**BLM 426 YAZILIM MÜHENDİSLİĞİ**

**BAHAR 2016**

**Yrd. Doç. Dr. Nesrin AYDIN ATASOY**

**7. HAFTA: YAZILIM TASARIMI**

# NESNEYE YÖNELİK TASARIM

**Nesneye yönelik çözümleme ve tasarım** günümüz yazılım geliştirme ortamlarının en yaygın kavramlarından biridir. Çoğu zaman büyük bir kurtarıcı olarak algılanmasına rağmen gerçekte böyle bir kurtarıcı bulunmamaktadır. Nesneye yönelim, daha gele­neksel olan işlevsel çözümleme ve veri akış yöntemlerine göre daha değişik bir yaklaşım gerektirmektedir.

Nesneye yönelik çözümleme ile nesneye yönelik tasarım arasında kesin bir ayrım yap­mak bazen mümkün değildir. Temel olarak, çözümleme bir sınıflandırma etkinliği olarak değerlendirilebilir. Bu aşamada problem çözümü için kullanılacak nesnelerin ait olacağı sınıfların belirlenmesi için çalışma yapılır. Tasarım aşamasında ise sınıflar­dan oluşturulan nesneler ve onların aralarındaki ilişkiler tanımlanır.

Nesneye yönelik tasarım en basit şekliyle, işlemlerden çok ***nesne*** kullanan bir düşün­ceye dayanır. Bu düşünceyi destekleyen programlama dilleri de programı ***nesne*** adı verilen küçük parçalara bölebilmeyi sağlayan özel yapılara sahiptir. Nesnenin içyapısında durum bilgileri ve ara yüz olarak tanımlanan işlemler, yani yordamlar kümesi bulunur. Bu işlemlerle durum bilgisi üzerinde değişiklik yapılabilir.

Nesneye yönelik tasarımın gerçekleştirilebilmesi için mutlaka nesneye yönelik bir programlama dili kullanılması zorunlu değildir; ancak nesneye yönelik tasarımın tüm özelliklerden yarar sağlayabilmek, bu yararı özellikle kodlamada görebilmek için nesneye yönelik bir dil seçilmelidir. Eğer bu sağlanamazsa, nesneye dayalı bir dil ile veya o hale getirilmiş yapılarla desteklenen bir dil kullanılmalıdır.

Tasarımcılar artık problemleri önceden tanımlanmış veri ve denetim yapıları kulla­narak çözmek zorunda değillerdir. Nesneye yönelim ile kendi tasarladığı soyut veri yapılarım ve işlevsel kapsamayı kullanarak gerçek dünya problemlerini çözebilir duruma gelmişlerdir. Bu amaçla kullanılan temel soyutlama birimi de nesnedir.

Diğer tasarım yöntemleri gibi nesneye yönelik tasarım yöntemi de bir dizi tanım, gös­terim şekli ve yordam kullanmaktadır. Bunlar, kullanılan araç veya dile göre küçük farklılıklar gösterseler de temelde aynıdırlar. En önemli ortak özellik olan nesne gerçek dünyadaki varlıkların yazım alanına uyarlanmasında kullanılır. Her nesnenin bir ara yüzü, bir veri yapısı ve bir dizi işlemi vardır.

Nesneler birbirleriyle ilişkili veri ve işlemleri bir arada tutarak kapsama (encapsula- tion) ve iyi tanımlanmış bir arayüz ile modülerlik (modularity) ilkelerini desteklerler. Nesnenin genel kısmı herkese açık ve paylaşılır verileri içerirken özel kısmında yalnızca kendisi tarafından kullanılan veriler bulunur. Böylelikle de Bilgi Gizleme (informatıon hiding) ilkesi desteklenmiş olur.

Nesneye yönelik tasarımın aşamalarına geçmeden önce nesneye yönelimin temel özel­liklerine değinmekte yarar vardır.

# TEMEL ÖZELLİKLER

Nesneye yönelik tasarım yönteminin en temel özellikleri şunlardır:



Şekil 1. Nesne özellikleri

### **Kapsama (encapsulation)**

Bir veri yapısı ve onun üzerinde işlem yapan bir grup yordamın iyi tanımlanmış bir arayüzü olan, erişimi ve kullanımı kolay bir yapı içine konmasına kapsama denir. Şekil 1(a)’ da görüldüğü gibi yordamlar ve özelliklerden oluşmaktadır. Nesneye yönelik tasarımda bu yapı nesnedir ve gönderilen İletilerle arzu edilen işi yapması sağlanır. Her nesne belirli bir veri grubunu ve işlevleri kap­sar; farklı veriler farklı nesnelerde yer alır. Sınıfların ve onlardan oluşturulan nesnelerin adları kapsamlarını en iyi tanımlayacak şekilde seçilir.

### **Sınıf (class)**

Nesneye yönelik programlamanın önemli özelliklerinden biri olan sınıf, kap­samayı sağlayan bir gruplandırma yapısıdır. İçeriğinde hem veriler hem de işlemler bulunur. Sınıf kendisinden oluşturulacak nesneler için bir şablon görevi görür.

### **Çokşekilcilik (polymorphism)**

Çokşekillilik bir ismin birbirleriyle bağlantılı fakat aslında değişik olan birden fazla amaç için kullanılmasıdır. Çokşekilliliğin amacı, bir tek ismi genel bir işlevler kümesini tanımlamada kullanmaktır. Hangi işlevin gerçekleşeceği üzerinde işlem yapılacak veri tipi tarafından belirlenir.

### **Kalıtım (inheritance)**

*Kalıtım*, bir hiyerarşik şeklinde, bir sınıftan veri ve işlemlerin tekrar kullanım amacıyla devralmasıdır. Durağan kalıtımda devralınan kod derleyici tara­fından türetilmiş nesneye kopyalanır. Dinamik kalıtımda ise, sistem en uygun aktif yöntemi yürütme sırasında seçer ve devralınan kısımla ilgili bir görevcik başlatır. Durağan kalıtım bellek kullanımı dışında oldukça etkin ve güvenlidir. İsteğe bağlı dinamik kalıtımda ise işlemler ve veriler ortak bir yerde tutulur, gerektiğinde bu kod dinamik olarak bağlanır. Tekrar kullanımı en yükseğe çıka­rabilmek için birden fazla sınıfın temel sınıf olduğu çoklu kalıtım (multiple inheritance) ortaya çıkmaktadır.

Bu özelliklerin tamamının tasarım sırasında dikkate alınması gerçekleştirimde kul­lanılacak programlama diline bağlıdır. Bazı diller kapsamayı desteklerken sınıf yapısını desteklemezler; bazıları da kalıtımı desteklemezler. Bu nedenle, nesneye yönelik tasarımdan yararlanabilmek için tasarım aşaması sırasında hangi dilin kul­lanılacağına karar verilmelidir.

# TİPLER VE SINIFLAR

Tip kavramı önceleri belirli bir topluluğun genel özelliklerini temsil etmek üzere küme oluşturmada kullanılmıştır. Örneğin, **integer** tipi tüm tamsayılar kümesini temsil etmiş, bu kümenin elemanları üzerinde de çeşitli işlemler yapılabilmiştir. Küme içinde *altküme* adı verilen gruplar da oluşturulabilir. Bu altkümeleri temsil eden tipler *alttip* (subtype) adını alırlar. Örneğin, tamsayılar, gerçel sayıların bir altkümesi olduğundan, C'deki **int** tipi **float** tipinin alttipidir. C ve C++ dillerinde kullanılan **unsigned** int ve **unsigned char** gibi tipler de aslında birer altkümedir.

İlk zamanlardaki sayısal hesaplamalar mühendislik ve matematik uygulama alanlarına yönelik olduğundan, hesaplarda kullanılan nesneler ve tipler, günün gereksinimlerine göre, kullanılan dilin değişkenlerine ve tiplerine karşı düşmekteydi. Fakat program­lama dilleri gerçek dünyayı makine dünyasına yansıtmak için bir takım yaklaşımlar ve sınırlamalar yapmak zorundadır. Örneğin, sayı kümeleri gerçek anlamda sınırsız değildirler; gösterilebilecek en büyük ve en küçük sayı değerleri kullanılan bilgisayar donanımının sözcük uzunluğuna ve uygulanan sayı biçimlerine bağlıdır. Bu nedenle **int** ve **float** tipleri makinede farklı şekillerde işlem görürler.

Nesneye yönelimin yapı taşı olan sınıflar da tiplerin taşıdığı özellikleri taşırlar. Bun­lara ek olarak, kullanıcının kendi isteklerine uygun tipler oluşturulması, bunlar üzerinde kendi istediği yordamlarını tanımlayabilmesi ve sınıfları birbirlerine çevirme işlevleri kazandırması mümkündür. Bir tipi diğerine çevirme işlemi bazı durumlara göre değişebilir.

İyi bir nesneye yönelik tasarımda, uygulama alanında (application domain) tipler, çözüm alanında (solutıon domain) da sınıflar kullanılır. Uygulama alanında ortaya konan varlıklar (entity) nesnelerle, tipler de sınıflarla gösterilir. Uygulama alanı varlıklarına örnek olarak bir banka sisteminin işlem, para, hesap ve otomatik makine­lerini, bir haberleşme sistemine ait telefonları, konuşmaları, hatları ve anahtarlarma kutularını verebiliriz. Bu varlıkları genelleştirerek tip haline dönüştürür ve bireylere ait varlıklar yerine genel olarak işlemlerden, hesaplardan, telefonlardan bahsedebili­riz. Uygulama alanına ait bu kavramlar yazılımın gerçekleştirildiği çözüm alanında tiplerin sınıflara, varlıkların da nesnelere dönüştürülmesinde kullanılırlar. Varlıkların ve bunların birbirleriyle olan ilişkilerinin belirlenmesinde yapılacak bir hata daha son­raki gelişmeleri aksatacak boyutta sorunlar oluşturabilir. Bu sorunlar özellikle alttip ilişkileri bulunduğunda üst düzeye çıkar. Bu nedenle uygulama alanındaki çözümle­melerin iyi yapılmasına, varlıkların ve ilişkilerin en doğru şekilde belirlenmesine çalışılmalıdır.

# NESNEYE YÖNELİK TASARIM AŞAMALARI

Nesneye yönelik tasarımı; nesneye yönelik olmayan programlama dilleri için ve nesneye yönelik diller için olmak üzere iki şekilde incelemekte yarar vardır:

### **Nesneye Yönelik Olmayan Diller İçin Tasarım**

Nesneler çözümleme aşamasında genel olarak belirlendikten sonra tasarım tanımla­ması yapılır. Bu amaçla nesnelerin iki türlü tanımlaması olabilir:

***İletişim tanımlaması,*** nesnenin ara yüzünü, alabileceği iletileri ve aldığı bir iletiyle ne gibi bir işlem yapacağını anlatır. Genellikle iletinin adım ve her bir ileti için yapılacak işlemleri anlatan bir liste şeklindedir. Çok sayıda ileti içeren büyük sistemler için iletilerin gruplanması mümkündür.

***Gerçekleştirim tanımlaması***, iletiyi aldıktan sonra nesnenin yürüteceği yor­damın ayrıntılarını, nesnenin genel ve özel kısımlarının yapısını içerir. Veri tipleri ve yordamlar ya tanımlanır ya da başka yerde tanımlanmış olanlara ilgi yapılır.

Tasarım aşamasında çeşitli araçlar ve yöntemler kullanılabilir. En doğru ve en iyiye ulaşan birden fazla yol olabileceğinden, tasarım, hala bir sanat olma niteliğini korur. Aşağıda belirtilen tasarım basamaklarını kılavuz olarak kullanarak ilerlemek başarılı bir sonuç verir:

İsterler çözümlemesi aşamasında;

* Problem tanımlanarak kapsam belirlenir. Problemin yazılım ile çözülecek kısmı metinsel bir şekilde anlatılır.
* Nesneler ve öznitelikleri belirlenir.
* Nesnelere uygulanacak işlemler belirlenir.
* Nesneler ve işlemler arasındaki ilişki ortaya konur ve arayüzler oluşturulur.
* Gerçekleştirim için uygun bir programlama dili seçilmesine gayret edilir ve tasarımda kullanılacak yönteme karar verilir.
* Tasarım aşamasına geçildiğinde, çözümleme sırasında yapılanlar gözden geçi­rilir, soyut veri sınıfları, varsa sınıflar, yordam ve ileti özellikleri ile diğer ayrıntılar göz önüne alınarak iyileştirme yapılır.
* Nesnelerin öznitelikleri olarak nesne içinde kullanılacak veri tipleri ve veri yapıları tanımlanır.

### **Nesneye Yönelik Diller İçin Tasarım**

Nesneye yönelik bir programlama dili kullanmak için yapılacak tasarım nesneye yöne­limin tüm özelliklerini oraya çıkaracaktır. Nesneye yönelik bir programlama dilinin en büyük özelliği sınıf yapısıdır. Sınıfların kullanılmasıyla yapılacak bir tasarımın adımları şunlardır:

* Çözümleme sırasında belirlenen nesneler için veri soyutlaması yapılacak sınıflar tanımlanır. Genelde, İsterler Belirtimi belgesinde yer alan her fiziksel nesne bir sınıfa karşılık düşer.
* Soyutlama sağlayan her sınıf için öznitelikler birer değişken (instance variable) haline getirilir. Bunun için çözümleme sırasında belirlenmiş olan öznitelikler kullanılır; herbiri birer sınıf değişkeni, diğer bir deyişle snııf üyesi (class member) olarak tanımlanır.
* Her nesne üzerinde uygulanacak işlemler birer sınıf yordamı haline dönüştürülürler. Nesneler arasındaki iletişim tanımlanır. Bu amaçla, iletiler alındığında çağrılacak yordamlar belirlenir.
* Uygun olan yerlerde kalıtım kullanılır. Eğer yukarıdan aşağıya doğru bir kap­sama süreci izleniyorsa, kalıtım, sınıfların tanımlanması anında, aşağıdan yukarı bir süreç izleniyorsa öznitelik, işlem ve iletişimin tanımlanmasından sonra uygulanır. Amaç, olabildiğince yüksek düzeyde tekrar kullanımı sağlamaktır.
* Tasarım bir senaryoya göre kağıt üzerinde sınanır. Tatmin edici bir başarım sağlaması, yeterli bir hata hoşgörüsüne sahip olması gibi durumlar gözden geçi­rilir ve tüm ileti alış-veriş seçenekleri denenir.
* Bu aşamaların bir çevrim şeklinde tekrarlanabileceği, hatta bir kısmının paralel olarak yürütülebileceği dikkate alınmalıdır. Bundan sonra ayrıntılı tasarım adımına geçilir ve işlemlerin yordamsal ayrıntıları tanımlanır. Bunun için sözde kod kullanımı gibi bir yöntem izlenebilir.

### **Nesneye Yönelimde Nitelik Ölçütleri**

Nesneye yönelimin en büyük özelliği sınıflardır. Herbiri soyut bir tip tanımlayan bu sınıflardan nesneler üretilir. Sınıf yordamları, yani metotlar, nesnenin içyapısına eriş­mek için kullanılır. Bir sınıfın sahip olduğu yordamların sayısı ve her bir yordamın kar­maşıklığı bir metrik olarak kullanılabilmektedir. Bu şekilde sınıfı geliştirmek ve bakı­mını yapabilmek için yaklaşık ne kadar zaman ve emek gerektiği konusunda bir kestirimde bulunulabilir. Sınıfların kalıtım yoluyla devraldıkları yordamlar bu metriğin değerini arttırıcı, nitelikte olduğundan dikkatli bir çözümleme yapılması gereklidir.

Bir sınıfın gelen bir çağrı sırasında işlem yaparken kendi içinde kaç adet yordamı çağırdığı da bir metrik olarak kullanılır. Buna bir de başka sınıfların yordamlarının çağrılma sayısı eklenebilir. Eğer bu sayı çok artarsa o sınıfı test etmek ve hatalarını düzeltebilmek çok zorlaşır. Sınıf yanıtları için en kötü olarak değerlendirilen sayı dikkate alınarak ortalama bir test süresi belirlenebilir. Bu nedenle bu metrik kul­lanılabilirlik ve test edilebilirlik yanında sistem tasarımının da bir ölçütüdür.

Nesneye yönelik sistem tasarımının önemli bir özelliği de kalıtımdır. Kalıtım ile temel sınıfların tekrar kullanımı mümkün olmakta, türetilen sınıflarda temel sınıfın veri ve yordamları kullanılabildiği için karmaşıklık azalmaktadır. Ancak, fazlaca kalıtım kul­lanılması ile tasarım ve bakım güçleşmektedir. Kalıtımın derecesini belirlemek için iki metrik kullanılır:

* Kalıtım hiyerarşisini *derinliği* ve *genişliği.*
* Kalıtımın derinlik ağacı

Kalıtım hiyerarşinin oluşturduğu ağacının tepesinden sınıfın bulunduğu yere kadar olan düzey sayısıdır. Bu sayı ne kadar büyük olursa sınıfın kalıtımla devraldığı yordam ve veri sayısı da o kadar artmış olur. Bu da davranışını belirlemede bir miktar karmaşaya neden olur. Devralman yordam sayısı da bazen bir ölçüt olarak kullanılır.

### **Nesneye Yönelik Tasarım İlkeleri**

Nesneye yönelik tasarımda genel olarak aşağıdaki önerilerimizin dikkate alınması iyi bir yazılım ortaya çıkarmak için atılmış önemli bir adım olacaktır:

* Probleme çözüm olabilecek fikirler somut bir şekilde ve gerçek yaşamdakine uygun bir gösterimle ifade edilmelidir. Yazılım yapısının bu fikirleri açıkça gös­termesi sağlanmalıdır.
* Sistemin tasarımı yapılırken sistemin ne yaptığı değil, ne üzerinde, ne gibi işlem yaptığı göz önünde bulundurulmalıdır.
* Bir şey apayrı birfikir olarak değerlendiriliyorsa onu bir sınıf haline getirmek uygun olacaktır.
* Bir şey apayrı bir varlık olarak değerlendiriliyorsa onu belirli bir sınıfa ait nesne haline getirmek uygundur.
* Eğer iki sınıf arasında önemli ölçüde ortak yanlar varsa bu ortak özellikler bir temel sınıf içinde toplanmalıdır.
* Eğer bir sınıf birkaç nesneyi üzerinde tutan saklayıcı bir özellik gösteriyorsa onu kalıp (template) yapmak yararlı olur (dil uygun olduğu takdirde).
* Evrensel veri ve yordam kullanılmamalıdır. Bu tür gereksinimler yine sınıflarla karşılanmalıdır.
* Bir başka nesnenin iç verilerine doğrudan erişmeye çalışılmamak, bunun yerine o nesnenin yordamları kullanılmalıdır. Tasarım ve gerçekleştirim bu ilke üzerine dayanmalıdır.
* Bir yazılımı veri yapıları ve bunlarla işlem yapan yordamların oluşturduğu bir grup olarak değil de sınıflar ve nesnelerle gösterilen birbirleriyle ilişki halindeki kavramlar olarak düşünmeye çalışmak daha olumlu sonuç verir.

# VERİYE YÖNELİK TASARIM

Veriye yönelik ya da diğer adıyla veri yapılarına yönelik tasarımda veriyle ilgili işlemler fazla dikkate alınmadığı için veri akış diyagramı kullanılmasına gerek yoktur; onun yerine veri yapılarının ve yordamların tanımlamaları yapılır. Tasarım, veri akışından çok bilgi yapısına dayandırılır. Bu yöntem daha çok, giriş ve çıkışın dosyalar şeklinde olduğu iş dünyası uygulamalarında, veri yapılarının yoğun bir şekilde kulla­nıldığı işletim sistemi geliştirmede, endüstriyel üretime ilişkin veri yapılarının üre­tildiği bilgisayar destekli tasarım ve üretim sistemlerinde kullanılır.

Tasarım sırasında genellikle aşağıdaki işler gerçekleştirilir:

* Veri yapılarının özellikleri değerlendirilir.
* Verilerin saklama, arama ve erişim yöntemleri, veri yapılarına ait temel gös­terim şekilleriyle tanımlanır.
* Veri yapıları yazılımın denetim hiyerarşisi içindeki yerine göre uyarlanır. Gerekirse yazılım hiyerarşisi veri yapıları dikkate alınarak yalınlaştırılır.
* Bundan sonraki yazılımın genel tasarımı yordamsal tanımlama şeklinde yapılır.

### **Tasarım Kalıpları**

Yazılım mühendisliğinde en yaygın ve etkin yöntemlerden biri olan nesneye yönelik çözümleme ve tasarım yeni bir konunun doğmasına neden olmuştur: Tasarım kalıpları ya da diğer çevirilerle tasarım örüntüleri veya tasarım desenleri. Tasarım kalıpları, elde edilen deneyimlerin, çıkarılan derslerin ve en iyi sonuçların belgelendirilmesiyle oluşmuş modern mimarilerle problem çözme disiplinleridir. Aslında, teknolojiye odaklanmak yerine, güvenilir ve sağlıklı bir mimariye dayalı tasarım yapılarak belge­lendirilmesi için bir kültür oluşturma ilkesine dayanır.

### **Kalıp Tanımı ve Özellikleri**

Tasarım kalıpları çözümleyicinin, tasarımcının ve kodlayıcının aynı dili konuşmala­rını sağlayarak iletişimi kolaylaştırır. Bir tasarım kalıbı belirli bir bağlam içindeki bir problemi ve onun çözümünü kapsayan bir kuraldır. Genelde yazılım geliştirme sırasında karşılaşılan tekrarlanan problemlere tekrar kullanılabilir çözümler üretmeyi amaçlar. Kalıpların algoritmalar gibi belirli kuralları vardır. Ancak, algoritmalar ve bunlara bağlı olarak veri yapıları genellikle sıralama, arama gibi küçük çaplı bilgi işleme problemlerini çözmede kullanılırlar. Kalıplar ise daha geniş etkileri olan ve mimariye yönelik konularla ilgilenir ve bu düzeydeki problemlere çözüm getirir. Öte yandan, çatılar (frameworks) genellikle yürütülebilir kod şeklinde olup bir ya da birden fazla kalıp çözümlerinin gerçekleştirilmesinden oluşur. Kalıpların ortak özel­likleri arasında şunlar vardır:

* Deneyimlerle ortaya çıkmışlardır.
* Daha büyük problemleri çözmek için beraberce kullanılabilirler.
* Çeşitli soyutlama düzeylerinde bulunurlar.
* Yapısal bir biçimde yazılırlar.
* Tekrar kullanılabilir öğelerdir.
* En iyi uygulamalar ile tasarımlar arasında bir tür iletişim sağlarlar.
* Yine de her çözüm, algoritma, pratik uygulama veya deneyim mutlaka bir kalıp oluşturmaz. Gerçekten bir kalıp olabilmesi için hiç değilse üç farklı sistemde tekrarlanan bir probleme çözüm olması gereklidir. İyi bir kalıbın özellikleri arasında da şunları sayabiliriz:
* Yalnızca bir takım ilke ve stratejilerden oluşmaz, belirli bir probleme belirli bir çözüm sağlar.
* Çözümleri kayıt altına alarak herhangi bir şüpheye yer bırakmaz.
* Kalıp yalnızca modülleri değil, daha alt düzeydeki yapı ve düzenekleri de tanımlar.
* İnsanlara kullanım kolaylığı ve yarar sağlar.
* Gerçek dünyadaki bir probleme uyarlanabilecek şekilde kullanışlıdır.
* Kullanılmıştır ve bu şekilde belgelendirilmiştir.
* Bir kalıp çeşitli Öğelerden oluşmaktadır. Genel olarak, bir kalıpta bulunması gerekli öğeler şunlardır:

**KAYNAKLAR**

1. Yazılım Mühendisliği; M. Erhan Sarıdoğan.