
İÇERİK

4. Nesne Yönelimli Programlamaya Giriş	2
4.1. Nesne Yönelimli Programlama (Object Oriented Programming) Tekniği Nedir?	2
4.2. Nesne Yönelimli Programlama Tekniğinin Avantajları	5
4.2.1. Nesnelerin Bilgi Gizleme Özelliği	5
4.2.2. Nesnelerin Kalıtım Özelliği	10
4.2.3. Nesnelerin Çok Biçimlilik Özellikleri	14
4.3. Sınıf Nedir?	16
4.4. Kendin Pişir Kendin Ye! Örneği	18

ÜNİTE HAKKINDA

Programlamaya yeni bir bakış açısı ile karşı karşıyayız. Daha doğrusu önceden bu teknikle çalışmamış iseniz yeni bir bakış açısı sayılabilir: Nesne Yönelimli Programlama Tekniği.

Bu teknik sizi, klasik yordamsal programlama yöntemlerinden kurtararak, size daha dinamik ve daha modüler bir programlama yapmanıza olanak sağlayacaktır.

Aslına bakarsanız bu üniteye kadar zaten biz bu teknigi bir şekilde kullanıyorduk. Zira birçok örnekte nesneler ve özellikleri ile ilgili programlar yazmıştık. Yalnız, daha çok sözel bir şekilde geçecek olan bu üniteyle birlikte, Görsel Programlama için olmazsa olmazlarından biri olan Nesne Yönelimli Programlama teknigi hakkında daha çok bilgi sahibi olacak ve bu teknigi görece daha bilinçli bir şekilde inceleyerek mantığını kavramaya çalışacağız.

ÖĞRENME HEDEFLERİ

Bu ünite bitiminde;

- Nesne ve Sınıf kavramlarını açıklayabilecek,
- Nesnelerin özelliklerini ve olaylarını tanıabilecek,
- Nesnelerin temel prensiplerini bilebilecek,
- Delphi'de ki sınıf ve nesneleri ayırt edebilecek ve
- Var olan sınıflardan yararlanarak ve RAD Studio olmadan (sırf Delphi kodlarıyla) çalışma zamanında kendi nesnelerinizi oluşturabileceksiniz.

ÜNİTEYİ ÇALIŞIRKEN

Günümüzdeki programlama dillerinin çoğu NYP teknüğine az ya da çok destek vermektedir. NYP teknüğü ile ilgili olarak öğrendiklerinizi Delphi haricinde ki başka programlama dillerinde de (C#, java, C++ gibi) uygulama yoluna gidebilirsiniz.

Bu size dilden bağımsız bir şekilde tekniğin nasıl çalıştığını dair fikir verecektir.

ANA METİN

4. Nesne Yönelimli Programlamaya Giriş

Artık Delphi ortamını ve projelerini tanadığımıza göre elimizi iyiden iyiye Delphi programlamaya bulaştırmamızın zamanı gelmiştir. İşin aslı zaten elimizi daha ilk üniteyle birlikte Delphi programlamaya bulaştırmıştık, fakat bazı konularda durumun çok da bilincine varmadan yaptığımız örneklerdi bunlar. Artık bu üniteyle birlikte, programcılık hayatımızda önemli bir yeri bulunması gereken yeni bir programlama tekniğinin de farkına varmaya ve bu tekniği kullanarak da bilinçli bir şekilde *görsel programlama* yapmaya başlayacağız.

Görsel programlama kavramını açıklamaya çalışırken yaptığımz tanımlamalarda işletim sistemi tarafından sağlanan *nesnelerden* bahsetmiştim.

• Peki, ama neydi bu nesneler?

Metin giriş kutuları, butonlar, seçenek düğmeleri... dediğinizi duyar gibiyim. Doğrudur, bunlar işletim sistemi tarafından programlarımızda kullanılmak üzere sunulan ve kolaylıkla kullanabildiğimiz *daha doğrusu Delphi gibi bir görsel programlama dili yardımıyla* kolaylıkla kullanabildiğimiz yapılardır. Daha da sayamadığımız yüzlercesi bulunmaktadır.

• Fakat bütün bunlar neden *nesne* olarak ifade edilmektedir ve bu nesnelerin de *gerçek hayatı* *nesnelerle* bir ilişkisi var mıdır?

İşte tüm bu soruların cevapları *Nesne Yönelimli Programlama* diye tabir edilen yeni bir programlama tekniğinde yatmaktadır.

4.1. Nesne Yönelimli Programlama (Object Oriented Programming) Tekniği Nedir?

Bu teknik, programlamaya yeni –yeni derken en azından bizim için yeni- bir bakış açısındandır. Uygulamaların tasarımını, yeniden düzenlenmesini ve birçok kişinin aynı uygulama üzerinde birlikte çalışmasını kolaylaştmak amacıyla geliştirilmiştir. Bu tekniğin avantajları daha çok, *büyük çaplı uygulamalarda* kendisini göstermektedir (Hatta küçük çaplı uygulamaların çoğunda ise “*ben bunu şu yolla daha kolay yapardım*” dedirttiği de olmaktadır. Özellikle de nesneleri kendiniz oluşturmak zorunda kaldığınızda, ama iyi haber: biz daha çok, önceden oluşturulmuş nesnelerle programlama yapacağız, işte bu da görsel programlama olmaktadır).

NYP (Nesne Yönelimli Programlama) tekniği birkaç küçük dezavantajı yanında yine de birden fazla kişinin aynı proje üzerinde çalışması ve yeniden düzenlenenebilirlik söz konusu olunca görsel programlama konusunda vazgeçilmez tekniklerimizden birisi olacaktır.

Programlama konusunda ki bu yeni yaklaşıma göre programlama ortamında ki her şey potansiyel bir nesnedir ve dolayısıyla programlama da bu *Nesne Yönelimli* ortamda yapılmaktadır.

?
Peki, ama bu nesnelerin bildiğimiz nesnelerle bir ilişkisi var mıdır?

Bu güne kadar bilgisayar programlama sanatını öğrenmeye çalışırken, hep bizler bilgisayara yaklaşmıştık, yani hep bizler onun o garip dilini öğrenmeye çalışmıştık. Hatırlayın, algoritma derslerimizde problemlerimizin algoritmalarını kurarken *daha böyle bir bilgisayar dili ile düşünmeye gayret edip*,

- ☞ Başla,
- ☞ Şu oluncaya kadar şu işi şu kadar yap,
- ☞ Sonra şu satırdan devam et...
- ☞ Dur.

gibi komutlarla derdimizi ona anlatmaya çalışırdık.

İşte şimdi bu teknik yardımıyla bilgisayarı az da olsa bizim tarafa çekmeyi başarıbiliyor ve programlama ortamını *nesne yönelimli* bir hale getirerek derdimizi ona biraz daha *bizden bir şeylerle ifade etmeye* çalışıyoruz. Bu anlamda Nesne Yönelimli Programlama tekniğinde bahsedilen nesnelerin *evet, gerçek hayatı nesnelerle bir bağlantı bulunmaktadır*.

?
Peki, nesne nedir?

Aslına bakarsanız nesnenin tam tanımını yapmaya çalıştığımızda, elle tutulabilen, gözle görülebilen, kütlesi ve hacmi bulunan diye bir giriş yapıp, uzun uzadıya da tanımlamak mümkün değildir. Fakat bizim işimize yarayacak şekilde bir tanımlama yapmaya çalıştığımızda ise işin özünde *algılamak* olduğu ortaya çıkmaktadır. Sonuçta biz *nesneleri algılarız*. Bunu da bazı özelliklerini ortaya koyarak yaparız. Algılamak içinde illaki o nesneyi görmemiz, duymamız veya bir şekilde ona dokunmamız da gerekmekz.

Bir şekilde nesnelerin özellikleri çok iyi bir şekilde ortaya konulabilirse -yani tarif edilebilirse- karşı taraf onu rahatlıkla algılayabilecektir. İşte bu algılama da o nesneyi oluşturmuş olacaktır. (Aslında bende felsefeye bu kadar girilmemesi gerektiğini düşünenlerdenim, maazzallah felsefeye girip de çıkamayan çok kişi olduğunu da biliyoruz☺)

-Onun için- kısaca şunu demek istiyorum; örneğin şimdi ben, şu an yanımda duran ve sizin hiç görmediğiniz bir nesneyi, size çok iyi bir şekilde tarif edebilsem (yani özelliklerini çok iyi bir şekilde size aktarabilsem, şu renktedir, şu malzemeden yapılmıştır gibi bunun yanında eni-boyu-yüksekliği bilgilerini de size tam olarak verebilirsem eğer) o nesneyi en azından sizin kafanızda canlandırmış olabilirim. Yani bir anlamda o nesnenin özelliklerini ortaya koyarak onu sizin algı dünyانızda oluşturmuş olurum.



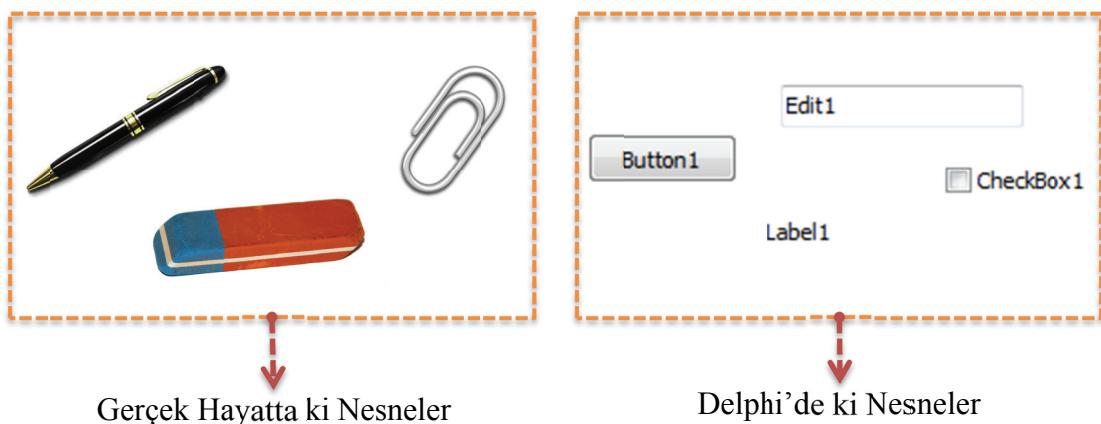
Buradan da anlıyoruz ki bir nesneyi tanımlayan özellikleri bulunmaktadır. Örneğin yukarıdaki gibi bir fareyi,

- ☞ Siyah renkli,
- ☞ İki temel butonu,
- ☞ Bir tekerleği bulunan,
- ☞ ...
- ☞ Genelde sert-parlak bir malzemeden yapılmış olmasına rağmen
- ☞ Başparmağımızın denk olduğu yerde daha mat ve kaydırırmaz bir madde kullanılarak üretilmiş bir nesnedir şeklinde tanımlamak mümkündür.

Bunun yanında bu fare ile neler yapılabilir sorusuna da:

- ☞ Tıklanabilir,
- ☞ Çift Tıklanabilir,
- ☞ Kaydırma yapılabilir,
- ☞ Sağ tıklanabilir... gibi daha birçok olaydan da (iş, oluş, eylemden) oluşan cevaplar verilebilir.

İşte tüm bu özellik ve olaylar bir nesne üzerine atfedilen *tanımlayıcılardır*. Bu tanımlayıcılar sayesinde de nesnelerimiz oluşmaktadır.



Nesne Yönelimli Programlama teknigi de programlamaya tamda bu açıdan yaklaşmaktadır. Uygulamalarda kullanılan modüller (parçalar) birer nesne olarak

tasarlanmakta ve o nesnelerinde özellikleri ve olayları kısaca tanımlayıcıları ortaya konmaktadır. Sonrasında ise oluşturulan bu nesneler birleştirilerek uygulamaları meydana getirmektedirler.

Nesnelerden meydana gelen uygulamalar ise nesnelerin özelliklerinden dolayı daha *müdahale edilebilir* olmaktadır.

...1960'lı yılların sonuna doğru ortaya çıkan bu yaklaşım, o dönemin yazılım dünyasında beliren bir bunalımın sonucudur. Yazılımların karmaşıklığı ve boyutları sürekli artıyor, ancak belli bir nitelik düzeyi korumak için gereken bakımın maliyeti zaman ve çaba olarak daha da hızlı artıyordu. NYP'yi bu soruna karşı bir çözüm haline getiren başlıca özelliği, yazılımda birimselliği (modularity) benimsemesidir. NYP ayrıca, bilgi gizleme (information hiding), veri soyutlama (data abstraction), çok biçimlilik (polymorphism) ve kalıtım (inheritance) gibi yazılımın bakımını ve aynı yazılım üzerinde birden fazla kişinin çalışmasını kolaylaştıran kavramları da yazılım literatürüne kazandırmıştır. Sağladığı bu avantajlardan dolayı, NYP günümüzde geniş çaplı yazılım projelerinde yaygın olarak kullanılmaktadır.

NYP'nin altında yatan birimselliğin ana fikri, her bilgisayar programının (izlence), etkileşim içerisinde olan birimler veya nesneler kümesindenoluştuğu varsayımdır. Bu nesnelerin her biri, kendi içerisinde veri işleyebilir ve diğer nesneler ile çift yönlü veri alışverişinde bulunabilir. Hâlbuki NYP'den önce var olan tek yaklaşım (Yordamsal programlama), programlar sadece bir komut dizisi veya birer işlev (fonksiyon) kümesi olarak görülmekteydi. (Nesne Yönelimli Programlama - Vikipedi)

4.2. Nesne Yönelimli Programlama Tekniğinin Avantajları

! *Nesne Yönelimli Programlama Tekniği* ili *Görsel Programlama* kavramlarını karıştırmamak gerekmektedir. Şöyle söyleyebiliriz: Nesne Yönelimli Programlama teknigi sağladığı avantajlardan dolayı görsel programlama sürecinde yararlanacağımız başta gelen tekniklerden olacaktır.

NYP'nin sağladığı avantajlara da kısaca değinmek gerekirse eğer; yukarıdaki Wikipedi'adan yaptığımız alıntıda bahsedilen özelliklerine göz atmak gerekmektedir. Zira avantajları bu özelliklerini içerisinde saklı durumdadır.

NYP teknigi ile oluşturulan nesneler temelde üç özelliğe sahiptirler:

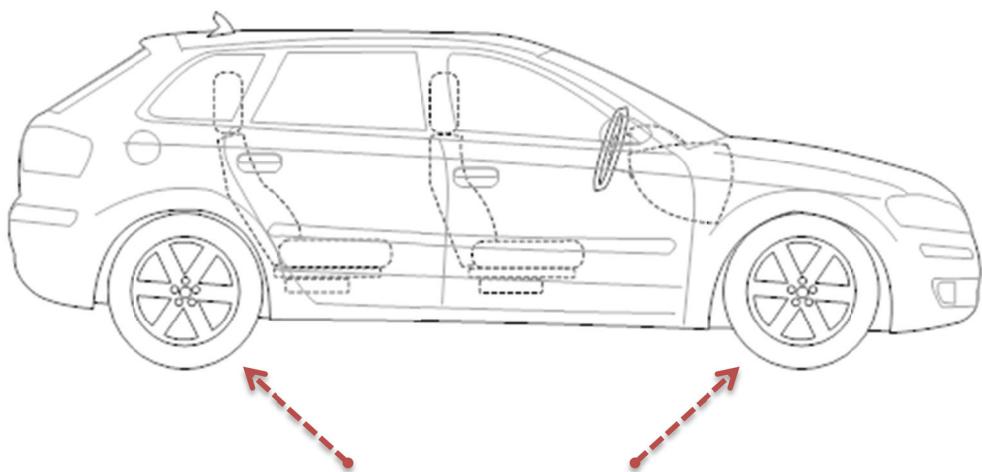
- Bilgi gizleme (information hiding), bazen Veri Soyutlama (data abstraction) da denilebilmektedir,
- Kalıtım (inheritance) ve
- Çok biçimlilik (polymorphism).

İşte tüm bu özelliklerinden dolayı da nesnelerden oluşturulmuş uygulamalar yukarıda da belirttiğimiz gibi daha *müdahale edilebilir* yapılar olarak karşımıza çıkmaktadırlar.

4.2.1. Nesnelerin Bilgi Gizleme Özelliği

Örneğin aşağıdaki gibi irili ufaklı ve belki de binlerce nesneden oluşan bir otomobil sistemini düşündüğümüzde tüm bu nesnelerin bir şekilde belli bir amaç için bir araya

getirildikleri ve aralarında da bir uyumun –ahenkli bir çalışmanın– olduğu aşıkârdır. Zira siz direksiyonu *sola* çevirdiğinizde tekerlekler *sağa* gitmek isteseydi otomobil sistemini oluşturan nesneler arasındaki bir uyumdan söz edilemezdi. Oysaki bildiğimiz gibi günümüz teknolojisiyle donatılmış araçlar bırakın direksiyonu sola çevirdiğinizde tekerleklerin sola yönelmesini, o esnada aracın değişen yeni ağırlık merkezini de hesaba katarak ilgili tekerleklerin fren sistemini kontrol edebilen yapılarla donatılmış durumdadırlar. İşte tüm bunlar otomobili oluşturan nesnelerin aralarındaki uyumlu çalışmadan kaynaklanmaktadır.



Bir otomobil sistemini oluşturan tüm nesnelerin görevleri ve birbirleriyle olan ilişkiler önceden belirlenmiş durumdadır.

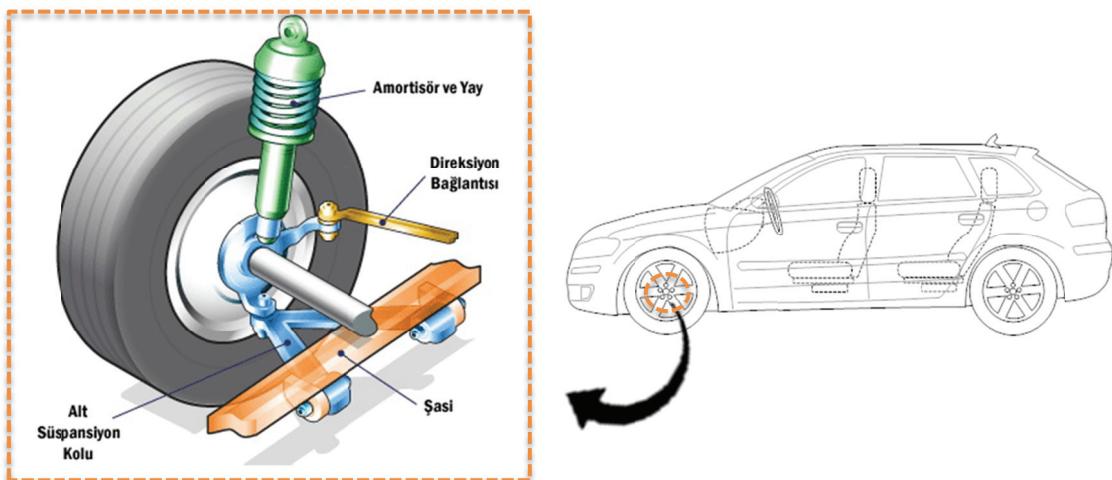
Birbirleriyle bu kadar uyumlu çalışan nesneler ise tek tek düşünüldüğünde kendi içerisinde ayrı bir dünyadır: Örneğin otomobilin lastikleri.

Otomobilin lastikleri de kendi içerisinde belki de yüzlerce nesneden oluşmaktadır ve geliştirilmeye de son derece açıktrırlar. Yol tutuşlarındaki artıstan tutun da ömrlerinin uzunluğuna varıncaya kadar otomobil lastiklerinde bir sürü iyileştirmeye gidilebilir.

• İşte burada ki en önemli sorulardan bir tanesi de bu örneğe göre, lastiklerde yapılan iyileştirmeler tüm sistemi yani otomobili nasıl etkiler?

Şüphesiz olumlu yönde etkiler, zira kim istemez ki yola daha iyi tutunan uzun ömürlü lastikleri demi? Peki, her lastikleri değiştirdiğimizde aracımızı da değiştiriyor muyuz? Değiştirmiyoruz tabii!

İşte Nesne Yönelimli Programlama tekniğinin arkasında yatan mantıklardan biriside budur. Uygulamalarımı meydana getiren nesneler belli bir amaç için bir arada çalışabilirlerken kendi içerisinde de *adeta ayrı bir dünya gibidirler*. İşin iyi tarafı ise tüm sistemin, ilgili nesnenin içeriğiyle ilgilenmesine gerek yoktur. Yeter ki aralarındaki bağlantı noktaları iyi belirlenmiş ve ilgili nesnenin de tüm sistem içerisindeki kendisine düşen görevi layıkıyla yerine getiriyor olsun.



Yine otomobil lastiklerinden yola çıkacak olursak yukarıdaki şekilde de gösterildiği gibi lastiklerin otomobilin şasisine, amortisörlerle ve direksiyona bağlantısı önceden ayarlanmıştır. Hatta öyle ki biz son kullanıcılar olarak, bu bağlantıları **bilmek** zorunda bile değilizdir. Bizi ilgilendiren kısım daha çok, lastiği araca tutturacak olan somunların sıkılmasıdır. Aslına bakarsanız bağlantılar tamam olduktan sonra, **tüm sistemin** ilgili nesnenin içeriğiyle ilgili bir fikri yoktur.

Buradan şu noktaya gelmek istiyorum: Aynen yukarıda verdiğimiz örnekteki gibi NYP tekniğinde de uygularımız birden fazla nesneden oluşacak ve bu nesneler kendi aralarında bir şekilde haberleşebiliyor olacaktır. Yine de tüm sistemin ilgili nesnenin içeriğiyle ilgili *doğrudan bir müdaħale* *ise söz konusu değildir*. Çünkü dediğimiz gibi “*her bir nesne kendi içinde adeta ayrı bir dünya gibi çalışmaktadır*” ve dolayısıyla nesnenin kendi içindeki bu çalışmadan tüm sistem doğrudan haberdar değildir. İşte bu da nesnelerin *Bilgi Gizleme* veya *Veri Soyutlama* özelliklerine karşılık gelmektedir.



Daha da basit düşünecek olursak, yine nesnelerin bilgi gizleme özellikleriyle ilgili olarak çay-kahve otomatları örnek olarak gösterilebilir. Bu tür cihazlarda biz son kullanıcılar olarak makinenin nasıl çalıştığını ilgili olarak çok da fikir sahibi değilizdir. *Nasıl çalıştığını ilgili olarak* derken bize hitap eden *arayüzü* değil de *makinenin iç çalışmasını* kastediyorum. Bizi ilgilendiren kısmı ise elimizdeki bozuk paralarla çayımızı nasıl alabileceğimizdir. Bunun içinde genellikle elimizdeki bozuk paralardan yeterli bir kısmını bize gösterilen bir hizneden makinenin içine atarız. İstediğimiz içeriği de belirledikten sonra belki şeker miktarını falan girerek makineden istediğimizi alabiliriz. Tabii bu arada şansımız varsa para üstünü de alabiliriz belki😊 Fakat bu süreçte makinenin içinde olan biten konusunda ise bihaberizdir (habersizizdir).

İşte bu örneğimizde de varmak istediğimiz nokta, daha doğrusu nesnelerle ilgili olarak altını çizmek istediğimiz özellik nesnelerin bilgi gizleyebilme yetenekleridir.

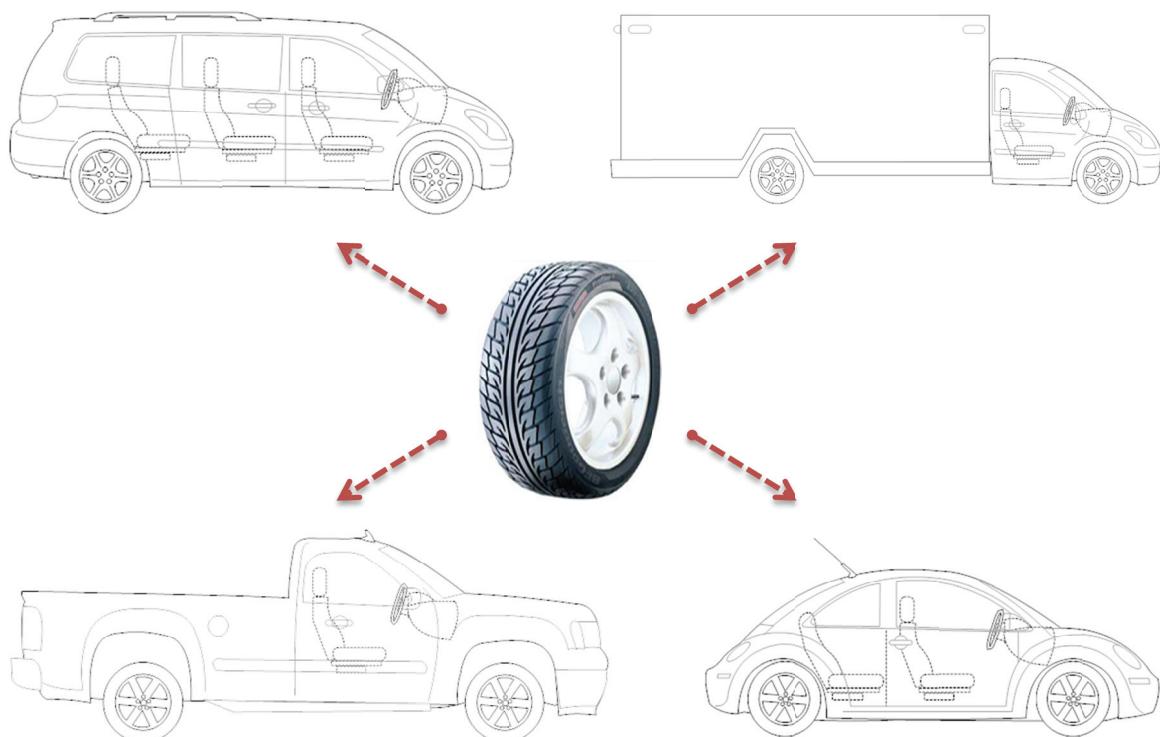
! Nesneler kendi içlerindeki çalışmayla ilgili olan verileri genellikle dış dünya ile paylaşmadan kendi içlerinde işlerler. Dış dünyadan kasıt ise bazen son kullanıcı olabilirken bazen de bağlı bulunduğu sistemin ta kendisidir. Ancak yukarıda ki çay-kahve otomatları örneğinde olduğu gibi dış dünyaya açılan da bir arayüzleri vardır. Otomobil lastiklerinin de bir şekilde otomobile bağlı olmaları gibi.

Bir yandan da nesnelerin bu özellikleri tüm sistemi müdahale edilebilir kılmaktadır. Yine yukarıdaki örneklerimizde olduğu gibi otomobilleri geliştirmek için otomobilleri oluşturan nesnelerde iyileştirmelere gidilebilmektedir. Düşünün, otomobilinizin lastiklerini daha iyi yol tutabilir hale getirdiğinizde tüm sistem bundan nasıl etkilenir veya çay-kahve makinesinin -tamam kendi içinde nasıl çalıştığını bilmiyoruz fakat bu aleti yapan kişiler- aleti arayüzüne değiştirmeseler bile bizim isteklerimize daha hızlı tepki verebilen bir hale getirdiklerinde buna kim itiraz eder ki demi😊

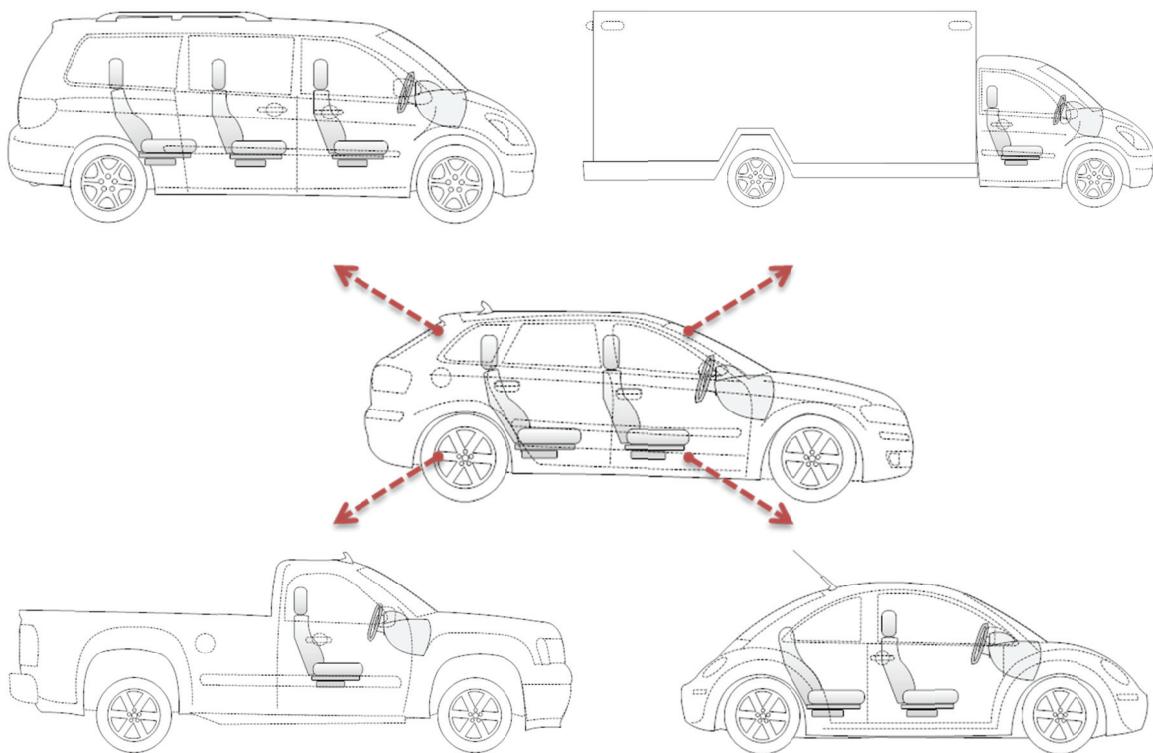
İşte Delphi ile NYP tekniğini kullanarak geliştirdiğimiz uygulamalarda da durum böyledir. Tüm uygulamayı meydana getiren nesneler de zamanla kendi içlerinde geliştirilme yoluna gidilecektir. Hatta nesnelerin yazarları (programcılar) farklı olsalar bile. Kimse

birbirinin nesnesinin çalışması hakkında hiçbir fikir sahibi olmasa bile herkes kendi geliştirdiği nesneyi güncellediğinde tüm uygulama bundan olumlu etkilenecektir ve bu da daha müdahale edilebilir bir yapı demektir.

Kaldı ki oluşturduğumuz ve kendi içlerinde ayrı bir dünya olan bu nesneleri sadece tek bir uygulamada da kullanmak zorunda değiliz.



Yukarıdaki şekilde de anlatılmamaya çalışıldığı gibi oluşturduğumuz bir lastik nesnesini küçük rötuşlarla kolaylıkla diğer uygulamalara da taşıyabiliriz.



Keza koltuklar, direksiyon sistemi ve hava yastıkları gibi pek çok sistem küçük rötuşlarla diğer uygulamalarda da kullanılabilir olmaktadır. Tüm sistemi oluşturan programcının artık koltuğun kumaşının terletip terletmeyeceğine kafa yorması gerekmez. O koltuğu programlayan programcının düşünmesi gereken bir konudur bu. Tüm sistemi programlayan programcı daha çok koltuğun bağlantı noktaları ve sisteme entegrasyonu üzerinde kafa yormaktadır.

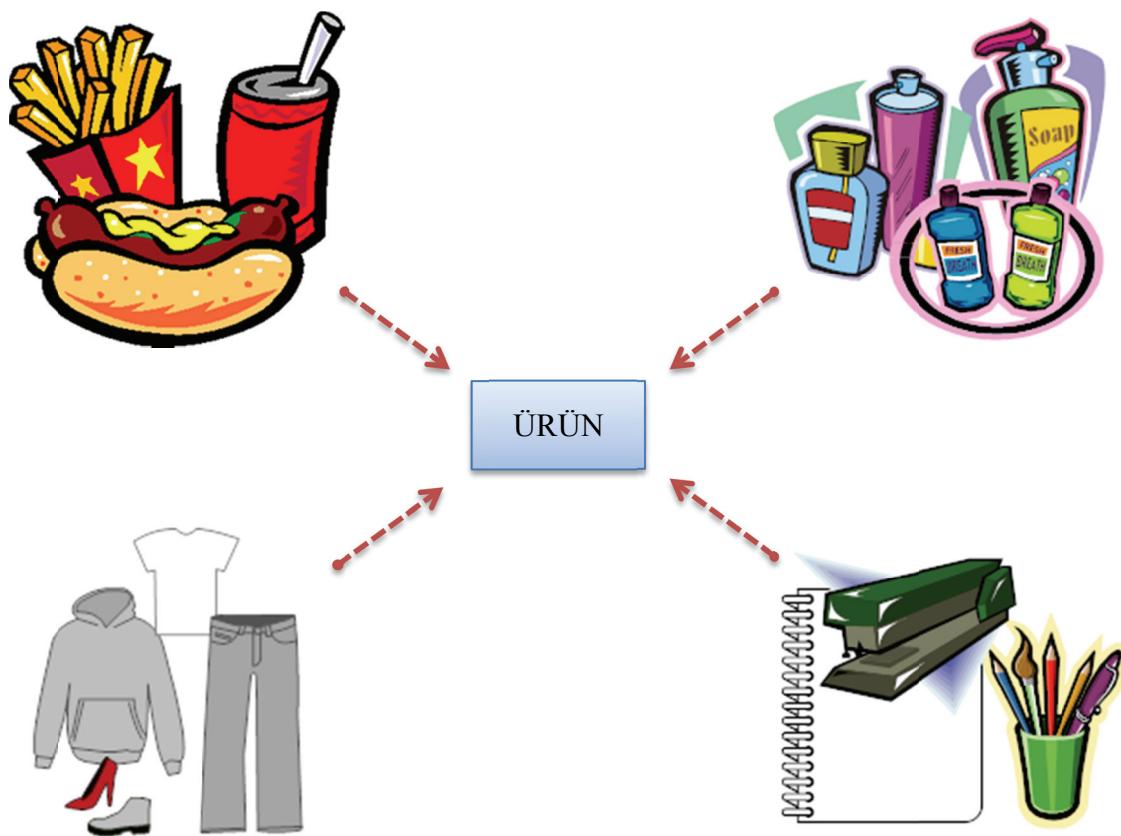
4.2.2. Nesnelerin Kalıtım Özelliği

NYP tekniğinin diğer önemli bir avantajı da nesnelerin sahip oldukları kalıtım özelliğinden ileri gelmektedir.

Nesneler, sahip oldukları bu kalıtım özellikleri sayesinde temel bazı özelliklerini ebeveynlerinden kalıtım yoluyla devralabilmektedirler.

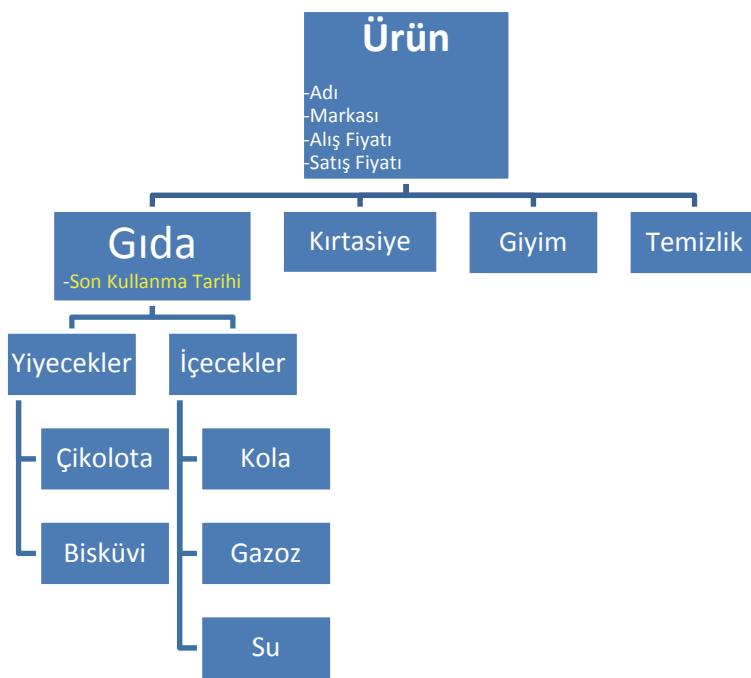
Bu sayede de uygulamalar daha kolay oluşturulabilmekte, düzeltilebilmekte ve daha kolay da genişletilebilmektedir. Tüm bunlar da zaten yine sistemlerin müdahale edilebilir olmalarına hizmet eden özelliklerdir.

Örneğin bir *süper market* için geliştireceğiz *stok takip uygulamasında* NYP tekniğini tercih etmeniz durumunda süper marketteki *her bir ürünü bir nesne olarak oluşturmak* iyi bir çözüm olacaktır.



Bir süre sonra ise fark edeceksiniz ki oluşturduğunuz nesnelerin bir sürü ortak noktası bulunmaktadır. Örneğin bir kırtasiye ürünü olan dolma kalemini düşünün: bir adı, markası, alış fiyatı, satış fiyatı gibi özellikleri bulunmaktadır. Benzer özellikler ele alacağınız bir çikolata da karşımıza çıkabilmektedir. Yalnız dikkat edilecek olursa çikolatanın gıda ürünü olmasından kaynaklı olarak bir *son kullanma tarihi* de olmak zorundadır.

İşte yine de ortak özellikleri nesnelere tekrar kodlamak yerine bunları belli bir hiyerarşiyi takip edecek şekilde ebeveynlerinden devralabilmeleri sağlanabilmektedir.



Marketteki ürünlerimizi ilk etapta yukarıdaki gibi hiyerarşik bir düzene sokmak mümkündür. Tabii bu düzen çok daha iyileştirilebilir durumdadır. Simdilik konumuzun mantığını açıklamaya yetecek kadarıyla yetinelim.

Bu hiyerarşik düzende dikkat edecek olursanız en tepeye *ürün* adlı nesnemiz oturmuştur. Bu nesnemizin de kendine özgü özellikleri bulunmaktadır. Aslına bakarsanız *ürün nesnesinin özellikleri çok da kendine özgü değildir*. Hatta hiç değildir! (Tamda bu noktaya dikkat çekmek için böyle bir geri dönüş manevrasında bulundum;))

Zira ürün nesnesi hiyerarşik düzenin en tepesinde bulunduğuundan dolayı taşıdığı bütün özellikler kendisinden türeyen tüm nesnelerde de bulunmaktadır. Örneğin bu tabloda biraz daha detaylandırdığımız gıda ürünlerinden çikolatayı ele alırsak, yukarıya doğru çıktığımızda nihayetinde çikolatanın da bir ürün olduğunu söylemek yanlış olmaz. Bu durumda *çikolata nesnesinin siz kodlamamış* bile olsanız, bir;

- ☞ Adı,
- ☞ Markası,
- ☞ Alış Fiyatı ve
- ☞ Satış Fiyatı

Özellikleri otomatikman olmaktadır. Çünkü o sonuçta bir ürünüdür. Bir gıda ürünü olmasından kaynaklı olarak da bir *son kullanma tarihine* sahiptir.

? Şimdi düşünün: bu örnekte *son kullanma tarihi* özelliğini *gıda ürünlerinden* alıp en tepedeki *ürün nesnesine* vermiş olsaydık neler olurdu?

Cevap çok basit, tüm ürünlerin birer son kullanma tarihi olmuş olurdu. Bu durumda giyim ürünleri de sonuçta bir ürün olduğu için onlarında birer son kullanma tarihleri olmuş olacaktır. Bu kula yapısıdır, elbette bir son kullanma tarihi olmalıdır demeyin sakın, zira

anlatmaya çalıştığımız sadece gıda ürünlerinde bulunması gereken bozulmadan kalabilecekleri maksimum süreyi gösterir tarihtir.

Evet, NYP tekniğinde görüldüğü üzere nesnelerimizin belli bir hiyerarşik düzene göre oluşturulmaları da mümkündür. Bu düzen içerisinde ki herhangi bir nesne özelliklerinin bazılarını üstündeki nesnelerden kalıtım yoluyla devralabilmektedir.

! Bu hiyerarşik dönemin oluşturulmasında ki önemli kural ise şudur:

-dir, -dır, -dur, -dür... kuralı ☺ İngilizcesi daha kolay: “is a...” kuralı. Eğer ki herhangi bir nesne için şudur-budur şeklinde bir genellemeye yapabiliyorsanız genellemeye yaptığınız nesneyi hiyerarşik düzende bir üst basamağa yerlestirebilirsiniz demektir.

Örneğin çikolata, *tüm çikolatalar sonuca bir yiyecektir* ifadesini söyleyebiliyorsanız, çikolataları *yiyecek* grubunun altına yerlestirebiliriz demektir. Aynı şekilde *tüm yiyeceklerde bir gıda üriiniür* ve *tüm gıda ürünleri de marketteki herhangi bir ürünüür* şeklinde genellemeler yapılmaktadır. Bu genellemeler yardımıyla da hiyerarşik düzen oluşturulabilmektedir.

İşte bu hiyerarşik düzen çerçevesinde de herhangi bir nesne özelliklerinin bazılarını bir üst basamağından devralmaktadır. Bu da NYP tekniğinde *nesnelerin kalıtım özellikleri* olarak bilinmektedir.

Nesnelerin kalıtım özelliği nesneleri oluşturmamızda kolaylık sağlama, kodlamayı kısaltması gibi avantajları yanında bir de uygulamaların kolaylıkla genişletilebilmelerine olanak sağlamaktadır.

Düşünün bir kere, herhangi bir süper market için bir uygulama geliştirmiyorsunuz veya geliştirdiniz. Yukarıdaki gibi de bir hiyerarşik düzen içerisinde nesnelerinizi birbirlerinden tureterek kodlama yoluna gittiniz ve sistem tikır tikır işliyor. Sonra bir gün marketin patronu barkod sistemine geçmeye karar veriyor (bu arada da marketin patronu bizim müşteriniz olmaktadır, müşteri de veli-nimetimiz, ne derse yapmak durumundayız demi☺).

Bu sisteme göre de marketteki her bir ürünün barkod numarasının olması gerekmektedir. Haydaa!...-amiyane bir tabirle- *buyur burdan yak!* derler ya, tam öylesine bir durum demi ☺ Fakat, tabi n'apıyoruz, öncelikle panik yapmıyoruz!

Eklemek istediğimiz bu yeni özelliğin hangi tür nesnelerde bulunması gerektiğini biliyor muyuz: biliyoruz, marketteki tüm ürünlerde barkod numarasının bulunması gerekmektedir. Bizim nesnelerimiz temel özelliklerini bir üst basamağından devralarak geliyordu, bu durumda olması gereken en temel özelliklerini kimden devralır? Tabii ki en tepede ki ürün nesnesinden demi?

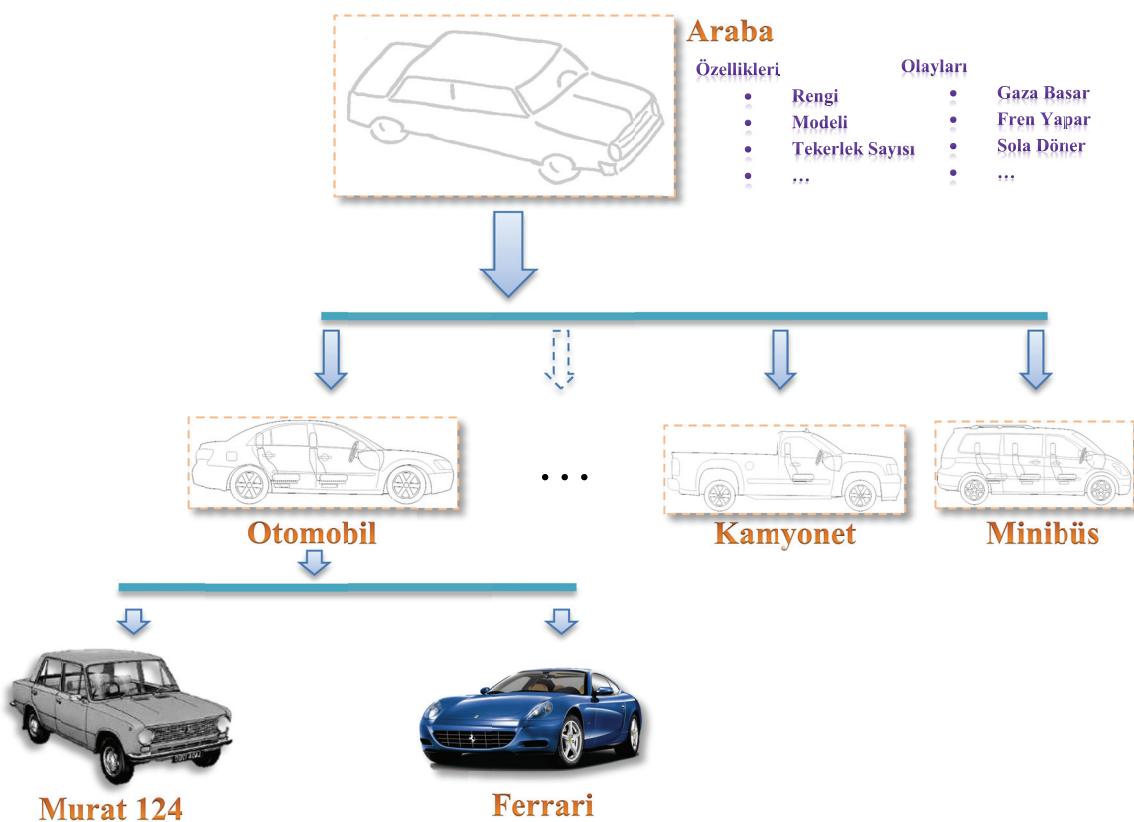
İşte sisteme yapmamız gereken müdahale de tam da bu noktada, *ürün nesnesine bir barkod numarası özelliğini eklemek* şeklinde olacaktır. Diğer tüm nesneler bir şekilde bu nesneden türeyerek üretildikleri için otomatik marketteki tüm nesnelere de bu özelliği eklemiş oluruz. Dolayısıyla çikolatada nihayetinde bir ürün olduğundan, bir barkod numarasına, herhangi bir giyim malzemesi de nihayetinde marketteki bir ürün olduğundan dolayı otomatik olarak bir barkod numarasına sahip olmaktadırlar. Görüldüğü üzere de bu sayede uygulamalarımız daha müdahale edilebilir bir yapıya bürünmektedir.

4.2.3. Nesnelerin Çok Biçimlilik Özellikleri

Yine NYP tekniğinde uygulamalarımızın esnekliğine ve onların hep dediğimiz gibi daha müdahale edilebilir bir yapıya bürünmelerine hizmet edebilecek özelliklerinden bir tanesi de nesnelerin *çok biçimlilik* özellikleridir.

Bu özelliği de kısaca özetlemek gerekirse, nesneler çok biçimlilik özelliklerinden yararlanarak, türlerine göre aynı komutu (aslında metotları) farklı şekillerde yorumlayabilmektedir.

Örneğin kendinize bir araba nesnesi oluşturduğunuzu düşünün ve artık biliyoruz ki nesnelerin kalıtım özellikleri sayesinde bu nesneyi temel alan bir sürü türevlerini –tür olarak çeşitlerini- de üretebiliriz.



Araba nesnesi diğer nesnelere ebeveynlik yapacağı için, diğer tüm türlerin, araba olmak adına olan ortak özelliklerini üzerinde barındırmalıdır.

Örneğin bir arabanın nesi vardır, yani ne gibi özellikleri bulunur:

- ☞ Rengi,
- ☞ Modeli,
- ☞ Tekerlek Sayısı,
- ☞ ...

gibi şu anda hepsini sayamayacağımız daha bir sürü özellik. Yalnız dikkat edin bu özellikler tüm arabalarda olur. Fakat *kasa hacminden* bahseden bir özellik olsaydı, onu araba

nesnesine yerleştirmemiz doğru olmazdı. Çünkü tüm arabaların kasası bulunmamaktadır. Belki sadece kamyonetlerin kasası olabilir.

Diger taraftan bir araba ile neler yapılır:

- ☞ Gaza basılır,
- ☞ Frene basılır,
- ☞ Sola dönülür,
- ☞ Sağa dönülür... gibi daha pek çok eylem yapılabilir.

Bunlar da genel olarak arabalarla yapılabilecek eylemlerdir. Bu noktada şimdiye kadar öğrenciklerimize göre bu özelliklerin ve olayların ortak özellik ve olaylar olduğundan dolayı alt basamaklarda ki türlerde de tekrar tekrar tanımlanmaması gerektiğini düşünebilirsiniz.

! Yalnız bazen nesnelerin öyle özellikleri veya olayları olur ki ilgili özelliğin veya olayın, *temel türde de bulunması gerekirken tür değişince alt türlerde de yeniden tanımlanması* gerekebilir.

Örneğin yukarıdaki şekillerde de gösterildiği gibi arabaların gaza basması. Tüm arabalarda olması gereken bir özellikle aslında (ozellikten ziyade bir olaydır, fakat bazen özellik kelimesi daha geniş anlamda tanımlayıcı kelimesinin yerine kullanılmaktadır). Yalnız arabalardan türeyen yeni arabalarda da gaza basma işlemi giderek değişmektedir. Zira ikisi de birer otomobil olan Murat 124 ile Ferrari'nin gaza basma işlemi sonunda verdikleri tepkiler takdir edersiniz ki farklı olmaktadır ☺



Murat124.GazaBas;

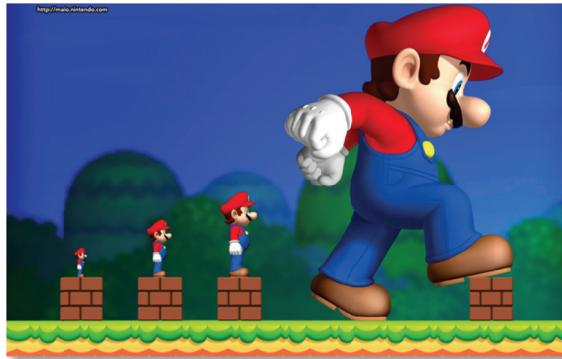


Ferrari.GazaBas;



Bir araba gaza basınca n'apar? Cevap çok basit: gider! demi ☺ Yalnız bizim *Haci Murat'in* gidişiyle *Ferrari'nin* gidişi farklı olacaktır.

Tam bu noktada nesnelerin çok biçimlilik özellikleri devreye girmektedir. Bazen olur ki bir özelliği hem temel nesneye de oluşturmanız gereklidir hem de temel nesneden türeyen yeni nesnelerde oluşturmak zorunda kalırsınız. İşte böyle bir durumda bu nesnelerin çok biçimlilik özelliklerinden dolayı bu işlem mümkün olabilmektedir.



Bir başka örnekte de, Mario benzeri küçük bir oyun programı yazdığınıza düşünelim. Bu oyunda kahramanımız Mario yediği her bir mantarla birlikte giderek büyümektedir. Dolayısıyla da tehlikelere karşı giderek daha dirençli olmaktadır.

Kötü niyetli katil kaplumbağalar ve ekip arkadaşları ise kahramanımız Mario'yu sürekli takip etmekte ve sıkıştırdıkları anda da eğer ki kaçamazsa onu vurmaktadırlar😊

Bu oyunda kahramanımızı temsil eden temel bir nesne oluşturarak Mario'nun özelliklerini bu nesneye yükleyebiliriz. Kahramanımız büyündükçe de bu temel nesneden yeni özelliklere sahip daha gelişmiş nesneler tureterek Mario'nun büyündükçe kazanması gereken özellikleri de bu yeni nesnelere yükleyebiliriz.

Oyunda gerektiğinde, Mario'nun da vurulması gerekiyordu ve vurulma özelliğinin Mario'nun tüm büyülüklerinde olması gerekiyor. Yani Mario büyük de olsa küçük de olsa yeri geldiğinde bir şekilde vurulması gerekiyor.

İşte bu vurulma özelliğini gerçekten Mario'nun tüm büyülüklerinde yeniden kodlamak gerekmektedir. Çünkü Mario'nun değişik büyülüklerinde ki vurulmalar da farklı olacaktır.

Mario'nun en büyük halinin vurulduğunu düşünün, bu onu fazla etkilemezken (belki sadece sendeleyecek veya sadece bir canı eksilecek) Mario'nun en küçük halinin vurulduğunu düşünün, bu da onun aktif futbol hayatının bitmesine ve onun yeşil sahalardan silinmesine yol açacaktır😊

Oyun boyunca Mario'muzu temsil eden ise tek bir değişkenimiz olacaktır. Biz gerektiğinde Mario'muzun vurulması için kodun herhangi bir yerinde;

☞ *eğer (şunlar şunlar olduysa) mario.Vurul*

şeklinde bir kod çalıştıracağız ve gerisini nesnelerin çok biçimlilik özelliği halledecektir. Mario'muzu temsil eden değişkenimize o anda Mario'nun hangi türü yüklendi ise vurulma olayı da Mario'nun o türüne göre gerçekleşecektir.

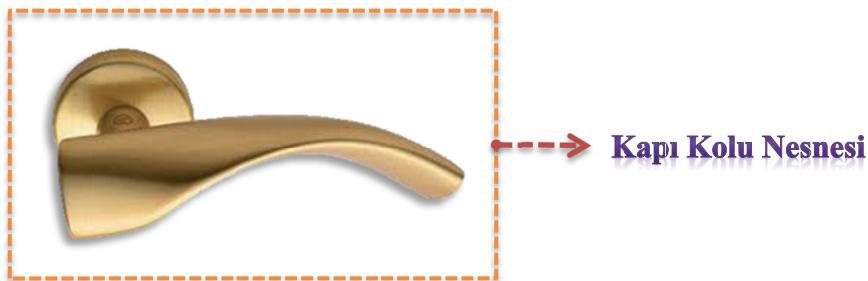
4.3. Sınıf Nedir?

NYP tekniğini ne kadar da kısaltırsak kısaltalım bahsetmeden geçmeyeceğimiz diğer bir yapıda sınıflardır. Zira sınıflarımız olmadan maalesef nesnelerimizde olamamaktadır. Aslında yukarıda ki özellikleri anlatırken çok defa klavyemizin (!) ucuna gelip geri döndüğü

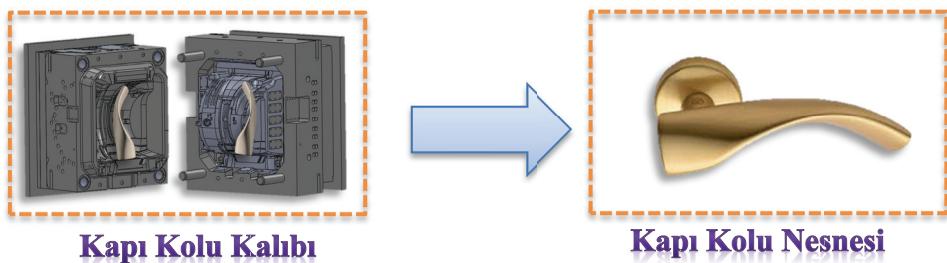
de olmuştur sınıfların. Çünkü dediğimiz gibi herhangi bir nesneyi oluşturmak için öncesinde onun sınıfını (bir anlamda şablonunu) ortaya koymamız gerekmektedir.

Sınıflar nesnelerin hangi özelliklere sahip olmaları gereği, nerde ve nasıl davranışları gereği, hangi parametreler karşısında ne gibi tepkiler vermeleri gereği gibi durumları ortaya koyan yapılardır.

Bir anlamda nesnelerin şablonu gibidirler. Sınıflar gerçekte iş yapmazlar, asıl iş yapan ilgili sınıfından oluşturulan nesnedir. Sınıflar soyut, nesneler ise somuttur.



Örneğin bir kapı kolunu düşünün. Bu kapı kolunun üretilmesi için bir kalıba ihtiyaç vardır demি? Bir üretim tekniğine göre genellikle kalıbin yarısına kapı kolunun bir yarısı diğer yarısına da kapı kolunun diğer yarısı oyularak oluşturulur bu kalıp. Bu sayede kalıbin iki yarısı birleştirildiğinde aralarında kalan boşluk kapı kolunu oluşturacak şekilde olmuş olur. Sonrasında ise bu boşluğa bırakılan küçük bir delikten içeriye eritilmiş metal enjekte edilerek eritilmiş metalin kalıbin şeklini alması sağlanır. Metal soğuduğunda da zaten aynen bir kapı kolu şeklini almış durumdadır. İnanmıyorsanız [adresindeki](#) videoyu seyredebilirsiniz ☺

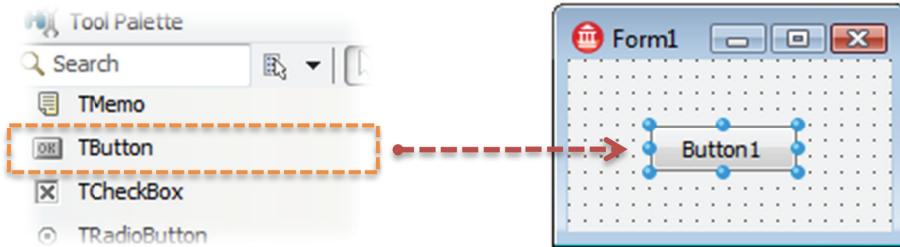


SINIF		NESNE
Soyut		Somut
İş Yapmaz		İş Yapar

İşte bu örneğimizde bahsedilen kalıp bizim sınıfımız, o kalıptan üretilen kapı kolu da nesnemiz olmaktadır. Asıl iş yapan görüldüğü üzere nesnelerimizin kendisidir. Yani herhangi bir kapıya tutup da *kapı kolunun kalibini* takamayız. Ancak o kalıptan üretilmiş olan *kapı kolunun* kendisini takabiliriz. Tabii ki burada kapı kolu kalibinin iş yapmadığı ifadesi kapıları açmak anlamında kullanılmaktadır. Yoksa gereksizdir manasının çıkartılmaması

gerekmektedir. Zira kapı kolunu üretebilmek için de şiddetle onun bir kalıbına ihtiyacımız vardır.

Delphi'de de dikkat ederseniz tüm nesnelerin birer sınıfı yani birer kalıbı bulunmaktadır.

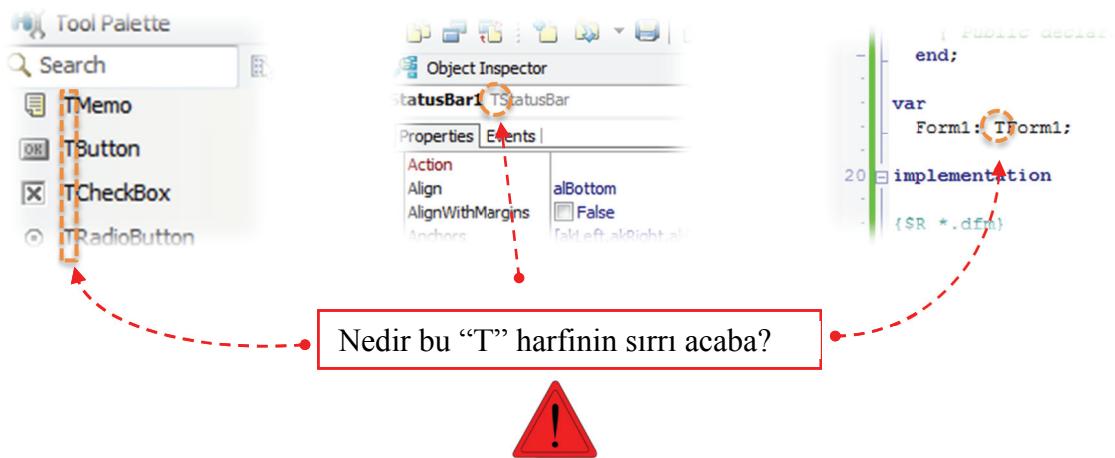


Aslına bakarsanız Bileşen Paleti üzerinde bulunan nesnelerimizin hepsi orda durdukları sürece *sınıf* olarak bulunmaktadırlar. Ne zaman ki biz onları formun üzerine sürüklüyoruz işte o anda, Delphi tarafından otomatik olarak oluşturularak birer *nesne* haline dönüştürülmektedirler. Yalnız ağız alışkanlığından dolayı Bileşen Paleti üzerindeki nesnelere (aslında sınıflara) nesne diyoruz. (Tam doğruluğunu bileyemiyorum ama zamanın birisinde Osmanlı Devletinde bir kanun çıkmış: “*Gâvura gâvur demek yasaktır!*” diye, bu biraz ona benzedi☺).

4.4. Kendin Pişir Kendin Ye! Örneği

Evet, başlık sizi güzel bir bahar gününde ki güzel bir mangal partisine götürmüş olabilir fakat yine de benim amacım Delphi'nin yapımcıları tarafından daha önceden oluşturulmuş olan nesne kalıplarını (yani sınıfları) kullanarak kendi nesnelerimizi kendimizin oluşturması idi (Umarım sizi hayal kırıklığına uğratmamışdım☺).

! Başlamadan önce bir noktaya dikkatinizi çekmek istiyorum. Delphi'de sıkılıkla kelimelerin başlarında bir “T” harfi görmekteyiz.

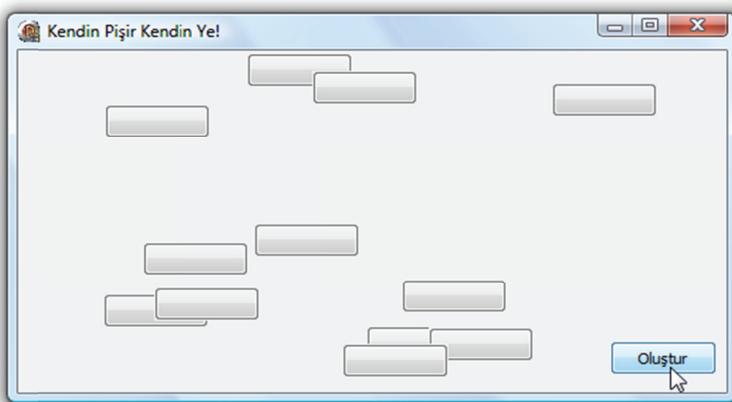


Özellikle de Bileşen Paleti üzerinde ve bileşenlerin önünde bulunmaktadır.

Evet, bu “T” harfi o bileşenin bir sınıf olduğunu göstermektedir. (Yanılmıyorsam eğer Turbo kelimesinin baş harfinden gelmektedir. Turbo Pascal’ı hatırladınız mı? ☺ Turbo Pascal editörü de Delphi’nin ilk yapımcılarından olan Borland firmasına ait bir yazılımdı)

Dolayısıyla da şunu söyleyebiliriz ki başında “T” olan bileşenleri doğrudan kullanmak mümkün değildir. Önce oluşturulmaları (yani Create edilmeleri) gerekmektedir.

Şimdi ki bu örneğimizde de ilgili butona her tıklayışımızda rastgele yerlerde türeyen yeni butoncuklar oluşturmaya çalışacağız.



Butoncuklar dediğimize bakmayın! Zira sonradan (çalışma zamanında) oluşturulmuş olan her bir butonun, normal RAD Studio’da iken Bileşen Paleti üzerinden sürükleyip forma koyduğumuz butonlardan hiçbir eksiği bulunmamaktadır (Caption’larının olmadığına bakmayın, isteseydik onları da yazabilirdik).

Çalışma zamanında oluşturulan nesnelerle, tasarım aşamasında oluşturulan nesneler arasındaki tek fark, oluşturma şekilleridir.

Çalışma zamanında nesneler üretebilmek için ise nesne üretme kodlarını sizin yazmanız gerekmektedir. Bunun içinde aşağıdaki gibi “*oluştur butonuna tıklandığında çalışan bir prosedür*” rastgele bir koordinata, bir buton nesnesi üretmek için yeterli olacaktır.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  bizimButon: TButton;
begin
  bizimButon := TButton.Create(nil);
  bizimButon.Parent := self;
  bizimButon.Left := random(Form1.ClientWidth - bizimButon.Width);
  bizimButon.Top  := random(Form1.ClientHeight - bizimButon.Height);
end;
```

! Çalışma zamanında nesne oluşturma konusuna ilerde daha detaylı olarak değineceğimiz için şimdilik örnekte geçen bilmediğiniz (nil, self, parent... gibi) ifadelere takılmanızı gerek yoktur.

Bu örnekteki amacımız basit bir şekilde, NYP tekniğinden yararlanarak nesnelerin şablonlarından (yani sınıflarından) oluştuğunu göstermekti.