성 제거

```
<script>
// 변수를 선언합니다.
var student = {};
student.이름 = '윤인성';
student.취미 = '악기';
student.특기 = '프로그래밍';
student.장래희망 = '생명공학자';

// toString() 메서드를 만듭니다.
student.toString = function () {
    var output = '';
    for (var key in this) {
        // toString() 메서드는 출력하지 않게 합니다.
        if (key != 'toString') {
            output += key + '\t' + this[key] + '\n';
        }
    }
    return output;
};

// 출력합니다.
alert(student.toString());
// 속성을 제거합니다.
delete(student.장래희망)
// toString() 메서드를 사용하지 않아도 toString() 메서드를 사용합니다.(다음장 떡밥)
alert(student);
</script>
```

그림 6-9 '장래희망' 속성 제거



6.6 객체와 배열을 사용한 데이터 관리

❖ 학생들의 성적 총점과 평균 계산하는 예제

- 추상화 - 현실에 존재하는 객체의 필요한 속성 추출하는 작업
 - Ex) 아래 코드와 같은 데이터 작업

코드 6-16 학생 데이터 생성

```
<script>
  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 };
</script>
```



6.6 객체와 배열을 사용한 데이터 관리

❖ 학생들의 성적 총점과 평균 계산하는 예제

- 배열의 push() 메서드 사용 - 배열에 요소를 집어 넣음

코드 6-17 배열에 데이터 추가

```
<script>
  // 학생 정보 배열을 만듭니다.
  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 });
</script>
```



6.6 객체와 배열을 사용한 데이터 관리

❖ 학생들의 성적 총점과 평균 계산하는 예제

■ 객체에 메서드 추가

코드 6-18 메서드 추가

```
<script>
    // 학생 정보 배열을 만듭니다.
    /* 생략 */

    // 모든 students배열내의 객체에 메서드를 추가합니다.
    for (var i in students) {
        // 총점 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;
        };
    }
</script>
```



6.6 객체와 배열을 사용한 데이터 관리

❖ 학생들의 성적 총점과 평균 계산하는 예제

■ 학생 성적 출력

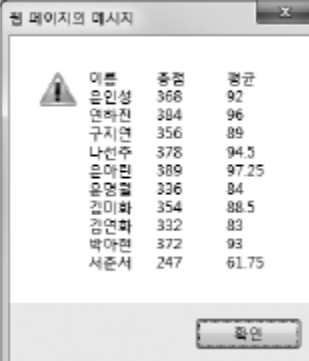
- 배열 students의 객체 이름, 총점, 평균 출력
- for in 반복문 사용해 배열students의 요소에 접근 후 문자열 만들어 출력

코드 6-19 학생 성적 출력

```
<script>
// 학생 정보 배열을 만듭니다.
/* 생략 */

// 모든 students배열내의 객체에 메서드를 추가합니다.
/* 생략 */
// 출력합니다.
var output = '이름\t총점\t평균\n';
for (var i in students) {
    with (students[i]) {
        output += 이름 + '\t' + getSum() + '\t' + getAverage() + '\n';
    }
}
alert(output);
</script>
```

그림 6-11 객체를 사용한 학생 성적 관리 (1)



A screenshot of a web browser's alert dialog box. The dialog has a title bar that says '웹 페이지의 메시지' and a close button. Inside, there is a warning icon (a triangle with an exclamation mark) on the left. To the right of the icon is a table with three columns: '이름' (Name), '총점' (Total Score), and '평균' (Average). The table contains 10 rows of student data. At the bottom right of the dialog is a '확인' (OK) button.

이름	총점	평균
은인성	368	92
은하진	384	96
구지연	356	89
나현주	378	94.5
은하린	389	97.25
윤영철	336	84
김미화	354	88.5
김연화	332	83
박아현	372	93
서준서	247	61.75



6.7 함수를 사용한 객체 생성

❖ 객체를 개별적으로 생성할 때의 이점

- 서로 다른 형태의 객체를 배열 안에 넣을 수 있다는 장점

코드 6-20 객체를 개별적으로 생성할 때의 이점

```
<script>
var students = [];
students.push({
  이름: '윤인성', 국어: 87,
  수학: 98, 영어: 88, 과학: 95,
  장래희망: '생명공학자'
});
students.push({
  이름: '연하진', 국어: 92,
  수학: 98, 영어: 96, 과학: 98,
  특기: '요리', 취미: '일렉기타'
});
students.push({
  이름: '구지연', 국어: 76,
  수학: 96, 영어: 94, 과학: 90,
  장래희망: '프로그래머'
});
</script>
```



6.7 함수를 사용한 객체 생성

❖ 객체 생성 함수

- 빠르고 쉽게 생성하는 객체

코드 6-21 객체를 생성하는 함수 (1)

```
<script>
  function makeStudent(name, korean, math, english, science) {
    var willReturn = {

      };
    return willReturn;
  }
</script>
```



6.7 함수를 사용한 객체 생성

❖ 객체 생성 함수

■ 매개 변수 추가

코드 6-22 객체를 생성하는 함수 (2)

```
<script>
function makeStudent(name, korean, math, english, science) {
    var willReturn = {
        // 속성
        이름: name,
        국어: korean,
        수학: math,
        영어: english,
        과학: science,

        // 메서드
        getSum: function () {
            return this.국어 + this.수학 + this.영어 + this.과학;
        },
        getAverage: function () {
            return this.getSum() / 4;
        },
        toString: function () {
            return this.이름 + '\t' + this.getSum() + '\t' + this.getAverage();
        }
    };
    return willReturn;
}
</script>
```



6.7 함수를 사용한 객체 생성

❖ makeStudent() 함수 사용

■ 학생 목록을 만들어 출력

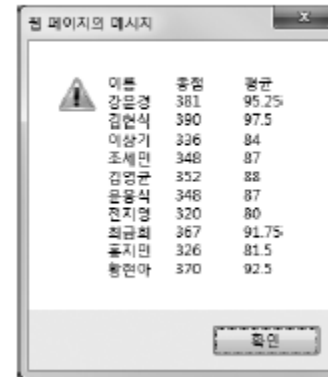
코드 6-23 함수를 사용한 객체 생성

```
<script>
    function makeStudent(name, korean, math, english, science) {
        /* 생략 */
    }

    // 학생 정보 배열을 만듭니다.
    var students = [];
    students.push(makeStudent('강윤경', 98, 98, 90, 95));
    /* 생략 */
    students.push(makeStudent('황현아', 98, 92, 82, 98));

    // 출력합니다.
    var output = '이름\t총점\t평균\n';
    for (var i in students) {
        output += students[i].toString() + '\n';
    }
    alert(output);
</script>
```

그림 6-12 객체를 사용한 학생 성적 관리 (2)



The image shows an alert dialog box titled '웹 페이지의 메시지' (Web page message). It contains a table with student performance data. The table has three columns: '이름' (Name), '총점' (Total Score), and '평균' (Average). The data is as follows:

이름	총점	평균
강윤경	381	95.25
김현식	390	97.5
이상기	336	84
조세민	348	87
김영준	352	88
윤용식	348	87
전지영	320	80
최금희	367	91.75
홍지민	326	81.5
황현아	370	92.5

At the bottom of the dialog box, there is a button labeled '확인' (OK).



❖ 생성자 함수

- new 키워드를 사용해 객체 생성할 수 있는 함수

```
<script>
    function Student() {

    }
</script>
```

- This 키워드 사용해 생성될 객체의 속성 지정

코드 6-24 생성자 함수(1)

```
<script>
    function Student(name, korean, math, english, science) {
        this.이름 = name;
        this.국어 = korean;
        this.수학 = math;
        this.영어 = english;
        this.과학 = science;
    }
</script>
```



❖ 생성자 함수

- this 키워드 사용해 속성 생성하고 함수를 넣어줌

코드 6-25 생성자 함수 (2)

```
<script>
function Student(name, korean, math, english, science) {
    // 속성
    this.이름 = name;
    this.국어 = korean;
    this.수학 = math;
    this.영어 = english;
    this.과학 = science;

    // 메서드
    this.getSum = function () {
        return this.국어 + this.수학 + this.영어 + this.과학;
    };
    this.getAverage = function () {
        return this.getSum() / 4;
    };
    this.toString = function () {
        return this.이름 + '\t' + this.getSum() + '\t' + this.getAverage();
    };
}
</script>
```



❖ 생성자 함수

- Student 생성자 함수 사용해 객체 생성
 - 해당 객체는 이름, 국어, 수학, 영어, 과학속성과 getSum(), getAverage(), toString() 메서드를 가짐
 - 생성자 함수를 사용해 객체 생성할 때에는 new 키워드 사용

코드 6-26 생성자 함수를 사용한 객체의 생성

```
<script>
    function Student(name, korean, math, english, science) {
        /* 생략 */
    }
    var student = new Student('윤하린', 96, 98, 92, 98);
</script>
```



❖ 생성자 함수

■ 총점과 평균 출력하는 예제

코드 6-27 생성자 함수를 사용한 객체의 생성과 출력

```
<script>
    function Student(name, korean, math, english, science) {
        /* 생략 */
    }

    // 학생 정보 배열을 만듭니다.
    var students = [];
    students.push(new Student('윤하린', 96, 98, 92, 98));
    /* 생략 */
    students.push(new Student('윤인아', 96, 96, 98, 92));

    // 출력합니다.
    var output = '이름\t총점\t평균\n';
    for (var i in students) {
        output += students[i].toString() + '\n';
    }
    alert(output);
</script>
```



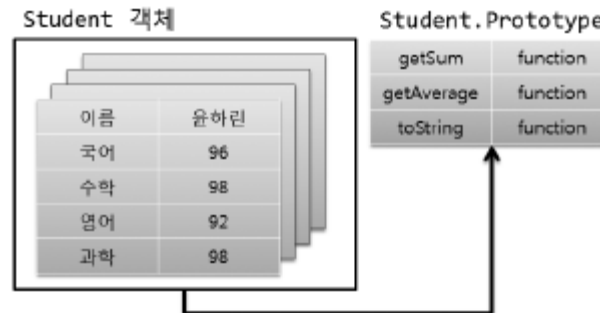
❖ 생성자 함수의 장점

- 속성은 모든 객체가 다른 값을 가지지만 메서드는 모두 같은 값
 - 각기 객체를 생성할 때마다 동일한 함수를 계속 생성하는 것은 낭비

그림 6-13 기존의 객체 구조



그림 6-14 프로토타입을 사용한 객체 구조



❖ 생성자 함수 사용

- 내부에는 속성만 존재
- 메서드는 프로토 타입에 넣을 것

```
<script>
    function Student(name, korean, math, english, science) {
        this.이름 = name;
        this.국어 = korean;
        this.수학 = math;
        this.영어 = english;
        this.과학 = science;
    }
</script>
```

그림 6-15 모든 자바스크립트의 함수는 prototype 객체를 갖습니다.

Student.prototype



❖ 프로토 타입 사용한 메서드 생성

코드 6-28 프로토타입을 사용한 메서드 생성

```
<script>
    function Student(name, korean, math, english, science) {
        this.이름 = name;
        this.국어 = korean;
        this.수학 = math;
        this.영어 = english;
        this.과학 = science;
    }

    Student.prototype.getSum = function () { };
    Student.prototype.getAverage = function () { };
    Student.prototype.toString = function () { };
</script>
```



❖ instanceof 키워드

- 생성자 함수를 통해 만들어진 객체가 인스턴스instance
- 해당 객체가 어떠한 생성자 함수를 통해 생성됐는지 확인할 때 사용

코드 6-29 instanceof 키워드

```
<script>
    // 생성자 함수를 선언합니다.
    function Student(name) { this.name = name; }
    // 변수를 선언합니다.
    var student = new Student('윤인성');

    // 출력합니다.
    alert(student instanceof Student);
    alert(student instanceof Number);
    alert(student instanceof String);
    alert(student instanceof Boolean);
</script>
```



❖ new 키워드

- 생성자 함수 사용해서 생성할 때 사용

코드 6-30

```
<script>
  // 생성자 함수를 선언합니다.
  function Constructor(value) {
    this.value = value;
  }

  // 변수를 선언합니다.
  var constructor = new Constructor('Hello');

  // 출력합니다.
  alert(constructor.value);
</script>
```

코드 6-31

```
<script>
  // 생성자 함수를 선언합니다.
  function Constructor(value) {
    this.value = value;
  }

  // 변수를 선언합니다.
  var constructor = Constructor('Hello');

  // 출력합니다.
  alert(value);
</script>
```

그림 6-16 변수 value 출력 결과



❖ 생성자 함수 Rectangle 선언

코드 6-32 생성자 함수 Rectangle 선언

```
<script>
// 생성자 함수를 선언합니다.
function Rectangle(width, height) {
    this.width = width;
    this.height = height;
}
Rectangle.prototype.getArea = function () {
    return this.width * this.height;
};

// 변수를 선언합니다.
var rectangle = new Rectangle(5, 7);

// 출력합니다.
alert('AREA: ' + rectangle.getArea());
</script>
```

코드 6-33 잘못된 속성의 사용

```
// 변수를 선언합니다.
var rectangle = new Rectangle(5, 7);
rectangle.width = -2;

// 출력합니다.
alert('AREA: ' + rectangle.getArea());
```



❖ 캡슐화

- 잘못 사용될 수 있는 객체의 특정 부분
 - 사용자가 직접 사용할 수 없게 막는 기술
 - 클로저 활용

코드 6-34 캡슐화를 사용한 Rectangle 생성자 함수

```
// 생성자 함수를 선언합니다.
function Rectangle(w, h) {
    // 변수를 선언합니다.
    var width = w;
    var height = h;

    // 메서드를 선언합니다.
    this.getWidth = function () { return width; };
    this.getHeight = function () { return height; };
    this.setWidth = function (w) {
        width = w;
    };
    this.setHeight = function (h) {
        height = h;
    };
}
```



❖ 캡슐화 예제

■ 게터와 세터

- throw 키워드 - 웹 페이지 오류를 발생시키는 키워드로 에러 처리

코드 6-35 게터와 세터

```
<script>
// 생성자 함수를 선언합니다.
function Rectangle(w, h) {
    var width = w;
    var height = h;

    this.getWidth = function () { return width; };
    this.getHeight = function () { return height; };
    this.setWidth = function (value) {
        if (value < 0) {
            throw '길이는 음수일 수 없습니다.';
        } else {
            width = value;
        }
    };
    this.setHeight = function (value) {
        if (value < 0) {
            throw '길이는 음수일 수 없습니다.';
        } else {
            height = value;
        }
    };
}
```

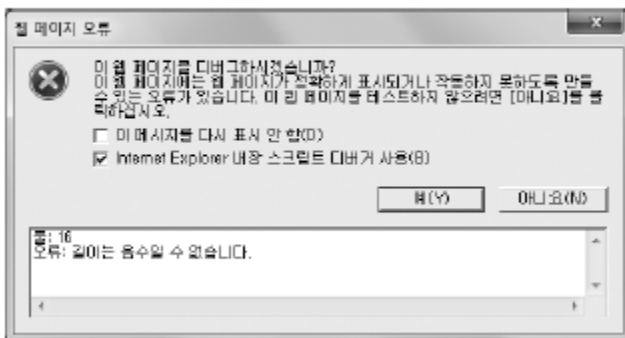


❖ 캡슐화 예제

■ 게터와 세터 (2)

```
// 변수를 선언합니다.  
var rectangle = new Rectangle(5, 7);  
rectangle.setWidth(-2);  
  
// 출력합니다.  
alert('AREA: ' + rectangle.getArea());  
</script>
```

그림 6-17 웹 페이지 오류



❖ 캡슐화 예제

■ 게터와 세터 (3)

- 게터 (Getter) `get○○()` 형태의 메서드와 같이 값을 가져오는 메서드
- 세터 (Setter) `set○○()` 형태의 메서드와 같이 값을 입력하는 메서드

■ 캡슐화

- 만일의 상황 대비해 특정 속성이나 메서드를 사용자가 사용할 수 없게 숨겨 놓는 것



❖ 상속

- 기존의 생성자 함수나 객체를 기반으로 새로운 생성자 함수나 객체를 쉽게 만드는 것
- 기존의 객체를 기반으로 생성
 - 상속을 통해 새로 만들어지는 객체에는 기존 객체의 특성 존재
- 예제 코드 6-36]
 - 정사각형 객체를 생성하는 생성자 함수 Square

코드 6-36 생성자 함수 Square 선언

```
<script>
  // 생성자 함수를 선언합니다.
  function Square(length) {
    this.width = length;
    this.height = length;
  }
  Square.prototype.getArea = function () {
    return this.getWidth() * this.getHeight();
  };
</script>
```



❖ 예제 코드 6-37]

- 생성자 함수 Rectangle 기반으로 생성자 함수 Square 작성

코드 6-37 상속

```
<script>
// 생성자 함수를 선언합니다.
function Rectangle(w, h) {
    var width = w;
    var height = h;

    this.getWidth = function () { return width; };
    this.getHeight = function () { return height; };
    this.setWidth = function (value) {
        if (value < 0) {
            throw '길이는 음수일 수 없습니다.';
        } else {
            width = value;
        }
    };
    this.setHeight = function (value) {
        if (value < 0) {
            throw '길이는 음수일 수 없습니다.';
        } else {
            height = value;
        }
    };
}
```



❖ 예제 코드 6-37]

```
Rectangle.prototype.getArea = function () {  
    return this.getWidth() * this.getHeight();  
};  
  
// 변수를 선언합니다.  
var rectangle = new Rectangle(5, 7);  
rectangle.setWidth(-2);  
  
// 출력합니다.  
alert('AREA: ' + rectangle.getArea());  
</script>
```

- 생성자 함수 Square 내부에서 작성
 - base 속성에 생성자 함수 Rectangle 넣고 실행
 - **Rectangle 객체 속성 Square 객체에 추가**
 - 생성자 함수 Square의 프로토타입에 Rectangle의 프로토타입 넣은 것
 - **Rectangle 객체의 프로토타입이 가지고 있는 속성 또는 메서드를 Square 객체의 프로토타입으로.**



❖ Rectangle 객체 상속받은 Square 객체 활용

코드 6-38 상속의 활용

```
<script>
// 생성자 함수를 선언합니다.
function Rectangle(w, h) {
    var width = w;
    var height = h;

    this.getWidth = function () { return width; };
    this.getHeight = function () { return height; };
    this.setWidth = function (value) {
        if (value < 0) {
            throw '길이는 음수일 수 없습니다.';
        } else {
            width = value;
        }
    };
    this.setHeight = function (value) {
        if (value < 0) {
            throw '길이는 음수일 수 없습니다.';
        } else {
            height = value;
        }
    };
}
```



❖ Rectangle 객체 상속받은 Square 객체 활용 (2)

```
Rectangle.prototype.getArea = function () {  
    return this.getWidth() * this.getHeight();  
};  
</script>  
<script>  
    // 생성자 함수를 선언합니다.  
    function Square(length) {  
        this.base = Rectangle;  
        this.base(length, length);  
    }  
    Square.prototype = Rectangle.prototype;  
    // 변수를 선언합니다.  
    var rectangle = new Rectangle(5, 7);  
    var square = new Square(5);  
  
    // 출력합니다.  
    alert(rectangle.getArea() + ' : ' + square.getArea());  
</script>
```

그림 6-18 실행 결과



❖ 상속

- 생성자 함수 Square의 프로토타입
 - constructor() 메서드 Square를 다시 넣는 부분은 없어도 정상적 작동
 - square 객체의 constructor() 메서드 출력
 - 생성자 함수 Rectangle 가리킴
 - 프로토타입의 생성자 함수를 재정의한 것

❖ 상속여부 판단

- instanceof 키워드를 사용할 때 true 출력하면 상속됐다고 판단

코드 6-39 상속의 확인

```
// 변수를 선언합니다.  
var square = new Square(5);  
alert(square instanceof Rectangle);
```

그림 6-19 상속의 확인



❖ call() 메서드

- 다른 함수의 메서드를 자신의 메서드처럼 사용할 수 있게 하는 메서드

코드 6-40 call() 메서드를 사용한 상속

```
<script>
    // 생성자 함수를 선언합니다.
    function Square(length) {
        Rectangle.call(this, length, length);
    }
    Square.prototype = new Rectangle();
    Square.prototype.constructor = Square;

    // 변수를 선언합니다.
    var square = new Square(5);
    alert(square.getArea());
</script>
```



❖ ECMAScript 5

- 인터넷 익스플로러 8 이하에서는 지원하지 않음
 - 인터넷 익스플로러 9 이상이나 그 외의 브라우저에서 테스트 필요
- 객체 속성 추가하는 메서드

표 6-1 ECMAScript 5의 객체 속성 추가 메서드

메서드 이름	설명
Object.defineProperty()	객체에 속성을 추가합니다.
Object.defineProperties()	객체에 속성들을 추가합니다.



❖ defineProperty() 메서드

- 첫 번째 매개 변수에는 속성 추가하려는 객체 입력
- 두 번째 매개 변수에는 속성 이름 입력
- 세 번째 매개 변수에는 속성과 관련된 옵션 객체 입력

코드 6-41 Object.defineProperty() 메서드

```
<script>
  var object = {};
  Object.defineProperty(object, 'name', {

  });
</script>
```

표 6-2 ECMAScript 5의 객체 속성 관련 옵션

옵션 이름	설명
value	속성의 값을 의미합니다.
writable	객체의 속성 값을 변경할 수 있는지를 의미합니다.
get	게터를 의미합니다.
set	세터를 의미합니다.
configurable	속성의 설정을 변경할 수 있는지를 의미합니다.
enumerable	for in 반복문으로 검사할 수 있는지를 의미합니다.



❖ value 옵션, writeable 옵션, enumerable 옵션

- name 속성 추가
- 옵션 객체에 writeable 속성과 enumerable 속성이 false
 - 속성 변경할 수 없음
 - 반복문 사용해 출력할 수도 없음

코드 6-42 value 옵션, writeable 옵션, enumerable 옵션

```
<script>
    // 변수를 선언합니다.
    var object = {};
    Object.defineProperty(object, 'name', {
        value: 'RintIanTta',
        writeable: false,
        enumerable: false
    });

    // 확인합니다.
    object.name = 'OTHER';
    for (var i in object) {
        alert(i + ' : ' + object[i]);
    }

    // 출력합니다.
    alert(object.name);
</script>
```



6.14 ECMAScript 5 객체 속성 추가

❖ get 옵션과 set 옵션

- get 옵션과 set 옵션 사용해 게터와 세터 만들기
 - get 옵션과 set 옵션 사용할 때 value 옵션 함께 사용할 수 없음

코드 6-43 get 옵션과 set 옵션

```
<script>
// 변수를 선언합니다.
var object = {};
var value = 'RINTIANTTA';
Object.defineProperty(object, 'name', {
  get: function () {
    alert('GETTER...!!');
    return value;
  },
  set: function (newValue) {
    alert('SETTER: ' + newValue);
    value = newValue;
  }
});

// 출력합니다.
object.name = 'ALPHA';
alert(object.name);
</script>
```

그림 6-20 게터 실행



❖ configurable 옵션

- configure 옵션의 기본값은 false
 - true로 설정하면 이미 지정한 속성의 설정 변경 가능

코드 6-44 configurable 옵션

```
<script>
  // 변수를 선언합니다.
  var object = {};
  var value = 'RINTIANTTA';
  Object.defineProperty(object, 'name', {
    get: function () { return value; },
    set: function (newValue) { value = newValue; },
    configurable: true
  });
  Object.defineProperty(object, 'name', {
    enumerable: true
  });
</script>
```



❖ defineProperty0 메서드

- 한 번에 하나의 속성만 지정 가능

❖ defineProperties() 메서드

- 한꺼번에 여러 개의 속성 지정 가능
- 첫 번째 매개 변수에는 속성 추가하고자 하는 객체
- 두 번째 매개 변수에는 속성과 그에 대응하는 옵션 객체.

코드 6-45 Object.defineProperties() 메서드

```
<script>
  // 변수를 선언합니다.
  var object = {};

  Object.defineProperties(object, {
    name: {
      value: 'RintIanTta'
    },
    gender: {
      value: 'Male'
    }
  });
</script>
```



❖ create() 메서드

- ECMAScript 5에서 배열을 생성할 때 쓰이는 메서드
 - 생성자 함수 - 틀을 기반으로 객체 찍어내 객체 생성
 - create() 메서드 - 기존에 있던 객체 복제, 새로운 속성 추가해 객체 생성

표 6-3 ECMAScript 5의 객체 생성 메서드

메서드 이름	설명
Object.create()	객체를 생성합니다.

코드 6-46 Object.create() 메서드

```
<script>
// 변수를 선언합니다.
var object = Object.create({}, {
  name: { value: 'RintanTta', enumerable: true },
  gender: { value: 'Male', enumerable: true }
});

// 출력합니다.
alert(Object.keys(object));
</script>
```

그림 6-21 create() 메서드를 사용한 객체 생성



❖ create() 메서드

■ 상속 구현한 예제

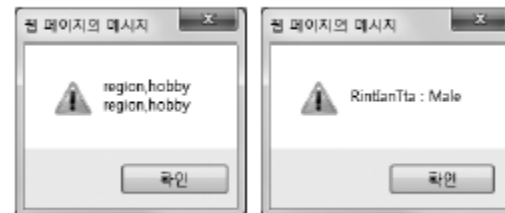
코드 6-47 상속

```
<script>
// 변수를 선언합니다.
var object = Object.create({}, {
    name: { value: 'RintianTta', enumerable: true },
    gender: { value: 'Male', enumerable: true }
});

var person = Object.create(object, {
    region: { value: 'Seoul, Korea', enumerable: true },
    hobby: { value: 'Guitar', enumerable: true }
});

// 출력합니다.
alert(Object.getOwnPropertyNames(person) + '\n' + Object.keys(person));
alert(person.name + ' : ' + person.gender);
</script>
```

그림 6-22 실행 결과



❖ ECMAScript 5 동적 속성 추가 제한

■ 제한 메서드 사용

표 6-4 ECMAScript 5의 동적 속성 추가를 제한하는 메서드

메서드 이름	설명
Object.preventExtensions()	객체의 속성 추가를 제한합니다.
Object.isExtensible()	객체에 속성 추가가 가능한지 확인합니다.

코드 6-48 객체의 동적 속성 추가

```
<script>
// 변수를 선언합니다.
var object = {};

// 간단한 객체 속성 추가 방법
object.gender = 'Male';

// 복잡한 객체 속성 추가 방법
Object.defineProperty(object, 'name', {
  value: 'RintlanTta',
  writable: false
});
</script>
```



❖ ECMAScript 5 동적 속성 추가 제한

- preventExtension() 메서드 사용
 - 객체에 gender 속성과 name 속성 추가
 - preventExtensions() 메서드 호출 → 사용 후에는 객체 추가 불가
 - dream 속성과 showMeTheMoney 속성 추가



6.16 ECMAScript 5 동적 속성 추가 제한

❖ ECMAScript 5 동적 속성 추가 제한

■ preventExtension() 메서드 예제 코드

코드 6-49 preventExtensions() 메서드

```
<script>
// 변수를 선언합니다.
var object = {};

// 간단한 객체 속성 추가 방법
object.gender = 'Male';

// 복잡한 객체 속성 추가 방법
Object.defineProperty(object, 'name', {
    value: 'RintianTta',
    writable: false
});

alert(Object.isExtensible(object));
Object.preventExtensions(object);
alert(Object.isExtensible(object));

// 간단한 객체 속성 추가 방법
object.dream = '생명공학자';

// 복잡한 객체 속성 추가 방법
Object.defineProperty(object, 'showMeTheMoney', {
    value: true
});
</script>
```

그림 6-23 객체 확장 불가능 오류

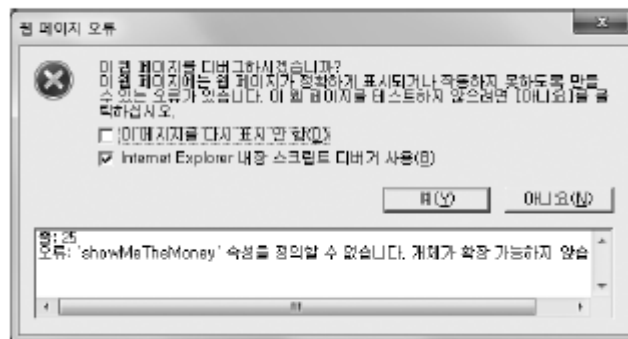


그림 6-24 오류가 발생하지 않아도 추가 안 됨

```
▼ object: Object
  gender: "Male"
  name: "RintianTta"
  ▶ __proto__: Object
    : undefined
```



6.17 ECMAScript 5 동적 속성 삭제 제한

❖ 동적 속성 삭제 제한 메서드

표 6-5 ECMAScript 5의 동적 속성 삭제를 제한하는 메서드

메서드 이름	설명
Object.seal()	객체의 속성 삭제를 제한합니다.
Object.isSealed()	객체에 속성 제거가 가능한지 확인합니다.

코드 6-50 Object.seal() 메서드

```
<script>
// 변수를 선언합니다.
var person = {
    name: 'RintianTta',
    gender: 'Male',
    region: 'Seoul, Korea',
    hobby: 'Guitar'
};

Object.seal(person);
delete person.name;

// 출력합니다.
alert(person.name);
</script>
```

그림 6-25 삭제되지 않은 name 속성



표 6-6 ECMAScript 5의 동적 속성 삭제 및 수정 제한 메서드

메서드 이름	설명
Object.freeze()	객체의 속성 삭제와 수정을 제한합니다.
Object.isFrozen()	객체에 속성 삭제와 수정이 가능한지 확인합니다.



❖ ECMAScript 5 객체 보조 메서드

표 6-7 ECMAScript 5의 객체 보조 메서드

메서드 이름	설명
Object.keys()	반복적으로 순환 가능한 객체 자신 소유의 속성을 배열로 만듭니다.
Object.getOwnPropertyNames()	모든 객체 자신 소유의 속성을 배열로 만듭니다.
Object.getOwnPropertyDescriptor()	특정 속성의 옵션 객체를 추출합니다.

코드 6-51 keys() 메서드와 getOwnPropertyNames() 메서드

```
<script>
// 변수를 선언합니다.
var object = { name: 'RintianTta' };

// 속성을 추가합니다.
Object.defineProperty(object, 'gender', {
    value: 'Male'
});

// 출력합니다.
alert(Object.keys(object));
alert(Object.getOwnPropertyNames(object));
</script>
```

그림 6-26 keys() 메서드와 getOwnPropertyNames() 메서드



❖ ECMAScript 5 객체 보조 메서드

- `getOwnPropertyDescriptor()` 메서드 - 특정 속성의 옵션 객체 추출
- 예제 코드 6-52]
 - object 객체의 name 속성 옵션 객체, gender 속성 옵션 객체 추출 후 출력

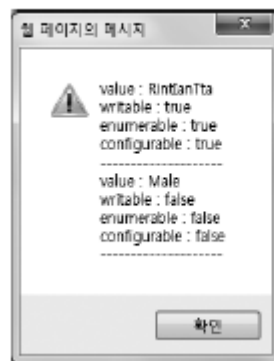
코드 6-52 `Object.getOwnPropertyDescriptor()` 메서드

```
<script>
// 변수를 선언합니다.
var object = { name: 'RintianTta' };
Object.defineProperty(object, 'gender', { value: 'Male' });

// 설정 배열을 추출합니다.
var descriptors = [];
descriptors.push(Object.getOwnPropertyDescriptor(object, 'name'));
descriptors.push(Object.getOwnPropertyDescriptor(object, 'gender'));

// 출력합니다.
var output = '';
for (var i in descriptors) {
    var item = descriptors[i];
    for (var key in item) {
        output += key + ' : ' + item[key] + '\n';
    }
    output += '-----\n'
}
alert(output);
</script>
```

그림 6-27 `getOwnPropertyDescriptor()` 메서드를 사용한 옵션 확인





Thank You !

모던 웹을 위한 Javascript jQuery 입문