웹 앱 개발을 위한 JavaScript 기초 강의 노트

제 10회차

객체 지향 프로그래밍

▣ 학습목표

- 프로토타입을 이해하고 활용할 수 있다.
- 상속에 대해 이해하고 다양한 방법으로 구현할 수 있다.
- 캡슐화에 대해 이해하고 활용할 수 있다.

▣ 학습내용

- 프로토 타입
- 상속
- 캡슐화

1. 프로토타입

1) 객체 지향 언어

- 객체 지향 프로그래밍

  - 모든 데이터를 객체 단위로 조합하고 프로그래밍 하는 것
  - 특징 : 상속, 캡슐화 등

- 클래스 기반의 언어

  - 클래스로 객체의 기본적인 형태와 기능을 정의
  - 생성자로 인스턴스를 만들어서 사용
  - 모든 인스턴스가 클래스에 정의된 대로 같은 구조이고, 보통 런타임에 변경할 수 없음
  - Java, C++

- 프로토타입 기반

  - 객체의 자료 구조, 메서드를 동적으로 변경 가능
  - JavaScript

- 클래스기반의 언어와 프로토타입 기반의 언어 장점 비교

클래스 기반의 언어	프로토타입 기반의 언어
정확성, 안정성, 예측가능성	동적으로 객체의 구조와 동작 방식을 바꿀 수 있음

1. 프로토타입

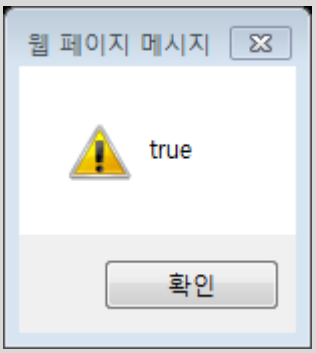
2) 표기법을 이용한 객체 생성 시 프로토타입

- 표기법을 이용한 객체 생성 시 프로토타입의 예와 결과

- 예

```
<script>
    var car = {
                    name: 'car1',
                    model: 400,
                color: 'black'
            };

    alert(car.hasOwnProperty('model'));

</script>
```

- 결과



- 프로토타입(prototype)

- 속성과 메서드 포함, 객체
- 모든 객체는 자신의 프로토타입을 가리키는 prototype이라는 숨겨진 속성을 가짐
- 모든 객체는 속성을 상속하는 프로토타입 객체에 연결됨
- 표기법({ })을 이용하여 생성된 객체는 JavaScript의 표준 객체인 Object의 속성인 프로토타입(Object.prototype) 객체에 연결
- 객체 생성 시 결정된 프로토타입 객체는 다른 객체로 변경 가능

1. 프로토타입

3) 생성자 함수를 이용한 객체 생성 시 프로토타입

- 생성자 함수를 이용해 객체를 생성하는 프로그램

• 예

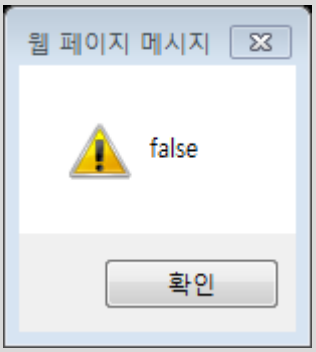
```
<script>

    function Make(name){
       this.name = name;
    }

  ① var person = new Make('홍길동');
  ② alert(person.name);
  ③ alert(person.hasOwnProperty('model'));

</script>
```

① 리턴값이 없으므로 this가 가리키는 객체가 생성자 함수의 리턴값이 되어 person 변수에 저장됨

② 생성된 person 객체의 name 속성의 값에 접근

③ 객체 내 model 속성이 있는지 확인

• 결과



1. 프로토타입

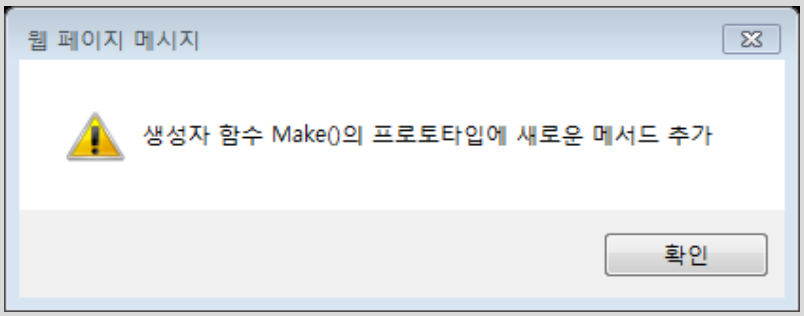
3) 생성자 함수를 이용한 객체 생성 시 프로토타입

- 생성자 함수를 이용해 객체를 생성하는 프로그램

```
<script>

    function Make(name){
       this.name = name;
     ① Make.prototype.print = function () {
          alert('생성자 함수 Make()의 프로토타입에 새로운 메서드 추가');
       }
    }

    var person = new Make('홍길동');
    alert(person.name);
    alert(person.hasOwnProperty('model'));
    person.print();

  </script>
```

① Make() 함수 객체의 prototype에 print()메서드를 동적으로 추가

- 결과



1. 프로토타입

4) 프로토타입 체이닝

- 프로토타입 체인 : 상위의 프로토타입 객체들과 연결된 프로토타입 링크들의 집합
- 프로토타입 체이닝 : 프로토타입 체인을 따라 상위 프로토타입 객체를 차례로 검색하는 것

```
<script>

    function Make(name){
        this.name = name;
        Make.prototype.print = function () {
            alert('생성자 함수 Make()의 프로토타입에 새로운 메서드 추가');
        }
    }

    var person = new Make('홍길동');
    alert(person.name);
① alert(person.hasOwnProperty('model'));
    person.print();

</script>
```

① person객체의 hasOwnProperty()메서드 호출

2. 상속

1) 상속

- 사전적 의미

> • 일정한 친족 관계가 있는 사람 사이에서 한쪽이 죽었을 때 다른 한쪽이 호주권이나 재산에 관한 권리, 의무 일체를 이어받는 것

- 객체 지향 프로그래밍에서의 의미

> • 한쪽이 다른 한쪽으로 자신의 속성과 메서드 등을 물려주어 사용할 수 있도록 하는 것

- 클래스 기반의 언어

> • 클래스 단위의 상속
> • 코드의 재사용의 한 형태
> • 클래스들의 관계는 계층을 형성

- 프로토타입 기반의 언어(JavaScript)

> • 객체의 특성을 그대로 물려 받는 또 다른 객체를 만들 수 있는 기능을 의미
> → 사용자에게 메시지를 전달하기 위한 대화 상자 출력
> • 기존의 특성을 수정하고 변경해서 파생된 새로운 객체를 만들 수 있음
> • 객체에서 중요한 점은 계층보다는 동작
> • 객체가 다른 객체로 바로 상속 가능

2. 상속

2) 상속 구현 방법

- 상속 구현하기

- 부모에 해당하는 함수를 통해 객체를 생성 후 자식에 해당하는 함수의 프로토타입 속성으로 생성한 객체를 참조
- 예

```
<script>
    function Make(name) {
        this.name = name;
    }

    ① Make.prototype.print = function () {
            alert('생성자 함수 Make()의 프로토타입에 새로운 메서드 추가');
    }

    ② function Person(name) {
        this.name = name;
    }

    Person.prototype = new Make();
  ⑤ Person.prototype.hello = function () {
        return 'hello world';
    }
    var person = new Person('홍길동');
    person.print();

   ⑥ alert(person.hello());

</script>
```

① 프로토타입 객체의 print 메서드 추가 부분을 함수의 밖으로 이동하도록 수정

② Person 생성자 함수 추가 선언

2. 상속

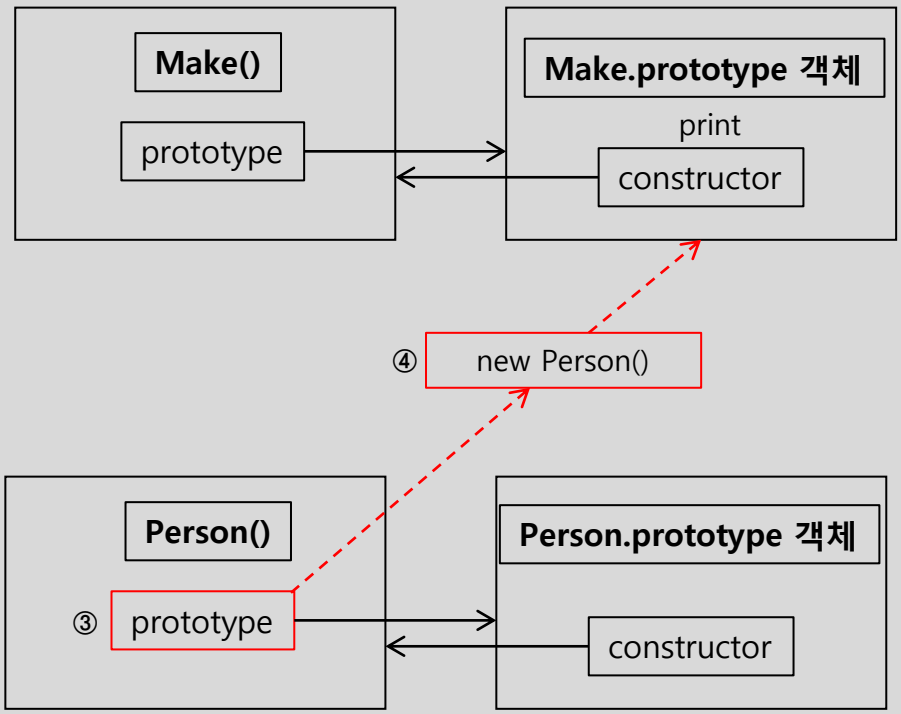
2) 상속 구현 방법

- 상속 구현하기

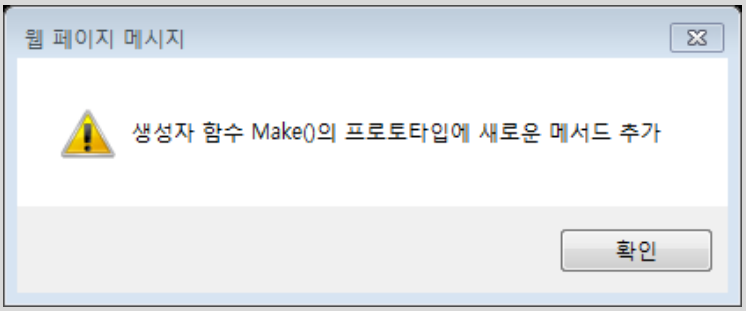
- 부모에 해당하는 함수를 통해 객체를 생성 후 자식에 해당하는 함수의 프로토타입 속성으로 생성한 객체를 참조
- 예

③ 자바스크립트에서 함수 생성 시 함수는 자신과 연결된 프로토타입 객체를 동시에 생성

④ Make() 생성자 함수는 new 키워드로 객체 생성



- 결과



2. 상속

2) 상속 구현 방법

- 상속 구현하기

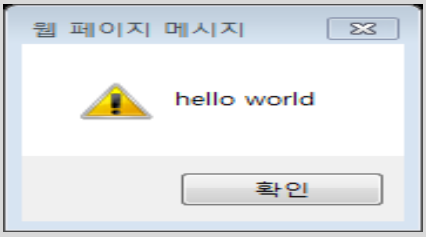
- 부모에 해당하는 함수를 통해 객체를 생성 후 자식에 해당하는 함수의 프로토타입 속성으로 생성한 객체를 참조
- 예

```
<script>
    function Make(name) {
        this.name = name;
    }

    ① Make.prototype.print = function () {
            alert('생성자 함수 Make()의 프로토타입에 새로운 메서드 추가');
    }

    ② function Person(name) {
        this.name = name;
    }

    Person.prototype = new Make();
  ⑤ Person.prototype.hello = function () {
        return 'hello world';
    }
    var person = new Person('홍길동');
    person.print();

  ⑥ alert(person.hello());

</script>
```

⑤ Make.prototype 객체의 속성을 물려받은 Person의 프로토타입 객체에 hello world 값을 돌려주는 hello() 메서드 추가

⑥ Person 생성자 함수에 의해 생성된 person 객체에서 새로 추가된 hello 메소드 호출

- 결과



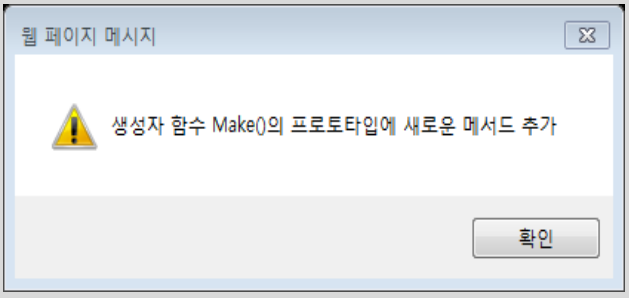
2. 상속

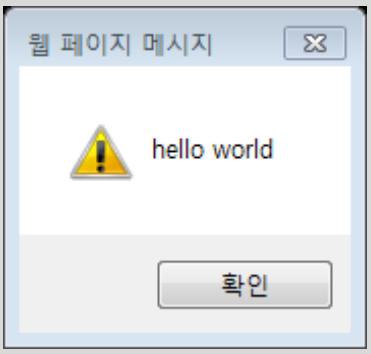
2) 상속 구현 방법

- 프로토타입 공유

```
<script>
    function Make(name) {
        this.name = name;
    }

    Make.prototype.print = function () {
        alert('생성자 함수 Make()의 프로토타입에 새로운 메서드 추가');
    }

    function Person(name) {
        this.name = name;
    }

  ① Person.prototype = Make.prototype;
    Person.prototype.hello = function () {
        return 'hello world';
    }

    var person = new Person('홍길동');
    person.print();
    alert(person.hello());
</script>
```

① 자식이 되는 함수를 통해 생성된 객체는 부모가 되는 함수의 프로토타입 객체와 프로토타입 연결을 갖게 됨





2. 상속

2) 상속 구현 방법

- 객체에서 객체로 상속 구현하기

- Object.create()

  → 괄호 안에 객체를 지정하여 새로운 객체를 생성하는 메서드

  → 새로 생성된 객체는 지정된 객체의 프로토타입과 링크로 연결

- 형식

```
var object1=Object.create(x)
```

→ x가 객체일 경우, 새로 생성되는 object1 객체는 x 객체의 프로토타입 객체와 링크로 연결

→ x 객체의 속성을 object1 객체가 상속받게 됨

- 객체가 다른 객체로 상속

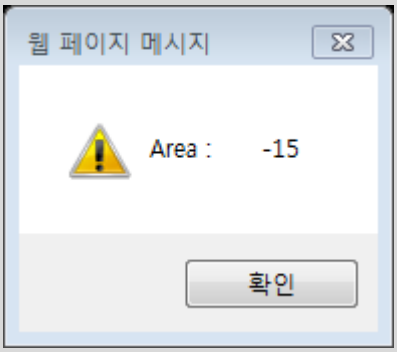
```
<script>
    var person = {
        age : 20,
        print: function () {
            alert('생성자 함수 없이 속성 구현');
        },
        getName: function () {
            return this.name;
        }
    };

    var person1 = Object.create(person);
    alert(person1.age);
    person1.print();


    person1.name = '홍길동';
    alert(person1.getName());

    person1.hello = function () {
        alert('hello world');
    }
    person1.hello();
</script>
```

3. 캡슐화

1) 캡슐화

- 사용자가 수정하지 말아야 할 것을 수정하거나 잘못된 입력을 방지
- 객체의 특정 부분을 사용자가 사용할 수 없게 막는 것
- 클로저를 이용하여 구현
- 예) 생성자 함수 외부에서 width와 height에 접근하여 잘못된 값 입력

```
<script>
    function Rectangle(width, height){
      this.width=width;
      this.height = height;

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

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

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

    var rectangle = new Rectangle(4, 5);
    alert('width : ₩t ' + rectangle.getWidth());
    alert('height : ₩t ' + rectangle.getHeight());
    alert('Area : ₩t' + rectangle.getArea());

  ① rectangle.width = -3;
    alert('Area : ₩t' + rectangle.getArea());
</script>
```



- rangle 객체의 width 값을 (마이너스 삼)-3으로 설정한 후 getArea메서드 호출 시 넓이가 -15로 출력됨 → 사각형의 가로, 세로, 넓이는 0보다 작을 수 없음
- 작성자와 사용자가 다를 시 유효한 값만 입력할 수 있도록 설정 필요 → 캡슐화

3. 캡슐화

2) 캡슐화의 구현

- 예) 생성자 함수 외부에서 width와 height에 접근을 하지 못하도록 수정
  - 게터(getter) : get으로 시작하여 값을 가져오는 메서드
  - 세터(setter) : set으로 시작하여 값을 설정하는 메서드

```
<script>
    function Rectangle(){
①     var width;
      var height;

②     this.setWidth = function (w) {
        if (w < 0) {
          return alert('음수는 입력할 수 없습니다.');
        } else {
          width = w;
        }
      };

③     this.setHeight = function (h) {
        if (h < 0) {
          alert('음수는 입력할 수 없습니다');
        } else {
          height = h;
        }
      };
```

① 생성된 객체의 속성에 추가되지 못하도록 일반 변수로 선언

② setWidth 함수의 매개값 w가 0보다 작으면 음수를 입력할 수 없다는 메시지 출력

③ setHeight 함수의 매개값 h가 0보다 작으면 음수를 입력할 수 없다는 메시지 출력

3. 캡슐화

2) 캡슐화의 구현

- 예) 생성자 함수 외부에서 width와 height에 접근을 하지 못하도록 수정
  - 게터(getter) : get으로 시작하여 값을 가져오는 메서드
  - 세터(setter) : set으로 시작하여 값을 설정하는 메서드

```
};
④ this.getWidth = function () {
        return width;
  };
  this.getHeight = function () {
        return height;
  };
  this.getArea = function () {
        return width * height;
  };

⑤ var rectangle = new Rectangle();
  rectangle.setHeight(10);
  alert('height : ₩t' + rectangle.getHeight());
  rectangle.setWidth(4);
  alert('width : ₩t' + rectangle.getWidth());
  alert('Area : ₩t' + rectangle.getArea());
</script>
```

④ width와 height를 일반 변수로 선언했기 때문에 자신을 가리키는 this 키워드 없이 변수의 값과 곱을 돌려주게 됨

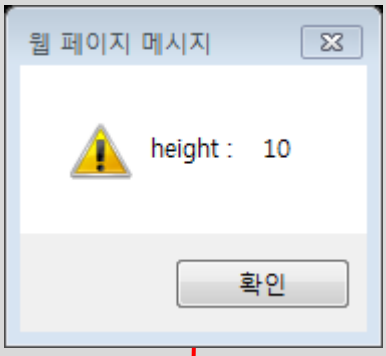
⑤ setWidth와 setHeight를 통해 값 설정

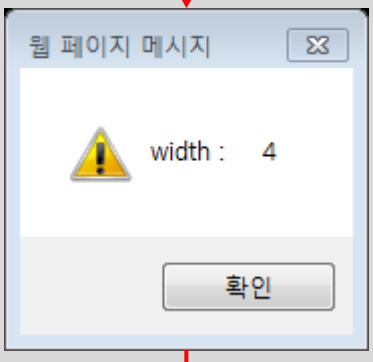
3. 캡슐화

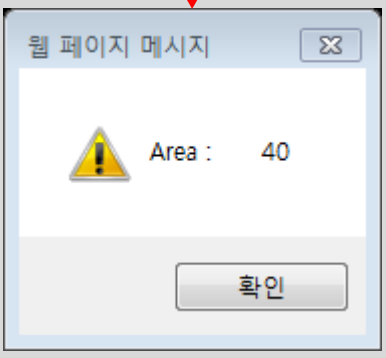
2) 캡슐화의 구현

- 예) 생성자 함수 외부에서 width와 height에 접근을 하지 못하도록 수정
  - 게터(getter) : get으로 시작하여 값을 가져오는 메서드
  - 세터(setter) : set으로 시작하여 값을 설정하는 메서드

- 결과







▣ 정리하기

1. 프로토타입

- 모든 객체는 자신의 프로토타입을 가리키는 prototype이라는 숨겨진 속성을 가짐
- 표기법({ })을 이용해 생성한 객체는 Object.prototype 객체에 연결되며, Object.prototype 객체는 프로토타입 체이닝의 종점임
- 생성자 함수를 통해 생성된 객체는 생성자 함수의 프로토타입 객체와 링크로 연결되어 있음
- 객체는 프로토타입 링크로 연결된 프로토타입 객체의 속성에 접근할 수 있음

2. 상속

- 상속은 객체의 특성을 그대로 물려 받는 또 다른 객체를 만들 수 있는 것을 의미하며, 상속을 통해 코드의 재사용이 가능함
- 부모에 해당하는 함수를 통해 객체 생성 후, 자식에 해당하는 함수의 프로토타입 속성으로 생성한 객체 참조하도록 함으로써 구현이 가능함
- 프로토타입 공유와 Object.create() 메서드를 이용하여 객체에서 객체로 상속을 구현할 수 있음

▣ 정리하기

3. 캡슐화

- 캡슐화는 내부의 동작 방법을 숨기고, 사용자에게는 사용 방법만을 알려주는 것을 의미함

- JavaScript에서는 클로저 개념을 이용하여 구현할 수 있음