NESNE TABANLI PROGRAMLAMA NEDİR?

Javascript birçok gelişmiş yazılım dilleri gibi OOP *(Object Oriented Programming) (nesne tabanlı programlama)* imkânı sunmaktadır. Üst düzey programlama geliştirmek zor, karmaşık, zahmetli, yüksek boyutlu ve güncellenmesi uzun süreler alan yazılımlardır. Bu anlayış üzerinden yola çıkacak olursak nesne tabanlı programlama sayesinde daha modüler yapılara sahip, sistematik, çok biçimli ve çok daha güvenli yazılımlar yapılmaktadır. Nesne tabanlı programlama içerisinde belirttiğimiz türde yazılımları yapabilmemizi sağlayan bir takım kavramlar ve unsurlar bulunmaktadır. Bu kavramlar ve unsurlardan bahsedecek olur isek;

- Objects *(nesneler)*
- Classes *(sınıflar)*
- Encapsulation *(kapsülleme)*
- Aggregation *(bütünleme)*
- Inheritance *(kalıtım)*
- Polymorphism *(çok biçimlilik)*

Objects *(nesneler)*

Nesne kavramı nesne tabanlı programlamanın en temel unsurudur. Burada nesne aslında soyut bir kavramdan ibarettir. Nesneler bir isme ve özelliklere sahiptir. Belirlenecek olan tüm opsiyonel işlemleri yapar. Gerçek hayattan örnekle açıklayacak olur isek; Örneğin; İnsan bir nesnedir. Programlamadaki nesne kavramı ile karşılaştırıldığında yine aynı kavrama sahiptir. Çünkü, her insanın bir adı vardır *(Volkan, Hakan, Onur vs.)*, her insanın bir takım yaptığı işler vardır *(yürümek, koşmak, uyumak vs.)* ve her insanın bir özelliği vardır *(boy, kilo, göz rengi vs.)*. Programlamada da oluşturulan her nesnenin, bir insan gibi adı, işi ve özelliği vardır. Yapmak istediğimiz işe ve özelliğe göre nesneleri kendimiz oluşturabiliriz. Ayrıca gelişmiş yazılım dillerinde, tıpkı Javascript'te de olduğu gibi hazır işlevlere ve özelliklere sahip nesnelerde vardır.

Classes *(Sınıflar)*

Nesneyi anlatırken soyut kavramına değindik ve örnek ile yaptığımız açıklamada bir ismin, yaptığı bir işin ve kendine ait özelliğin olduğundan bahsettik. Peki bu nesneleri bilgisayar ortamında nasıl barındırabiliriz? Bu değer ve özellikleri nasıl tanımlayabiliriz? İşte bu soruların cevabında sınıf kavramı devreye giriyor ve isim, iş ve özellik bakımından nesneleri sınıflandırmamızı sağlıyor. Programlamada her zaman bir nesneyi sınıf veya sınıflar temsil eder. Javascript'te sınıf kavramı yoktur ve sınıf kavramı yerine fonksiyonları kullanmaktadır.

Encapsulation *(Kapsülleme)*

Kapsülleme nesnelerin özelliklerini diğer nesnelerden yada dışarıdan saklanmasını sağlayan bir kavramdır. Nesne tabanlı programlamalarda kapsülleme üç erişim kanalı ile sağlanır. Bu erişim kuralları göz önünde bulundurulduğunda Javascript'te işler bazen daha farklı yürüyebiliyor.

- Public *(açık)*
  Belirlenen özellik ve metotlara her nesneden erişilebilir.
- Private *(özel)*
  Belirlenen özellik ve metotlara sadece kendi içerisinden erişilebilir.
- Protected *(korumalı)*
  Belirlenen özellik ve metotlara sadece kendi içerisinden ve kendisinden türeyen nesneden erişilebilir.

Aggregation *(bütünleme)*

Her bir nesne alt nesnelerden oluşur. Verdiğimiz örnekte insan bir nesnedir. Fakat insanı oluşturan beyin, kalp, akciğer, böbrek vs. de kendi başına birer nesnedir. Kısacası birden fazla nesnenin birleşip yeni bir nesne ortaya çıkardığı durumlara bütünleme denir.

Inheritance ***(kalıtım)***

Bir nesnenin başka bir nesneden türediği durumlara kalıtım denir. İnsanı örnek vermiş ve insan bir nesne demiştik. Bu insan nesnesi belirli özellik *(isim, boy, kilo vs.)* ve metotlara *(yürümek, koşmak, uyumak vs.)* sahiptir. Pekala o zaman iki farklı nesne daha ele alalım. Örneğin biri doktor, diğeri ise futbolcu olsun. Bu durumda insan nesnesinden türeyen yeni bir doktor birde futbolcu nesnemiz daha oldu. Bu iki nesnenin insan nesnesinden türemesi bu nesnelerin kalıtıldığı nesne özelliklerine sahip olduğu anlamına gelir. Yani doktor nesnesi de, futbolcu nesnesi de isim, boy ve kilo özelliklerine sahiptir. Bu özelliklerin yanında kalıtıldığı nesne olan insan nesnesinin yürümek, koşmak ve uyumak gibi metotlarına da sahiptir.

Polymorphism ***(çok biçimlilik)***

En anlaşılır şekilde anlatılacak olur ise, polymorphism *(çok biçimlilik)*, nesnelerin belirli bir işi farklı yöntemler ile yapmasıdır. Bu yazılım dili içerisindeki nesne arayüzleri sayesinde gerçekleşir. Nesne arayüzleri nesnenin bulundurması gereken özellikleri belirten yükümlülük ilkeleridir. Örneğin X adında bir arayüz oluşturduğumuzu düşünürsek ve bunun içerisinde de XA ve XB adında iki tanede metot tanımladığımızı düşünelim. Eğer oluşturduğumuz bir A nesnesini bu nesne arayüzünden yükümlü tutarsak, bu A nesnesinin içerisinde XA ve XB metotlarını uygulamak zorunda olduğumuz anlamına gelir.

Daha net bir örnek vererek açıklayacak olur isek; Hareket adına sahip bir arayüzümüz olduğunu düşünelim. Bu Hareket arayüzünün içerisinde A'ya Git adında bir metot tanımlı olsun. Daha sonrada bir minibüs ve birde otobüs nesnemiz olsun ve bu iki nesne Hareket arayüzüne yükümlü tutulsun. Bu, bu iki nesnenin A'ya Git adında bir metoda sahip olması gerektiği anlamına gelir. İşte bu noktada çok biçimlilik devreye giriyor. Bu iki nesnenin de aynı işi yapmak zorunda olması demektir. Fakat aynı şekillerde yapmak zorunda değildirler. Yani minibüs A noktasına giderken a, b ve c işlemlerini yaparken, otobüs A noktasına giderken d, e ve f işlemlerini yapabilir. Gerçek hayattan anlatacak olursak, minibüs A noktasına x sokaktan, otobüs A noktasına y sokaktan gidebilir. Ama işlem aynıdır. İkisi de A noktasına gitmek zorundadır. Buna da çok biçimlilik denir.