NESNEYE YÖNELİK TASARIM

Bir program tasarlanırken, gereksinimleri programın gerçekleştireceği işlemler açısından incelemek en alışılmış yaklaşımdır. Bu yaklaşımda, programın yapması gereken işlemler üzerinde yoğunlaşılır. Örneğin; bir öğrenci bilgi sisteminde işlemlere göre gereksinimler aşağıdaki şekilde belirlenebilir:

- Belirli bir dönem için açılan derslerin ekranda listelenmesi,
- Bir öğrencinin kayıt olacağı ders kodunun girilmesi,
- Öğrencinin derse kaydının onaylanması / reddedilmesi.

Bu gereksinimlere göre ayrıştırılan bir yazılımda oluşan birimler arasında veri paylaşımı yoğunluktadır. Yazılımın gerçekleştireceği işlemlere göre ayrıştırma başlangıçta kolaylık sağlasa da, veri paylaşımının çok olması gereksinimlerdeki küçük değişikliklerin bile yazılımda kapsamlı değişiklikler yapılmasını gerektirir. Bu durum ise hem sistemin geliştirilmesi hem de bakımı aşamalarında sorun yaratabilir.

Nesneye yönelik yaklaşımda ise bir problem, nesneler ve nesnelerin sorumlulukları açısından incelenir. Örneğin; bir öğrenci bilgi sisteminde *öğrenci*, *ders* ve *öğretim üyesi* nesneler, *derse kayıt olma* ve *ders verme* gibi işlemler ise nesnelerin sorumluluklarıdır. Bu yaklaşımda tasarım **yapılması gereken işlemler yerine, işlemlerin üzerinde çalışacağı nesneler üzerinde** yoğunlaşır. Böylece, yapısı daha güçlü ve değişikliklere daha uygun yazılımlar oluşturulması amaçlanmaktadır. Bu yaklaşımın önceki yaklaşımdan temel farkı, tasarımın odak noktasının işlemlerden nesnelere yani veriye değiştirilmesidir.

Nesneye yönelik yaklaşımdaki problem çözme yaklaşımı günlük hayattaki problemleri çözmek için kullanılan yöntemlere benzer. Bu, nesneye yönelik programlamanın temel fikirlerinin, bilgisayar programlama konusunda deneyimsiz olan kişiler tarafından daha kolay öğrenildiğine ilişkin görüşler öne sürülmesine neden olmuştur.

Nesneye yönelik tasarımın başarıya ulaşması için, gerçekleştirimde kullanılan programlama dilinin yaklaşımı desteklemesi gereklidir.

Nesne

Nesneye yönelik yazılım geliştirmede, tasarımda belirlenen her nesnenin gerçekleştirilmesi için gerekli tüm verileri ve işlemleri içeren bir modül oluşturulması gereklidir.

NESNEYE YÖNELİK PROGRAMLAMA

Büyük programların oluşturulması için gerekli iki kavram, dokuzuncu bölümde tartışıldığı gibi, modülerlik ve soyutlamadır. Nesneye yönelik programlamada modülerlik birimi, soyut bir veri tipi gerçekleştirimidir. Nesneye yönelik programlama ve "geleneksel" programlama arasındaki önemli bir fark, geleneksel programlamanın önceden tanımlanmış soyutlamalarla sınırlanmış olması, nesneye yönelik programlamada ise kullanıcı tanımlı soyutlamaların yer alabilmesidir.

Nesneye yönelik programlamada sınıflar, soyut veri tipleri tanımlamak için kullanılabilir. Buna ek olarak, varolan sınıflara ekler yapılarak, yeni nesne sınıfları tanımlanabilir.

Nesneye yönelik programlama, soyut veri tipi kavramına önemli fikirler de eklemektedir.

Temel Kavramlar

Nesneye yönelik programlamanın dayandığı temel kavramlar, aşağıdaki animasyonda özetlenmiştir.

Nesne	Veriler ve işlemler topluluğu.
Sınıf	Bir dizi nesne kümesinin tarifi.
Altsınıf	Bir sınıfın ek özellikler de içerebilen altkümesi.
Metod	Bir sınıfın işlemlerini gerçekleştiren kod.
Mesaj	Bir metodun çalıştırılmasını istemek için bir çağrım.
Kalıtım	Bir sınıftan yeni bir sınıf türetilmesi.
Çokyapılılık	Dinamik bağlama ile aynı mesajın farklı sınıflara ilişkin nesneler tarafından yanıtlanabilmesi ve farklı davranışlar gösterebilmesi.

Nesneye yönelik programlama, *kalıtım* ve *çokyapılılık* özellikleriyle yazılım geliştirmede yenider kullanımı da destekler.

Sınıflar ve Nesneler

Nesne:

Nesneye yönelik programlamada bir program, verileri ve ilgili işlemleri içeren nesneler üzerinde kurulur.

Bir **nesne**, üzerinde işlemlerin gerçekleştirilebileceği verilerle bir bütündür. Örneğin, *yığıt* bir *nesne*, yığıtın içerikleri de yığıt nesnesinin *verileri*dir. Yığıtın metodları, *yığıta veri eklenmesi*, *yığıttan veri çıkarılması* gibi yığıt nesnesinin sorumluluklarıdır.

Sınıf:

Nesneye yönelik programlamada **sınıflar**, soyut veri tipleridir. Sınıflar, özellikleri ve metodları açısından tanımlanır. Özellikler, sınıfın verilerini, metodlar ise sınıfın sorumluluklarını gösterir.

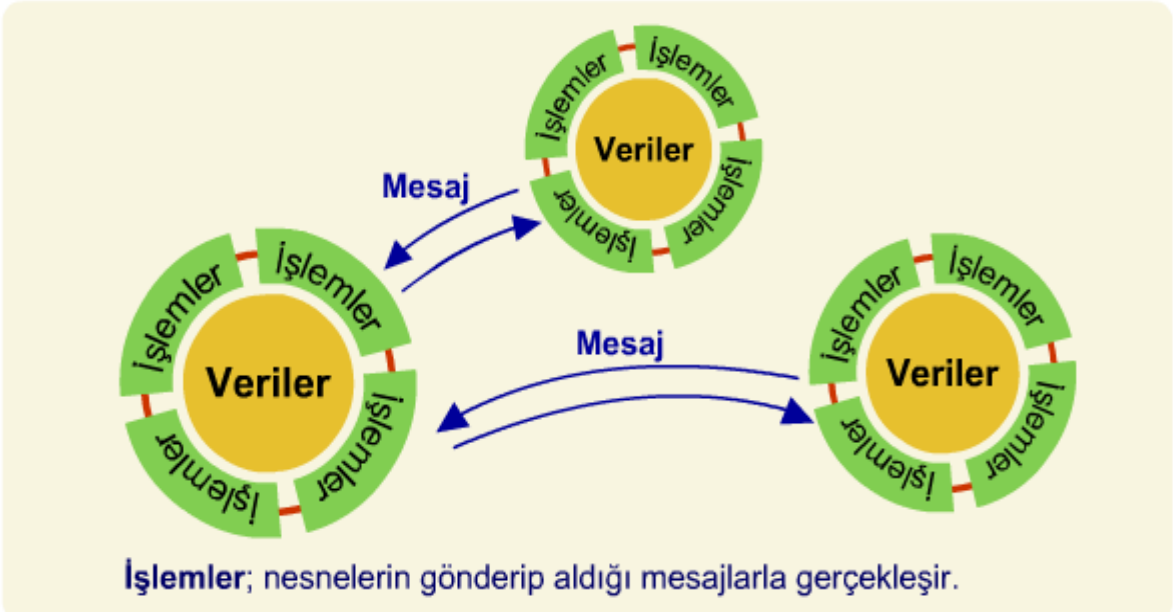
Nesneler ise sınıfların (kullanıcı tanımlı tiplerin) örnekleridir. Bu bağlamda, bir programda bir sınıf tanımını yeni bir tip tanımı, bir nesneyi de bir sınıf tipinde bir değişken tanımlanması olarak düşünebiliriz.

Sınıflar ve Nesneler

Sadece veri içeren bir nesne, işlevsel olarak C'deki bir *structure*'a veya bir Pascal *record*'a benzer olarak düşünülebilir.

Sınıf metodlarında yer alan metod kodları, bir sınıftan türetilen her nesne için yinelenmez. Sadece bir kez sınıf metodunda yer alır.

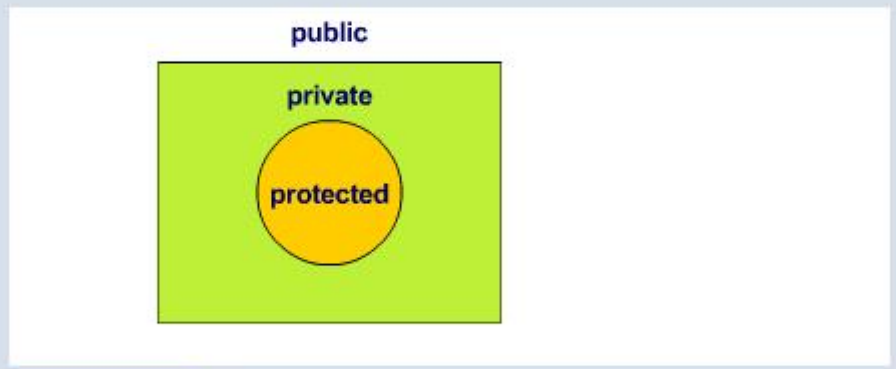
Nesneler birbirlerine mesaj yollayarak etkileşirler. Bir işlem, bir programdaki nesneler tarafından gönderilen ve alınan mesajlarla gerçekleştirilir. Yani bir işlem, davranışın nasıl gerçekleştirildiğinin ayrıntıları açısından değil, nesnelerin görülebilen davranışı açısından tanımlanır.



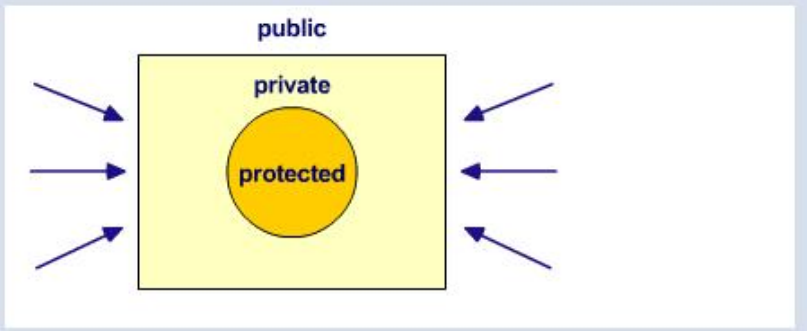
İşlemler; nesnelerin gönderip aldığı mesajlarla gerçekleşir.

Sınıf Veri ve Metodlarının Saklanması

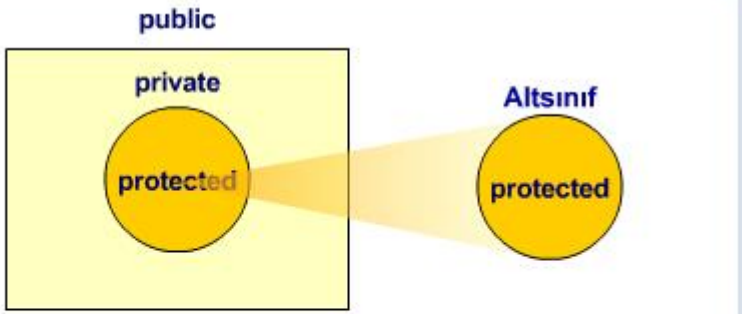
Çeşitli programlama dilleri, bir sınıfa ilişkin veri ve metodların diğer sınıflardan saklanması için olanaklar sunarlar. C++ ve Java'da üç tür tanımlayıcı ile değişkenlerin erişilebilirlikleri sınırlanabilir.



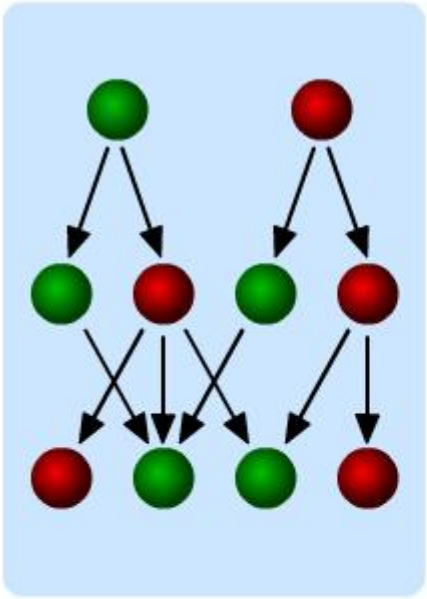
- **private** — Sadece o sınıf içinde erişim yapılabilir.
- **public**
- **protected**



- **private**
- **public** — Erişim için sınırlama yoktur.



- **private**
- **public**
- **protected** — Sadece o sınıf ve alt sınıfları içinde erişim tanımlıdır.



Kalıtım, varolan sınıfların tanımlarını kullanarak, yeni sınıflar oluşturmayı sağlar.

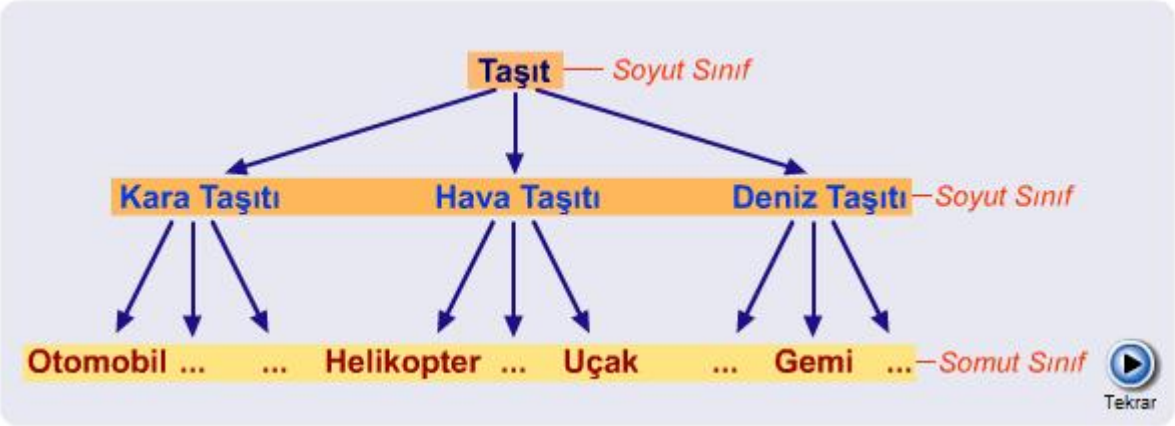
10.3.3. Kalıtım

Nesneye yönelik programlamanın temel bir kavramı olan **kalıtım** (*inheritance*), varolan sınıfların tanımlarını kullanarak, yeni sınıflar oluşturmayı sağlar. Kalıtım ile ilişkili sınıflar arasında kod paylaşımı sağlanarak kod büyüklüğü azaltılır.

Kalıtım ile Sınıflar ve Sınıf Hiyerarşileri Oluşturma

Kalıtım ile sınıflar ve sınıf hiyerarşileri oluşturmayı bir örnekle açıklayalım.

Ulaşım araçları için bir sınıf hiyerarşisi oluşturmak istiyorsak, ulaşım araçlarını *kara taşıtı, hava taşıtı* ve *deniz taşıtı* olarak gruplayabiliriz. Öncelikle, her üç gruptaki taşıtların ortak özelliklerini içeren *taşıt* sınıfı tanımlanabilir. Daha sonra hava taşıtlarının uçak, helikopter vb. şeklinde ayrılması gibi, her grup içinde alt sınıflar tanımlanabilir. Aşağıdaki animasyon, ulaşım araçları için oluşturulan sınıf hiyerarşisini göstermektedir.



Bu sınıf hiyerarşisinde bir alt sınıf, hiyerarşide daha yukarıda olan bir üst sınıftan özellikleri kalıtımla alabilir. Başlangıç sınıfı olan *Taşıt*, tüm taşıtların ortak özelliklerini içermektedir. *Taşıt* sınıfı, soyut bir kavramı göstermekte ve sadece tanım olarak bulunmaktadır. *Kara taşıtı*, *hava taşıtı* ve *deniz taşıtı*, *taşıt* sınıfının özelliklerini kalıtımla alan ve kendi özgün özelliklerini ekleyen yeni sınıflardır. Çeşitli somut nesnelerin ortak özelliklerini soyut düzeyde ayırarak, onların birçok kez tanımlanması yerine bir kez tanımlandığı sınıflar, soyut sınıfları oluştururlar.

Bir soyut sınıf, doğrudan örneklenemez, sadece alt sınıflar üretmek için kullanılabilir.

Örneğin, *taşıt* gibi bir soyut sınıf tanımlandıktan sonra, bu soyut sınıftan türetilen bir alt sınıf olan *kara taşıtı* gibi alt sınıf, sadece kendisini özel bir taşıt türü yapan özelliklerinin belirtilmesi ile tanımlanabilir.

Is-a ve *Has-a* İlişkileri

Nesneye yönelik tasarımda *is-a* ve *has-a* ilişkileri aranmalıdır. Bu ilişkiler, hem sınıflar arası kalıtım ilişkilerini hem de sınıfların özelliklerini belirlemekte yardımcı olur.

***Is-a*:**

Is-a ilişkisinde ilk eleman, ikinci elemanın özel bir şeklidir. Böylece daha soyut olan bir fikrin davranışının ve verisinin daha özel bir fikir ile bir altkümesi oluşturulur. Yani *is-a* ilişkisi, bir sınıf - alt sınıf hiyerarşisini gösterir.

***Has-a*:**

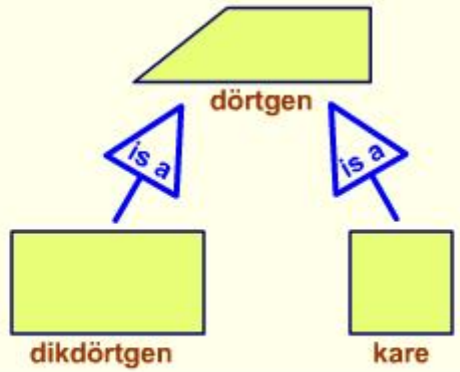
Has-a ilişkisi, ikinci kavramın birinci kavramın bileşeni olduğu zaman görülür. Bu nedenle *has-a* ilişkisi, bir sınıfta tutulacak veriyi yani bir sınıfın özelliklerini belirlemektedir. Bir önceki sayfada ulaşım araçları için oluşturulan sınıf hiyerarşisinde *otomobil is-a kara taşıtı* ve *otomobil has-a direksiyon* ilişkileri vardır.

Kalıtım ile türetilmiş bir sınıf, taban sınıfa işlevsellik eklemeye ek olarak, yeni özellikler tanımlayabilir ve üst sınıfta tanımlanmış işlemlerin bir kısmını yeniden tanımlayabilir. [Budd,98], kalıtımın 5 türlü amaçla kullanılabileceğini belirtmiştir.

- **Özelleşme İçin Kalıtım**
- Spesifikasyon İçin Kalıtım
- Genişletme İçin Kalıtım
- Kısıtlama İçin Kalıtım
- Birleştirme İçin Kalıtım

Kalıtımın en yaygın kullanımını göstermektedir. Bir alt sınıf, üst sınıfının tüm özelliklerini almakta ve üst sınıfın bir alttipi (subtype) olmaktadır.

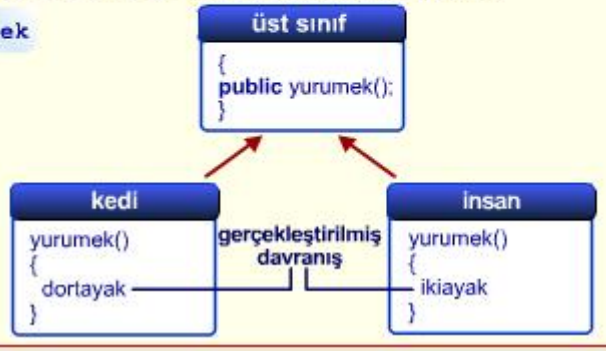
Örnek



Spesifikasyon İçin Kalıtım

Bu tür kalıtımda üst sınıf, gerçekleştirilmiş işlemler ve gerçekleştirilmesi altsınıfa ertelenmiş işlemlerden oluşur. Alt sınıf ise üstsınıfta tanımlanan ancak gerçekleştirilmeyen davranışı sağlar. Spesifikasyon için kalıtımda alt sınıflar, varolan bir tipin yenilenmesi ile değil, eksik ya da soyut bir spesifikasyonun gerçekleştirilmesi ile oluşurlar.

Örnek



- Özelleşme İçin Kalıtım
- Spesifikasyon İçin Kalıtım
- **Genişletme İçin Kalıtım**
- Kısıtlama İçin Kalıtım
- Birleştirme İçin Kalıtım

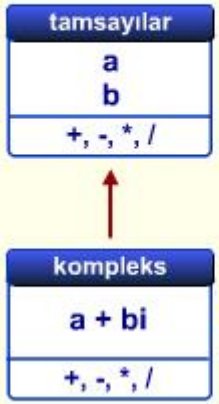
Bu tür kalıtımda bir alt sınıf, üst sınıftan aldığı özellikleri hiç değiştirmeden sadece üst sınıfa yeni davranış ekler. Genişletme ile kalıtımda, üst sınıfın işlevselliği aynı kaldığı için, alt sınıf, üst sınıfın bir alttipi olarak kalır.

Örnek

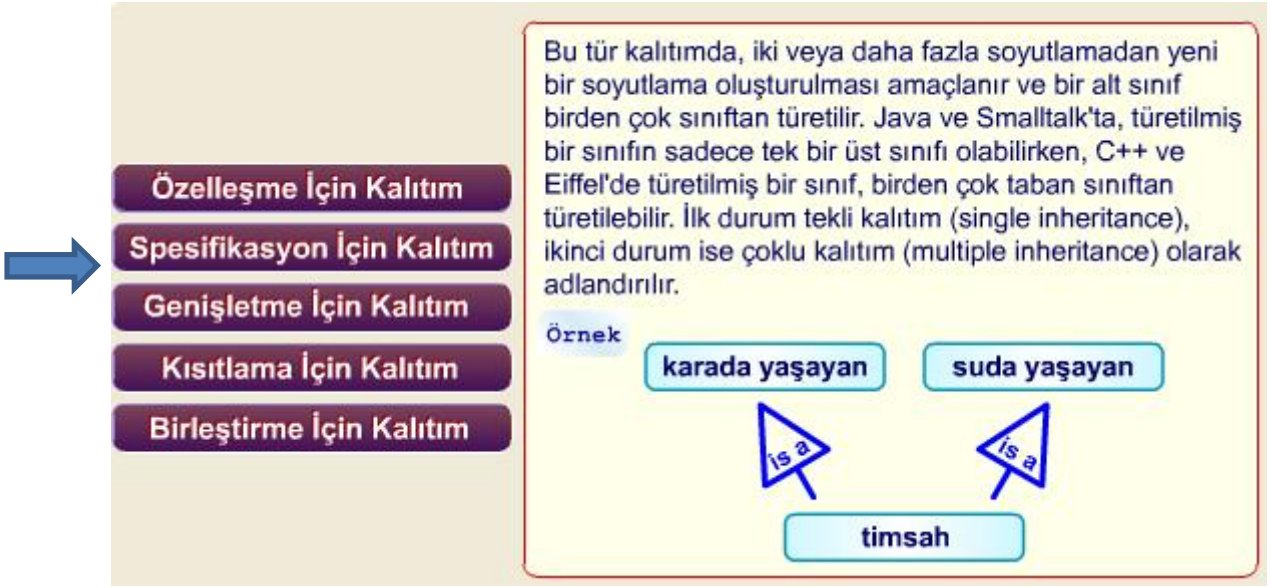
- Özelleşme İçin Kalıtım
- Spesifikasyon İçin Kalıtım
- Genişletme İçin Kalıtım
- **Kısıtlama İçin Kalıtım**
- Birleştirme İçin Kalıtım

Bu tür kalıtımda, alt sınıf, üst sınıfın davranışından daha az davranış gerçekleştirir. Örneğin, üst sınıfın bazı metodları, alt sınıfta *üzerine yazma* ile değiştirilir.

Örnek



Sayısal işlemciler kompleks sayılarda daha kısıtlı işlerler.

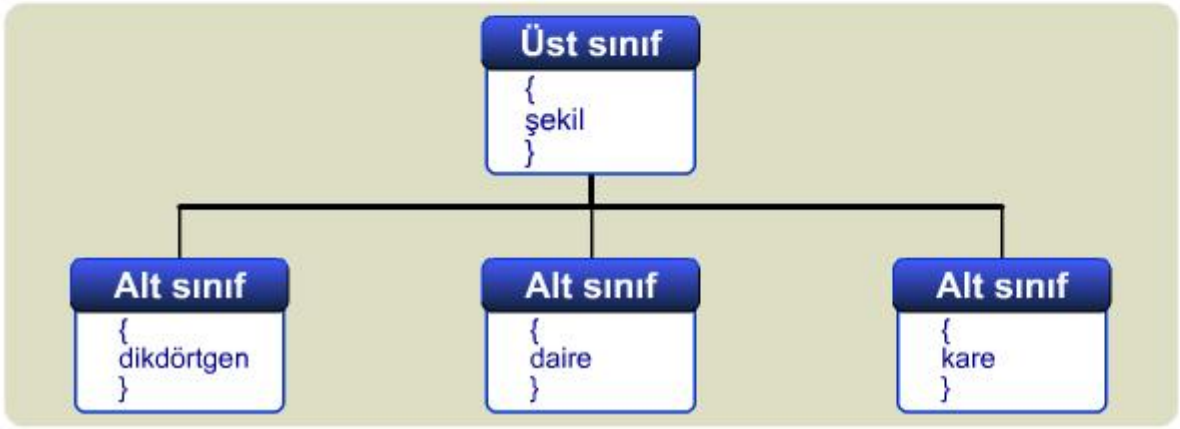

Özelleşme İçin Kalıtım
Spesifikasyon İçin Kalıtım
Genişletme İçin Kalıtım
Kısıtlama İçin Kalıtım
Birleştirme İçin Kalıtım
Bu tür kalıtımda, iki veya daha fazla soyutlamadan yeni bir soyutlama oluşturulması amaçlanır ve bir alt sınıf birden çok sınıftan türetilir. Java ve Smalltalk'ta, türetilmiş bir sınıfın sadece tek bir üst sınıfı olabilirken, C++ ve Eiffel'de türetilmiş bir sınıf, birden çok taban sınıftan türetilebilir. İlk durum tekli kalıtım (single inheritance), ikinci durum ise çoklu kalıtım (multiple inheritance) olarak adlandırılır.
Örnek
karada yaşayan
suda yaşayan
is a
is a
timsah

Çokyapılılık

Çokyapılılık *(polymorphism)*, *is-a* ilişkisinin bir sonucudur. Çokyapılılık, mesaj iletme ile etkileşim, kalıtım ve alttip kavramlarından yararlanır. Çokyapılılığın çeşitli türleri vardır. Bu ders kapsamında söz edilecek olan çokyapılılık, kalıtım ile oluşturulan sınıflar arasındaki çokyapılılıktır.

Metod Yükleme:

Aynı isme sahip bir çok metodun bulunması ve bu metodların aldıkları argümanlar ile ayırd edilmeleri metod yükleme *(overloading)* olarak adlandırılır.



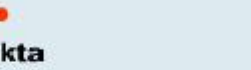
Çokyapılı bir değişken, bir sınıfın bir nesnesini veya o sınıftan türetilmiş sınıflardan bir nesneyi gösterebilir. Çokyapılı bir değişkenin hangi değeri göstereceği önceden bilinemediği için, gerekli bellek miktarı derleme zamanında bilinemez. Bu nedenle bu türdeki nesneler, yığıt bellek yerine yığın bellekte bulunur. Benzer şekilde bir çokyapılı değişkene gönderilen bir mesajın karşılığı, değişken tarafından gösterilen nesne türüne bağlı olarak çalışma zamanında belirlenir. İki ayrı nesnenin aynı mesaja farklı metodları gerçekleştirerek karşılık vermeleri ise inamik bağlama ile gerçekleşir.

Sınıf, Alt Sınıf ve Çokyapılı Değişken Tanımlama

Aşağıdaki şekilde Java'da *sekil* sınıfının tanımı ve *sekil* sınıfından türetilmiş *dikdortgen* ve *daire* alt sınıflarının tanımları verilmektedir.

Şekil Sınıfının Tanımı

```
class sekil{
protected int a;
protected int  b;
public sekil(int  aa, int bb)
{a =aa; b=bb;}
public string  kimlik()
{return "tanımsız sekil";}
}
```



Nokta tanımlanır.

Dikdörtgen Sınıfının Tanımı

```
class dikdortgen extends sekil{
protected int  kenar1;
public dikdortgen(int aa, int bb, int cc,int dd)
{super(aa, bb); kenar1= cc; kenar2=dd;}
public string kimlik()
{return  "seklin kimligi dikdortgen";}}
```

Noktadan başlayan iki kenar tanımlanır ve bundan dikdörtgen oluşturulur.

Daire Sınıfının Tanımı

```
class daire extends sekil{
protected int yaricap;
public daire(int aa, int bb, int ee)
{super(aa, bb); yaricap= ee;}
public string kimlik()
{return  "seklin kimligi daire";}}
```

Noktadan belli bir uzaklıkta bir yarıçap tanımlanır ve bundan daire çizilir.

Çokyapılılı Değişkenin Kullanımı:

Aşağıda *sekil* sınıfından türetilen alt sınıflar, *daire* ve *dikdortgen* sınıflarına ilişkin bir çokyapılı değişken olan *form*, *kimlikyaz* sınıfında *sekil* tipinde tanımlanmaktadır. *form* değişkeni gerektiğinde *daire* sınıfının örneklenmesi, gerektiğinde ise *dikdortgen* sınıfının örneklenmesi için kullanılabilir. Bununla bağlantılı olarak *kimlikyaz* ın çalışması sırasında *form* değişkeni hangi türde nesneyi gösteriyorsa, *form* nesnesine gönderilen *kimlik* mesajına karşılık olarak çalıştırılacak metod buna göre belirlenecektir. Bir mesajın alıcısı metodun kimliğinin çalışma zamanında belirlenmesi, değiştirilebilir bağlama ile mümkün olmaktadır.

Metod Yükleme

Bir metod ismi, iki veya daha fazla metod kodu ile ilişkilendirilmişse, metod yükleme (method overloading) gerçekleştirilmiş olur. Bu durumda çokyapılılık, metod ismi için uygulanmaktadır.

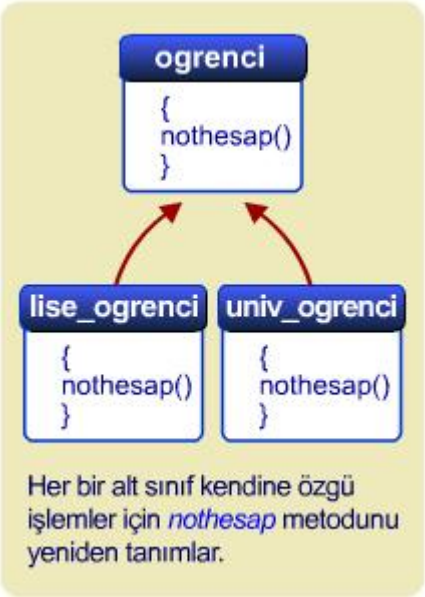
Metod yükleme, aşağıdaki iki durumda gerçekleşebilir.

1. Kalıtımla ilişkilenmemiş iki veya daha fazla sınıfta aynı metod ismini paylaşan metodlar olması durumunda veya
2. Aynı sınıf tanımı içinde aynı metod ismini paylaşan metodlar olması durumunda.

Metod yükleme ile aynı isimi paylaşan metodların, farklı veri tipleri üzerinde benzer işlemler gerçekleştiren metodlar olması alışılmış bir uygulamaysa da, bu metodların anlamsal olarak benzer olmaları şart değildir.

Metod yükleme, işlemci yükleme ile benzerlikler taşımaktadır. Aynı ismi paylaşan birden çok metod içinden hangisinin belirli bir mesaja karşılık vereceği, metodun parametreleri ile belirlenir. Örneğin; yukarıdaki şekilde görülen Java kodunda, *int* ve *double* veri tiplerindeki iki parametre için metod yükleme uygulanmış *kub* metodu görülmektedir.

Metod Üzerine Yazma



Metod Üzerine Yazma

özelleşme için kalıtımda tanımlandığı gibi türetilmiş bir sınıf, üst sınıfa ekler yapmaktan başka, üst sınıftan kalıtımla alınmış bir metodu yeniden tanımlayabilir. Bir sınıfta, üst sınıfındaki bir metodla aynı isimde bir metodun bulunması, **metod üzerine yazma** (*method overriding*) olarak nitelendirilir.

Metod üzerine yazma, bir sınıfta tanımlanmış bir metodun o sınıfın en az bir alt sınıfında aynı isimle yeniden tanımlanmasını gerektirir.

Alt sınıftaki tanımlama, üst sınıftaki metod tanımına erişmeyi engellemektedir. Bu durumda bir mesaja karşılık olarak hangi metodun kullanılacağı, çalışma zamanında mesajı alan nesneden başlayarak, üst sınıflarda devam eden arama ile belirlenir.

Örneğin yandaki şekilde görüldüğü gibi; *ogrenci* sınıfında tanımlanmış *nothesap* metodu, *ogrenci* sınıfının alt sınıfı olan *lise_ogrenci* ve *univ_ogrenci* sınıflarında, alt sınıflara özgü işlemlerin gerçekleştirilmesi için yeniden tanımlanabilir.

C++, Java gibi dillerde her iki metod, aynı parametrelere ve aynı dönüş tipine sahip olmalıdır.