6.5.3. Nesne ve Sınıf İlişkisi

Nesnelerle sınıflar arasında bazı ilişkiler kurulmaktadır. Bunlardan en temel olanları şöyle sıralayabiliriz:

- **Tanımlama ilişkisi**
  Bu ilişki tipler ve sınıflar arasında tek yönlü bir tanımlama için kullanılır. Örneğin, `Kişi` sınıfından türetilen `Öğrenci` sınıfı bu ilişkiye bir örnektir. `Öğrenci` sınıfı `Kişi` sınıfı ile tanımlanabilir, ancak bunun tersi doğru değildir. Bir başka deyişle, bir öğrenciler kümesi aynı zamanda bir kişiler kümesidir. Bu ilişki daha çok genel özellikleri kapsamak için kullanılır.
- **Sahip olma ilişkisi**
  İki sınıf, bir sınıf ve bir nesne veya iki nesne arasındaki bir ilişkiyi tanımlayan bu tip bir *taşıma* belirtir. Örneğin, bir `BölümBaşkanı` bir `Sekreter` ve üç `Asistan`'a sahiptir. Buradaki asistanlar `Öğrenci` sınıfının birer nesnesidir ve `BölümBaşkanı` sınıfının dışında başka bir sınıf tarafından doğrudan kullanılamazlar. `Kişi` sınıfı belirli bir kavramı tek başına ifade edebilmektedir. Ancak, kavramlar düzeyinde, `Öğrenci` sınıfı bu sınıfın sahibi konumundadır. Bu düzeyde `Kişi` sınıfı artık bir başka sınıfın özellikleriyle birleşik hale gelmiştir.
- **Kullanılma ilişkisi**
  Bu ilişki, bir sınıfın üye yordamının bir başka sınıfın bir nesnesini parametre olarak kullanması ile meydana gelir. Kullanılma ilişkisi bir sahip olma ilişkisi değildir, çünkü ilk sınıf ikincisini kendi yapısında kullanmamaktadır.
- **Yaratma ilişkisi**
  Bir sınıf ve bir nesne için geçerli olan yaratma ilişkisi, bir sınıfın yordamı yürütülürken bir başka sınıfın nesnesini yaratmak için onun yapıcısını çağırması şeklinde ortaya çıkar. Bu durum sahip olma ve kullanma ilişkilerinden daha farklıdır.

Burada belirtilen dört ilişki, varlıklar ve tipler arasında ortak olan mantıksal bağları ifade etmekte, bunların sınıf ve nesneler şeklinde gerçekleştirilmelerinin nasıl yapılabileceğini açıklamaktadır. Dilin özelliklerine göre bu ilişkiler dışında temel bir nitelik taşımayan başka ilişkiler de bulunabilir. Hangi tür ilişkinin kullanılacağının belirsiz olması durumunda, o uygulamanın özelliklerine göre en uygun sonucu veren ve programlama ilkelerine uyan ilişki tipi seçilir.

6.5.4. Nesneye Yönelik Tasarım Aşamaları

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

5.4.1. 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 arayüzünü, alabileceği iletileri ve aldığı bir iletiyle ne gibi bir işlem yapacağını anlatır. Genellikle iletinin adını ve herbir 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.
- Her işlem için birer yordam ve yordamlar için ayrıntılar tanımlanır (yordamsal tasarım).

5.4.2. 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 yaratım değişkenleri (instance variable) haline getirilir. Bunun için çözümleme sırasında belirlenmiş olan öznitelikler kullanılır; herbiri birer sınıf değişkeni, diğer bir deyişle sınıf üyesi (class member) olarak tanımlanır.
- Her nesne üzerinde uygulanacak işlemler birer sınıf yordamı haline dönüştürülürler. Bir kısım yordamlar yaratım değişkenlerine erişip değerlerini değiştirirken bir kısmı da sınıfa özgü işlemleri gerçekleştirirler.
- 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.

10.5.5. Nesneye Yönelimde Nitelik Ölçütleri

Nesneye yönelimin en büyük özelliği sınıflardır. Herbiri soyut bir tip tanımlayan bu sınıflardan nesneler üretilir. Sınıf yordamları, yani metotlar, nesnenin iç yapısına erişmek için kullanılır. Bir sınıfın sahip olduğu yordamların sayısı ve herbir 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 sıradüzeninin *derinliği* ve *genişliği*.

- **Kalıtımın derinlik ağacı**
  Kalıtım sıradüzeninin 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. Devralınan yordam sayısı da bazen bir ölçüt olarak kullanılır.
- **Türetim sayısı**
  Kalıtım sıradüzeni içinde bir sınıftan türetilen başka sınıfların toplam sayısıdır. Bu sayı, tasarım ve dolayısıyla da sistem üzerinde bir sınıfın potansiyel etkisinin göstergesidir. Türetim sayısının fazla olması temel sınıfın tekrar kullanımı iyi sağladığını gösterebilir. Öte yandan, türetim mantığının yanlış kullanımını da gösterebilir. Sayının çok düşük olması da temel sınıfın soyutlamasının iyi yapılmadığının bir göstergesidir. Bir sınıftan fazla sayıda alt sınıf türetiliyorsa o sınıfın tüm işlemlerinin ve davranışının çok iyi test edilmesi gereklidir. Bu da toplam test süresini artıran bir etkendir.

8.5.6. 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 bir *fikir* 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 çalışılmamalı, 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.

5. Hafta
1. Grup
Biyoloji

10.5. Nesneye Yönelik Tasarım

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.

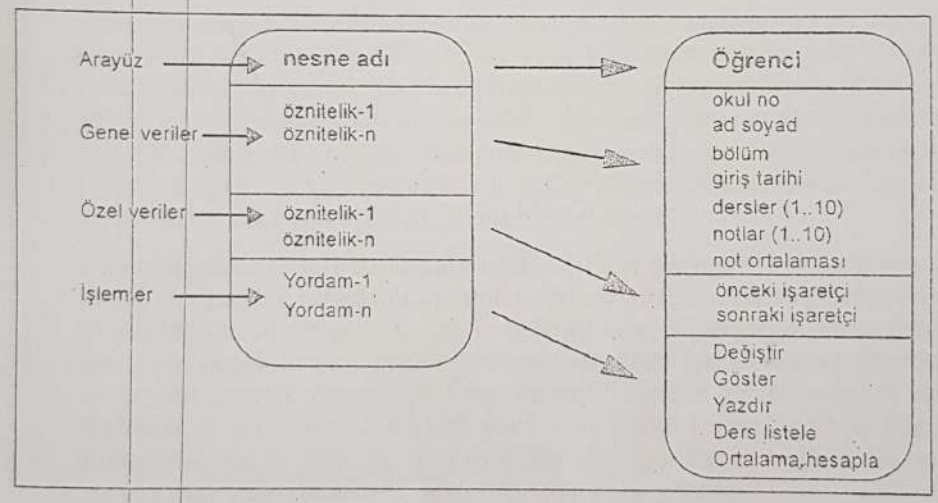
Nesneye yönelik çözümleme ile nesneye yönelik tasarım arasında kesin bir ayrım yapmak bazen mümkün değildir. Temel olarak, çözümleme bir sınıflandırma etkinliği olarak değerlendirilebilir. Bu aşamada problem çözümü için kullanılacak nesnelerin ait olacağı sınıfların belirlenmesi için çalışma yapılır. Tasarım aşamasında ise sınıflardan yaratılan 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 iç yapısında durum bilgileri ve arayüz olarak tanımlanan işlemler, yani yordamlar kümesi bulunur. Bu işlemlerle durum bilgisi üzerinde değişiklik yapılabilir. [4] [5]

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 yapılarını 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 yazılım alanına uyarlanmasında kullanılır. Her nesnenin bir arayüzü, bir veri yapısı ve bir dizi işlemi vardır. Şekil-6.11 de bir nesnenin ayrıntılı yapısı gösterilmekte ve bir örnek verilmektedir:



Şekil-6.11. Nesne yapısı gösterimi.

Nesneler birbirleriyle ilişkili veri ve işlemleri bir arada tutarak kapsama (encapsulation) 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 (information hiding) ilkesi desteklenmiş olur.

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

6.5.1. Temel Özellikler

Nesneye yönelik tasarım yönteminin en temel özellikleri şunlardır:

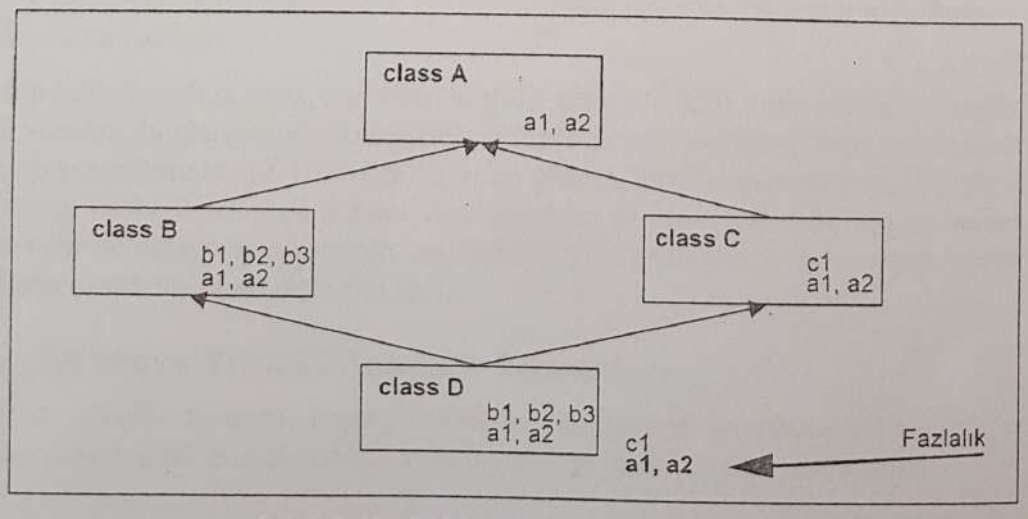
- **Kapsama (encapsulation)**
  Bir veri yapısı ve onun üzerinde işlem yapan bir grup yordamın iyi tanımlanmış bir arayüzü olan, erişimi ve kullanımı kolay bir yapı içine konmasına *kapsama* denir. Nesneye yönelik tasarımda bu yapı nesnedir ve gönderilen iletilerle arzu edilen işi yapması sağlanır. Her nesne belirli bir veri grubunu ve işlevleri kapsar; farklı veriler farklı nesnelerde yer alır. Sınıfların ve onlardan yaratılan 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 yaratılacak nesneler için bir şablon görevi görür.

- **Çokşekillilik (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.

- **Temsilcilik (delegation)**
  Bir nesnenin kullanımı *temsilcilik* ya da *kalıtım*la (inheritance) genişletebilir. Temsilcilik, bir işlemin o nesne tarafından yapılamaması durumunda işlemin o işi yapabilecek bir başka nesneye aktarılması esasına dayanmaktadır.

- **Kalıtım (inheritance)**
  Kalıtım, bir sıradüzen şeklinde, bir sınıftan veri ve işlemlerin tekrar kullanım amacıyla devralınmasıdır. 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. Bazı dil ve ortamlarda bu yöntem Şekil-6.12 de gösterilen olası bir soruna neden olabilir: [2]



Şekil-6.12. Çoklu kalıtımda olası sorun.

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.

10.5.2. Tipler ve Sınıflar

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 yaratması, 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 (solution 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 yaratabilir. Bu sorunlar özellikle alttip ilişkileri bulunduğunda üst düzeye çıkar. Bu nedenle uygulama alanındaki çözümlemelerin iyi yapılmasına, varlıkların ve ilişkilerin en doğru şekilde belirlenmesine çalışılmalıdır.