BİL211
Ders 3: UML ile Nesnesel Modelleme (Ch.2 Jia)

Erdoğan Doğdu
TOBB Ekonomi ve Teknoloji Üniversitesi
Bilgisayar Mühendisliği Bölümü
Ankara

İçerik

- Esaslar ve Kavramlar
- Yapıların ve ilişkilerin modellenmesi
- Dinamik hareketlerin (behavior) modellenmesi
- Gereksinimlerin 'kullanım şekilleri' (use cases) ile modellenmesi
- Örnek bir çalışma: E-Kitapçı

UML tarihçesi

- UML: Unified Modelling Language
  - Many modelling methodologies by 1990s
  - Booch method, Rumbaugh's OMT, Jacobson's OOSE unified in Rational
  - Object Management Group (OMG) www.omg.org
  - UML 1.0 (1997)
  - UML 2.0 (2004)

UML nedir?

- Yazılım tasarımı için kullanılan bir modelleme dili
- Tasarlanan sistemin 3 önemli yönünü modeller
  - Fonksiyonel (işlevsel) model
    - Kullanıcının bakışı ile sistem (kullanım şekilleri)
      - UML Use Case Diagrams
  - Nesne modeli
    - Sistemin yapısı: nesneler, özellikleri, işlemler, yapısal ilişkiler
      - UML Class Diagrams
  - Dinamik model
    - Sistemin iç işleyişi
      - UML Sequence, Avtivity, State-Chart Diagrams

Nesneler ve Sınıflar

- Nesne (object)
  - Gerçek dünyada, ayrı ayrı tanımlanabilen herşey bir nesnedir.
  - Modelde, her nesnenin bir kimliği, durumu, ve davranışı vardır.
- Sınıf (class)
  - Gerçek dünyada, benzer karakteristik ve davranışlara sahip nesneler bir sınıf (class) ile temsil edilir.
  - Modelde, bir sınıf, nesneler tarafından paylaşılan durum ve davranışları temsil eder.

Nesneler ve Sınıflar

	Gerçek Dünyada Yorumlanması	**Model**de Gösterimi
Nesne (Object)	Ayrı ayrı tanımlanabilen herşey bir nesnedir	Her nesnenin bir kimliği (identity), durumu (state), ve davranışı (behavior) vardır
Sınıf (Class)	Benzer karakteristik ve davranışlara sahip nesneler bir sınıf (class) ile temsil edilir	Bir sınıf, nesneler tarafından paylaşılan durum ve davranışları temsil eder

Nesne (object)

- Kimlik (*identity*)
  - Nesneyi birtek (*unique*) olarak tanımlar ve onu diğer nesnelerden ayırır
- Durum (*state*)
  - Özellikler (*fields* veya *attributes*) ile belirtilir
  - Özellik =
    - **ad** (*name*) + **tür** (*type*) + **değer** (*value*)
- Davranış (*behavior*)
  - Metotlar (*methods* veya *operations*): nesnenin durum bilgilerine erişebilen ve değiştirebilen işlemler.
  - Metot, metot adı, aldığı parametre türleri, ve döndürdüğü tür ile tanımlanır. Herhangi bir değer döndürmeyen metotlar *void* ile belirtilir.

Nesne (object): Örnek

- Öğrenci123456:
  - Durum:
    - ad: "Serdar Doğdu"
    - öğrenciNo: "st123456"
    - yıl: 2005
  - Metotlar:
    - dersEkle()
    - dersSil()
    - danışmanAta()

Nesne (object)

- İki nesne:
  - **"eşit"** (*equal*): durumları aynı; özellik değerleri aynı.
  - **"aynı"** (*identical*): aynı nesne
- Metotlar:
  - "erişim metotları" (*accessors*): özellik değerlerini değiştirmeyen metotlar
  - "değişim metotları" (*mutators*): özellik değerlerini değiştiren metotlar

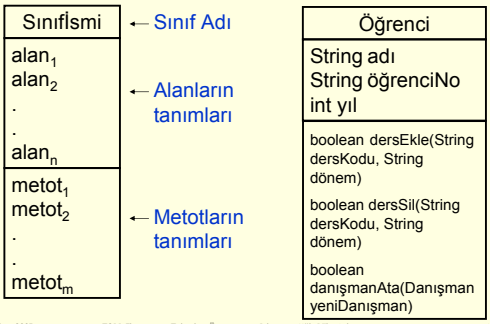
Sınıf (*class*)

- Nesneler (*objects* veya *instances*) **sınıf** (*class*) tanımı kullanarak oluşturulurlar (*instantiation*).
- Sınıf (*class*) aşağıdakileri tanımlar:
  - Alanlar (*fields*): Nesne özelliklerini tanımlayan değişkenler, adları ve türleri ile.
  - Metotlar (*methods*): Metot adları, döndürdüğü tür, parametreleri, ve metotu gerçekleştiren program kodu

Sınıf (*class*): Örnek

- Öğrenci
  - Alanlar (*fields*)
    - String ad
    - String öğrenciNo
    - int yıl
  - Metotlar (*methods*)
    - boolean dersEkle(String dersKodu, String dönem)
    - boolean dersSil(String dersKodu, String dönem)
    - boolean danışmanAta(Danışman yeniDanışman)

UML'de sınıf (*class*) gösterimi



UML sınıf tanımları

- Alanlar
  - [Erişim][Tür] Ad [[Sayı]] [=İlkDeğer]
  - Örnek: private String ad = "Serdar"
  - Standart UML gösterimi
    - [Erişim] Ad [[Sayı]] [:Tür] [=İlkDeğer]
    - Örnek: private yıl:int = 2005
- Metotlar
  - [Erişim][Tür] Ad ([*Parametre*, ...])
  - Örnek: public boolean dersAl (String dersKodu)
  - Standart UML gösterimi
    - [Erişim] Ad ([*Parametre*, ...]) [:Tür]
    - Örnek: public danışmanAta(Danışman d):boolean

Erişim (*visibility*)

- Public: diğer sınıflar erişebilir
- Protected: aynı paketteki (*package*) diğer sınıflar ve bütün alt sınıflar (*subclasses*) tarafında erişilebilir
- Package: aynı paketteki (*package*) diğer sınıflar tarafında erişilebilir
- Private: yalnızca içinde bulunduğu sınıf tarafından erişilebilir (diğer sınıflar erişemezler)

Erişim (*visibility*)

Erişim (*visibility*)	Java	UML
public	**public**	+
protected	**protected**	#
package		~
private	**private**	-

UML alan örnekleri

```
Date doğumGünü      (Java)
doğumGünü:Date      (UML)

public int süre = 100 (Java)
+süre:int = 100       (UML)

private Öğrenci
  öğrenciler[0..MAX_ÖĞR]        (Java)
-öğrenciler[0..MAX_ÖĞR]:Öğrenci(UML)
```


UML metot örnekleri

```
boolean dersEkle(String dersKodu,
  String dönem)                (Java)
~dersEkle(String dersKodu, String
  dönem):boolean               (UML)

public int getSize()           (Java)
+getSize():int                 (UML)
```


UML sınıf örnekleri

UML (Java syntax):

Point
private int x private int y
public void move(int dx, int dy)

UML (standart):

Point
- x:int - y:int
+move(dx:int, dy:int)

UML (kısaltılmış):

Point veya

Point
x y
move()

UML nesne (*object*) gösterimi

nesneAdı : SınıfAdı
$alan_1$= $değer_1$ $alan_2$= $değer_2$ . . $alan_n$= $değer_n$

← Nesne ve Sınıf Adı; altı çizili

← Alanlar ve aldıkları değerler

UML nesne gösterimi

UML

p1:Point
x = 0 y = 0

p2:Point
x = 24 y = 40

Java

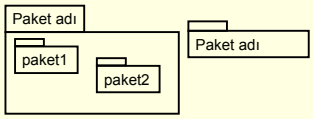
```
Point p1 = new Point();
p1.x = 0;
p1.y = 0;

Point p1 = new Point();
p1.x = 24;
p1.y = 40;
```


Mesaj Geçme (message passing)

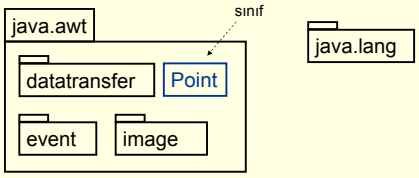
- Nesneler birbirleriyle mesaj geçerek iletişim kurarlar
  - Mesaj geçme (*message passing*) veya
  - Metot çağırma (*method invocation*)
- Mesajı bir nesneye gönderilir (alıcı - *recipient*) ve alıcı nesne çağırılan metodu çalıştırır
- Örnek: `p1.move(10, 20)`
  (p1 nesnesi: x'de 10, y'de 20 pixel kay)
  Alıcı: `p1`, metot: `move()`, parametreler: `(10,20)`

UML Paket (package) gösterimi

- Birbirleriyle ilişkili sınıflar bir paket (package) içine yerleştirilirler.
- Paket isimler küçük harflerle yazılır.
- Yaygın olarak kullanılacak paket isimleri internet domain ismini tersten yazarak kullanırlar.
  - tr.edu.etu.bil



UML Paket: Örnekler



Nesnesel Yazılım Geliştirme Prensipleri

- Modülerlik (*modularity*)
- Soyutlama (*abstraction*)
- Kapama (*encapsulation*)
- Polymorphism

Modülerlik (*modularity*)

- Karmaşık sistemler, parçalara bölünerek geliştirilmeli ve denetlenmelidir
- Parçala-fethet (*divide-and-conquer*) tekniği
- Sistem modüllere parçalanmalıdır (decomposition)
  - Tek bir yolu yok; bu bir sanat
  - Kriter:
    - Cohesion: Modül içindeki nesneler birbirleriyle fonksiyonel olarak ilişkili olmalıdır
    - Coupling: Modüller arası iletişim basit olmalıdır

Soyutlama (*abstraction*)

- Esas olanın, esas olmayandan ayrılması
- Bir modülün davranışları, veya işlevleri, bir kontrat arayüzü (contractual interface) ile belirlenmelidir. Bu arayüz modülün soyutlanmış halidir; modülün ne yaptığını belirtir.
- Modül: servis sağlayıcı (service provider)
- Kontrat arayüzü: servis kontratı
  - **Hangi** servisler sağlandığı belirtilir (*what*)
  - Servislerin **nasıl** sağlandığı **belirtilmez** (*not how*)

Soyutlama: Örnek

- Telefon
  - Oldukça karmaşık bir sistem: sesin elektronik sinyallere ve tekrar sese çevrimi, sinyallerin analog ve dijital olarak aktarımı, sinyallerin enkript/dekript edilmesi, bağlantı kurulması vs.
- Telefon kullanıcı açısından
  - El kitabı kullanılarak
    - (1) Arama, (2) konuşma, ve (3) kapama
    - Telefon servisinin kullanıcı açısından soyutlanmış hali
    - Bu servislerin telefon servisi tarafından "*nasıl*" yapıldığı (detaylar) servis kullanıcıyı ilgilendirmemektedir

Kapama (*encapsulation*)

- Müşteri (servis kullanıcı) servisi kullanmak için servis kontratı dışında hiçbirşey bilmek zorunda değildir
- Bir modül gerçekleştirilirken, detaylar kullanıcılardan gizlenmelidir; yalnızca kontrat arayüzü aracılığı iletişim kurulmalıdır
- Bilgi gizleme (*information hiding*)
- Arayüz sabit kaldığı sürece, program kodu diğer kullanıcıları etkilemeden değiştirilebilir..

Kapama: Örnek

- Telefon servisi eskiden yalnızca analog'du
- Şimdi dijital telefon servisi de sağlanıyor, ayrıca encription (kodlama) ile güvenlik de sağlanabiliyor
- Fakat servis arayüzü (*interface*) değişmedi
- Kullanıcılar servisi eskisi gibi kullanıyorlar, fakat ses ve servis kalitesi arttı
- Java'da servis arayüzü, gerçekleştirimden tamamen ayrılabilir
  - *interface* ve interface'i gerçekleştiren **sınıf** *(class)*

Polymorphism

- Birden fazla servis sağlayıcı aynı servis kontratını kabul edebilir (honor)
- Servis sağlayıcı, kullanıcıyı etklilemeden, yer değiştirilebilir (başka bir servis sağlayıcıya geçilebilir)

Polymorphism: Örnek

- Bazı cep telefonları hem analog hem dijital telefon kullanma kapasitesine sahipler
- Kullanılan alan hangisi ise, telefon otomatik olarak dijital veya analog servise geçmektedir.
- Kullanıcılar hangi servisin kullanıldığını bilmemekteler; servis arayüzü her iki durumda da aynı
- (GSM, TDMA, CDMA servislerini, yada değişik frekansları kullanma kapasitesine sahip telefonlar da bunun bir örneğidir)

İkişkilerin ve Yapıların Modellenmesi

UML Class Diagram

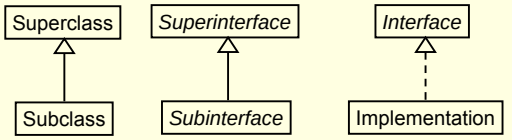
- Inheritance (kalıt)
- Association (ilişkili)
- Aggregation ve Composition (agregasyon ve kompozisyon)
- Dependency (bağımlılık)

Inheritance (kalıt)

- Sınıf ve arayüzler arasındaki ilişki
- 3 tür
  - Extension: üstsınıf (superclass) ve altsınıf (subclass) arasında
  - Arayüzler arası extension
  - Implementation: bir sınıf bir arayüzü gerçekleştiriyor
- UML'ce:
  - Specialization (extension)
  - Realization (implementation)

Inheritance (kalıt)

- UML notasyonu

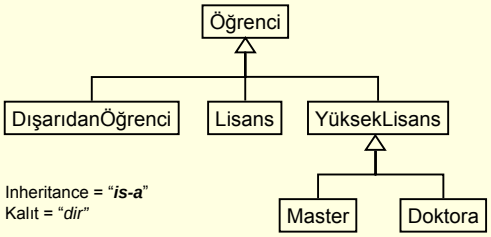


Extension of classes

Extension of interfaces

Implementation of interfaces

Inheritance: Örnek



Inheritance = "***is-a***"
Kalıt = "*dir*"

Her `Lisans` öğrencisi bir `Öğrenci`**dir.**
Her `Master` öğrencisi bir `YüksekLisans` öğrencisi**dir**

Inheritance

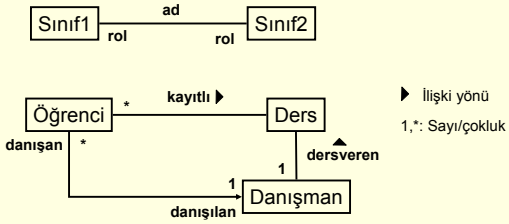
- Multiple Inheritance (çoklu kalıt): Bir sınıf birden fazla üstsınıftan kalıt alabilir (inherit from multiple superclasses)
- Single Inheritance (tekli kalıt): Çoğu nesnesel programlama dili tekli kalıta izin verir. Java gibi.
- Java'da kısıtlı olarak çoklu kalıta izin vardır; ancak bu arayüzlerden (interface) olabilir.

Soyutlama Düzeyleri (*levels of abstraction*)

- Kalıt ilişkileri sınıfları bir hiyerarşi içerisine koyar.
- Hiyerarşi içinde yukarı çıkıldıkça, sınıflar "genelleşir", aşağıya inildikçe "özelleşir".
- Süpersınıflar daha genel soyutlamalardır.
- Altsınıflar daha özel soyutlamalardır.

Association (ilişkili)

- Sınıflar arası ikili (binary) ilişkiler



İlişkinin çokluğu (sayısı)

- *a .. ü* : alt değerden üst değere kadar.
- *i* : tek bir değer
- * : *0 .. n*

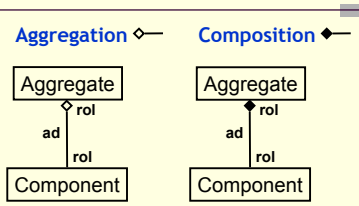
Örnekler:

0 .. *	0 veya daha fazla
1 .. *	1 veya daha fazla
2 .. 5	2'den 5'e kadar
2, 5, 7	2, 5, veya 7
1, 3, 5 .. *	1, 3, 5, veya daha fazla

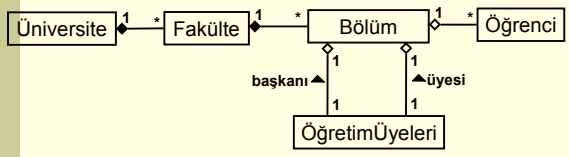
Aggregation ve Composition (agregasyon ve kompozisyon)

- Association'ının özel bir hali
- Parça-Bütün (part-whole)ilişkisini temsil eder
- Parça veya bütün'ün hayat süresi konusunda bir yaptırımı yoktur
- Aggregasyonun daha güçlü bir hali komposizyon (composition) dur.
- Kompozisyonda, parçalar bütün olmadan olmazlar (ortadan kalkarlar)
- Association'daki multiplicity, isimlendirmeler, navigasyon (ilişkilerin yönü) (agregasyon ve kompozisyonda) aynen uygulanır.

Aggregation ve Composition



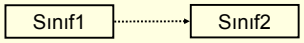
Aggregation/Composition : Örnek



- Fakülte olmadan bölümler olamaz (kompozisyon)
- Üniversite olmazsa fakülteler de olamaz (kompozisyon)
- Bölüm olmadan öğretim üyeleri ve öğrenciler(?) olabilir (agregasyon)

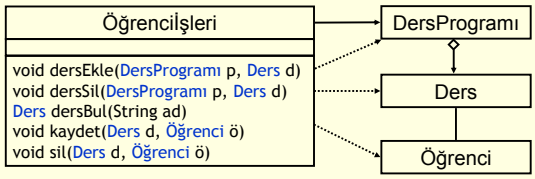
Bağımlılık (dependency)

- Bir sınıfın diğer bir sınıfa bağlı olması
- Örneğin: bir sınıfın diğer bir sınıfı "kullanması" (use) (sınıfın diğer sınıfın metotlarını çağırması gibi, yada o sınıftan bir nesneyi döndürmesi gibi..



Sınıf1, Sınıf2'ye bağımlı (dependent)

Bağımlılık (dependency): Örnek

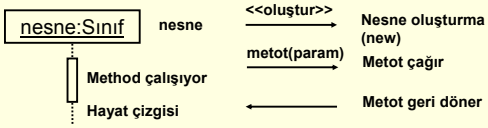


Dinamik davranışların modellenmesi

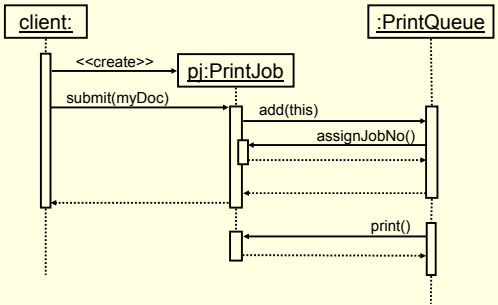
- "Class diagrams" : statik yapı
- Dinamik davranışlar?
  - Nesneler arası aktiviteler, bir nesne üzerinde olayların ve hareketlerin sıralanması
- "Sequence" diyagramları
- "State" (durum) diyagramları
  - İç-içe durum (nested state) diyagramları

Sequence Diagrams

- Nesneler arası metot çağırma işlemlerinin zaman çizgisinde sıralı gösterimi
- Zaman çizgisi y-ekseni üzerinde yukarıdan aşağı gösterilir
- İlgili nesneler x-ekseni üzerinde en üstte soldan sağa sıralanır



Sequence Diagrams : Örnek

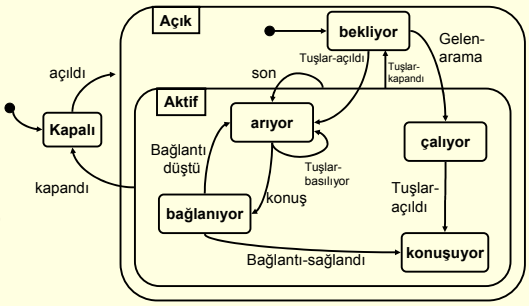


State Diagram (durum diyagramı)

- Finite State Machine (sonlu durum makinesi) nin genelleştirilmişi
- Durumlar (states) ve bunlar arasındaki geçişlerin (transitions) gösterilmesi
- Geçiş (transition): nesnenin bir durumdan diğerine geçmesi. Bu bir olayla tetiklenebilir (triggered) ve hiçbir sebep olmadan olabilir (triggerless).
  - [Olay-listesi][[Kontrol]][/Aksiyon]

State Diagram (durum diyagramı)

State Diagram: Telefon örneği



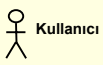
Gereksinimlerin modellenmesi

- Sistem gereksinimleri UML Use Case (kullanım şekilleri) diyagramları ile belirtilir
- Yazılım geliştirme için gerekli değildir, fakat gereksinimler ve nesnesel modeller arasında en önemli bağlantıdır
- Use Case: Kullanım şekli
  - Bir sistem fonksiyonunun dışarıdan gözlemlenen davranışı
  - Sistemle, sistem dışı aktörler (kullanıcı veya diğer sistemler gibi) arasında etkileşimler..
  - Sistem "ne" yapıyorla ilgili, "nasıl" yapıyorla ilgili değil..

Gereksinimlerin modellenmesi

- Use Cases (Kullanım şekilleri)
  - Bir adı var
  - Birkaç senaryordan oluşabilir
    - Bunlardan birisi ana senaryo, diğerleri alternatif senaryolar olabilir

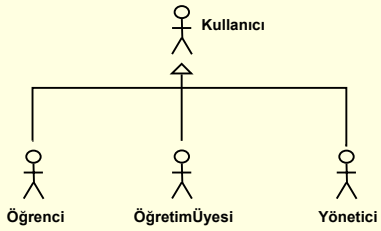
Use Cases (grafik gösterim)



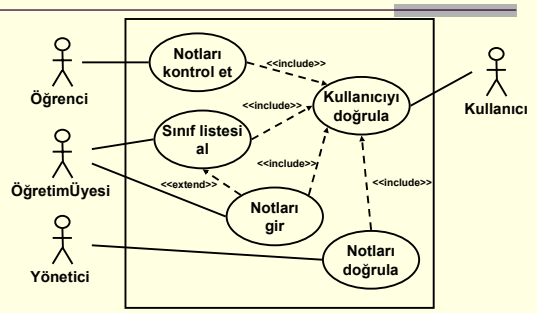
Aktör

Kullanım şekli (Use case)

Aktörler arası ilişki



Kullanım şekilleri: Örnek



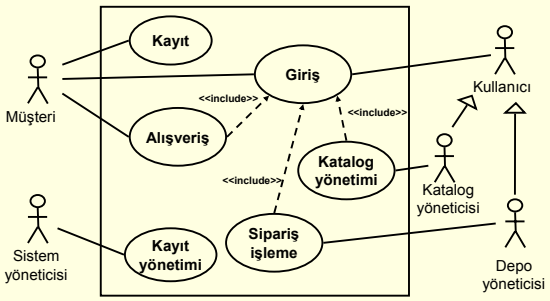
Örnek Çalışma: eKitapçı

- Kavramsallaştırma (conceptualization)
- Kullanım şekilleri (use cases)
- Nesnesel model (class diagrams)

eKitapçı: Kavramsallaştırma

- Sitemin temel gereksinimleri:
  - Müşterilerin kitapları, müzik CD'lerini, ve bilgisayar yazılımlarını İnternet üzerinden taramalarını ve satın almalarını sağlamak
- Sistemin temel fonksiyonları:
  - Müşterilere satın almalarında yardımcı olacak bilgileri vermek,
  - Müşterilerin kayıtlarını, siparişlerini, ve adres bilgilerini almak,
  - Sistem yönetimi: kayıt girme, silme, ve değiştirme; müşteri bilgilerinin güncellenmesi

eKitapçı: Kullanım şekilleri (Use Case Diagram)



eKitapçı: Use Case

- Önkoşul (precondition): Müşteri kayıtlı
- Ana senaryo:
  - Aktör: Müşteri

Giriş eylemi	Sistem eylemi ve cevabı
Giriş isteği	Hoşgeldin mesajı, kullanıcı adı ve şifresi isteği
Kullanıcı adı ve şifre girilir	Kullanıcı doğrulanır ve girişe izin verilir
Tekrarla:	
Arama ve listeleme	Satılan şeylerin listelenmesi
Almak üzere seçme	Seçilen şeyin sepete eklenmesi
Alışveriş tamamlandı	Sepettekileri, ödeme/gönderme adresleri göster
Siparişi doğrula, ve ödeme yap	Ödeme metodunu doğrula Ödeme tamamlanır Sipariş işlenir, e-fiş verilir; depoya sipariş emri
Çıkış	

- Alternatif senaryo:
  - Müşteri alışverişi tamamlamadan, alışveriş sepetini saklar, çıkar
- Aykırı senaryo (exceptional):
  - Müşteri girişi başarısız olur; giriş yinelenir
- Aykırı senaryo (exceptional):
  - Ödeme işlemi başarısız olur; müşterinin başka bir ödeme yöntemi girmesi..

eKitapçı: Nesne modelleri

- Sınıfların belirlenmesi
  - Use case'lerde geçen sınıfların belirlenmesi
  - Ne sınıftır, hangi özellikler sınıfta olmalıdır?
    - Sınıflar, nesneleri ifade etmelidir, eylemleri değil
      - Fiziki nesneler (araç, gereç, ürün, vs.)
      - Kişiler (öğrenci, öğretim üyesi, müşteri, ve bunların rolleri, gibi)
      - Organizasyonlar (üniversite, şirket, bölüm, vb.)
      - Yer (bina, oda, koltuk, vb.)
      - Olaylar (farenin tıklanması, servis isteği, alım siparişi, vb.)
      - Kavramlar (çokboyutlu uzaylar, işlemler, hava raporu haritaları, vb.)
    - Eylemler, sınıfların metotları olarak modellenmeli
    - Basit kural:
      - Sınıf -> İSİMLER (NOUN)
      - Metotlar -> EYLEM (VERB)

eKitapçı: Sınıfların belirlenmesi

- Müşterilerin kitapları, müzik CD'lerini, ve bilgisayar yazılımlarını İnternet üzerinden taramalarını ve satın almalarını sağlamak
- Sistemin temel fonksiyonları:
  - Müşterilere satın almalarında yardımcı olacak bilgileri vermek,
  - Müşterilerin kayıtlarını, siparişlerini, ve adres bilgilerini almak,
  - Sistem yönetimi: kayıt girme, silme, ve değiştirme; müşteri bilgilerinin güncellenmesi

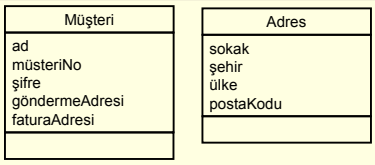
eKitapçı: Sınıflar

- EKitapçı: Tüm sistem
- Müşteri: eKitapçı müşterileri
- Kitap: eKitapçıda satılan kitaplar
- MuzikCD
- Yazılım

Ayrıca:

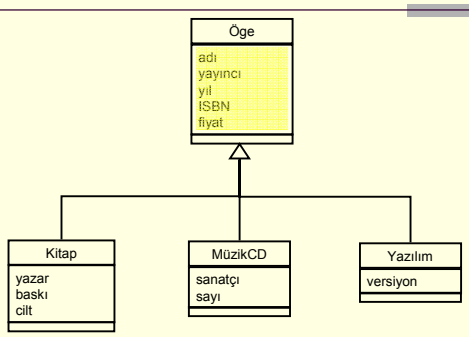
- Sepet: Müşterinin almak istediklerini tutan geçici liste
- Sipariş: Müşterinin siparişi
- Adres: Müşteri adresi

eKitapçı: Classes

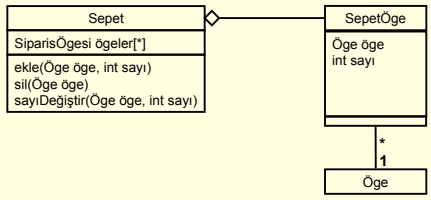


eKitapçı: Classes

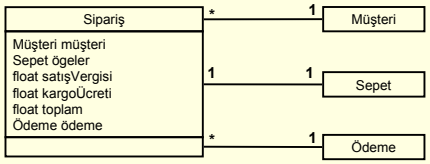
eKitapçı: Classes



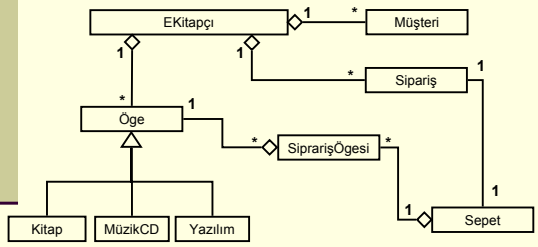
eKitapçı: Class Diagram



eKitapçı: Class Diagram



eKitapçı: Class Diagram (son)



Ödev

- Exercise 2.3 (Jia) Sayfa 52-53 (Teslim: **28 Eylül**)
  - Aşağıdaki sistemin nesnesel modelini geliştirin
    - Aktörleri ve 'use case'leri belirleyin, UML 'use case' diyagramını çizin
    - Sistemin UML 'class diagram'ını çizin
  - Havayolu Rezervasyon Sistemi:
    - Sistem bir müşterinin uçak seyahati rezervasyonu yapmasını sağlar. Müşteri, seyahatin başlangıç şehrini, gidilecek yeri, gidiş ve dönüş tarihlerini, uçuş saatlerini, tercih edilen havayolu şirketini, kaç kişilik yer ayırtılacağını belirtir. Sistem müşterinin tercihlerine uygun uçuşları ve yer olup olmadığını listeler. Müşteri uygun bir uçuşu ve yeri seçer, biletleri satın alır.

Özet

- Nesnesel yazılım geliştirmede temel esaslar: modularity, abstraction, encapsulation, levels of abstraction. Amaç: karmaşıklığı azaltmak, esnekliği artırmak.
- Decomposition (sistemin parçalarına ayrılması): İki temel kıriter uygulanır, "cohesion" (modüller içinde sınıfların uyumluluğu) ve "coupling" (modüller arasında bağların gevşek olması).
- UML nesnesel modelleme için yaygın olarak kullanılan bir modelleme dilidir.
- Sistemin statik yapısı "class" diyagramları ile modellenir,
- Sistemin dinamik yapısı ise "sequence" ve "state" diyagramları kullanılarak modellenir.
- "Use case" diyagramları ise sistemin kullanım gereksinimlerini modellemek için kullanılır (to model "what" the system does, not "how").