CONSTRUCTOR VE PROTOTYPE NEDİR?

Javascript'te classes *(sınıflar)* kavramı bulunmamaktadır ve Javascript classes *(sınıflar)* kavramı yerine fonksiyonları kullanmaktadır. Bu bir dezavantaj değildir. Aksine bazı durumlarda çok daha avantajlıdır.

Javascript'te class *(sınıf)* yapısı tanımlamak için oluşturulan fonksiyonlar, constructor *(yapıcı metot)* olarak adlandırılır. OOP *(Object Oriented Programming) (nesne tabanlı programlama)* terminolojisinde new *(yeni)* anahtarı ile bir object *(nesne)* türetildiğinde constructor'lar *(yapıcı metotlar)* çağarılmaktadır. Constructor'lar *(yapıcı metotlar)*, class'dan *(sınıftan)* türetilecek object'lerin *(nesnelerin)* özelliklerine ilk değer atamalarını yapmak için kullanılır. Yani constructor'lar *(yapıcı metotlar)* object'leri *(nesneleri)* ilk kullanıma hazırlamaktadır. Bu açıdan Javascript'te class *(sınıf)* yapısını tanımlamak için oluşturduğumuz fonksiyonlar, aynı zamanda constructor *(yapıcı metot)* olarak çalışmaktadırlar.

Prototype *(prototip)* özelliği Javascript'te Inheritance'ın *(kalıtımın)* temelini oluşturmaktadır. Inheritance *(kalıtım)* var olan class'ları *(sınıfları)* kullanarak, yeni class'lar *(sınıflar)* tanımlanmasına olanak sağlamaktadır. Bir class'dan *(sınıftan)* başka bir class *(sınıf)* türetildiğinde, türetilen class *(sınıf)*, ana class *(sınıf)* tarafından tanımlanan tüm özellikleri, metotları vs. class *(sınıf)* üyelerini kalıtım yoluyla elde etmektedir. Bu yapı bir özellik veya metot grubunun farklı class'lar *(sınıflar)* içerisinde tekrar tanımlanmasını engellemek için iyi bir yol sunar.

Javascript'te Inheritance *(kalıtım)*, prototype *(prototip)* tabanlıdır. Yani aslında Javascript'te önceden tanımlı yada kullanıcı tarafından tanımlanan tüm object'ler *(nesneler)*, object *(nesne)* nesnesinin prototype *(prototip)* özelliği ile erişilebilen prototype *(prototip)* nesnesinden türetilmektedir. Javascript'te bir fonksiyon oluşturulduğunda prototype *(prototip)* özelliği ilgili fonksiyona otomatik olarak eklenir. Fonksiyonlarında birer object *(nesne)* olduklarını unutmamak lazım. Javascript'te sadece constructor'lar *(yapıcı metotlar)* ve ön tanımlı object'ler *(nesneler)* prototype *(prototip)* özelliğine sahiptir. Yani kısaca kullanıcı tarafından tanımlanan object'ler *(nesneler)* prototype *(prototip)* özelliğine sahip değildir. Prototype *(prototip)* özelliği constructor'lara *(yapıcı metotlara)* Inheritance *(kalıtım)* olarak gelen prototype object'ine *(prototip nesnesine)* ulaşmak için kullanılır. Default *(varsayılan)* olarak constructor'lar *(yapıcı metotlar)* kullanılarak oluşturulan her object *(nesne)* oluşturduğu constructor'ın *(yapıcı metodun)* prototype *(prototip)* özelliği ile tanımlanan prototype object'inin *(prototip nesnesinin)* özellik ve metotlarını Inheritance *(kalıtım)* olarak alır. Bir constructor *(yapıcı metot)* prototype object'ini *(prototip nesnesini)*, object *(nesne)* nesnesinin prototype object'inden *(prototip nesnesinden)* Inheritance *(kalıtım)* olarak almaktadır. Kısaca Javascript'te oluşturulan tüm object'ler *(nesneler)* aslında Object.Prototype object'inden *(nesnesinden)* türetilmektedir. Object.Prototype object'inin *(nesnesinin)* kendisine ait birtakım özellik ve metotları da bulunmaktadır. Bu özellik ve metotlardan bahsedecek olur isek;

- **constructor *(yapıcı metot)***

Object'in *(nesnenin)* oluşturduğu yapıcı metoda *(işleve)* erişmek için kullanılan özelliktir.

- **__proto__**

Object'i *(nesneyi)* oluşturan prototype object'ini *(prototip nesnesini)* elde etmek için kullanılan özelliktir.

- **hasOwnProperty**

Bir object'in *(nesnenin)* parametrik olarak girilen özelliğinin kullanılıp kullanılmadığını test etmek için kullanılır.

- **isPrototypeOf**

Parametrik olarak verilen object'in *(nesnenin)* prototype *(prototip)* zincirinde, bir constructor'ın *(yapıcı metodun)* bulunup bulunmadığını test etmek için kullanılır.

- **unwatch**

Belirtilen bir özelliğinin değeri değiştiğinde, eklenmiş olan herhangi bir işlevi kaldırmak için kullanılır.

- **propertyIsEnumerable**

Bir object'in *(nesnenin)* parametrik olarak girilen özelliğinin kullanılıp kullanılmadığını ve bu özelliğin numaralandırılabilir olup olmadığını test etmek için kullanılır.

- **toLocaleString**

Bir object'in *(nesnenin)* karakter dizesi olarak temsil eden halini döndürmek için kullanılır. *(geçerli lokasyona görede işlem yapabilir. örneğin; lokasyonda geçerli saat dilimi vs.)*

- **toString**

Bir object'in *(nesnenin)* karakter dizesi olarak temsil eden halini döndürmek için kullanılır.

- **toSource**

Bir object'in *(nesnenin)* kaynak kodunu temsil eden halini döndürmek için kullanılır.

- **valueOf**

Bir object'in *(nesnenin)* temel değerini elde etmek için kullanılır.

- **watch**

Belirtilen bir özelliğinin değeri değiştiğinde, çalıştırılacak herhangi bir işlevi eklemek için kullanılır.