SAKARYA ÜNİVERSİTESİ

Nesneye Dayalı Programlama

Hafta 9

Prof. Dr. Ümit KOCABIÇAK
Öğr. Gör. Özgür ÇİFTÇİ

9.1. Modelleme

Gerçek hayatta karşılaşılan sistemler genellikle karmaşıktır ve bu sistemlerin kişiler tarafından tüm yönleriyle bir defada kavranabilmesi için, sistemin parçalar halinde incelenebilmesini ve tasarlanabilmesini sağlayacak başka yöntemlerin kullanılması gerekir. Modelleme kavramı, insanlığın sistemlerin karmaşıklığı ile baş etmede kullandığı bilinen en eski ve en etkin yöntemdir.

Bir sistem modellenirken, sistemin özelliklerinden o an için ilgilenilen kısımlar öne çıkarılırken, diğerlerini arka planda saklayan soyut yapılar kurulur.

"Hiçbir model, tanımı gereği, sistemin tüm özelliklerini gerçeğe özdeş biçimde içermez. Her model aslının yalınlaştırılmış bir kopyasıdır ve tüm ayrıntıları içinde barındırması olanaklı değildir."

Yazılımcı tümüyle soyut yapılar üzerinde çalışır. Tasarım ve programla süresince kullandığı listeler, döngüler, nesneler, veritabanı tabloları gibi yapıların hiçbirisi elle tutulabilir somut nesneler değildir. Geliştirdiği sistemin, örneğin mimarların yaptığı gibi, bir maketinin yapılması ve masanın üzerine konarak incelenmesi de mümkün değildir. Bu nedenle yazılımcı soyut yapılarla çalışabilme ve soyut düşünebilme yeteneğini olabildiğince geliştirmek zorundadır.

Modeller, sistem karmaşıklığını yönetilir boyutlara indirgemenin yanı sıra, tasarımcılar arasında bir etkileşim biçimi olarak da hizmet veriler. Bir tasarımın temel özelliklerinin açıklanması ve çeşitli seçeneklerin değerlendirilmesi bir model üzerinde daha etkin olarak yapılabilir. Bunun yapılabilmesi için modellemede ortak bir gösterim biçiminin, bir başka deyişle ortak bir modelleme dilinin kullanılması zorunludur.

9.2. Unified Modelling Language (Birleşik Modelleme Dili)

UML (Unified Modelling Language) Türkçe olarak "**Birleşik Modelleme Dili**" şeklinde adlandırılabilir. UML bir programlama (ya da yazılım geliştirme) dili olmaktan ziyade iş sistemlerinin nasıl modellenebileceğini belirleyen ve açıklayan yöntemlerin bir araya toplanmış halidir. Daha çok yazılım geliştiriciler tarafından kullanılıyor olsa da UML ile yapılan modellemeler sadece yazılım projelerinde kullanılmak zorunda

değildir: Resmi UML dokumantasyonlarında UML 'in yazılımın yanısıra "İş Sistemleri Modellenmesi" 'nde de kullanılabilir.

Örneğin bir iş sistemin yapısını sade ve anlaşılır şekilde ortaya çıkarmak için Paket Diyagramı ("Package Diagram") kullanılabilir. Sınıf Diyagramı ("Class Diagram") vasıtası ile Nesnel Yönelimli Programlamada temel teşkil eden sınıflar net şekilde gösterilebilir ve böylece sağlanan ek görsellik ile yazılım tasarlamanın ilerleyen aşamalarında daha yüksek verimlilik sağlanabilir.

UML 'in belki de en kullanışlı diyebileceğimiz diyagram türü olan Etkinlik Diyagramları ("Activity Diagram") ile yazılım haline getirilmek istenen süreçler herkesin anlayabileceği şekilde görüntülenebilir. Bu açıdan faaliyet diyagramları hem yazılımcıya hem de yazılımı kullanacak olan kişilere net bir görüş sağlar.

Diyagramları oluşturmada altın bir kural vardır: Diyagramdaki elemanlar ve elemanları açıklayıcı yazılar ne kadar az olursa, diyagram o kadar açıklayıcı olur. Detayların tümünü bir diyagramda göstermeye çalışırsanız hem kendiniz hem de yazılımı kullanacak olanlar temel bakış açısını yitirebilirler.

1989-1994 yılları arasında yazılım dünyasında sistemleri modellemek için onlarca modelleme dili kullanılıyordu. Bu dilleri kullanan yöntemler belirli tip sistemler için tasarlanmıştı ve yazılım yaşam döngüsünü tam olarak tanımlyamıyorlardı. Bu yöntemlerden zaman içinde en çok tercih edilenleri Booch, OMT (Object Modelling Technology, Nesne Modelleme Teknolojisi) ve OOSE (Object Oriented Software Engineering, Nesne Yönelimli Yazılım Mühendisliği) oldu. Booch, Nesne Yönelimli Tasarım konusunda; OMT, Nesne Yönelimli Analiz gerektiren sistemlerde mükemmeldi. OOSE yöntemi ise sistemin genel yapısını anlamayı çok kolaylaştıran Kullanım Senaryosu (Use-Case) adlı güçlü bir tekniği içeriyordu.

1994 yılında Grady Booch (Booch yönteminin yaratıcısı) ve Jim Rumbaugh (OMT yönteminin yaratıcısı) Rational firmasının çatısı altında sahip oldukları iki yöntemi tekrar gözden geçirerek birleşik bir yöntem yaratmak için çalışmaya başladılar. Firmaya 1995 yılında Ivar Jacobson'ın (OOSE yönteminin yaratıcısı) da katılmasıyla, 3 Amigolar olarak bilinen grup, kendi yöntemlerinin güçlü yönlerini birleştirip eksiksiz bir sistem modelleme dili yaratmak üzere çalışmaya başladılar.

1996 yılında 3 Amigoların önderliğinde, Unified Modelling Language (UML) ismi verilen birleşik modelleme dilinin sonuçlandırılması amacıyla UML Partners isimli bir şirkeler birliği kuruldu. Microsoft, IBM, Oracle,

Hewlett-Packard, Texas Instruments gibi dev şirketleri de içeren birlik üyeleri UML'yi kendi modellemelerinde kullanmaya başladılar. UML 1.0, taslak olarak 1997 Ocak ayında OMG'ye (Object Management Group, nesne yönelimli sistemler için standartlar belirlemek amacıyla kurulan, kar gütmeyen bir organizasyon) tanıtıldı. Birkaç aylık bir çalışmanın ardından UML 1.1, Ağustos'ta OMG'ye sunuldu ve Kasım ayında kabul edilerek, belirli bir kişi ya da şirkete ait olmayan bir modelleme dili yazılım dünyasına kazandırılmış oldu.

UML 1997'den bu yana 1.1, 1.3, 1.4 ve 1.5 gibi versiyonların ardından 2005'te çıkan 2.0 versiyonu ile birçok yönden geliştirilmiş, dildeki eksikler tamamlanmış ve hatalar ortadan kaldırılmıştır. Son versiyonu UML 2.1.2, 2007 Kasım ayında çıkmıştır.

Grafiksel bir dil olan UML, modelleme için değişik diyagramlar kullanır. Diyagramlar, bir sistem modelini kısmen tarif eden grafiklerdir. UML diyagramları bir sistem modelini 3 farklı açıdan ele alırlar. Modelin,

- **İşlevsel gereksinimler açısında,** kullanıcının bakış açısından sistemin gereksinimleri vurgulanır. Kullanım Senaryosu (Use-Case) diyagramını içerir.
- **Statik yapısal açısında,** nesneler, nesnelere ait özellikler ve ilişkiler kullanılarak sistemin statik yapısı incelenir. Sınıf (Class) ve Birleşik Yapı (Composite Structure) diyagramlarını içerir.
- **Dinamik davranış açısında,** nesneler arası ortak çalışmalar ve nesnelerin durumlarındaki değişiklikler gösterilerek sistemin dinamik davranışı incelenir. Sıralama (Sequence), Faaliyet (Activity) ve Durum (Statechart) diyagramlarını içerir.

UML 2.0, 3 ana bölümde 13 çeşit diyagram içerir. Yapısal diyagramlarda modellenen sistemde nelerin var olması gerektiği vurgulanır. Davranış diyagramlarında modellenen sistemde nelerin meydana gelmesi gerektiğini belirtir. Davranış diyagramlarının bir alt kümesi olan Etkileşim diyagramlarında ise modellenen sistemdeki elemanlar arasındaki veri ve komut akışı gösterilir.

Yapısal Diyagramlar

1. **Sınıf (Class) diyagramı**, sistemin yapısını anlatmak için sistemde var olan sınıfları, sınıfların özelliklerini ve sınıflar arası ilişkileri kullanır. Nesneye yönelik sistemleri modellemede kullanılan en yaygın diyagramdır.
2. **Nesne (Object) diyagramı**, modellenen sistemin yapısının belirli bir andaki bütün ya da kısmi görünüşü tarif edilir.

3. **Bileşen (Component) diyagramı**, bir yazılım sisteminin hangi tür bileşenlere ayrıldığını ve bu bileşenlerin nasıl birbiriyle ilişkili olduğunu betimler. Bir bileşen genellikle bir veya birden fazla sınıf, arayüz ve iletişime karşılık gelir.
4. **Paket (Package) diyagramı**, bir sistemin hangi mantıksal gruplara bölündüğünü ve bu gruplar arasındaki bağımlılıkları betimler.
5. **Dağılım (Deployment) diyagramı**, sistemde kullanılan donanımları, bu donanımların içinde yer alan bileşenleri ve bu bileşenlerin arasındaki bağlantıları gösterir.
6. **Birleşik Yapı (Composite Structure) diyagramı**, bir sınıfın iç yapısını ve bu yapının mümkün kıldığı iletişimleri tarif eder.

Davranış Diyagramları

1. **Kullanım Senaryosu (Use-Case) diyagramı**, modellenen sistem tarafından sağlanan işlevselliği sistemde yer alan aktörleri, aktörlerin sahip olduğu kullanım senaryolarını ve bu senaryolar arasındaki bağımlılıkları göstererek açıklar.
2. **Durum (Statechart) diyagramı**, bilgisayar programlarından iş süreçlerine kadar birçok sistemi tarif eden standartlaşmış bir gösterimdir. Durumlar, geçişler, olaylar ve faaliyetler gösterilir.
3. **Faaliyet (Activity) diyagramı**, modellenen sistemdeki iş akışını adım adım gösterir. Faaliyet diyagramı kapsamlı bir komut akışını tarif eder. Faaliyetler arası akışı gösteren Durum diyagramıdır.

Etkileşim Diyagramları

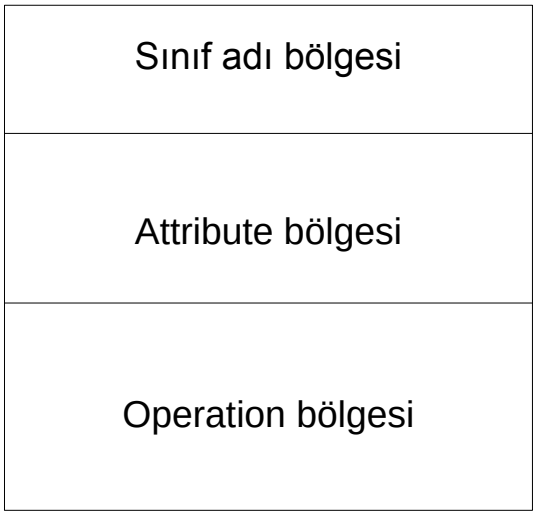
1. **Sıralama (Sequence) diyagramı**, nesnelerin birbiriyle nasıl iletişim sağladıklarını sıralı iletiler şeklinde gösterir. Ayrıca nesnelerin yaşam süreleri de gösterilir.
2. **İletişim (Communication) diyagramı**, nesneler ve parçalar arasındaki etkileşimi sıralı iletiler olarak gösterir. Sınıf, Sıralama ve Kullanım Senaryoları diyagramlarındaki bilgileri kullanarak sistemin hem statik yapısını hem de dinamik davranışını gösterir.
3. **Etkileşime Bakış (Interaction Overview) diyagramı**, farklı etkileşim diyagramları kullanarak, bunlar arasındaki komut akışını gösterir. Bir başka deyişle, elemanları etkileşim diyagramları olan faaliyet diyagramlarıdır.
4. **Zaman Akış (Timing) diyagramı**, odağın zaman kısıtlamaları olduğu etkileşim diyagramıdır.

9.2.1. UML'nin Yararları

- ✓ Takım çalışmasında yardımcı olur, UML standartlaşmış uluslararası bir dildir ve bu dili bilen herkes diyagramlardan aynı şeyleri anlar. Müşteri ve teknik sorumlular diyagramlar üzerinden rahatça iletişim kurabilirler. Ekibinizde yer alan çalışma arkadaşlarınızla uyumlu bir şekilde çalışabilirsiniz ve ekibe yeni giren bir çalışan da projeye rahatlıkla dahil edilebilir.
- ✓ Kodlamayı kolaylaştırır, UML ile uygulamanızın tasarımı analiz aşamasında yapıldığı için, modellemeniz bittikten hemen sonra kod yazmaya başlayabilirsiniz.
- ✓ Hataları en aza indirir, UML ile bütün sistem tasarlandığı için sistemde hata çıkma olasılığı azdır. Çıkan hataları düzeltmek ise çok daha kolaydır.
- ✓ Tekrar kullanılabilir bileşenleriniz artar, UML ile tüm sistem ve sistemin bileşenleri daha baştan belirlendiği için, o bölümler tekrar tekrar yazılmayacaktır.
- ✓ Program kararlılığı artar, UML ile ayrıntılı gereksinim analizleri yapıldıktan sonra senaryolar belirlenir. Senaryoların baştan belirli olması programınızı daha kararlı hale getirmenizde size yardımcı olur.

9.3. Class UML Diyagramları

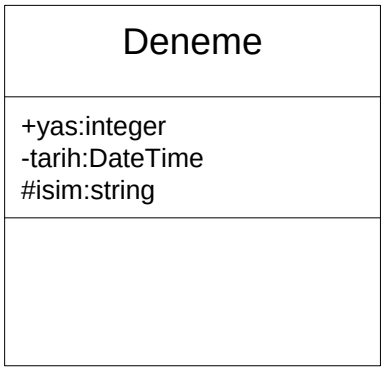
- Bir sınıf, ortak yapısı, ortak davranışları, ortak ilişkileri ve ortak semantiği bulunan nesneler koleksiyonudur.
- UML' de üç farklı alanı olan bir dikdörtgen şeklinde gösterilirler.
- Bu üç bölümden ilki sınıf ismini, ikinci kısım yapısını(attributes), ve üçüncü bölüm ise davranışını(operations) gösterir.
- Sınıfların gösteriminde sadece sınıf ismini, yapısını ya da davranışlarını veya her üçünü de birden görebilirsiniz.
- Sınıflar isimlendirilirken bir standardizasyon olması amacıyla bütün isimler büyük harf ile başlarlar.



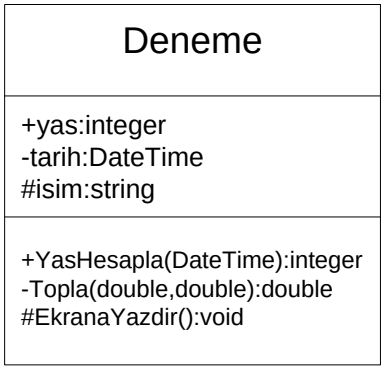
Sınıf Adı Bölgesi: Sınıfların ve Interfacelerin adının yazıldığı bölgedir. Sınıf abstract ise sınıf adı italik yazılır. Interface olması durumunda ise bu sınıfın arayüz olduğu belirtebilmek amacıyla <<interface>> şeklinde stereotypes olarak belirtilir.

Sınıf Abstract Sınıf Arayüz

Attribute Bölgesi: Bu bölgede sınıf üyelerinden alanlar yer alır. Öncelikle erişim belirtice yazılır (Bakınız 9.3.3 Erişim Belirteçleri). Üyenin adından sonra : (iki nokta üst üste) yazılır ve tipi belirtilir.



Operation Bölgesi: Bu bölgede sınıf üyelerinden metotlar yer alır. Öncelikle erişim belirtice yazılır (Bakınız 9.3.3 Erişim Belirteçleri). Üyenin adından sonra varsa metot parametrelerinin türleri parantez içersinde yazılır. : (iki nokta üst üste) yazılır ve geri dönüş tipi belirtilir.



9.3.1. Kalıtım

- Sınıflar arası türetme işlemi ucu açık üçgen ve düz bir çizgiyle yapılır.
- Ana sınıf(parent Class)
- Alt sınıf(sub Class)
- Türetme sınıflar arası ilişki açısından türetmenin “is kind of”(bir çeşit) ilişkisinin olduğu görülür(Bird is a kind of Animal)gibi

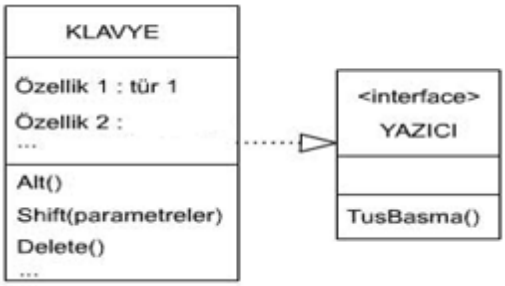
9.3.2. Arayüz (interface) kavramı

- Herhangi bir sınıfla ilişkisi olmayan ve standart bazı işlemleri yerine getiren sınıfa benzer yapılara arayüz denir. Arayüzlerin özellikleri yoktur. Sadece bazı işlemleri yerine getirmek için başka sınıflar tarafından kullanılırlar.
- Kesik çizgilerle ve çizginin ucunda boş bir üçgen olacak şekilde gösterilir.

Gerçekleme(Realization) Bir sınıfın bir arayüze erişerek, arayüzün fonksiyonlarını gerçekleştirmesine denir.

Ara yüz kavramı , nesnelerin davranışlarını belirleyen kurallar bütünü olarak düşünülebilir.

- Ara yüzler kuralları belirlerler ancak ,bu kuralların nasıl uygulanacağına karışmazlar.
- Bir sınıf , ilgili ara yüzün yordamlarını gerçekleyerek , ara yüzün belirlediği kurallara uymuş olur.



9.3.3. Görünürlük(Visibility)

Public: diğer sınıflar erişebilir. UML'de **+** sembolü ile gösterilir.

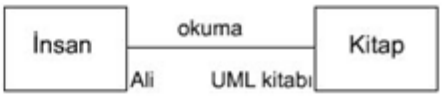
Protected: aynı paketteki () diğer sınıflar ve bütün alt sınıflar () tarafında erişilebilir. UML'de **#** sembolü ile gösterilir.

Package: aynı paketteki () diğer sınıflar tarafında erişilebilir. UML'de **~** sembolü ile gösterilir.

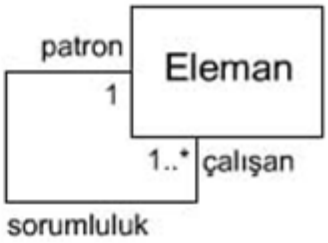
Private: yalnızca içinde bulunduğu sınıf tarafından erişilebilir (diğer sınıflar erişemezler). UML'de - sembolü ile gösterilir.

9.3.4. Sınıflar Arası İlişkiler

- Bire-bir
- Bire-çok
- Bire –bir veya daha çok
- Bire –sıfır veya bir
- Bire-sınırlı aralık
- Bire-n (*)



Reflexive(Kendine dönen)ilişki:Bir sınıfın sistemde birden fazla rolu vardır.



9.3.4.1. Sınıflar arası ilişki(Assocation)

- Birliktelik nesneler arası uzun süreli ilişkidir.
- Gerçek hayatta , örneğin , insanlar ve arabaları bir ilişki oluştururlar. Bu ilişki bir birlikteliktir, bir yerden başka bir yere gitme olayında , ne kullanıcı arabasız düşünülebilir nede araba kullanıcısız düşünülebilir.

İki tür birliktelik vardır:

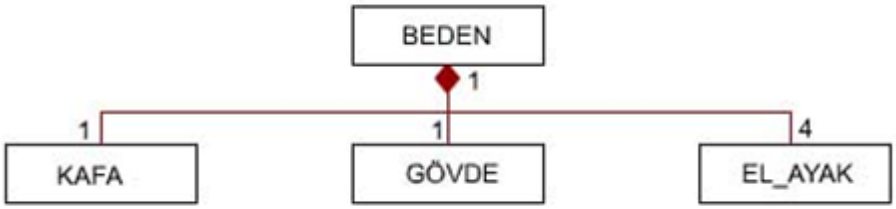
1. İçerme (Aggregation)
2. Kompozisyon - Oluşum (Composition)

9.3.4.2. İçerme(Aggregation)

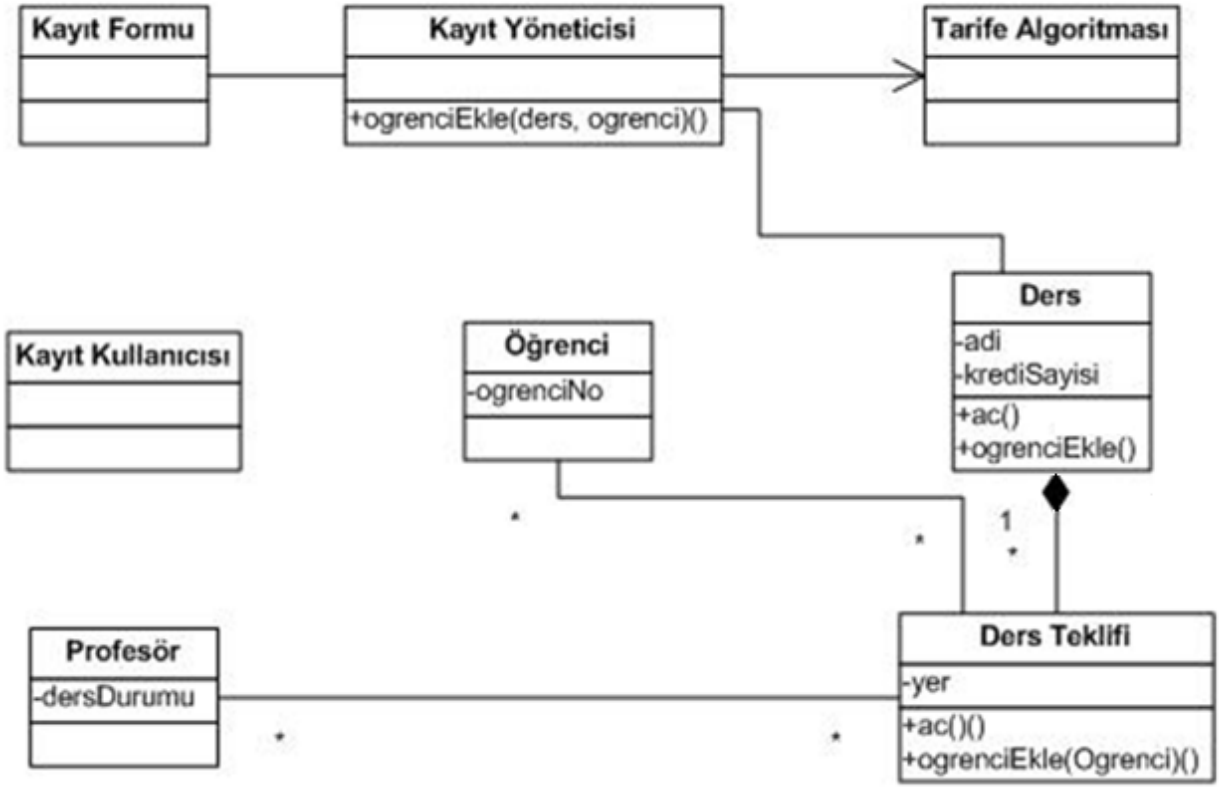
- Bütün parça yukarıda olacak şekilde ve bütün parçanın uçuna içi boş elmas yerleştirilecek şekilde gösteririz.
- İçi boş elmas ile gösterilen ilişkilerde herbir parça ayrı bir sınıftır ve tekbaşlarına anlam ifade eder. Parça bütün arasında sıkı bir ilişki yoktur.

9.3.4.3. Kompozisyon - Oluşum (Composition)

- Bazı durumlarda bütün nesne yaratıldığında parçalarının da yaratılmasını isteriz.
- Bu ilişki daha sıkıdır.


BEDEN
1
1
1
4
KAFA
GÖVDE
EL_AYAK

Örnek:


Kayıt Formu
Kayıt Yöneticisi
+ogrenciEkle(ders, ogrenci)()
Tarife Algoritması
Ders
-adi
-krediSayisi
+ac()
+ogrenciEkle()
Kayıt Kullanıcısı
Öğrenci
-ogrenciNo
1
*
*
*
*
Profesör
-dersDurumu
Ders Teklifi
-yer
+ac()()
+ogrenciEkle(Ogrenci)()

9.4. Use Case UML Diyagramları

Use Case sistemin kullanıcılara sunacağı bir hizmetin senaryo şeklinde anlatımıdır. Sistem gereksinimleri Use Case diyagramları ile belirtilir

Yazılım geliştirme için gerekli değildir, fakat gereksinimler ve nesnesel modeller arasında en önemli bağlantıdır.

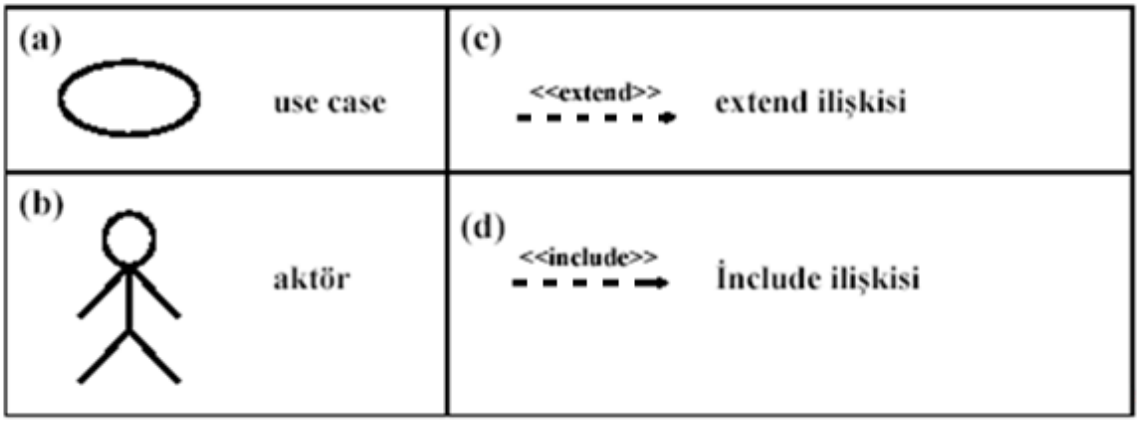
Bu diyagramlar ilk olarak aktörlere bakılarak oluşturulurlar. Aktör sistemin sunduğu hizmetleri kullanan bir kişi veya başka bir sistemdir. Aktörler sistemin dışında olan ve sistemle etkileşimde bulunması olası bir şahıs veya farklı bir sistem olarak belirtilirler. İlk olarak sorulacak soru sistemle kim iletişimde bulunacak sorusudur.

Diyagramı hazırlayanlar Sistem Analisti'dir.

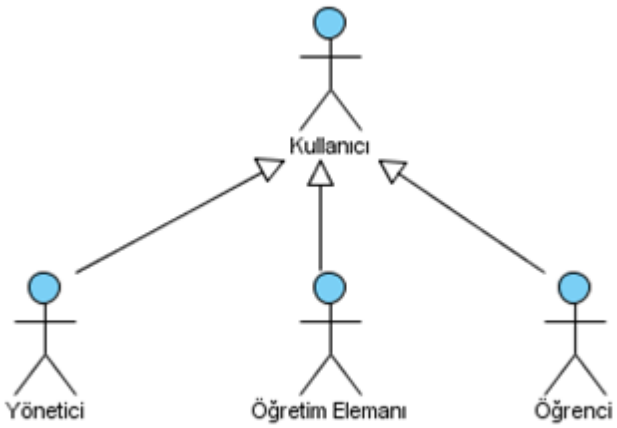
Diyagramı kullananlar ise Müşteri, Proje Yöneticisi, Tasarımcı, Veritabanı analisti, tasarımcı

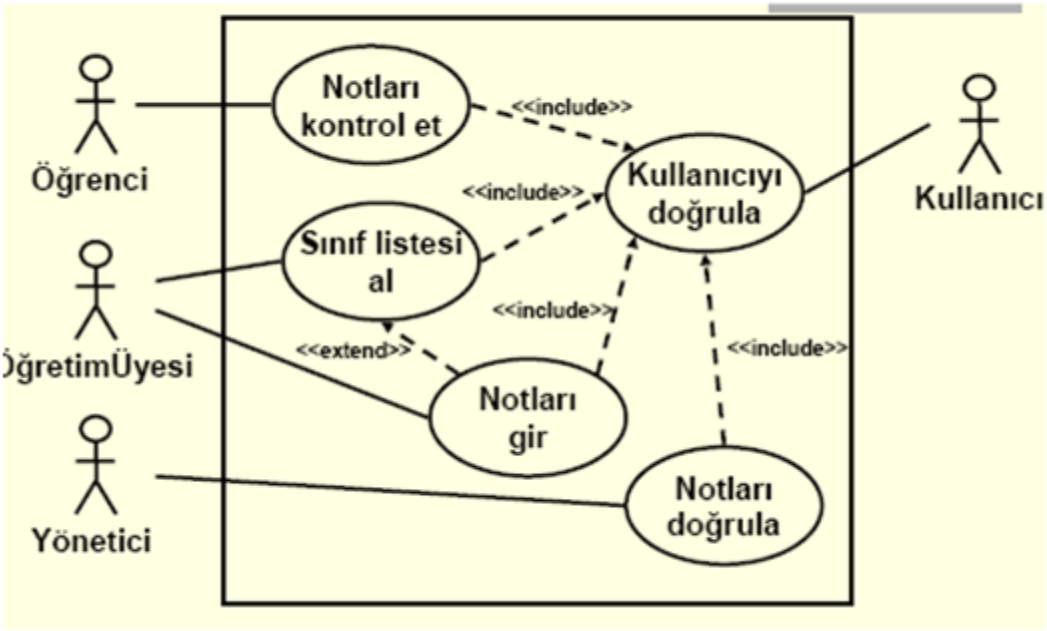
- Ders seçim modeli için aktör olarak şu hususlardan bahsedebiliriz. Kayıt memuru, öğrenci, profesör ve de dış bir ödeme sistemini ele alabiliriz. (aktör illaki bir insan olmak zorunda değildir farklı bir sitem de aktör olabilir)
- Kullanıcı durumu (use case) sistemdeki aktör tarafından sistem ile bir etkileşim esnasında gerçekleştirilecek işlemler dizisinin referansıdır. Kullanım durumları birer yazılım modülü değillerdir. Sistem ile etkileşimde bulunacak aktörlerin sistem için önemini gösterirler.
- Sistemdeki kullanım durumlarını bulmak için o sisteme kısaca aktörlerin sistemi ne amaçla kullanmak istediklerini sormak yeterli olacaktır.

Sembol	Açıklama
use case	Belli bir hedefe ulaşmak için kullanılabilecek tüm alternatif senaryolar
aktör	Sistemle etkileşen kişilerin canlandırdıkları roller ve dış sistemler
extend	Bir use case'in kullanıcısının seçimine bağlı olarak çalıştırdığı diğer bir use case'i vurgular
include	Bir use case'nin her zaman çalıştırdığı diğer bir use case'i vurgular



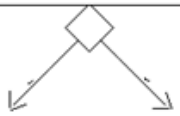
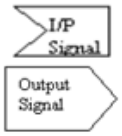
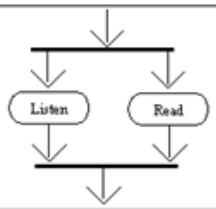
Örnek:

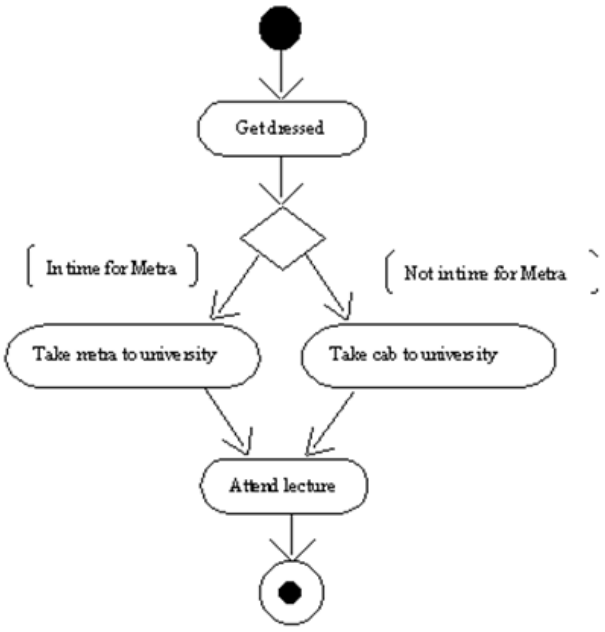



Notları kontrol et
<<include>>
Kullanıcıyı doğrula
Kullanıcı
Öğrenci
<<include>>
Sınıf listesi al
<<include>>
<<include>>
<<extend>>
ÖğretimÜyesi
Notları gir
Notları doğrula
Yönetici

9.5. Activity (Etkinlik) UML Diyagramları

- Etkinlik diyagramları sistemin kontrol akışını göstermektedir.
- Yuvarlatılmış dörtgen şeklindeki bütün parçalar birer aktiviteyi göstermektedir. Aktiviteler tipik olarak uygulama durumlarıdırlar.
- Her bir durum işlevinin tamamlanması ile diğer bir duruma otomatik olarak geçiş yapar.
- Aktivite diyagramı akış diyagramına oldukça benzemektedir. Aktivite diyagramları tipik olarak analiz aşamasının ilk safhalarında iş akışını belirtmek amacıyla kullanılırlar.

Sembol	Açıklama
	Başlangıç aktivite sembolü
	Aktivite
	Karar sembolü
	Sinyaller 2 çeşittir. Üstteki giriş sinyali, alttaki ise çıkış sinyalidir.
	Eşzamanlı aktiviteler. Paralel kalı çizgi ile gösterilir.
	Bitiş aktivitesi

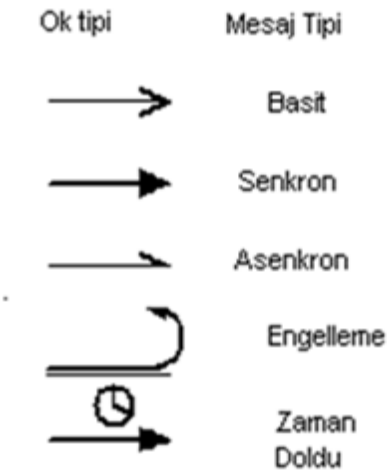


9.6. Sequence UML Diyagramları

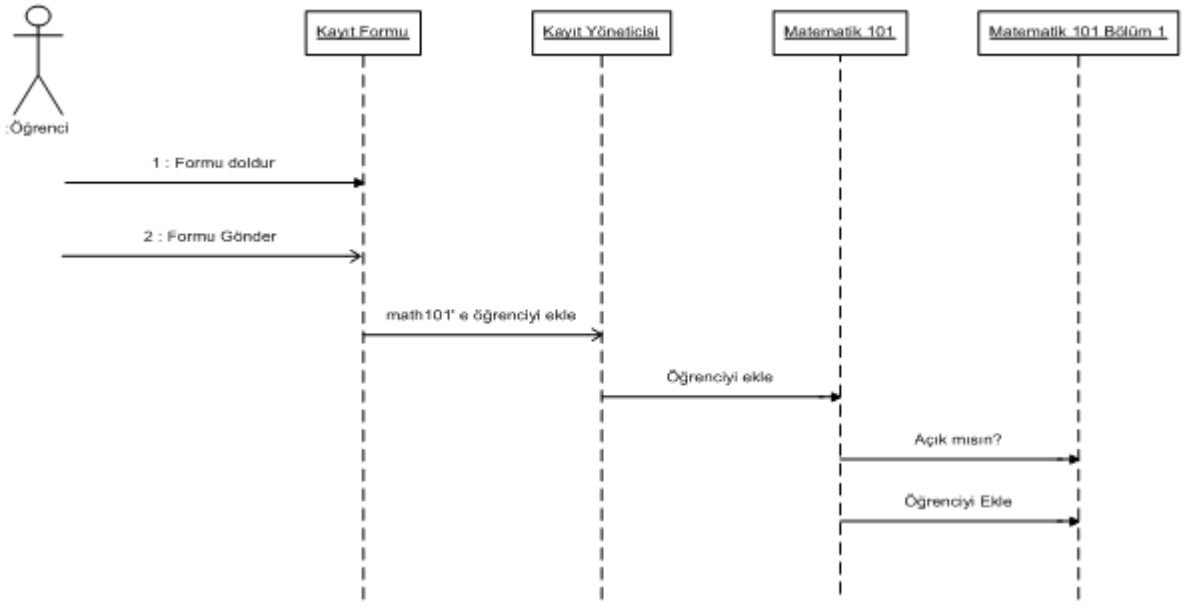
- Sequence (Sıralama etkileşim) diyagramları belli bir zaman diliminde sıralanmış nesne etkileşimlerini gösterirler. Nesne ve bu nesnelerin birbirleri ile etkileşimini gösterebilmek için olayların zaman içerisinde akışını belirtmemiz yeterli olacaktır. Hangi zaman diliminde hangi işlemin yapılması gerektiği ve bu işlem yapılırken ne gibi değişiklikler sistemde oluştuğu belirtilmelidir.
- Ardıl etkileşim diyagramları , belirli bir zaman dilimine yayılmış nesne etkileşimlerini gösteren bir tür etkileşim diyagramıdır.
- Nesneler ile , zamanla nesneler arasında taşınan , fonksiyoneliteyi gösteren mesajlara odaklanmıştır.
- Nesneler birbirleri ile mesajlaşma yapan yapılardır.
- Mesajlar

* **Senkron** mesajlar , yanıtı beklerler.

*** Asenkron** mesajlar , sinyal olarak düşünülebilir. Yanıt beklemezler. Çağrı yapan nesneyi bloke etmezler.



Örnek:



KAYNAKLAR

1)Head First Object-Oriented Analysis & Design

2) www.csharpnedir.com

3) Y.Doç.Dr. Feza BUZLUCA ders notları

4) Doç.Dr. Erdoğan DOĞDU ders notları

5) tr.wikipedia.org

6) ebergi.com