Uyguluma (Problem) Uzayının Modellenmesi (*Application Domain Model*)

Yazılım mühendislerinin sadece yazılım konusunda uzman olmaları yeterli değildir; geliştirecekleri yazılım ile ilgili dünyayı da tanımaları, anlamaları gerekir.

Analiz aşamasının temel amacı uygulama alanını (*application domain*) **tanımaktır**.

Eğer yazılım ekibi daha önce bu alanda çalıştıysa analiz aşaması daha kısa sürebilir.

Bu aşamada, çözülmek istenen probleme ilişkin (gerçek) dünyanın; doğru, özlü, **anlaşılır**, sınanabilir bir modeli oluşturulur.

Uygulama uzayındaki (*application domain*) modelleme, problemin çözümlenmesi (*analysis*), yani **anlaşılması** aşamasını oluşturur.

Amaç problemin çözülmesi değil anlaşılmasıdır.

"Ne?" sorusunun cevabı aranır. "Nasıl?" sorusunun cevabı tasarımın konusudur.

Müşterinin tarif ettiği sistemde hangi varlıklar var? Bu varlıklar arasındaki ilişkiler nelerdir?

Bu aşamada kendi yorumlarımıza değil müşterinin isteklerine yoğunlaşırız.

Uyguluma Modeli (devam)

İsteklerin çözümlenmesinde (modellenmesinde) oluşturulan kullanım senaryoları (*use case*) nesneye dayalı özellikler taşımaz.

Uygulama uzayının modellenmesinde ise nesneye dayalı yöntem kullanılacaktır.

Uygulama uzayının modelinde gerçek dünyayı oluşturan kavramsal sınıflar ve nesneler yer alır. Bu model oluşturulurken yazılım nesneleri (çözüm) düşünülmez.

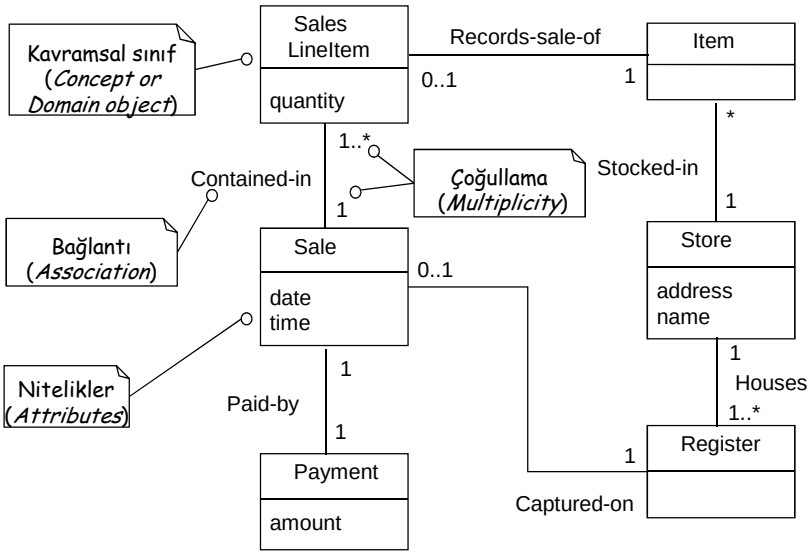
Uygulama uzayının modeli aşağıdaki bilgileri içeren sınıf diyagramları ile belirtilir:

1. Gerçek dünyadaki kavramsal sınıflar ve nesneler (Yazılım sınıfları/nesneleri değil)
2. Sınıflar arasındaki ilişkiler (bağlantılar) (*association*),
3. Sınıfların nitelikleri (*attributes*)

Oluşturulan analiz modeli iki amaca hizmet eder:

- Yazılım ile oluşturacağımız sistemi anlamak
- Tasarım aşamasına geçildiğinde sorumluluk atayacak sınıfları belirlerken kaynak olmak

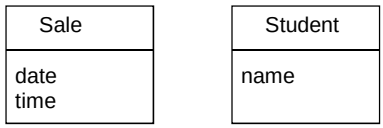
Örnek bir Kavramsal Sınıf Diyagramı



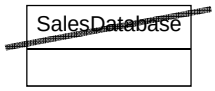
Uygulama modeli gerçek dünya kavramlarını gösterir, yazılım sınıflarını değil.

Gerçek dünya kavramları:

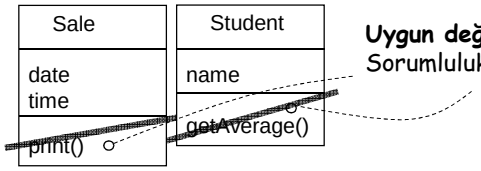
Uygulama modeli için uygundur.



Yazılım ile ilgili kavramlar uygulama modelinde yer almazlar.



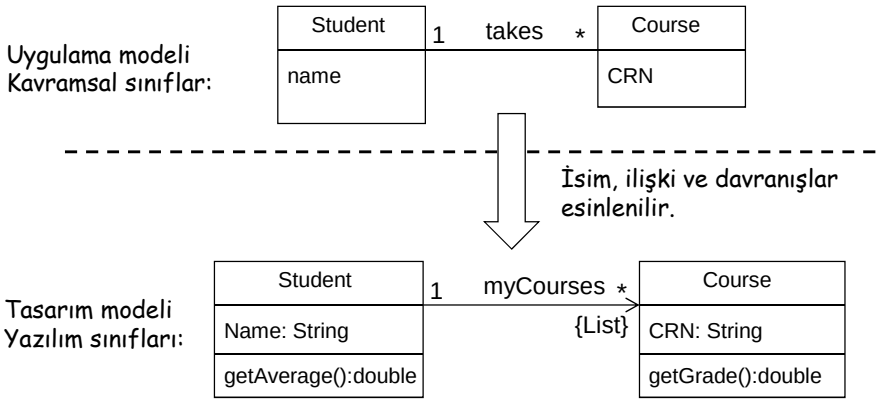
Uygun değil. Yazılım öğesi. Tasarım modelinde yer alabilir.



Uygun değil. Yazılım sınıfları.
Sorumluluklar tasarım aşamasında atanacaktır.

Gerçek dünya ile yazılım arasındaki yakınlık
(*Lower representational gap between real-world and software*)

Tasarım aşamasında yazılım sınıflarının isimleri ve sorumlulukları belirlenirken uygulama modelinden esinlenilecektir.



Uygulama (analiz) modeli ile yazılım (tasarım) modeli tamamen aynı olmazlar ancak benzerdirler.

Kavramsal Sınıfların Belirlenmesi (Bulunması)

Kavramsal sınıflar gerçek dünyadaki somut ve soyut varlıklara karşı düşen sınıflardır.

En çok kullanılan yöntemler:

1. Kavramsal sınıfların kategori listesinden yararlanma
2. Kullanım senaryolarındaki isimlerden (isim tamlamalarından) yararlanmak
3. Var olan (eski) modellerin güncellenmesi. Yayımlanmış modeller bulunmaktadır.

Bu yöntemler birlikte de kullanılabilir.

1. Kavramsal Sınıfların Kategorileri:

Deneyimlerden yararlanılarak uygulama uzayındaki sınıfların hangi kategorilerde yer aldığı belirlenmiş ve bir liste yapılmıştır. Modellenecek olan uygulama incelenerek bu listedeki kategorilere uyan unsurlar belirlenmeye çalışılır.

Bazı kategoriler ve örnek kavramsal sınıflar:

- Fiziksel ve somut nesneler : Ürün, terminal, uçak
- İşlem (*Transaction*): Satış, Ödeme, rezervasyon (Eğer kendi nitelikleri varsa)
- İşlem Kalemleri (*Transaction line itemes*): Satış kalemi
- İşlem veya hizmetle ilgili ürün ve kalemler: Ürün, uçuş, yemek
- İşlemin ve Hizmetin Yeri: Dükkan, havalimanı, sınıf
- Kişilerin ve kuruluşların rolleri: Müşteri, yolcu, öğretmen, havayolu şirketi, dükkan
- İşlem kayıtlarının tutulduğu yerler: Satış defteri, uçuş planı

Kavramsal sınıfların kategori listesinin devamı:

- Nesnelerin tanıtıcı bilgileri (*description*): Ürün tanımı, uçuş tanımı.
- Kataloglar (Nesnelerin tanıtıcı bilgilerini tutarlar): Ürün kataloğu, uçuş kataloğu.
- Başka nesneler içerebilen taşıyıcılar (*container*): Dükkan, kutu, uçak.
- Taşıyıcılarda yer alan nesneler: Ürün, yolcu.
- Finans elemanları: Çek, kart.

2. Kavramsal Sınıfların İsimler Yardımıyla Belirlenmesi:

Kavramsal sınıfların belirlenmesinde kullanılan ikinci yöntem ise senaryolarda ve problemin tanımında yer alan isim ve isim tamlamalarından yararlanılmasıdır.

Kullanım senaryolarında yer alan tüm isimler ve isim tamlamaları işaretlenir.

Çoğunlukla ilk aşamada gereğinden fazla sınıf elde edilir. Daha sonra uygulanan eleme yöntemi ile gereksiz sınıflar ayıklanır.

Yinelemeli (*iterative*) yazılım geliştirmede her yinelemede (*iteration*) senaryoların sadece bir kısmı üzerinde çalışılıyor olabilir.

Bu örnekte de sadece senaryo grubunun doğal akış kısmı ele alınmış ve burada isim ve isim tamlamaları işaretlenmiştir.

Her ismin bir defa işaretlenmesi yeterlidir.

Örnek:

Main Success Scenario (or Basic Flow):

1. **Customer** arrives at a **POS checkout** with **goods** and/or **services** to purchase.
2. **Cashier** starts a new **sale**.
3. Cashier enters **item identifier**.
4. System records **sale line item** and presents **item description**, **price**, and running **total**. Price calculated from a set of price rules.

Cashier repeats steps 3-4 *until indicates done.*

5. System presents total with **taxes** calculated.
6. Cashier tells Customer the total, and asks for **payment**.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external **Accounting** system (for accounting and **commissions**) and **Inventory** system (to update inventory).
9. System presents **receipt**.
10. Customer leaves with receipt and goods (if any).

Extensions:

7a. Paying by cash:

1. Cashier enters the cash **amount tendered**.
2. System presents the **balance due**.

.....

Gereksiz sınıfların elenmesi:

• Fazlalık Sınıflar (*Redundant classes*): Aynı unsuru ifade eden iki sınıftan daha tanımlayıcı olan alınır. Kişi – müşteri: müşteri

• İlgisiz sınıflar (*Irrelevant classes*): Problemin çözümü ile ilgisi olmayan ya da çözümlemenin o iterasyonunda ilgilenilmeyen unsurlar elenir. *Kredi kartı*

• Belirsiz sınıflar (*Vague classes*): Sınırları iyi çizilmemiş, fazla geniş (kaba) tanımı olan sınıflar elenir. Bunlar çoğunlukla başka sınıfların parçalarıdır ya da birden fazla sınıftan oluşurlar. *Muhasebe sistemi*

• Nitelikler (*Attributes*): Nitelikler de isimler ile ifade edildiğinden sınıflar ile karıştırılabilirler. Kendi başlarına varlıkları anlamlı olmayan sadece başka sınıfların niteliklerini oluşturan unsurlar olası sınıflar listesinden silinirler. *Miktar*

• İşlemler (*Operations*): Sadece başka nesneler üzerinde uygulanan işlemler sınıf olamaz. Kendi nitelikleri olan ve başka olaylardan etkilenen işlemler sınıftır. Örneğin "ödeme" bir işlem gibi görünmekte ancak kendine ait özellikleri (miktar, para birimi, tarih vb.) olduğundan tek başına bir sınıftır.

• Gerçekleme unsurları (*Implementation constructs*): Gerçekleme aşamasını ilgilendiren unsurlar uygulama uzayının çözümlenmesinde yer almazlar.

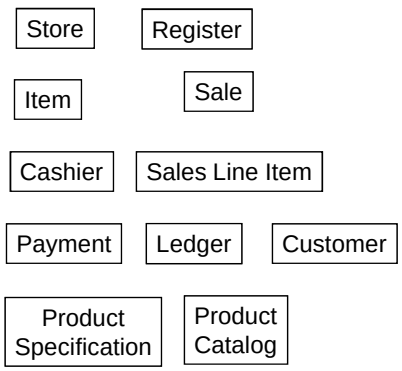
Bu elemeden geçenler uygulama uzayındaki unsurlara karşı gelen sınıflar olacaktır. Her sınıfın anlamını açıklayan bir **sözlük** hazırlanması yararlı olacaktır.

Problem Uzayı Modelini Oluşturmada Haritacı (*Mapmaker*) Yöntemi:

Gerçek dünyanın haritasını oluşturmada yararlanılan kavramlar burada da kullanılabilir:

1. Fiziksel dünyadaki gerçek isimleri kullanın.
2. İlgisiz özellikleri dışlayın.
3. Var olmayan şeyleri eklemeyin.

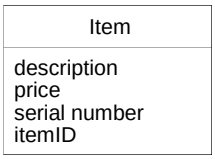
Örnek POS sistemine ait kavramsal sınıflar



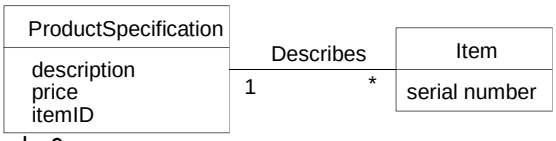
- Register
- Item
- Store
- Sale
- Sales Line Item
- Cashier
- Customer
- Ledger
- Payment
- Product Catalog
- Product Specification

Betimleme (*specification or description*) sınıflarına gerek duyulması

Dükkanda satılan malzemelerle ilgili fiyat, tip numarası gibi bilgilerin o malzemeye ilişkin nesnelerde tutulması öngörülebilir.
Ancak dükkandaki o cinsten tüm malzeme satıldığında (nesneler yok olduğunda) o malzemeyle ilgili bilgiler de kaybolur.
Böyle sistemlerde bir nesneyi betimleyen bilgilerin başka sınıflarda tutulması gerekir.



Kötü



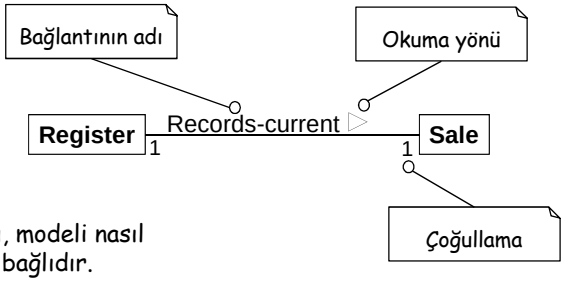
İyi

Ne zaman gerek duyulur?
- Bir malzeme ya da hizmetle ilgili bilgilere o malzeme ya da hizmetin var olup olmamasından bağımsız olarak erişmek gerekli ise,
- Bir nesnenin yok edilmesi bilgi kaybına neden oluyorsa,
- Gereksiz bilgi tekrarını önlüyorsa.

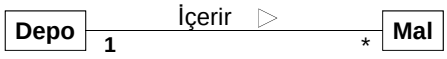
Kavramsal Sınıflar Arasındaki Bağlantıların Belirlenmesi

Uygulama uzayının modeli oluşturulurken ilk aşamada kavramsal sınıflar bulunur.
İkinci aşamada ise bu sınıflar arasındaki bağlantılar (*associations*) belirlenir.

Bağlantılara ilişkin UML notasyonu:



Çoğullama sayısı, modele nasıl kullanacağımıza bağlıdır.

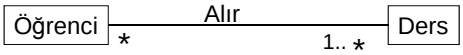


veya
0..1

Bir malın tükenip depoda bulunmaması bizi ilgilendiriyorsa Depo ucuna ait çoğullama sayısı 0..1 olabilir. Aksi durumda bu sayının 1 olması yazılım açısından daha uygundur.

Çoğullama (*multipicity*) Sayısı

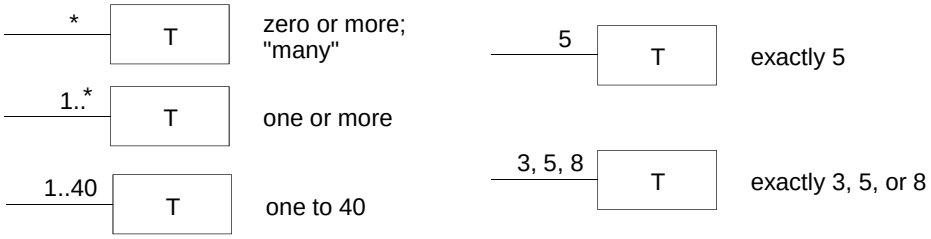
Çoğullama sayısı o sınıftan kaç tane **nesnenin**, diğer sınıftan kaç tane **nesne** ile belli bir anda geçerli olarak ilişkilendirilebileceğini gösterir.



Soldan sağa: Bir öğrenci en az bir (veya daha fazla) ders alır.

Sağdan sola: Bir ders hiçbir öğrenci tarafından alınmayabilir veya aynı anda birden fazla öğrenci tarafından alınabilir.

Çoğullama sayıları analiz aşamasında müşterinin isteklerine göre belirlenir.



Bağlantıların Bulunması

Bağlantı (*association*) iki sınıf arasında o sistemde belli bir süre geçerli olan ilişkidir.

Bu konuda da değişik yöntemler kullanılmaktadır:

1. Yaygın Bağlantılar Listesi (*Common Associations List*)
2. Kullanım senaryolarındaki fiillerden yararlanmak.

Yaygın Bağlantılar Listesi (*Common Associations List*):

- physical containment: Register - Store
- logical containment: Line Item - Sale
- log/record relation: Sale - Register
- usage relation: Cashier – Register, Manager - Register
- communication relation: Customer - Cashier
- description: Product Spec. - Item
- membership relation: Cashier - Store
- ownership relation: Store - Register
- transactional relation: Customer - Payment, Payment – Sale

Bu yöntemde, iki kavram arasındaki ilişkiye ait bilgi belli bir süre sistem tarafından bilinmesi gerekiyorsa (*need-to-know association*) göz önüne alınır. İki kavram arasındaki ilişki tasarlanan sistem açısından gerekli değilse dikkate alınmaz. Örneğin Satış – Müdür bağlantısı gerekli olmayabilir.

Diğer yöntemde ise kullanım senaryolarındaki fiiller dikkate alınarak tüm olası bağlantılar (ilişkiler) listelenir.
Aşağıdaki maddeler dikkate alınarak gereksiz bağlantılar ayıklanır:

- Önceki aşamada elenmiş olan sınıflar arasındaki bağlantılar gereksizdir.
- Sistemin amacı açısından gereksiz/ilgisiz olan bağlantılar.
- Gerçekleme aşamasını ilgilendiren bağlantılar.
- Faaliyet (*Actions*): Örnek ATM kredi kartı kabul eder. Bu cümle bir ilişkiyi değil müşteri ile ATM arasındaki etkileşimleri içerir.
- Üçlü (*ternary*) bağlantılar ikili bağlantılar şeklinde ifade edilmelidir.

"Memur hesap ile ilgili işlemleri girer." ifadesini
"Memur işlemleri girer.", "İşlemler hesapla ilgilidir." şeklinde ikiye bölmek gerekir.

- Başka bağlantılardan türetilebilen (*derived*) ilişkiler elenebilir.

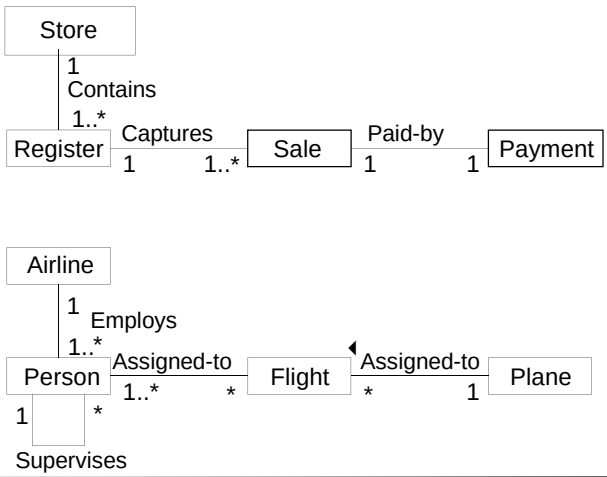
Örnek: Banka konsorsiyumu ATM'leri paylaşır ilişkisi aşağıdaki iki ilişkiden türetilebilir.
Banka konsorsiyumu merkezi bilgisayara sahiptir. Merkezi bilgisayar ATM'leri kontrol eder.
Genellikle UML diyagramında iki sınıf arasında birden fazla yol varsa, çözümlemede fazlalık (türetilebilir) ilişkiler olduğu düşünülebilir.
Her türetilebilir ilişkinin elenmesi doğru değildir. Sistem açısından önemli olanlar kalabilir. Örneğin toplumda Baba – kardeş yerine Amca ilişkisi kullanılmaktadır.

Öneriler

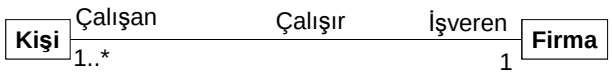
- Sistemin tasarımını ilgilendiren bağlantılara (*need-to-know*) yoğunlaşmak gerekir.
- Bağlantılara doğru isimler atanmalı. Tip adı – fiil – tip adı

Bağlantının adı sadece bir ya da bir kaç işlemin adı değil bir ilişkinin ifadesi olmalıdır.

Örnekler:



• Gerekli olan yerlere **rol** adları da yazılmalıdır. Roller bağlantının iki ucunu oluştururlar.

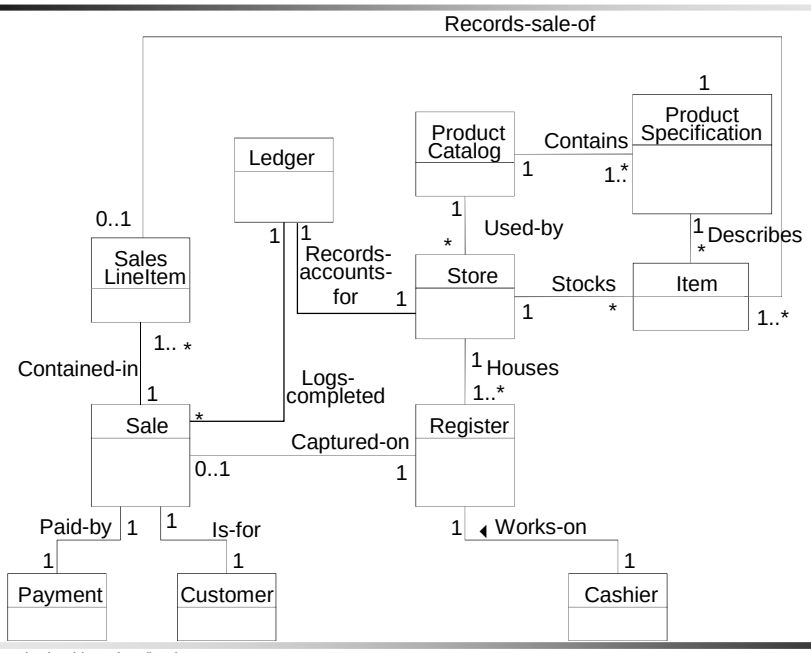


• Sahip olma, oluşma (*aggregation, compositon*) ilişkilerini ayrıntılı olarak belirlemek bu aşamada gereksizdir.

• Bu aşamada bazı bağlantıların unutulması sistem tasarımını çok etkilemez.

• Kavramsal sınıfların doğru olarak belirlenmesi bağlantılardan daha önemlidir.

• Gereğinden fazla bağlantı modelin anlaşırlılığını azaltabilir.



Kavramsal Sınıfların Niteliklerinin (*Attributes*) Belirlenmesi

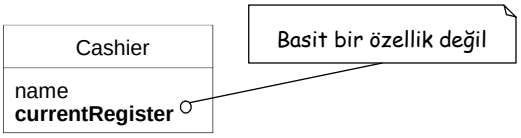
Bir sınıfın nitelikleri, o sınıftan nesneler yaratıldığında nesnelere özgü değerler alabilen verilerdir.

Bu aşamada amaç, üzerinde çalışılan senaryoları ilgilendiren nitelikler bulmaktır.

Nitelikler genellikle basit veri tipleri (int, string, bool, date) ifade edilirler.

Eğer bu aşamada nitelik olarak düşünülen veri daha karmaşık bir tipte ise o verinin bir bağlantı ya da ayrı bir sınıf olma olasılığı yüksektir.

Kötü



İyi

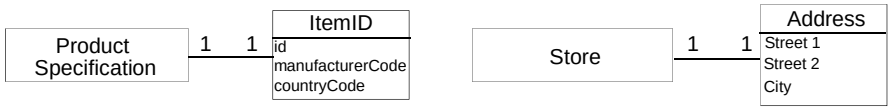
Bazı durumlarda nitelikler basit veri tiplerinden oluşmazlar.

Eğer bir nitelikte aşağıdaki özellikler varsa başka bir sınıf ile ifade edilmesi doğru olur.

- Birden fazla alandan oluşuyorsa: Telefon no, ad/soyad
- Üzerinde işlem yapılıyorsa: Kredi kartı numarası onaylanması
- Kendi nitelikleri varsa: Fiyatın geçerlilik tarihi olabilir.

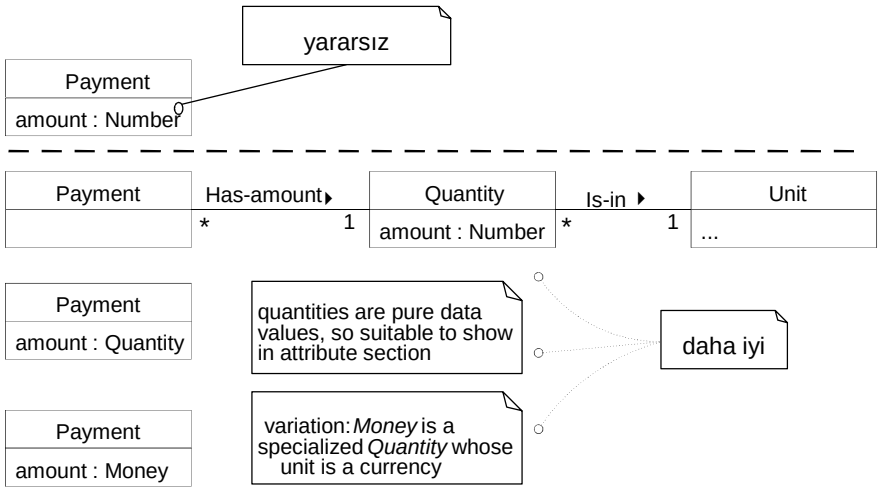
Bu tip nitelikler iki farklı şekilde de gösterilebilir.

OK

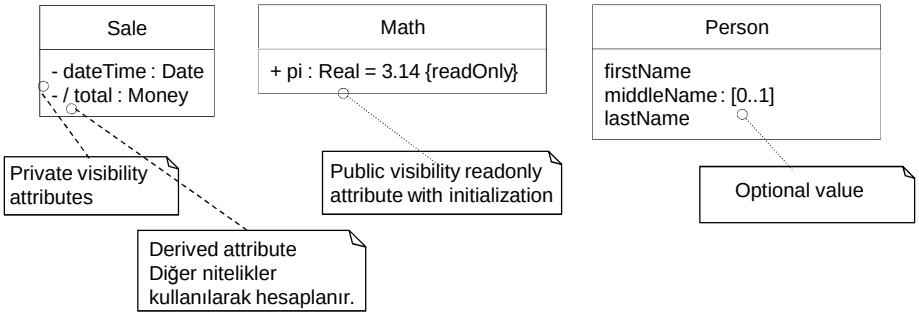


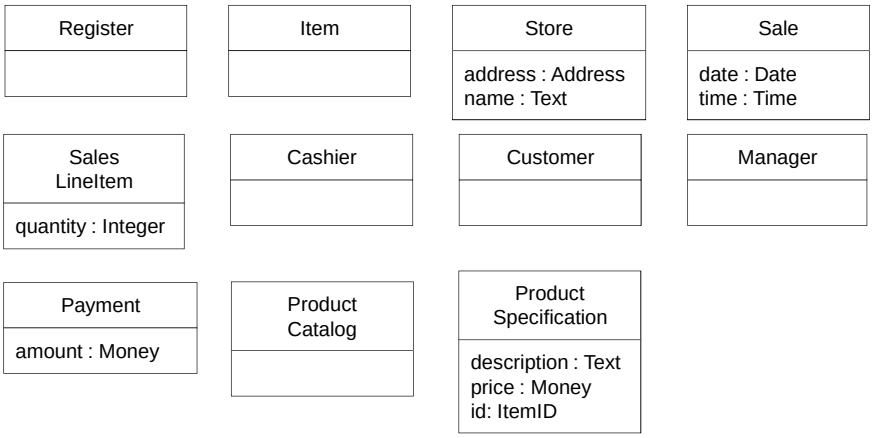
OK

Birimleri olan büyüklükleri de basit veri tipleri ile göstermek doğru olmaz.

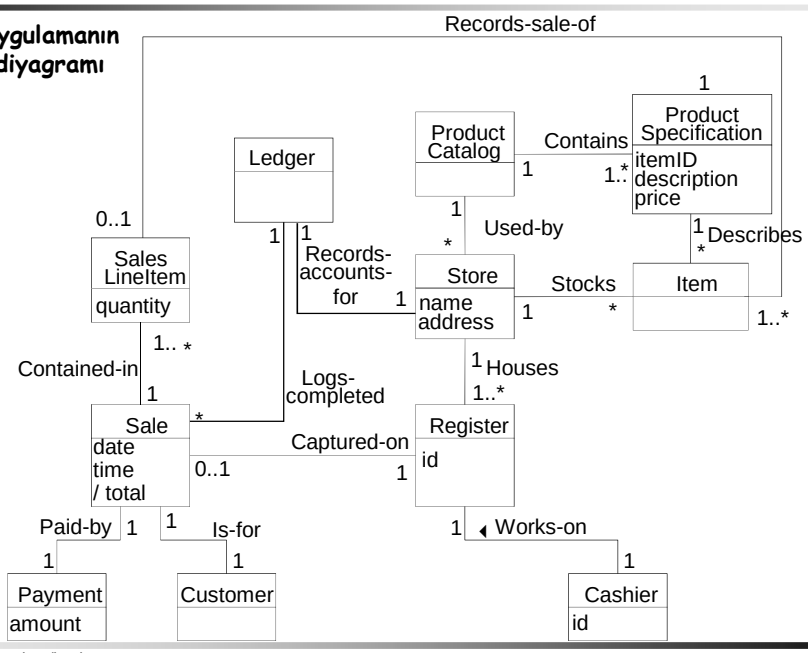


Eğer analiz sırasında kavramsal sınıfların özellikleri hakkında daha fazla bilgi elde edilmişse bunlar da diyagramlar da belirtilir.
Bu sınıflar sadece problemdeki kavramların resmidir, yazılım sınıfları değillerdir.
Tasarım aşamasında yazılım sınıfları oluşturulurken kavramsal sınıflar kaynak olarak kullanılacaktır.





Örnek uygulamanın analiz diyagramı



İşlem Sözleşmeleri (*Operation Contracts*)

Birçok uygulamada kullanım senaryolarını (*use-case*) yazmak isteklerin (ve sistemin davranışının) modellenmesi için yeterlidir.

Bazı durumlarda ise karmaşık bir işlemin daha iyi anlaşılabilmesi için o işlem için **sözleşme** (***contract***) yazmak yararlı olur.

Sözleşme; ön koşullar sağlandığında, sistemdeki bir işlem gerçekleştirildikten sonra, sistemin (uygulama uzayı nesnelerinin) alacağı durumların (son koşulların) tarif edilmesidir.

- Senaryolarda, aktörler ile sistem arasındaki etkileşim belirtilir.
- Sözleşmelerde ise sistem içi nesnelerdeki değişim belirtilir.

Sözleşmelerin yazılmasında izlenecek yöntem:

1. Sistem etkileşim diyagramlarından (senaryolardan) işlemler belirlenir.
2. Karmaşık işlemler için sözleşmeler yazılır.
3. Sözleşmelerde önemli olan son koşullardır. Son koşullar aşağıdaki kategorilerden oluşur:

- Bir nesne (*instance*) yaratma / yok etme (*instance creation*)
- Bir niteliğin güncellenmesi (*attribute modification*)
- Bir bağlantı oluşturma, koparma (*association formation*)

Sözleşmelerde sözü edilen nesneler **uygulama uzayı** (gerçek dünya) nesneleridir.

Senaryoların yazılmasına benzer şekilde, sözleşmelerin de bölümleri ve bir yazım biçimleri vardır.

Sözleşmelerin bölümleri:

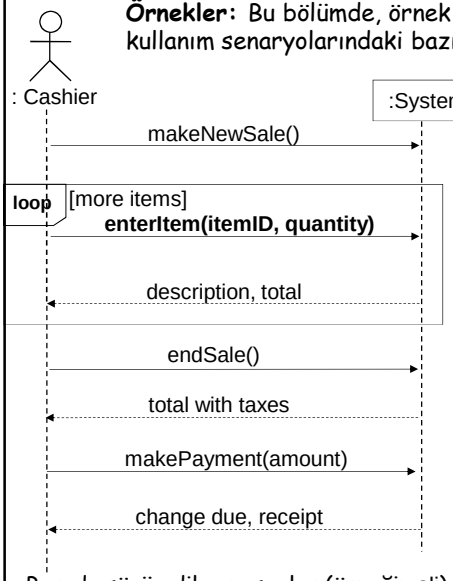
- Sözleşmenin numarası ve adı
- Sözleşmenin ait olduğu işlem
- Referans: Sözleşmenin ilgili olduğu kullanım senaryosu
- Ön koşullar (*Preconditions*): Sözleşmenin sağlanabilmesi (o işlemin gerçekleşmesi) için işleme gelinmeden önce gerçekleşmesi zorunlu olan ön koşullar.
- Son koşullar (*Postconditions*): Sistemde meydana gelmiş olan değişiklikler. Nesne yaratma, nesneleri ilişkilendirme, nitelikleri güncelleme. **Dikkat:** Buradakiler gerçek dünya nesneleridir.

  Son koşullar yazılırken geçmiş zaman kipi kullanılır. Çünkü bu bölümde yazılan maddeler, işlem gerçekleştikten sonra sistemdeki nesnelerin hangi durumlara geldiğini belirtmektedir.

  Son koşullar o işlemin nasıl yapılacağını gösteren adımlar değildir, nesnelerin gözlemlenen durumlarıdır.

  Bir işlemin nasıl yapılacağı sorusunun cevabı tasarımda aranır.

Örnekler: Bu bölümde, örnek POS sistemine ait **SG1: Satış İşlemleri** kullanım senaryolarındaki bazı işlemlere ilişkin sözleşmeler gösterilmiştir.



Örnek: enterItem

Contract CO2: enterItem
Operation:
enterItem(itemID: ItemID, quantity: integer)
Reference: Use Cases: Process Sale
PreCond.: There is a sale underway

PostCond.: - A SalesLineItem instance sli was created (instance creation)
- sli was associated with the current Sale (Association formed)
- sli.quantity became quantity (attribute modification)
- sli was associated with a ProductSpec. based on itemID match (association formed)

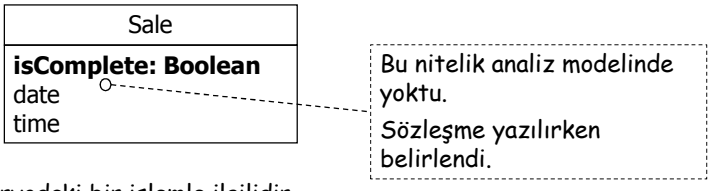
Burada sözü edilen nesneler (örneğin sli), bağlantılar ve niteliklerin hepsi **uygulama uzayı** ile ilgilidir. Bkz. Uygulama uzayı sınıf diyagramı

Sözleşmeler, uygulama modelinde bazı değişiklikler (eklemeler) yapılmasına neden olabilirler.
Aşağıdaki örnekte endSale() işlemi için sözleşme yazılmıştır.
Bu sistemde, biten satışların silinmediği sadece "bitti" olarak işaretlendiği varsayılmıştır.

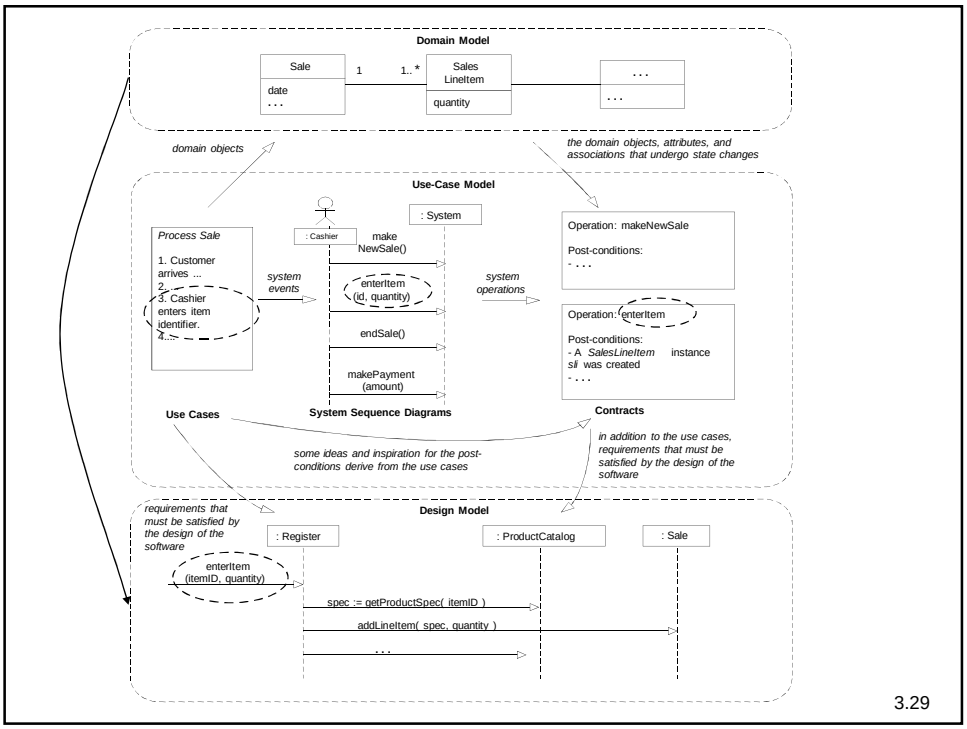
Örnek: endSale

Contract CO3: endSale
Operation: endSale()
Cross References: Use Cases: Process Sale
PreConditions: There is a sale underway

PostConditions: - Sale.isComplete became true (attribute modification)



Anımsatma:
- Bir sözleşme senaryodaki bir işlemle ilgilidir.
- Senaryolar yeterli ise sözleşme yazmaya gerek yoktur.
- Sözleşmeler sadece senaryolardaki karmaşık işlemler için yazılır.


Domain Model
Sale
date
...
1
1..*
Sales LineItem
quantity
...
...
domain objects
the domain objects, attributes, and associations that undergo state changes
Use-Case Model
Process Sale
1. Customer arrives ...
2. ...
3. Cashier enters item identifier.
4. ...
: Cashier
: System
make NewSale()
system events
enterItem (id, quantity)
endSale()
makePayment (amount)
system operations
Operation: makeNewSale
Post-conditions:
- ...
Operation: enterItem
Post-conditions:
- A SalesLineItem instance sli was created
- ...
Use Cases
System Sequence Diagrams
Contracts
some ideas and inspiration for the post-conditions derive from the use cases
in addition to the use cases, requirements that must be satisfied by the design of the software
requirements that must be satisfied by the design of the software
Design Model
: Register
: ProductCatalog
: Sale
enterItem (itemID, quantity)
spec := getProductSpec(itemID)
addLineItem(spec, quantity)
...

3.29