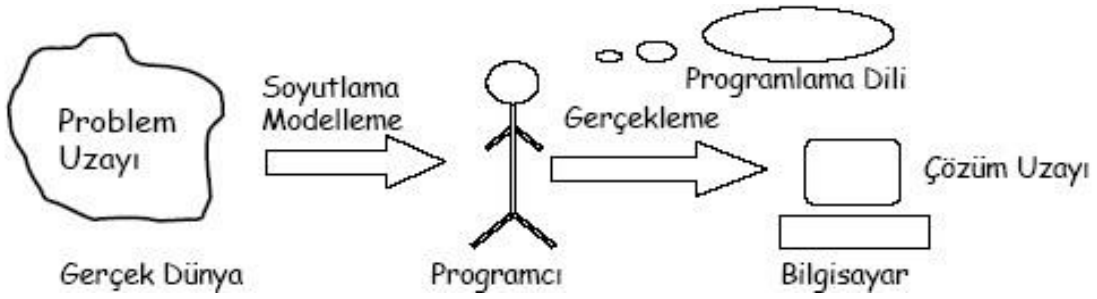
1. Bilgisayar Programlaması

İnsanlar günlük hayatta kullandıkları konuşma dilleri ile çeşitli kavramları birbirlerine anlatmaya çalışırlar. Benzer şekilde bilgisayar programcıları da çözülmesi gereken problemlerle ilgili kavram ve varlıkları, kullandıkları programlama dili ile bilgisayarda ifade etmeye çalışırlar.

Programlama, yaşadığımız gerçek dünyadaki problemlere ilişkin çözümlerin bilgisayarda ifade edilmesidir. Bunu yapabilmek için, kodlamaya geçilmeden önce tasarım aşamasında, problemi oluşturan varlıkların bilgisayarda ifade edilecek şekilde modellerinin oluşturulması gerekmektedir.



Şekil 1. Program Hiyerarşisi

2. Nesneye Dayalı Programlama

Nesneye Dayalı Programlama (OOP: Object Oriented Programming, ileride NDP olarak bahsedilecektir.) altında yatan birimselliğin ana fikri, her bilgisayar programının (izlence), etkileşim içerisinde olan birimler veya nesneler kümesinden oluştuğu varsayımıdır. Bu nesnelerin her biri, kendi içerisinde veri işleyebilir, diğer nesneler ile çift yönlü veri alışverişinde bulunabilir. Hâlbuki NDP'den önce var olan tek yaklaşımda (Yordamsal Programlama), programlar sadece bir komut dizisi veya birer işlev (fonksiyon) kümesi olarak görülmektedirler.

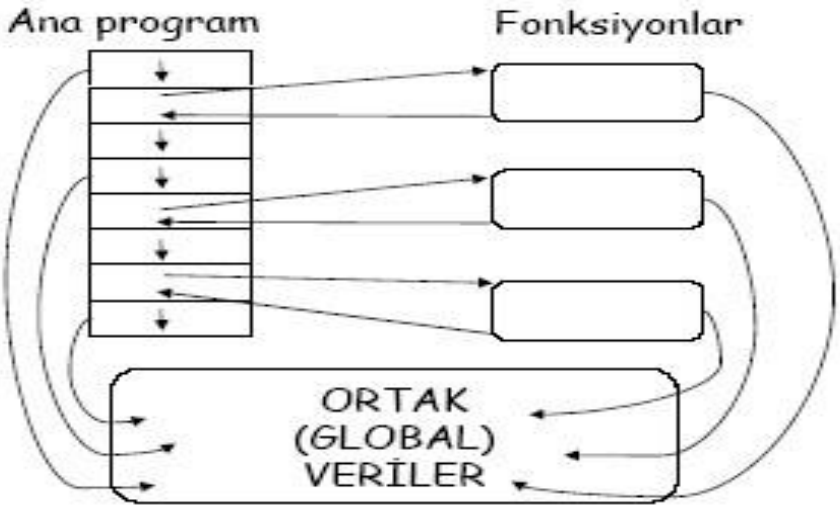
Kendi veri tiplerimizi ve veriyi işleyen yapılar oluşturmamıza izin veren NDP'nin temel prensibi "reuse" olarak adlandırılan kodun tekrar kullanılabilirliğidir. Değişik yerlerde değişik defaca kullanılan bir kod parçası kendisine ait bir hafıza bölgesinde saklanarak her seferinde çağrılabilir ve kullanılabilir.

Nesneye dayalı programlama gerçek dünyayı örnek almakta tıpkı gerçek dünyada olduğu gibi nesneler ve nesnelerin birbirleriyle iletimini sağlayan sistemler kurmaktadır.

2.1. Yordamsal Programlama ve Nesneye Dayalı Programlama Arasındaki Farklar

2.1.1. Yordamsal Programlama

Basic, Fortran, Pascal, C gibi programlama dillerinin desteklediği bu yöntemde öncelikle gerçeklemek istenen sistemin yapması gereken iş belirlenir. Büyük boyutlu ve karmaşık işler, daha küçük ve basit işlevlere (fonksiyonlara) bölünerek gerçeklenir.



Şekil 2. İşlevsel Programlama

İşleve dayalı programlama "Böl ve Yönet" mantığına dayanır. Amaç büyük programları küçük parçalara bölerek yazılım geliştirme işini kolaylaştırmaktır.

Ancak yazılımların karmaşıklıkları sadece boyutlarından kaynaklanmaz. Küçük problemler de karmaşık olabilir. Gerçek dünyadaki sistemler sadece fonksiyonlardan oluşmaz. Sistemin gerçeğe yakın bir modelini bilgisayardan oluşturmak zordur. Tasarım aşamasında verilerin göz ardı edilip fonksiyonlara ağırlık verilmesi hatalar nedeniyle verilerin bozulma olasılığını arttırır. Programcılar kendi veri tiplerini yaratamazlar. Programı güncellemek gerektiğinde, yeni öğeler eklemek ve eski fonksiyonları yeni eklenen unsurlar içinde kullanmak zordur.

İşleve dayalı programlama yöntemini kullanarak kaliteli programlar yazmak mümkündür. Ancak nesneye dayalı programlama yöntemi, kaliteli programların oluşturulması için, programcılara daha çok olanak sağlamaktadır ve yukarıda açıklanan sakıncaları önleyecek mekanizmalara sahiptir.

2.1.2. Nesneye Dayalı Programlama

Gerçek dünya nesnelerden oluşmaktadır. Çözülmek istenen problemi oluşturan nesneler, gerçek dünyadaki yapılarına benzer bir şekilde bilgisayara modellenmelidir.

Nesnelerin yapıları iki bölümden oluşmaktadır:

1. Nitelikler(Özellikler ya da durum bilgileri)

2. Davranışlar(Yetenekler): Tasarım yapılırken sistemin işlevi değil, sistemi oluşturan veriler esas alınır.

Bu nedenle tasarım yapılırken sorulması gereken soru, “bu sistem ne iş yapar?” değil, bu “sistem hangi nesnelerden oluşur?” olmalıdır. Hangi unsurlar nesne olarak modellenebilir:

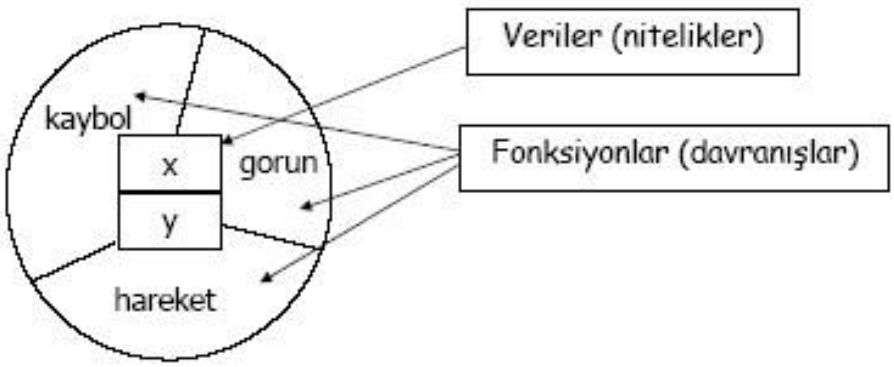
- İnsan kaynakları ile ilgili bir programda; memur, işçi, müdür, genel müdür.
- Grafik programında; nokta, çizgi, çember, silindir.
- Matematiksel işlemler yapan programda; karmaşık sayılar, matris.
- Kullanıcı arayüzü programında; pencere, menü, çerçeve.

Nesne örneği verilebilecek örnek ise; grafik programındaki noktadır. Düzlemdeki bir noktanın özellikleri; xy koordinatlarıdır. Davranışlar ise, noktanın düzlemde yer değiştirmesi, renginin değişmesi, ekranda görünmesi ve ekranda kaybolmasıdır. Buna göre, örnek olarak düşünülen nokta modeli şu bölümlerden oluşacaktır:

- x ve y koordinatları için iki adet tamsayı değişkeni: x ve y.
- Noktanın koordinatlarını değiştirerek düzlem üzerinde yer değiştirmesini sağlayan fonksiyon: hareket,
- Noktanın ekranda görünmesini sağlayan bir fonksiyon: gorun.
- Noktanın ekrandan silinmesini sağlayan bir fonksiyon: kaybol,

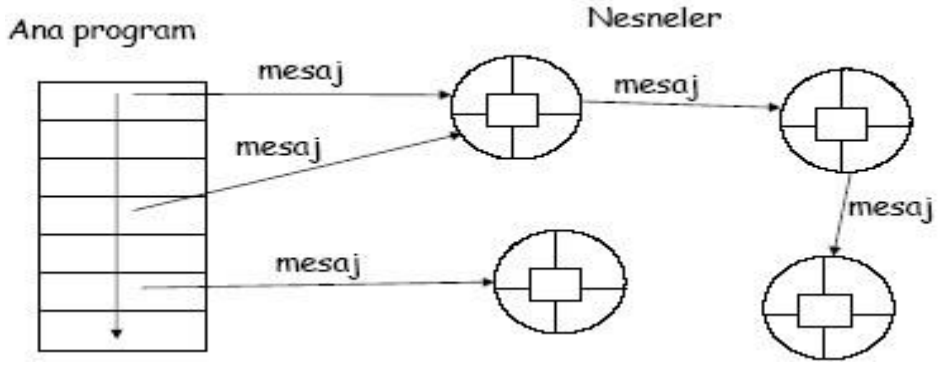
Model bir defa oluşturulduktan sonra, ana programda bu modelden birçok nesne yaratılabilir. Nokta nokta1, nokta2, nokta3, …….. Nokta1.gorun();

Bir Nesne Modeli:



Şekil 3. Program Analizi

Nesneye Dayalı Programın Yapısı:



Şekil 4. Nesneye Dayalı Programın Yapısı

Gerçek dünya nesnelerden oluştuğundan bu yöntem ile sistemin daha gerçekçi bir modeli oluşturulabilir.

Program daha anlaşılır olur.

Nesne modellerinin içindeki veriler sadece üye fonksiyonlarının erişebileceği şekilde düzenlenebilir. Veri saklama (data hiding) adı verilen bu özellik sayesinde verilerin herhangi bir fonksiyon tarafından bozulması önlenir. Programcılar kendi veri tiplerini yaratabilirler. Bir nesne modeli oluşturduktan sonra bu modeli çeşitli şekillerde defalarca kullanmak mümkündür. Nesneye dayalı yöntem takım çalışmaları için uygundur.

2.1.3. Yordamsal Programlama ve Nesneye Dayalı Programlama Arasındaki Farklar

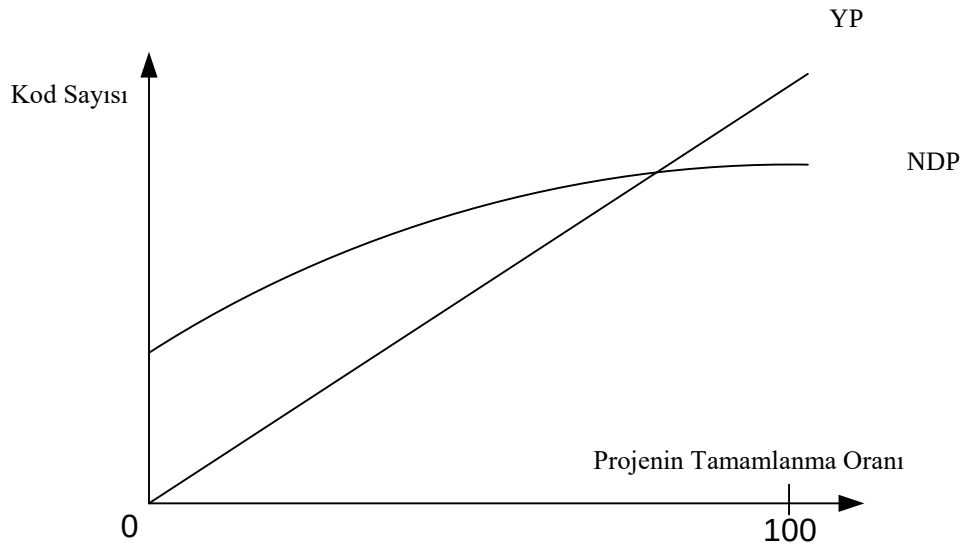
NDP bazı uygulama yeteneklerine sahip veri yığınlarının birbiri ile olan ilişkisine dayanan güçlü bir programlama tekniğidir. Zaman içerisinde ihtiyaçlara bağlı olarak yazılan programların boyutlarının artması, hata arama ve veri kontrollerini zorlaştırmıştır. Yordamsal programlamada verilerimiz ve fonksiyonlarımız (iş yapan kısım) vardır. Yani tüm program veri ve bu veriyi işleyen dokümanlar arasında oluşmaktadır. Kod karmaşıklığını artıran, hata yönetimi zorlaştıran, geliştirme sürecini direk etkileyen de veri ve veri üzerindeki operasyonların bir arada yapılmasıydı. İteratif yöntemlere dayanan

yordamsal programlama olarak adlandırılan bu programlama tekniği zamanla yerini veri ve veri üzerindeki operasyonları birbirinden ayıran, gerçek hayat olgularını gözlemleyip programlama dünyasına aktarabilen başka bir tekniğe yani NDP'ye bırakmıştır.

Biyolojideki "tür" ve "sınıf" farkını ortaya koyan yapı aslında NDP'nin de temelini oluşturmaktadır. Yordamsal programlama tür kavramıyla ilgilenirken, NDP sınıf kavramıyla ilgilenmektedir. Dolaysıyla yordamsal programlama her bir varlığı ayrı ayrı tüm özelliklerine kadar tanımlamak zorundadır. Oysaki NDP'de durum daha basittir. Öncelikle varlıkların ortak özelliklerini tanımla, gerekli olduğu kadar bu tanımdan yeni türler oluştur. Bu mantık günümüz yazılım dünyasının vazgeçilmez anlayışı durumundadır.

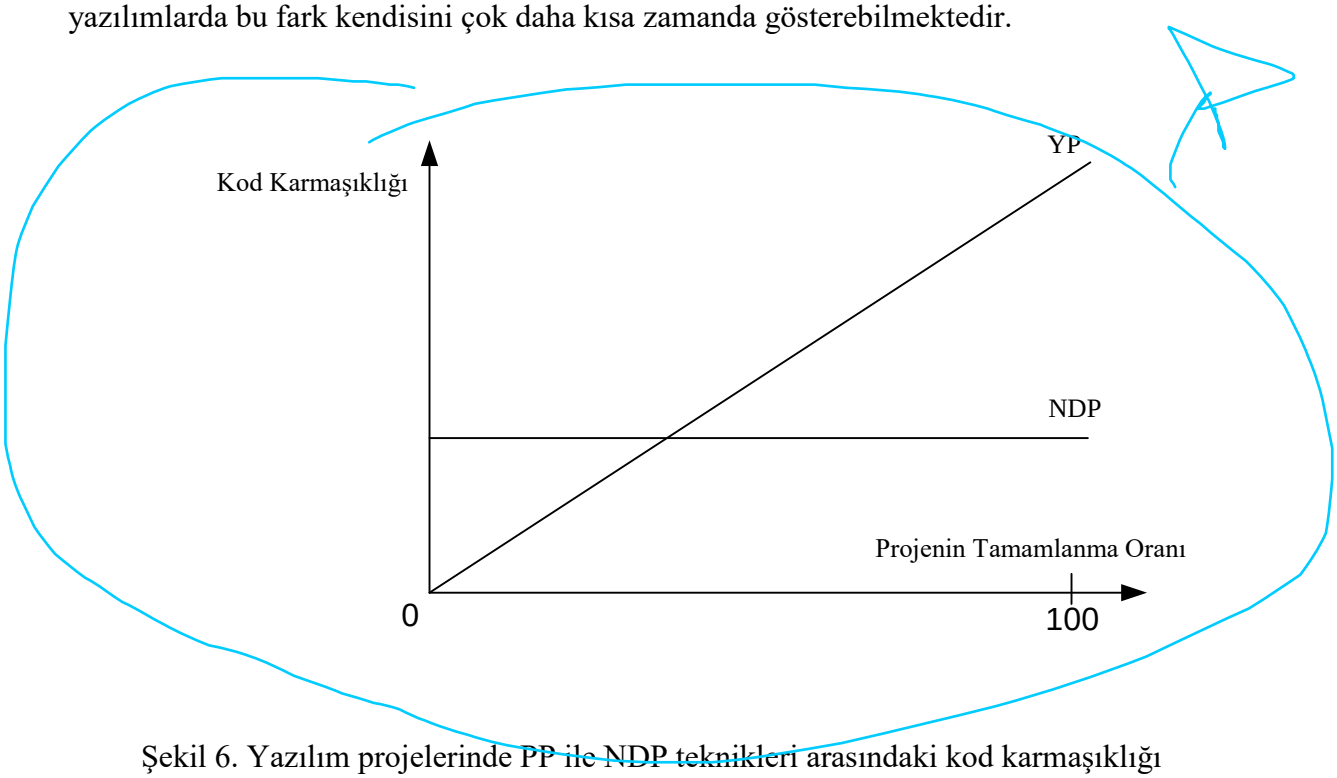
Özellikle veri ve veriler üzerindeki operasyonların birbirinden ayrılmış olması, tek bir veri (sınıf) ile temsil edilebilmesi harika bir fikir gibi gözükmesine rağmen, bazı dezavantajları da beraberinde getirmiştir.

Bunlardan en göze çarpanı ise yazılım geliştirme sürecine hemen başlanamaması gösterilebilir. Öncelikle gerekli olan alt yapılar(sınıflar) oluşturulmalı, bunlar arasında ki ilişkiler gösterilmeli, daha sonra ise geliştirme süreci başlayabilecektir. Şekil 2'i inceleyecek olursak;



Şekil 5. Proje tamamlanma oranları

NDP de belirsiz bir sayıda koda ulaşılırken hala projede herhangi bir tamamlanma oranı oluşmamaktadır. Ancak projelerin ilerleyen kısımlarında bu dezavantaj ortadan kalkmakta kod sayısındaki artış oranı oldukça azalmaktadır. Bankacılık, ERP, CRM gibi çok büyük sayılan, sürekli veritabanlarına verilerin işlenip gelen sonuçların değerlendirildiği yazılımlarda bu fark kendisini çok daha kısa zamanda gösterebilmektedir.



Şekil 6. Yazılım projelerinde PP ile NDP teknikleri arasındaki kod karmaşıklığı karşılaştırılması

Benzeri şekilde kod karmaşıklığı NDP de sabit ilerlerken YP'de bu oran projenin tamamlanmasına kadar artmaktadır. Tüm bunlarla beraber bakım maliyetleri (zaman ve emek) açısından da incelersek NDP nin getirmiş olduğu avantajlar başta belirtmiş olduğumuz dejavantajı ortadan kaldırmaktadır.

2.2. Sınıf Kavramı

2.2.1. Sınıflar

Yazacağımız programlarda sadece objeler vardır. Bu objeler, veri alışverişi yaparak, birbirleriyle haberleşirler.

Objeler OOP de sınıf olarak ifade edilen şablonlardan yaratılır. Her sınıf içinde, o sınıftan üretilecek nesnelerin özelikleri, davranışları tespit edilir. Örneğin "öğrenci" isminde bir sınıf yaratabiliriz. Bu sınıftan üretilecek objelerin ortak yani, bir numara, ad, soyad gibi özellikleri olacaktır. Sınıflar, obje üretme yanında aralarında akraba ve is ilişkileri kurmak mümkündür.

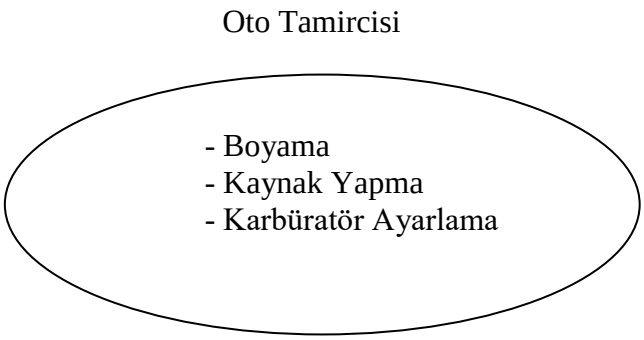
2.2.2. Nesneler

Nesneyi belirlenmiş bir işlevi yerine getiren, bunun içinde çeşitli fonksiyonlar içeren bir yapı olarak tanımlayabiliriz. OOP de objeler sınıflardan üretilir. Objeler, sınıfların aksine canlıdır ve kimlikleri vardır. Aynı sınıftan üretilmiş iki objenin sahip olduğu değişkenler değişik özelliklere sahiptir. Örneğin öğrenci sınıfından üretilen ayşe ve ali isimli iki öğrencinin numaraları değişiktir. Burada adı geçen numarası, sınıf içinde yer alan bir özelliktir. Sınıftan üretilen her obje bu özelliği alır. Objeler üretilirken, obje özellikleri, sahip oldukları yapıya göre, değişik olacaktır. Öğrenci örneğinde olduğu gibi, Ali ve Ayşe objeleri değişik numaralara sahiptir.

Bu yapı değişkenleri de bünyesinde bulundurabilir. Fakat esas, işlevini belli edecek fonksiyonları bünyesinde bulundurmasıdır. Bu özelliğe Sarmalama (Encapsulation) adı verilir. Bunun yani sıra ortaya çıkan bir diğer özellik de, paketlenecek fonksiyonların nasıl bir işlev göstereceği belirtilmeksizin, sadece nasıl kullanacağının belirtilmesidir ki, bu zorunludur. Buna da Soyutlama (Abstraction) adi verilir. Paketleme ve soyutlama bir nesneyi belli etmek için yeterli iki özelliktir.

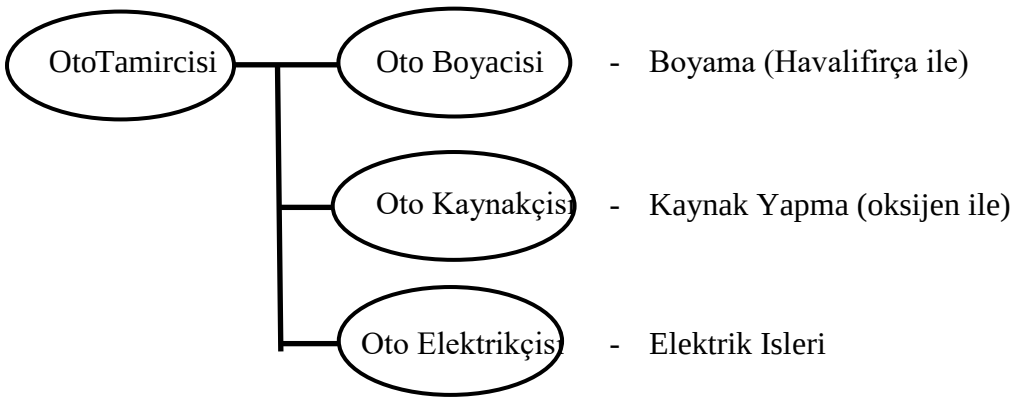
Nesneler için söylenebilecek bir diğer özellik ise türeme özelliğidir. Bir nesne tanımlanırken daha önceden tanımlanmış başka bir nesneyi kendisine taban olarak seçebilir. Bu durum, yeni tanımlanan nesnenin kendisine taban nesnenin özelliklerini kullandırma hakkına karşılık gelir. Ayni zamanda yeni nesnenin taban nesne ile ayni özellikleri taşımasına neden olur. Yeni nesne kendisine yeni özellikler katabileceği gibi devraldığı özellikleri de geliştirebilir. Bu özelliğe Türeme (Derviştin), özellikleri devralmaya da Miras alma veya Kalıtım (Inheritence) adi verilir.

Bu durumu söyle örnekleyebiliriz: Bir oto tamircisi, oto boyama (fırça ile), kaynak yapma (ark kaynağı) ve karbüratör ayarlama konusunda çalışıyor olsun.



Şekil 7. NDP için gerçek hayat örneği

Bu bir oto tamircisi nesnesidir. Bu oto tamircisinin üç oğlu olduğunu ve bunların her birini yetiştirdikten sonra, birer konuda uzmanlaşmalarını sağladığını düşünelim.



Şekil 8. NDP için gerçek hayat örneği (2. Şekil)

Burada Oto Boyacısı, Oto Kaynakçısı ve Oto Elektrikçisi türeyen nesnelerdir. Oto tamircisi ise taban nesnedir. Türeyen nesneler taban nesnenin özelliklerini göstereceklerdir. Oto boyayacaklar, kaynak yapacaklar, karbüratör ayarlayacaklar. Oto boyacısından boya yapmasını istediğimizde havali fırça ile boyama yapacaktır. Oysa Oto tamircisi sadece fırça ile boyamasını bilmekteydi. Oto tamircisi ile Oto boyacısının yaktıkları iş boyama isidir. Oto boyacısı babasından miras olarak aldığı boyama isini geliştirerek havali fırça ile yapmaktadır. Bu Oto kaynakçısı ve Oto elektrikçisi içinde geçerlidir.

Burada üç özellik göze çarpmaktadır:

1- Türeyen nesneler taban nesnenin özelliklerini koruyarak devam ettirip kullanabilirler.

2- Türeyen nesneler türedikleri nesnelerin (taban nesnelerin) özelliklerini değiştirebilir.

3- Türeyen nesneler yeni özellik kazanabilir veya mevcut özellikleri kaybedebilirler.

KAYNAKLAR

- Bt Akademi Kurs Notları http://www.btakademi.com
- Baransel, C., Mumcuoğlu, A., Web Tabanlı, Üç Katmanlı Yazılım Mimarileri: UML, EJB ve ORACLE İle Sistem Modelleme, Tasarım ve Gerçekleştirim, SAS Yayınları, 2003.
- Doç.Dr. Erdoğan DOĞDU ders notları
- Fowler, Martin (2002). Patterns of Enterprise Application Architecture. Addison-Wesley. ISBN 9780321127426.
- Head First Object Oriented Analysis & Design
- http://ceng.gazi.edu.tr/~hkaracan/NYPH1.pdf
- http://e-bergi.com
- http://e-bergi.com/2008/Subat/Nesne-Yonelimli-Programlama
- http://en.wikipedia.org/wiki/Enterprise_Messaging_System
- http://nesneyonelimliprogramlama.blogspot.com/
- http://tr.wikipedia.org
- http://tr.wikipedia.org/wiki/Hizmet-yönelimli_mîmârî
- http://tr.wikipedia.org/wiki/Nesne_Yönelimli_Programlama
- http://www.bilgininadresi.net
- http://www.bilisim-kulubu.com
- http://www.csharpnedir.com
- http://www.dofactory.com/Patterns/Patterns.aspx

- http://www.findikkurdu.com
- http://www.godoro.com
- http://www.msakademik.net
- http://www.onlineakademi.com/egitim-katalogu.html?page=show_ad&adid=258
- http://www.oreillynet.com/onlamp/blog/2006/10/design_patterns_are_signs_of_w.html
- http://www.programlama.com
- http://www.soamoa.org/
- http://www.soaturkiye.com/
- http://www.yazilimdevi.com

- http://wwwipd.ira.uka.de/~tichy/patterns/overview.html
- Object-Oriented Software Development Using Java – Principles, Patterns, and Frameworks,Xiaoping Jia, 2/ed. Addison-Wesley, 2003
- Prof. Dr. Çelebi S., "C++ ve Nesne Yönelimli Programlama, İlave Notlar Sürüm 1.1" Nisan 2005.
- Schmidt, Douglas C.; Michael Stal, Hans Rohnert, Frank Buschmann (2000). Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects. John Wiley & Sons. ISBN 0471-60695-2.
- Schildt, Herbert (2005) "Herkes İçin C#" , Alfa Yayınları
- www.csharpnedir.com
- Y.Doç.Dr. Feza BUZLUCA ders notları