NESNEYE YÖNELİK YAZILIM GELİŞTİRME VE TEMEL KAVRAMLARI

Sibkat KAÇTIOĞLU*
Y. Ziya AYIK**

Özet: Programlama teknikleri, günümüze kadar sürekli bir gelişme göstermiştir. Bunlardan, en yaygın kullanılan ve halen en son geliştirilmiş olanı, nesneye yönelik yaklaşım tekniğidir. "Dördüncü kuşak sonrası", olarak da adlandırılan bu teknik, daha çok nesneyi esas alır ve insan zekasını taklit eder. Nesneye yönelik yaklaşım tekniğinin bu derece ilgi görmesinin en önemli nedeni çok sayıda temel kavrama sahip olmasıdır. Bu kavramlar farklı özelliklere sahip olmalarına rağmen birbirlerini destekler niteliktedirler.

I. Giriş

Günümüzde gelişen bilgisayar teknolojisi, yeni bilgisayar kullanım tekniklerini de beraberinde getirmektedir. Bu tekniklerden bir tanesi ve en güncel olanı, nesneye yönelik yaklaşım tekniğidir. Bu yaklaşımda temel amaç, gerçek hayattaki nesnelerin hareketlerini örnek alarak ve nesneleri özelliklerine göre sınıflandırmak suretiyle kompleks problemlerin çözümünü gerçekleştirmektir (Blair, vd. 1988: 78).

Bu tekniğin uygulanabilmesi için, nesne kökenli programlama dilleri geliştirilmiştir. Bu diller yapıları gereği "obje" oluşumuna müsaade ederler. Obje oluşumunda nesne ve özellikleri, aynı anda tanımlanma imkanına kavuşurlar. Gerçek hayattaki olayları, doğrudan doğruya programdaki objelere yerleştirmek mümkündür. Nesneye yönelik programlama dilleri, nesneler üzerindeki genel ilkelerin, program yazımında kullanımına izin vermekle beraber nesneler arasındaki farkların da belirlenmesini sağlar.

II. Temel Kavramlar

Nesneye yönelim tekniği, kapsamının genişliği ve sağladığı avantajlar nedeniyle bir çok kavrama sahiptir. Nesneye yönelik yaklaşımın üstün olmasının ve tercih edilmesinin en büyük sebebi bu kavramların çokluğu ve özelliklerinin fazlalığıdır (Rumbaugh, 1990: 32). Birbirlerini destekleyen bu kavramların hepsi kendine göre öneme sahiptir.

En temel kavram, yaklaşımın isminde de yer alan nesne kavramıdır. Programlamada oluşturulan mantık da, nesne kavramı dikkate alınarak

*Prof. Dr. Atatürk Üniversitesi İİBF İşletme Bölümü.
** Yrd. Doç. Dr. Atatürk Üniversitesi Erzurum MYO

oluşturulmuştur. Ancak sadece nesne kavramının olması, elbette ki nesneye yönelik yaklaşımın fonksiyonlarının anlatılması için yeterli değildir. İşlemlerin her bir nesne üzerinde teker teker yapılmaya çalışılması, bu yaklaşımın avantajlarını yok eder. Bu nedenle, nesneleri bir araya toplayacak, onlar üzerinde toplu işlemler yapılmasını sağlayacak diğer temel kavramlara da ihtiyaç vardır. İşte bütün bu kavramların tümü, nesneye yönelik yaklaşımın mantığını oluşturmaktadır (Ayık, 1996: 24-50). Bu kesimde, bu yaklaşımla ilgili kavramlar, kavramlar arası ilişkiler ve yaklaşım özellikleri ele alınmaktadır.

A. Nesne (Object)

Gerçek dünyada karşılaştığımız, fonksiyonları olan bütün varlıklar, nesneye yönelik yaklaşım tekniğinde ve buna bağlı olarak geliştirilen nesneye yönelik programlamada nesne olarak kabul edilmektedirler. Nesneler canlı veya cansız durumda, soyut veya somut olabilen, ve en az bir özelliği bulunan varlıklardır. Herhangi bir masa, sandalye, telefon, bilgisayar, elma, nokta veya virgülden her biri, birer nesnedirler. Örneğin bunlardan, elma nesnesi ele alınarak incelenebilir. Bir elmanın yazılım terimleri ile ifade edilmesi mümkündür. Yapılması gereken ilk iş, elmanın özellikleri dikkate alınarak ayrıştırılmasıdır.

K'nın elmanın kabuğunu, S'nin elmanın taşıdığı sıvıyı, M'nin içindeki meyvesini, Ç'nin çekirdeklerini ifade ettiği düşünülebilir ve bu özellikler dikkate alınarak farklı sınıflar oluşturulabilir. Bu sınıfların hiçbirisi artık elma anlamında değildir. Bu sınıflarda, K, S, M ve Ç özelliklerine sahip nesneler vardır.

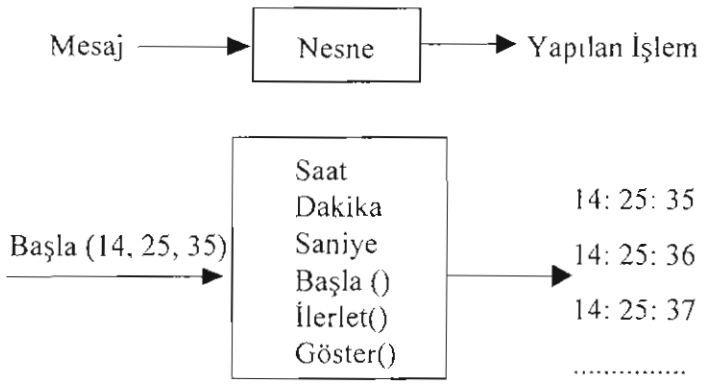
Bir başka şekilde olay, bir ressam ve boyacı gibi de tasavvur edilebilir. Bir elma resminin yapıldığı ve boyandığı düşünülebilir. Elmanın resmi, elma değildir. O, düzlem üzerinde oluşturulan bir semboldür. Ancak bütün haldedir. Sadece kabuğu, sıvısı, meyvesi veya çekirdeği boyanmış olsa, bütün bir elma resminden bahsedilemez. O zaman K, S, M, ve Ç nesnelerinin resminden bahsedilebilir. Elma nesnesini elde edebilmemiz için, yazılım için kodladığımız K, S, M ve Ç sınıflarını birbirleriyle ilişkilendirmesi gerekmektedir. Sınıflar, ortak özelliğe sahip nesneleri bir arada bulundururlar ve birbirleriyle olan ilişkileri sayesinde yeni nesnelerin elde edilmesine imkan sağlarlar.

Nesnenin tanımı, farklı şekillerde yapılabilmektedir:

Nesne, veri ile o veri üzerinde tanımlı bazı işlemlerden oluşan bir bütündür. Örneğin, 1 sayısı kendi başına bir anlam taşımaz. Sadece +, -, /, * gibi kendi üzerinde tanımlı işlemler ile beraber anıldığı zaman, nesne olma niteliğini kazanır (Paepcke, 1993: 45).

En basit anlamıyla bir nesne, mantıksal bir varlıktır. Programlama aracı olan nesne ise, verileri ve bu veriler üzerinde işlem yapmaya yarayan program kodunu içeren bir varlıktır. Programlama açısından nesneler, bir kara kutu

olarak değerlendirilir. Nesnelere mesaj göndermek ve bu mesajlarla yapılan işlemlerin sonuçlarını görmek mümkündür (Hunt, 1986: 183).



Şekil 1. *Nesne ve Kullanılışı*

Nesnelerin nasıl değerlendirildiği ve örnek kullanılışı şekil 1'de gösterilmektedir.

Şekilde, saat, dakika ve saniye nesneleri bir sınıfa ait olarak tanımlanmaktadır. Bu nesnelere, Başla() fonksiyonu ile mesaj gönderilerek başlangıç değerleri verilmekte, İlerlet() fonksiyonu ile mesaj gönderilerek zaman çalıştırılmakta ve sonuçların gösterilmesi Göster() fonksiyonu tarafından gönderilen mesajlar ile sağlanmaktadır. Nesne içinde nelerin olduğu, dışarıdan görülmemektedir. Nesne içinde bir kısım veri veya program kodu, özel (private) kısımda yer alır. Özel kısım, herhangi bir sınıftaki nesne ve fonksiyonlara sadece o sınıfa ait nesne veya fonksiyonlar tarafından erişilebilen ve kullanılabilen kesimdir (Morris, 1994: 21). Bu kesime diğer sınıflardan erişilemez ve bu kesimdeki nesne veya fonksiyonlar diğer sınıflar tarafından kullanılamaz. Ancak istenirse nesneler veya fonksiyonlar genel (public) olarak da tanımlanabilir. Bu durumda diğer sınıflardan da erişim ve kullanım mümkün olacaktır.

Ayrıca nesnelerin sadece kendisinden türeyen nesnelere, kendilerine erişmeleri ve kullanmaları için müsaade etmeleri durumunda korunmuş (protected) kesim oluşur. Yani korunmuş kısımda yer alan elemanlara, herhangi bir nesnenin kalıtım (inheritance) yoluyla türetilen alt nesneleri tarafından erişilebilir (Atkinson, 1991: 79). Program kodu ve verinin, bir nesne içinde bu şekilde birbirine bağlanmasına paketleme (encapsulation) denir. Özel ve korunmuş kısımlarla sağlanan bu korunma, istendiğinde arkadaş (friend) kullanım özelliği ile kaldırılabilir.

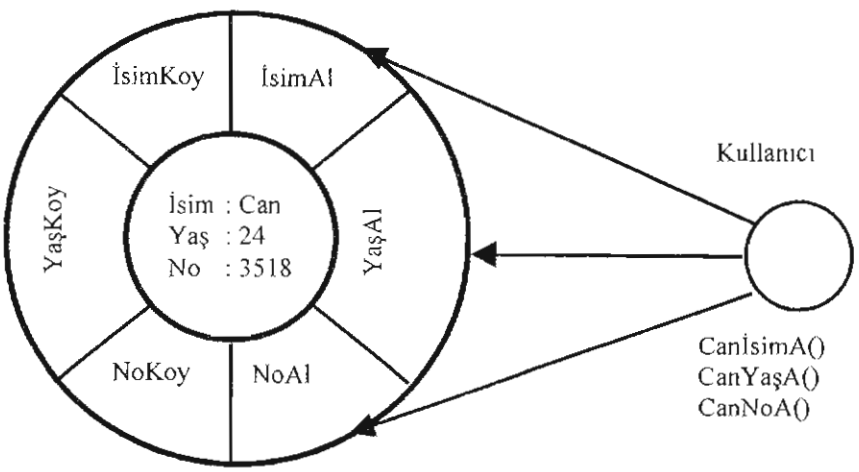
Nesneler kullanıldıkları yerlere göre, dört kategori halinde incelenebilir:

(1) Denetim: Nesneler bir programın normal akışı sırasında, bilgi işlem denetim elemanı olarak görev yapabilirler. Program içinde çalışacak olan sistem, bir nesne ile gösterilir ve denetim bu nesnenin fonksiyonları ile sağlanır.

(2) Uygulama: Bir sistemde belirli amaçların gerçekleştirilmesi için nesneler kullanılabilir. Çeşitli varlıkların bilgisayar ortamında gösterimi nesnelerle yapılabilir. Bu da daha kolay bir soyutlama sağlar.

(3) Veri Yönetimi: Üzerinde işlem yapılacak veriler, nesne haline getirilir. Bu nesneler üzerinde yapılan işlemler metot olarak tanımlanır. Nesneye yönelik veri tabanı yönetim sistemi buna bir örnektir.

(4) Arabirim: Kullanıcı ile uygulama arasındaki arabirim nesnelerle sağlanabilir. Bu da her iki düzey arasında bağımsızlık sağladığından, herhangi birinde yapılacak değişiklik veya gelişim diğerini de etkileyecek problemler çıkarmaz (Sarıdoğan ve Erhan, 1994: 87-88).



Şekil 2. *Üye Fonksiyonlarla Nesne Kullanımı*

Yazılım dünyasındaki nesneler, gerçek dünyadaki varlıklar gibi davranış gösteren program parçaları ve verilerdir. Bir nesnede onu tanımlayan bilgiler, veriler ve bunlar üzerinde işlem yapmaya yarayan metotlar bulunur (Demurlian vd., 1993: 58). Nesnelerin en büyük avantajı yeniden kullanıma açık olmalarıdır. Böylelikle önceden hazırlanmış, test edilmiş ve çalışır halc gelmiş programlar daha sonra çok kısa sürede yeni bir programa uyarlanabilir.

Nesneye yönelik programlamanın. temel birimi nesnedir. Yapısal programlamada yapılar ve işlevler ayrı ayrı ele alınırken, nesneye yönelik programlamada veriler ve işlevler nesne ile birleştirilir (Voss, 1991: 63). Bir nesne, verileri gizleyerek programlama dillerinin ve uygulamaların temel

prensiplerinden olan veri saklama ilkesini destekler. Bu verilere erişim o nesneye ait üye fonksiyonlarla sağlanır.

Şekil 2'de görüldüğü gibi üye fonksiyonlar nesnenin iç yapısını gizlemekte ve bir kabuk oluşturmaktadır. Verilere ulaşmak için arabirim kullanılmalıdır. İsim, Yaş ve No nesnelerine İsimKoy, İsimAl, YaşKoy, YaşAl, NoKoy ve NoAl üye fonksiyonları ile mesajlar gönderilmektedir.

Nesne, belleğin değiştirilebilen değerler veya belirlenmiş fonksiyonlarını yerine getiren isimlendirilmiş bir bölgesidir. Bu yaklaşımda tüm değişkenler birer nesnedir. Nesnelerin davranışlarına göre tasnif edilmeleri de, sınıf (class) kavramını ortaya çıkartır.

Nesneler, değişken ve fonksiyon içeren yapısal değişkenlerdir. Nesnelerin içerdikleri bu değişkenlere üye değişken (member variable), fonksiyonlara da üye fonksiyon (member function) adı verilir (Caruso ve Sciore, 1987: 109).

Bir nesnenin üye değişken ve fonksiyonlarının diğer nesneler tarafından doğrudan kullanılmasının istenmediği durumda, bu üyeler diğer nesnelere karşı korunabilirler.

Bir nesnenin, diğer nesneler veya fonksiyonlar tarafından kullanılabilmesi bazı sınırlamalara bağlıdır:

(1) Bir nesnenin veya fonksiyonun yalnız üyesi bulunduğu nesne içerisinde kullanılması halinde, özel (private) nesne veya fonksiyon tanımı yapılmış olur. Nesne ve fonksiyonlar özel durumdadır.

(2) Bir üyenin içinde bulunduğu nesne haricinde ancak bu nesneden türeyen yeni nesneler tarafından da kullanılabilmesi durumunda korumalı (protected) üye oluşur. Bu durumda, söz konusu nesneden türemeyen nesnelerin ve fonksiyonların erişimleri mümkün değildir.

(3) Bir üyenin, tüm nesneler ve fonksiyonlar tarafından ortaklaşa kullanılabilmesi ise ortak (public) üye kullanımı olarak adlandırılır.

(4) Bir nesnenin başka bir nesneyi arkadaş ilan ederek üyelerinin tümünün bu nesne tarafından kullanılmasına izin vermesi halinde arkadaş (friend) nesne özelliği gerçekleşir.

Bir nesne (object), bilginin özelliğini ve davranışını, yani işlemi tanımlar. Tablo 1'de verilen örnek ile, nesne yapısı ve kullanımı gösterilmektedir. Bu örnekte, Ucak adında bir obje tanımlanmış ve fonksiyonlar vasıtasıyla bu objeye özellikleri tanıtılmıştır.

Tablo 1. *Nesne Yapısı ve Kullanımı*

```
Type
        Ucak = Record
        Hız: Word;      Yükseklik: Word;
Kanatlar:(Yukarı,Asagı);
End;
{Uçağın hareketleri şu prosedürler ile tanımlanabilir:}
```

```
Procedure Hızlanma
        Begin
        End;
Procedure Yavaslama
        Begin
        .....
        End;
Procedure KanatKaldır
        Begin
        .....
        End;
Procedure KanatİndirBegin
        .....
        End;
```

Nesneye yönelik programlamada bilgilerin özelliği ve davranışı tek bir yapıda, yani nesne (object) de toplanır.

```
Type
        Ucak=Object
        Hız:Word;       Yükseklik:Word;
Kanatlar: (Yukarı, Asagı);
        Procedure İlkislemler;          Procedure Hızlanma;
        Procedure Yavaslama;  Procedure Yükselme;
        Procedure Alcalma;              Procedure KanatKaldır;
        Procedure KanatİndirBegin;
End;
```

Bu tanımlamanın ardından, nesnenin kullanımı ise şu şekildedir;

```
Procedure Ucak.İlkislemler;
        Begin
                Kanatlar:=Yukarı:               Hız:= 0; Yükseklik:=0;
        End;
        Var
                N:Ucak;
                With N do
                        Begin
                                İlkislemler;    KanatKaldır;
                                Hızlanma;       Yükselme;
                        End;
```

B. Sınıf (Class)

Aynı özelliği taşıyan nesnelerin, bir arada bulunarak, ortak hareket etmelerini sağlayan nesneler topluluğu, sınıf olarak tanımlanır. Eğer

sınıflandırma özelliği olmasaydı nesneye yönelik yaklaşım, içinden çıkılması mümkün olmayan bir hal alırdı. Etrafımızda bulunan herhangi bir nesneden söz ederken ya da onu birisine tanıtırken, o nesnenin özelliklerinden bahsederiz. Nesnenin şeklinden, ağırlığından, canlı veya cansız oluşundan, boyundan, renginden vs. Nesneye yönelimde, nesne, özellikleri ve davranışlarıyla bir sınıf yapısının içinde yer alır. Her nesnenin mutlaka üye olduğu bir sınıf söz konusudur.

Bir sınıf, esas itibariyle bir veri yapısına (structure) benzer. Veriler üzerinde yöntem (method) tanımlama işlemi tek tek her veri üzerinde değil, benzer yapısal özellikler gösteren nesnelerin oluşturduğu, sınıflar üzerinde tanımlanır. Her nesne, bir sınıfın bireyi olmak zorundadır. Sınıf, aynı yapısal ve davranışsal özellikleri taşıyan bireylerden oluşan bir yapıdır. Sınıflar da aynı zamanda birer nesnedirler. O halde birer nesne olan sınıfların da bir sınıfı vardır. Sınıflar, bir hiyerarşi içinde yapılanmışlardır. En tepede en genel özelliklere sahip sınıf olan Object yer alır. Sınıf hiyerarşisi adı verilen bu hiyerarşi, bir ağaç yapısına sahiptir. Her sınıf sadece tek bir sınıfın alt sınıfıdır (Gilb, 1988: 106).

Nesneye yönelimin tercih edilmesinin en önemli özelliği, sınıflama sistemidir. Bir programı birimler halinde (module) yazmak, belirli bir veri tipine ait tüm bilgilerin, merkezi bir birimin kontrolünde olmasını sağlar.

Bir grafik sisteminde kullanılmak amacıyla resim sınıfını tanımladığımızı ve sistemin sadece daire, üçgen ve kare şekillerini desteklemesi gerektiğini varsayalım. Ayrıca nokta ve renk sınıflarının da daha önce tanımlandığını kabul edelim. Bir resim sınıfı, sınıflama özelliği kullanılarak Tablo 2'de gösterildiği gibi tanımlanabilir ve kullanılabilir:

Tablo 2. *Sınıf Tanımı ve Kullanımı*

```
enum sec {daire, ucgen, kare }
class resim {
                point nokta;
                color renk;
                sec k;
        public:
                point where()    //dönüş merkezi
                void move (point to)
                void rotate (int);
                // diğer işlemler};
```

Bu tanımlamada, resim sınıfına ait where(), move() ve rotate() adlı fonksiyonlar yardımıyla ve k adlı alanın değerine göre hangi şeklin çizileceği tespit edilir.

Bir sınıf private (özel) ve public (genel) kısımlara sahip olabilir. Sınıf içinde tanımlanan her eleman otomatik olarak (default), özel durumdadır. Yani özel bir alan tanımı için, private ifadesinin kullanılması mutlaka gerekli değildir. Özel ve genel alan tanımlamaları Tablo 3'de gösterilmektedir.

Tablo 3. *Sınıflamada Kullanılan Alanların Tanımlanması*

```
Class Kuyruk {int i[100];
                    int eleman;
              public :
                    void bos(void);
                    void doldur(int i);
                    int cıkar(void);
                    int eleman_sayısı();     };
```

Kuyruk adlı sınıf içerisinde tanımlanan i ve eleman isimli değişkenler, private sözcüğü kullanılmadığı halde özel durumdadır. Bu kısımlara, o sınıfa ait olmayan hiçbir fonksiyon tarafından erişilemez. Bu durum, nesneye yönelimin diğer temel bir özelliği olan paketleme (encapsulation) kuralının gereğidir. Bazı veri yapıları özel tanımlanarak, başkaları tarafından erişilmeleri önlenir. Aynı şekilde fonksiyonlar da özel olarak tanımlanabilir. Bu tip fonksiyonlar, sadece o sınıfa ait diğer üye fonksiyonlar tarafından kullanılabilir.

Programın, diğer kısımlar tarafından da erişilebilmelerine müsaade edilen bölümleri, genel (public) olarak tanımlanmalıdır. Sınıf içinde public sözcüğünden sonra tanımlanan her şeye, sınıf dışından erişilebilir. Genel olarak, sınıf üyelerinin public olarak tanımlanması sınırlı tutulmalıdır. Hatta gerekmedikçe kullanılmamalıdır. Nesneye yönelimin ilkeleri gereği bütün veriler özel (private) yapılmalı, bu verilere ulaşımlar ise genel (public) fonksiyonlarla gerçekleştirilmelidir.

Bundan başka, sınıflarda bir de korunmuş (protected) kısımlar vardır. Bu kısım protected sözcüğü ile belirlenir. Korunmuş veri ve fonksiyonlara yalnızca o sınıftan türetilen alt sınıflar doğrudan erişebilir. Diğer sınıflar erişemez.

Tablo 3'de yer alan, bos(), doldur(), cıkar() ve eleman_sayısı() fonksiyonlarına üye fonksiyonlar (member functions) denir. Sınıf içinde tanımlanan fonksiyonlar, o sınıfın özel (private) kısımlarına erişebilirler

Sınıf tanımlanmasının ve kullanılmasının genel şekli, nesneye yönelik programlama dillerinden C++ programlama dilinde, Tablo 4'de gösterildiği gibidir.

Tablo 4. *Sınıf Tanımlamaları*

```
Class SınıfAdı  {
                    //Özel veriler ve fonksiyonlar
                    public:
                    //Genel veriler ve fonksiyonlar
                    protected:
                    //Korunmuş veriler ve fonksiyonlar
                } NesneListesi;
```

Hiçbir tanımlama yapılmadan kullanılan nesneler ve fonksiyonlar özel (private) tanımlanmış kabul edilirler. Public ifadesi ile kullanılan nesne ve fonksiyonlar ise genel (public) tanımlıdır ve bütün nesne ve fonksiyonlar tarafından erişilebilirler. Protected ifadesi ile kullanılan nesne ve fonksiyonlar ise, sadece o sınıf tarafından türetilen nesne ve fonksiyonlar tarafından erişilebilen korunmuş (protected) kısımda bulunan nesne ve fonksiyonlardır.

C. Kalıtım (Inheritance)

Sınıflama, nesneye yönelimde önemli bir özellik olmasına rağmen, nesneye yönelik yaklaşımın fonksiyonlarının gerçekleştirilmesi için yeterli değildir. Sadece sınıflama özelliği ile, nesneye yönelimde istenilen esneklik elde edilemez.

Nesneye yönelimde bir sınıfı, diğer bir sınıfa altsınıf (subclass) olarak tanımlamak mümkündür. Zaten yeni bir sınıf tanımlanırken, daha önce tanımlanmış sınıflardan hangisinin ya da hangilerinin alt sınıfı olacağı da belirtilir. Her sınıf, altsınıfı olduğu sınıflara ait özellikleri kazanır (Halbert ve O'Brien, 1987: 89-91)

Kalıtım (inheritance), sınıflar arasında, birbirinin özelliklerini kazanma ilişkisine verilen addır (Morris 1994: 23).

Genellikle iki şekilde kullanılır. Birincisi, verilerin kapsanması maksadıyla sınıfların özelliklerine göre hiyerarşik bir yapı oluşturulması, ikincisi de, tekrar kullanma imkanını sağlamak amacıyla bir öncekine aynen benzeyen bir sınıf oluşturulmasıdır. Her iki şekilde de bir temel sınıf, özelliklerinin tümünü veya bir kısmını bir başka sınıfa aktarır.

Kalıtım hiyerarşisinin başında bulunan sınıfa temel sınıf, bazen de üst sınıf (base class) denir. Bu sınıfın özelliklerini alarak geliştirilen yeni sınıfa da türetilmiş sınıf veya alt sınıf adı verilir. Bu türetme işlemi, istenildiği kadar yapılabilir. Fakat pratikte üç veya dördü geçmez.

Kalıtım, mevcut bir sınıfın biraz daha geliştirilerek yeni bir sınıf oluşturulması olarak da açıklanabilir. Temel sınıfın, tam veya bir kısım özelliklerinin kullanılacağı, ancak yeni bir sınıfın da gerekli olduğu durumlarda, kalıtım kuralının kullanılması, programcılıkta çok etkili bir yöntem olarak kabul edilmektedir.

Daha önce de belirtildiği gibi, kalıtım iki şekilde ve iki farklı amaçta kullanılır. Bunlardan hiyerarşik yapı oluşturarak gerçekleştirilen kalıtıma, genel kalıtım (public inheritance), tekrar kullauıma imkan sağlamak amacıyla var olan kodun tekrar edilmesiyle oluşturulan kalıtıma ise özel kalıtım (private inheritance) denir.

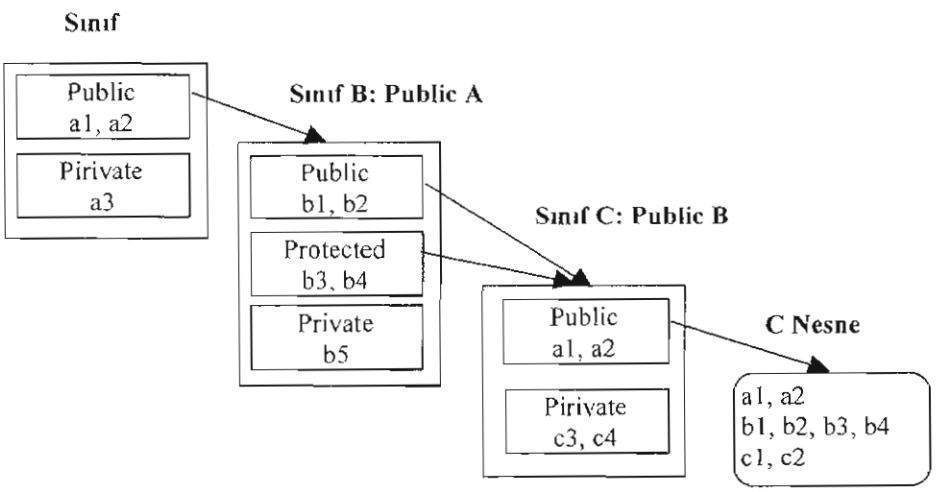
Kalıtım bir başka tasnif şekline göre de, basit kalıtım ve çoklu kalıtım şeklinde, iki kısımda incelenebilir:

(1) Basit Kalıtım: Basit kalıtımda yalnızca bir tek temel sınıf bulunur. Türetilmiş sınıf, temel sınıfın tüm özelliklerini taşır. Basit kalıtım sırasında veri ve üye elemanlara erişim bazı şartlara bağlıdır.

Türetilen sınıf, temel sınıfın özel kısımlarına erişemez. Bu sayede türetilmiş sınıfın, temel sınıfın veri gizleme ve kapsama özelliğine zarar vermesi önlenir.

Temel sınıfın elemanlarına hem türetilmiş sınıf, hem de diğer sınıflar tarafından kolaylıkla erişilebilmesi için, bu elemanların genel (public) kısımlara koyulması gerekir (Voss, 1991: 33). Elemanlar eğer korunmuş (protected) kısma koyulursa, bu sefer sadece türetilmiş sınıflarca erişilebilir. Yani basit kalıtımda, türetilmiş sınıflar, temel sınıfların hem genel, hem de korunmuş kısımlarına erişebilirler. Yalnızca özel kısımlarına erişemezler.

Eğer temel sınıfın özel tanımlı elemanlarına, alt sınıflardan bir kısmının erişmesi isteniyorsa, bu türetilmiş sınıfın, temel sınıfa erişiminde sınıflar arkadaş (friend) olarak tanımlanmalıdır. Friend tanım ile, temel sınıfın özel elemanını korunmuş yapmak aynı manadadır. Tüm alt sınıflar, temel sınıfın korunmuş (protected) kısımlarına erişebilirler.

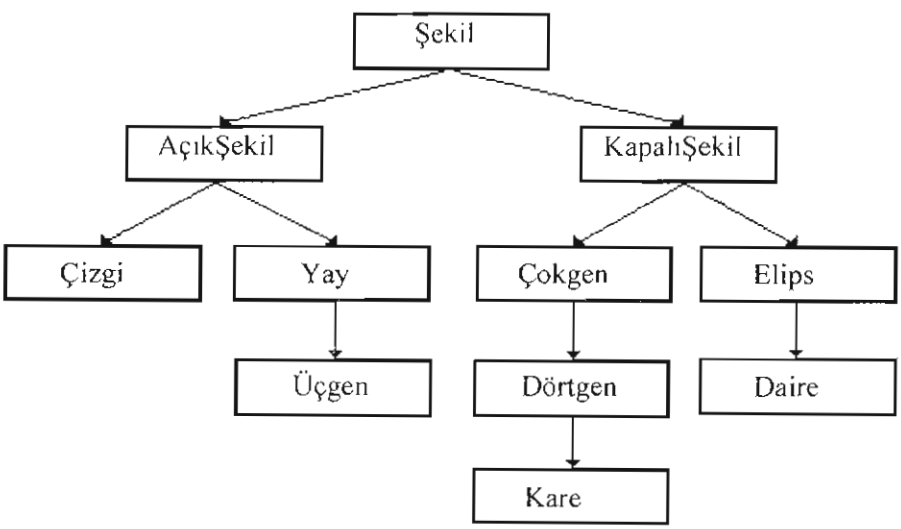


Şekil 3. *Basit Kalıtım Erişim Yöntemi*

Basit kalıtımın şeklinde, temel sınıfın genel kısmında bulunan bütün elemanlarına türetilmiş sınıf tarafından erişilebilmektedir. Basit kalıtımın erişim yöntemi şekil 3'de gösterilmektedir. Burada, a1, a2, b1, b2, b3, b4, c1 ve c2 elemanlarına, nesne tarafından erişilebilmekte, a3, b5, c3 ve c4 elemanlarına ise erişilememektedir.

Basit kalıtımda türetilmiş sınıfın elemanları, temel sınıfının genel (public) kısmında bulunan elemanlarına erişebilmekte, özel kısmında bulunan elemanlarına ise, özel kısmın tanımlı olduğu sınıf adı belirtilerek erişilebilmektedir.

(2) Çoklu Kalıtım: Tek bir sınıfta yapılan türetim, her zaman yeterli olmayabilir. Özelliklerin, birden fazla temel sınıftan alınmaları gerekebilir. Çoklu kalıtım (multiple inheritance) dediğimiz bu yöntemle çeşitli sınıfların özelliklerini, tek bir türetilmiş sınıfta birleştirerek, karmaşık yapıların bir arada oluşturulmaları sağlanır (Stein vd., 1989: 143).



Şekil 4.*Çoklu Kalıtım Erişim Yöntemi*

Şekil 4'deki şemada, örnek olarak iki boyutun alındığı, grafiksel şekiller üzerinde uygulanan, çoklu kalıtım erişim yöntemi gösterilmektedir. Burada AçıkŞekil ve KapalıŞekil sınıfları, Şekil sınıfına erişebilmekte, bu sınıfın eleman ve fonksiyonlarını kullanabilmektedirler.

D. Paketleme (Encapsulation)

Nesneleri oluşturan veri, değişken ve yordamların, aynı sınıfta toplanmasına paketleme veya kılıflama (encapsulation) denir.

Çevremizdeki nesneleri onların niteliklerini kullanarak tanımlarız. Daha önce de bahsedildiği gibi, nesneye yönelik programlamada nesneler, nitelikleri ve davranışlarıyla bir sınıf yapısını oluştururlar.

Örneğin, bir nesne olarak düzlemde yer alan bir noktayı düşünelim. Bu nokta, Turbo Pascal programlama dilinde Tablo 5'deki gibi tanımlanabilir:

Tablo 5. *Turbo Pascal'da Nesne Tanımlanması*

```
Type
        nokta=object    x,y=integer;    renk=char;
        procedure hareket(x,y,renk);
End;
```

Burada, noktanın düzlemdeki yerini ve rengini belirleyen x,y ve renk, nokta nesnesinin nitelikleridir. Hareket adlı prosedür ise, noktanın davranışlarını belirlemektedir. Bu durum, nesnelerin, değişkenlerin ve yordamların aynı sınıf içerisinde bulunmalarını sağlamakta ve nesneye yönelimin paketleme (encapsulation) özelliğini gerçekleştirmektedir (Bancilhon vd., 1992: 7). Bu şekilde daha modüler bir yapı oluşturulmakta ve problemlerin anlaşılması ve çözümlenmesi kolaylaşmaktadır.

Bir başka yaklaşıma göre paketleme (encapsulation), kod ve verilerin birlikte objeler içerisinde bir araya getirilmesidir (Morris, 1994: 18). Turbo Pascal nesneye yönelik programlama dili, özel bir yönlendirici kullanımıyla encapsulation özelliğine imkan sağlar. Burada, nesne sahalarına sadece ilgili nesneye ait prosedürler tarafından erişilebilir. Tablo 6'da, sahalar ve metotlar için özel bölümler oluşturulmadan Location ve Point adlı nesnelerin kullanımı gösterilmektedir.

Tablo 6. *Turbo Pascal'da Nesnelerin Kullanılması*

```
type
        Location = object
        X,Y:Integer;
        procedure Init(InitX, InitY:Integer);
        function getX:Integer;   function getY:Integer;
        end;
        Point = object(Location)
        Visible:Boolean;
        procedure Init(InitX,InitY:Integer);
        procedure Show;                 procedure Hide;
        function IsVisible:Boolean;
```

```
procedure MoveTo (NewX,NewY:Integer);
end;
```

Burada yalnızca üç data sahası vardır; X, Y ve Visible. MoveTo prosedürü X ve Y için yeni değerleri yükler ve getX ve getY fonksiyonları X ve Y'nin değerlerini geri getirir. Bu durum, X ve Y'ye direk olarak daha fazla erişimi gerektirmez. Show ve Hide prosedürleri, Visible değişkeninin true veya false almasını sağlar. IsVisible fonksiyonu da Visible değişkeninin o andaki değerini döndürür.

E. Çokşekillilik (Polymorphism)

Çokşekillilik (polymorphism) değişik yapılara sahip fonksiyonların, aynı isimle kullanılmaları işlemine verilen addır.

Bir başka ifade ile, birbirinden türeyen sınıflarda benzer davranışları belirlemek için yazılan yordamlar aynı isimde olabilirler. Bu yordamlar programın çalışması sırasında hangi sınıfla ilgili olarak çalışmışsa, onunla ilgili bir işlevi yerine getirir.

Nesneye dayalı programların en önemli özelliklerinden biri olan çokşekillilik (polymorphism) sayesinde, aynı isimdeki bir yordam ile, değişik varlıkların davranışlarını belirtmek mümkün olmaktadır (Florentin, 1991: 29).

Birbirlerinden türeyen sınıflarda yer alan varlıkların davranışları da benzer olmaktadır. Ancak programın çalışması sırasında, herhangi bir anda bu varlıklardan hangisinin kullanılacağı belirsiz olabilir. Bu durumda, sınıfların davranışlarını belirleyen yordamlara aynı isim verilir ve belli bir anda hangi yordamın çağrılacağına program çalıştırılırken (run time) karar verilir. Çokşekillilik kavramı bir örnek üzerinde Tablo 7'de gösterilmektedir.

Tablo 7. *Çokşekilliliğin Uygulanışı*

```
Class daire
        {protected:
            int merkez_x,merkez_y,yarıçap;
          public
            virtual void ciz();   };
Class silindir:public daire
        { int yükseklik;   public:   void ciz(); };
```

Görüldüğü gibi, her iki sınıfta da ciz isimli bir yordam bulunmaktadır. İsimleri aynı olmakla birlikte, bu yordamların iç yapıları ve yaptıkları işler farklıdır. Bu yordamların çağırılma şekillerini izah edebilmek için, işaretçilerden (pointerlerden) bahsetmemiz gerekmektedir. Pointer kullanımı, nesneye yönelik programlamanın vazgeçilmez özelliklerindendir. Programcıya

esnek programlar yazmada kolaylıklar sağlar. Her değişkenin bellekte bir adresi vardır. Değişkenlerin isimleri verilmeden, doğrudan adresleri verilerek bu değişkenlere erişilebilir. Yani pointerler, herhangi bir değişkenin adresini gösterirler. Pointer kullanılmadan değişkenlerin, hafızada (memory) nerede saklandığı bilinemez. Normal değişkenlerde, o değişkenin değeri saklanır. Pointerlerde ise, yalnızca verinin saklandığı adres saklanır. Pointerler kullanılmadan önce tanımlanmalıdırlar ve değerleri atanmış olmalıdır. İki çeşit pointer işlemcisi vardır:

(1) &, bellek adresini gösterir.

(2) *, adres değerini gösterir.

&, işareti adres işlemcisidir. örneğin X, tanımlanmış bir değişken ise, &X onun değerinin bulunduğu adresi saklar. * işlemcisi ise bu adresteki değeri göstermek için kullanılır (Li, ve Shahidehpour, 1992: 81-83)

Pointerler hakkında verilen bu genel bilgilerden sonra, çokşekillilik kavramı için Tablo 7'de verilen örnek üzerinde, önceden tanımlanmış bulunan daire ve silindir sınıflarına işaret eden pointerler tanımlanabilir.

```
Daire *işaret_daire;
Silindir *işaret_silindir;
Bu iki tanımlamadan sonra atamalar yapılabilir.
işaret_daire=işaret_silindir;
```

Kural olarak alt sınıf pointer'i, ana sınıf pointer'ine atanabilir. Çünkü alt sınıf daha çok veri içermektedir. Bu sayede, ana sınıfa ait veri alanları doldurulabilir. Ancak ana sınıf pointer'i alt sınıf pointer'ine atanamaz. Bu atama şekli ile yordamların çok şekilli olması sağlanabilir. Bu durum Tablo 8'de verilmektedir.

Tablo 8. *Pointer Kullanımı İle Çokşekilliliğin Sağlanması*

```
Daire *işaret_daire, nesne_daire;
Silindir nesne_silindir
İf (getch()= = 'd') işaret_daire=& nesne_daire;
        Else işaret_daire=&nesne_silindir
işaret_daire Çiz();
```

Programın başında daire ve silinidir sınıflarına ait iki nesne tanımlanıyor. Bunlar nesne_daire ve nesne_silinidir. İşaret_daire adlı pointer ise her iki sınıf için de kullanılabilecek durumdadır. Programda kullanıcının girdiği karakter 'd' ise bu pointere nesne_daire'nin adresi, aksi halde nesne silindir'in adresi aktarılmaktadır. Böylece işaret_daire ciz() satırında gerçekte hangi ciz() yordamın çağrıldığı bilinmektedir.

Görüldüğü gibi, programın derlenmesi esnasında ilgili satırda hangi yordamın çalışacağı belli olmamasına rağmen, programın icrası sırasında hangi yordamın çalışacağı belirlenebilir (Blair, vd., 1988: 80).

Bu yapı sayesinde oldukça esnek ve verimli programların yazılması mümkün olmaktadır.

F. Erken ve Geç Bağdaştırma (Dynamic and Late Binding)

Nesneye yönelimin temel özelliklerinden olan, erken bağdaştırma ve geç bağdaştırma, programlama mantığında derleme zamanı ve çalıştırma zamanı olaylarına karşılık gelir.

Nesneye yönelim terimi olarak, erken bağdaştırma, bir nesnenin kendisini çağıracağı fonksiyonla bağlantısının, derleme anında yapılması anlamındadır. Bu da, hangi fonksiyonun çağrılacağının, programın derlenmesi sırasında belirlenmesi demektir. Buna örnek olarak standart fonksiyonları ve operatör fonksiyonlarının çağrılmasını verebiliriz. Erken bağlanmanın asıl faydası etkin bellek kullanımıdır (Bancilhon vd., 1992: 11).

Nesneye yönelik diller, güç ve esnekliklerinin önemli bölümünü geç bağdaştırmaya (late binding) borçludurlar. Geç bağdaştırma kabaca, bir programın hangi nesne üzerinde hangi yöntemi kullanılacağının çalışma anında belirlenmesidir. Nesneye yönelik olmayan dillerden birinden bir fonksiyon veya alt programa çağrı yapıldığında, derleyici derleme anında hangi rutini çağıracağını kesinlikle bilir. C++ veya Turbo Pascal gibi nesneye yönelik bir dilde ise çağrılacak olan kod nesnenin sınıfına bağlıdır. Bu durum, genellikle hangi rutinin çağrılacağını ancak çalışma anında belirleyebilmemize izin verir (Sarıdoğan, 1994: 216).

Bağdaştırma, bir ad ile bir nesne arasında ilişkinin kurulmasıdır. Yani programlama dilinde mevcut bir tanıtıcı ile, değişkenler ya da üniteler arasında kurulan ilişkidir. Derleyiciler sayesinde programcılar, bu ilişkinin derleme ve link etme sırasında yapılıp bittiğini düşünürler. Bir değişken veya fonksiyona verdiğimiz adın, tanımlandığı blok içerisinde hep ona ait olduğunu düşünürüz. Hemen hemen, nesneye yönelik olmayan bütün diller böyle çalışırlar. Bu diller, aslında erken bağdaştırma mekanizmasına sahiptirler. Çünkü adlandırma, program çalışmadan önce yapılır. Buna karşılık, yorumlama esasına göre çalışan pek çok dil, bir programın çalışma anında tanımlanabilmesine izin verir. Hatta nesneye yönelik programlama dillerinden Smalltalk, bu işi çok daha ileri götürerek, programcının çalışma sırasında işletim sistemini değiştirmesine bile izin vermektedir. Bu dilde var olan bir sınıf değiştirildiğinde, yeni sınıf, değişiklikler dışında tüm özelliklerini yukarıdan miras alabilir. Üzerinde çalışılan nesneler, işletim sistemine ait olsa bile sistem çalışmaya devam eder. Eski sınıfa ait tüm nesneler, yenileri ile değiştirildikten sonra, eski sınıf tümü ile devre dışı bırakılabilir. Bu durumlar, bir fonksiyon her çağrıldığında bir tablodan adına karşılık gelen kodun bulunup çalıştırılmasını sağlayan, geç bağdaştırma sayesinde gerçekleşir.

Geç bağdaştırmanın yararları, nesneye yönelimin diğer temel özelliği olan çokşekillilik (polimorphism) kavramı incelendiğinde de iyice anlaşılmaktadır.

Farz edelim ki nokta, daire, dikdörtgen gibi çeşitli nesneleri ekrana çizmeyi sağlayacak bir grafik paketi hazırlıyoruz. Bu iş için, bu çizimlerin ortak özelliklerinin tanımlandığı bir taban sınıf tanımlanmalıdır. C++ veya Turbo Pascal'da gerekli tanımlamalar Tablo 9'da gösterildiği gibi yapılabilir:

Tablo 9. *Nesnelerin Ortak Tanımlanması*

C++

```
Class Graphobj{
        int x,y; /* Nesne yeri */
      public:
      int getx {return x;};
          int gety {return y;};
      Void moveTo (int newx,int newy)
          {x=newx;y=newy;};};
```

Turbo Pascal'da

```
      Type Graphobj=object;
          x,y:integer;  {Nesnenin yeri}
      Function getx:integer;
      Function gety:integer;
      Procedure moveTo(newx,newy:integer);
      End;
```

Erken bağdaştırmada program daha hızlı çalışır, buna karşılık programcıya sağlanan esneklik azalır. Geç bağdaştırmada ise nesnenin hangi fonksiyonu çağırdığı, çalıştırma zamanında belirlenir. Bu özellik programcıya geniş esneklik getirir. Ayrıca tekrar kullanılabilen ve sonradan geliştirilme imkanı olan kütüphanelerin oluşturulmasına yardımcı olur.

Programlamanın, bu iki bağdaştırma türünden hangisini kullanacağı, tasarım amacına bağlıdır. Küçük hacimli programlar genellikle erken bağdaştırmayı kullanırken, daha geniş programlar her iki metodu birden kullanırlar. Geç bağdaştırma nesneye yönelik programlamanın üstün yönlerinden biri olmasına rağmen, bu özellik programların biraz yavaş çalışmalarına neden olur. Fakat bu göz ardı edilebilir bir yavaşlamadır. Bu nedenle de geç bağdaştırma, nesneye yönelik programlamada genellikle en çok tercih edilen bağdaştırma yöntemidir.

Derleyicinin, bir sınıf ile o sınıftan türetilen değerler arasındaki farkı anlayabilmesi ve farklı sınıflarda tanımlanan kodları, ayrı ad kullanarak çağırabilmesi çokşekilliliğin, geç bağdaştırma kullanılarak elde edildiği güzel bir örneğidir (Bancilhon vd., 1992: 14).

Geç bağdaştırma programın bellek ihtiyacını artıracaktır. Bunlara rağmen geç bağdaştırma, esnekliği nedeniyle programcının işini oldukça kolaylaştırır. Nesnelerin kendilerini düşünmeleri, birbirlerini takip etmeleri, programcının onları düşünüp takip etmesinden daha iyidir. Ayrıca, diğer programcıların kolayca kullanabileceği yazılımlar hazırlamak için de uygundur. Zaten piyasalarda geliştirilen paket uygulamalar, geniş ölçüde bu tür araçları kullanmaktadırlar.

Son olarak da, geç bağdaştırma yöntemi ile işin çoğu derleyiciye yüklenir. Çünkü geç bağdaştırmada, hangi nesne üzerinde hangi yöntemin kullanılacağı ve hangi rutinin çağrılacağı çalışma anında tespit edilir. Bu da derleyicinin bu bilgiler üzerindeki kontrolünün sürekli olmasını gerektirir. Bu sayede hem programın karmaşıklığı, hem de hata oranı azalmaktadır. Donanımın nispeten ucuzlayıp, yazılım hatalarının önemli masraf çıkarması nedeniyle ve bu durumun devam etmesi halinde, nesneye yönelim ve onun bir özelliği olan geç bağdaştırma, hiçbir programcının göz ardı edemeyeceği kavramlar olacaktır.

G. Soyutlama (Abstraction)

Kompleks problemlerin çözümlenmesi esnasında, problemin temel bir grup sahalara bölünmesi ve bölümler üzerinde bağımsız olarak çalışılması genel bir yaklaşımdır. Bu yaklaşım, soyutlama (abstraction) kavramına da uygun düşmektedir. Problemin anlamlı ve orantılı bir şekilde safhalara ayrılması soyutlama kavramının özüdür. Problemlerin çözümünde bilgisayar bilimcilerinin alakadar oldukları en önemli husus, problemin uygun safhalara ayrılması ve gereken cihaz ve tekniklerin hazırlanmasıdır. Bu aşamada, en uygun cihaz ve tekniklerin belirlenebilmesi önemlidir ve bunun için şu problemler çözümlenmelidir:

(1) Cihaz ve teknikler, problemi çözebilmek için spesifik problemlere uygun olmalıdır.

(2) Cihaz ve teknikler, donanıma göre etkin bir biçimde gerçekleştirilmiş olmalıdır.

Problem çözmeyi kolaylaştırmak amacıyla, uygun soyutlamanın belirlenmesi için birkaç yol düşünülmüştür. Soyutlamalar, bazen özel problem safhaları için geliştirilmişken, bazen de genel problem çözmeyi desteklemek için geliştirilmişlerdir. Yine de bilgisayar bilimcilerinin karmaşıklığı gidermek amacıyla, soyutlamayı cihaz ve tekniklerin geliştirilmesi ve gruplandırılması şeklinde, sınırlandırdıklarını söylemek mümkündür.

Nesneye yönelik soyutlama, özel problem safhalarının oluşturulması olarak kabul edilmektedir. Bu amaca ulaşmak için, muhtemel uygulamalar skalasında özel problem çözümünün prensiplerinin iyi algılanması gerekir. Bu safhaların oluşturulmasında, nesneye yönelik yaklaşımda uygulanabilecek temel prensipler aşağıdaki kesimlerde ele alınmaktadır (Blair, vd., 1992: 80).

1. Veri Soyutlama (Data Abstraction)

Nesneye yönelik yaklaşımın temel prensiplerinden, soyutlama kavramının ilk prensibi veri soyutlamadır (data abstraction). Veri soyutlama, hem veri, hem de veriyi kontrol etmek için gerekli olan algoritmalardan programcının ayrı ayrı haberdar olmasıdır. Soyutlamanın yapılmadığı durumlarda kullanıcılar, faaliyetlerin detaylarından tam olarak haberdar değildirler. Veri soyutlama, ayrı fakat yakınen alakalı iki yapıyı kapsar:

(1) Modüllere ayırma (Modularisation): Modularisation, kompleks sistemleri bir grup kendine has (self-contained) modüllere bölme ile alakalıdır. Özel bir kısma ait tüm bilgi, sistemde bu model ile tutulur. Bu durum, bu sistemin bir parçasını icra ettirecek algoritmaların ve veri yapılarının model tarafından kapsanmasını sağlar. Bu, eğer değişiklik yapılmak zorunluluğu varsa ve problemler çözülecekse en uygun işlemin yapıldığı anlamındadır. Daha temelde, programcıların, problem safhalarını bir grup tanımlanabilir özel içerikli parçalara böldüğü özel tasarım, modularisation kavramı ile açıklanabilir. Bu tasarım yaklaşımı, nesneye yönelik soyutlamanın temelini belirler. Bu, aynı zamanda soyutlamanın da ilk seviyesini oluşturur.

(2) Bilgi Saklama (Information Hiding): Modüllere ayırma işleminde detayların kullanıcıdan saklanması bir sonraki soyutlama seviyesini oluşturur. Bilgi saklama ile, korunmuş ara yüze (interface) kullanıcı, bir nesne ile ulaşmalıdır. Kullanıcının lokal prosedürler veya veri yapıları gibi iç detayları görmesine izin verilmez. Bu, karmaşıklığın giderilmesinde önemli bir kuraldır.

2. Davranış Paylaşımı (Behaviour Sharing)

Nesneye yönelik soyutlamanın ikinci prensibi, sistemlerin davranış paylaşımını desteklemektir. Davranış paylaşımı, aynı ara yüz (interface) setinin birçok kısımda ortak kullanımına müsaade etmek anlamındadır. Bu, sistemin esnekliğini arttıran bir durumdur. Davranış paylaşımının en yaygın kullanım şekli, sınıflandırma özelliği kullanılarak gerçekleştirilir. Sınıflama, kısımların bir grubunu genel davranış temeline göre yeniden şekillendirir. Örnek olarak, queue; insert, delete ve print'ı destekleyen bir kısım olarak tanımlandığında bütün queue'ler insert, delete ve print davranışlarını paylaşırlar.

3. Gelişme (Evolution)

Programcılıkta ihtiyaçların çok sık değiştiği, bir gerçektir. Belirli bir dönem içerisinde sistemler değişir ve ek fonksiyonlara ihtiyaç duyulabilir. Sürekli değişim gösteren ve gelişen ihtiyaçlar nedeniyle, nesneye yönelik soyutlamada gelişme (evolution) prensibi oluşmuştur. Bu değişim ihtiyaçlarının karşılanabilmesi için, geliştirilecek sistemin hedefleri iyi belirlenmelidir. Gelişme prensibi iki şekilde gerçekleştirilebilir:

(1) İhtiyaçların Gelişimi (Requirements Evolution): Bir kere sistem geliştirildiğinde, modifikasyon ve eklemeler gerektirecek değişikliklerin

oluşması muhtemeldir. Bu değişikliklere kısmen dinamik ortamlarda ihtiyaç duyulacağından, tasarım esnasında bu ihtiyaçların giderilmesi oldukça zordur. Genellikle sonradan ortaya çıkan ihtiyaçların giderilebilmesinde soyutlama kavramının bu prensibi kullanılır.

(2) Gelişme İle Birlikte Çözüm (Solution by Evolution): Problemlere daha genel çözümler üretmek amacıyla, ilk deneylerden yazılım üretiminin son aşamasına kadar artan tarzda çözümler geliştirmektir. Nesneye yönelik yaklaşımın felsefik bakış açısına göre, her iki görüş tek bir yaklaşım olarak kabul edilebilir. Bundan dolayı gelişme (evolution), sistemin tamamını, başlangıç adımından bakım aşamasına kadar her zaman etkileyen bir soyutlama prensibidir.

4. Doğrulama (Correctness)

Nesneye yönelik soyutlamanın son prensibi doğrulamadır. Doğruluk terimi program doğruluğu, uygunluk testi ve hata toleransı gibi işlemlerin kontrolünde kullanılır. Her bir nesne, sistemdeki diğer nesnelere özel (specific) davranışlarda bulunur. Diğer bir deyişle her nesne için özel davranışlar söz konusudur. Yani doğrulama prensibi, nesneye yönelik sistemlerde nesnelerin spesifik davranışlarının doğruluklarını kontrol eder.

III. Sonuç

Nesneye yönelik yaklaşım tekniği, gerçek dünya nesnelerini örnek alır ve programlama mantığını nesne ve özelliklerini dikkate alarak oluşturur. Bu tekniğin en temel kavramı, nesnedir. Veriler ile onlara ait özellikler, birlikte nesneyi temsil ederler. Ayrıca, bu tekniği oluşturan diğer üstün kavramlar kullanılarak, hem yazılım üretiminin bölümler halinde yapılabilmesi, hem üretilen yazılımlara ihtiyaca göre ilaveler yapılabilmesi ve hem de yazılımın, üretildiği donanımdan bağımsız olarak çalıştırılabilmesi mümkündür.

Saydığımız tekniklerin avantajları nedeniyle, nesneye yönelimin kullanımı her geçen gün biraz daha artmaktadır. Ancak bu yöntemle yazılım geliştirilirken, bazı problemlerle de karşılaşmak mümkündür. Kullanılan programlama tekniklerinde analiz ve tasarım safhalarının yeterince ayrı düşünülememesi, nesneye yönelimin takım çalışmasını gerektirmesi, gerçek dünya ile uyumlu nesnelerin bulunma zorlukları ve geleneksel çalışmalara uygun olmadığı için geleneksel model ile gerçekleştirilen yazılımlar üzerinde nesneye yönelik çalışma yapılamaması gibi önemli problemler nesneye yönelik yaklaşımın kullanımını güçleştiren faktörlerdir. Bu nedenle çok basit programların üretiminde ve fazla değişiklik gerektirmeyecek projelerin hazırlanmasında, bu tekniğin kullanılmasına gerek olmadığı, daha kompleks ve uzun vadeli projelerde bu teknikten yararlanılması gerektiğini düşünmekteyiz.

Summary: Programming techniques have shown a continuos development so far. Among these, the most commonly used and still the most recently developed one is the object-oriented approach technique. This technique which is also named as "beyond the fourth generation" takes the object as the base and imitates the human intellect. The most important reason that object-oriented approach attracts such an acceptance is its having numerous fundamental concepts. Though these concepts bear varying features, they support each other.

Kaynaklar

Atkinson, C. (1991) **Object Oriented Reuse Concurrency and Distribution**. Newyork, ACM Press.

Ayık, Y. Z. (1996) **Yazılım Geliştirmede Nesneye Yönelik Yaklaşım ve Nesne Tabanlı Bir Çizim Aracı Geliştirme Uygulaması**, Erzurum, Basılmamış Doktora Tezi, 1996.

Bancilhon, F., Delobel, C. ve Kanellakis, P. (1992) **Building an Object Oriented Database System the Story of O2**, USA, Morgon Kaufman Publishers.

Blair, G., Gallagher, J., Hutchisaon, D. ve Shepherd, D. (1988) **Object Oriented Languages Systems and Applications**, New York, Halsted Press.

Caruso, M. ve Sciore, E. (1987) **The Vision Object-Oriented Database Management System**, Newyork, ACM Press.

Demurlian, S. A., Beshers, G. M. ve Ting, T. C. (1993) "Programming Versus Databases in the Object-Oriented Paradigm". *Information and Software Technology*, No:2 USA, Butterworth Ltd.

Florentin, J. J. (1991) **Object Oriented Programming Systems Tools and Applications**, Newyork, Chapman & Hall.

Halbert, D, C. ve O'Brien, P. D. (1987) "Using Types and Inheritance in Object-Oriented Languages", *European Conference on Object-Oriented Programming*, Paris.

Hunt, V. D. (1986) **Artificial Intelligence & Expert Systems Sourcebook**, Newyork, Chapman & Hall.

Li, S. ve Shahidehpour, S. M. (1992) "An Object Oriented Power System Graphics Package for Personal Computer Environment", *IEEE Transactions on Power Systems*, No:3. USA, IEEE.

Morris, S. (1994) **Object Oriented Programming Under Windows**, British Library, Butterwort-Heinemann Ltd.

Paepcke, A. (1993) **Object Oriented Programming the Clos Perspective**, USA, Teach Books.

Rumbaugh, J. (1990) **Object-Oriented Modeling and Design**, U.S.A., Prentice-Hall.

Sarıdoğan, M. E. (1994) **C++ ve Nesneye Yönelik Programlama**, İstanbul, Sistem Yayıncılık.

Stein, L. A., Lieberman, H. ve Ungar, D. (1989) "A Shared View of Sharing: The Treaty of Orlando", In:**Object-Oriented Concepts, Databases and Applications**, New York, ACM Press.

Tom, G. (1988) **Principles of Software Engineering Management**, New York, Addison-Wesley.

Voss, G. (1991) **Object Oriented Programming: An Introduction**, Tokyo, Osborne McGraw - Hill.