JavaScript ve Nesne Yaratımı

Nesne Yaratımı

- Javascript'de iki türlü nesne oluşturulabilir
  - Literal syntax: {key1:val1,key2:val2,...}
  - New operatörü: new Object();

```
var p1 = { firstName : "Kenan", lastName : "Sevindik"};

var p2 = new Object();
p2.firstName = "Muammer";
p2.lastName = "Yücel";

function fullName() {
window.alert(this.firstName + " " + this.lastName);
}

fullName.call(p1); //Kenan Sevindik
fullName.call(p2); //Muammer Yücel
```

İlk yöntem

İkinci yöntem

Nesne Yaratımı

- Bir fonksiyonu new operatörü ile çağırarak da yeni bir nesne yaratılabilir
- İkinci yönteme benzer

Geleneksel olarak yeni nesne yaratmak için tanımlanan fonksiyonlar büyük harf ile başlarlar

```
function Person(firstName,lastName) {
    this.firstName = firstName;
    this.lastName = lastName;

    this.fullName = function() {
        window.alert(this.firstName +
                     " " + this.lastName);
    }
}
var p = new Person("Kenan","Sevindik");
p.fullName();
```

Nesne yaratan fonksiyon içerisinde Adım adım şunlar gerçekleşir:
1. this = {} ile implicit biçimde nesne yaratılır
2. this'e property ve metotlar eklenir
3. Fonksiyon sonunda this dönülür

Property

- Property tanımı nesne **yaratım aşamasında** veya **sonrasında** yapılabilir
- Tanımlı property'lere **dot** veya **square bracket notasyonu** ile erişilebilir
- Bir nesneye eklenmiş property'ler daha sonra **silinebilir**

```
p = {};
p.name = "Kenan";
alert(p.name);   //Kenan
alert(p["name"]);//Kenan
delete p.name;
alert(p.name);//undefined
```

Property

- Eğer bir property nesne içerisinde mevcut değil ise erişilmeye çalışıldığı vakit undefined değeri dönülür

```
p = {};
alert(p.name);                  //undefined
alert(p.name == undefined);     //true
alert("name" in p);             //false

p = {name : undefined};
alert(p.name);                  //undefined
alert(p.name == undefined);     //true
alert("name" in p);             //true
```

in operatörü ile property'nin nesne içerisinde tanımlı olup olmadığı kontrol edilebilir

Property

- Nesne içerisindeki tanımlı property'ler üzerinde iterate etmek de mümkündür

```
p = {firstName:'Kenan',lastName:'Sevindik',age:39};

for(key in p) {
    alert(p[key]);
}
```

in operatörü ile p nesnesinin sahip olduğu bütün property'lere erişilebilir. Bu property'lere ata nesneden gelen property'lerde dahildir

Property'nin ata nesneye ait olup olmadığı **hasOwnProperty()** metodu ile anlaşılabilir

Fuction

- Nesne üzerinde property'lerde sadece basit değeler değil, fonksyionlar da tutulabilir
- Tanımlı fonksiyonlar nesne üzerinde dot notasyon ile invoke edilebilir
- Javascript'de fonksiyonlarda aslında birer nesnedir

```
p = new Object();
p.firstName = 'Kenan';
p.lastName = 'Sevindik';

p.fullName = function() {
    alert(this.firstName + "" + this.lastName);
}

p.fullName();
```

Object Variables

- Bir nesnenin atandığı değişken bu **nesneye referans** tutmaktadır
- Başka bir ifade ile bir **pointer** ile asıl veriye refer etmektedir
- Bu değişken üzerinden yapılan **herhangi bir değişiklik** bütün diğer referanslara da yansıyacaktır
- Nesne bir fonksiyona parametre olarak geçtiğinde de **aynı durum** söz konusudur
- Nesnenin referansı **fonksiyona parametre** olarak geçmektedir

Built-in Nesneler

- Javascript standart kütüphanesinde built-in bir takım nesneler mevcuttur
- Bu nesnelerden en yaygın olanları: Math, Date, RegExp
- Bu nesneler global object olarak da bilinirler

```
alert(Math.abs(-2));      //2

var d = new Date(1999,06,27);
alert(d.getTime());       //933022800000

var exp = new RegExp("ab+c","i");
alert(exp.test("abBBc"));//true
alert(exp.test("ac"));    //false
```

Built-in Nesneler

- String, Number, Boolean gibi nesneler de built-in'dir
- Fakat bunlar hiçbir zaman explicit nesne yaratmak için kullanılmazlar
- Primitif karşılıkları doğrudan kullanılır ve bu nesnelerde tanımlı metotlara erişilebilir

```
alert("Kenan Sevindik".length);            //14
alert("Kenan Sevindik".substring(0, 3));   //Ken
alert(true.toString());                    //true
```

Statik Değişkenler

- Javascript'de Java'daki gibi static keyword mevcut değildir
- Ancak statik olmasını istediğimiz property (değişkeni) doğrudan function nesnesinin içerisine koyabiliriz

```
function sayHello(name) {
    sayHello.count = ++sayHello.count || 1;

    alert("Hello " + name);
}

sayHello("Kenan");
sayHello("Muammer");

alert("sayHello invoked " + sayHello.count + " times"); //2
```

Statik Metotlar

- Statik metotlar da statik değişkenler gibi function nesnesine eklenerek oluşturulurlar

```
function Person(firstName,lastName) {
    this.firstName = firstName;
    this.lastName = lastName;

    arguments.callee.count = ++arguments.callee.count || 1;
}

Person.showCount = function() {
    alert(Person.count);
}

p1 = new Person("Kenan","Sevindik");
p2 = new Person("Muammer","Yücel");

Person.showCount();
```

Bir önceki Örnekteki statik değişkeni tanımlarken function ismi kullanılmıştı. Function ismi yerine callee kullanılabilir

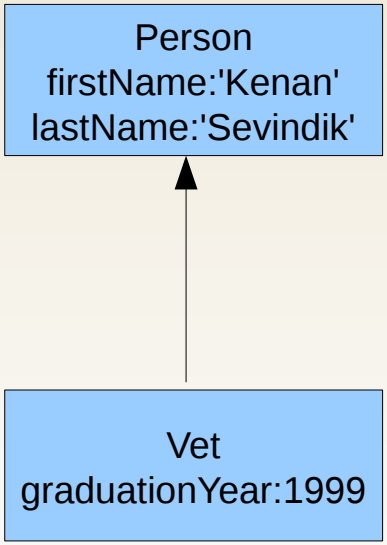
JavaScript ve Inheritance

harezmi bilişim çözümleri

Inheritance

- Java gibi bir OO programlama dilinde **nesneler sınıflardan** oluşturulur
- Inheritance'da **sınıflar arasında** kurulur
- Javascript'de ise **sınıf kavramı yok**tur
- Inheritance **nesneler arasında** gerçekleşir
- Javascript nesneleri **__proto__** şeklinde bir property'de inherit edilen ata nesneye referans barındırır

Prototip Tabanlı Inheritance




```
var p = new Object();
p.firstName='Kenan';
p.lastName='Sevindik';

var v = new Object();
v.__proto__=p;
window.alert(v.firstName + " " + v.lastName);
```

FirstName ve lastName property'lerine vet Nesnesi üzerinden erişildiği vakit interpreter Önce bu property'lerin vet nesnesi üzerinde Olup olmadığına bakar. Eğer vet nesnesinde Yoksa **__proto__** üzerinden ata nesneye geçilir ve property çözümlemesine oradan devam edilir

__proto__ property'si standart değildir. Firefox ve Chrome'a özeldir. Diğer tarayıcılarda da bu property mevcuttur. Fakat hidden'dır.

Diğer tarayıcılarda Object.create() metodu kullanılır

```
var v = Object.create(p);
window.alert(v.firstName + " " + v.lastName);
```

Prototype

- Standart biçimde __proto__ property'sini set etmeyi sağlar
- Bunun için constructor fonksiyonlarına ihtiyaç vardır

```
var person = new Object();
person.firstName = "Kenan";
person.lastName = "Sevindik";

function Vet() {
}

Vet.prototype = person;

var vet = new Vet();

alert(vet.firstName + " " + vet.lastName);
```

Bu tanım new Vet(); ile yaratılacak bütün nesnelerin __proto__ property'sinin değerinin person olmasını sağlar

Metot Dispatch

- Inherit edilen ata nesneye bir fonksiyon eklenirse bu fonksiyon aynı şekilde alt nesnelerden de erişilebilir olacaktır

```
var p = new Object();
p.firstName='Kenan';
p.lastName='Sevindik';

var v = Object.create(p);

p.fullName = function() {
window.alert(this.firstName + " " + this.lastName);
}

v.fullName();
```

JavaScript ve this

this Anahtar Kelimesi

- This anahtar kelimesinin **değeri dinamik**tir
- Değeri **fonksiyon çağrıldığı anda** tespit edilir
- This anahtar kelimesinin değeri Javascript içerisinde **dört farklı şekil**de tespit edilebilir
  - Metot olarak çağrıldığında
  - Fonksiyon olarak çağrıldığında
  - new operatöründe
  - Explicit olarak this

Metot Olarak Çağrıldığında

- Eğer bir fonksiyon bir nesne üzerinden çağrılmış ise bu durumda this nesneye refer eder
- Nesne üzerinden fonksiyon çağrılması “.” veya “[ ]” kullanım ile olabilir

```
var p = new Object();
p.firstName = "Kenan";
p.lastName = "Sevindik";

p.fullName = function() {
window.alert(this.firstName + " " + this.lastName); //"Kenan Sevindik"
window.alert(this == p); //true
};

p.fullName();
```

Fonksiyon Olarak Çağrıldığında

- Fonksiyon içerisinde this, window veya global object'e karşılık gelmektedir
- Bir fonksiyon içerisinde this kullanılması yeni tarayıcılarda undefined hatası üretebilir
- Tarayıcılar geri uyumluluk adına bu davranışı korumaktadırlar

```
function foo() {
  alert(this) // [object Window] veya [object global]
}

foo();
```

new Operatöründe

- Javascript'de nesne yaratma yöntemlerinden birisi **new operatörü**nden sonra bir fonksiyonun çağrılmasıdır
- Böyle bir durumda this **yeni bir Object** ile initialize edilir

```
function Person(firstName,lastName) {
    this.firstName = firstName;
    this.lastName = lastName;

    this.fullName = function() {
        window.alert(this.firstName +
                     " " + this.lastName);
    }
}
var p = new Person("Kenan","Sevindik");
p.fullName();
```

Nesne yaratan fonksiyon içerisinde
Adım adım şunlar gerçekleşir:
1. this = {} ile implicit biçimde nesne yaratılır
2. this'e property ve metotlar eklenir
3. Fonksiyon sonunda this dönülür

Explicit This

- Herhangi bir fonksiyon explicit this değeri ile çağrılabilir
- Bu **call** veya **apply** metotları üzerinden gerçekleşebilir
- Syntax call(context,args...) ve apply(context,args[]) şeklindedir

```
function Person(firstName,lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}
fullName = function() {
    window.alert(this.firstName + " " +
                 this.lastName);
}

var p = new Person("Kenan","Sevindik");
fullName.call(p);
```

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com