NESNEYE DAYALI YAZILIM GELİŞTİRME

Yrd.Doç.Dr. Feza BUZLUCA
İstanbul Teknik Üniversitesi
Bilgisayar Mühendisliği Bölümü
http://www.buzluca.info/ndyg

GİRİŞ

Dersin Hedefi:

- Bilgisayar donanımları geliştikçe yaşantımızın değişik alanlarında her geçen gün daha **karmaşık problemlerin** çözümü için bilgisayardan yararlanmaya çalışıyoruz.
- Bu durum yazılımlardan beklenen işlevleri ve performansı da arttırıyor.
- Gerçek dünyanın karmaşıklığı yazılımlara yansıyor.
- Bir programlama dilini iyi bilmek kaliteli bir yazlım geliştirmek için yeterli olmuyor.
- İyi bir yazılım oluşturabilmek için uygun yazılım geliştirme tekniklerini de bilmek ve uygulamak gerekiyor.
- Uygun teknikler kullanılmadığında bir çok değişik problemle karşılaşılmaktadır.

  Örneğin; yazılımın zamanında tamamlanamaması, bütçenin aşılması, bir çok hata çıkması ve bu hataların giderilememesi, yazılımın yeni gereksinimlere göre uyarlanamaması, bakım maliyetlerinin yüksek olması, eski projelerde hazırlanan yazılım modüllerinin yeni projelerde kullanılamaması gibi.

Bu **dersin amacı**, yukarıda kısaca sıralanan sorunları gidererek kaliteli yazılımlar geliştirmeyi sağlayan Nesneye Dayalı Çözümleme ve Tasarım (*Object-Oriented Analysis and Design – OOA/D*) yöntemlerini tanıtmaktır.

Dersin Kapsamı:

- Dersin ilk bölümlerinde istekleri ve problemi (gerçeklenecek olan sistemi) **anlamak** için yapılması gereken çözümleme (*analysis*) üzerinde durulacaktır.

  Ardından sistemin, işbirliği yapan nesneler halinde nesneye dayalı olarak nasıl **tasarlanacağı** (*design*) açıklanacaktır.
- Tümleşik modelleme dili (*The Unified Modeling Language* – UML), analiz ve tasarımların ifade edilmesinde yaygın olarak kullanılan ve endüstri standardı haline gelmiş görsel bir yöntemdir. Bu derste de çözümleme ve tasarım sonuçları UML ile ifade edilecektir.
- UML'i ayrıntılı olarak öğretmek dersin hedefleri arasında yer almamaktadır, ancak UML diyagramları kullanılırken aynı zamanda bu dilin ders kapsamında kullanılan özellikleri tanıtılacaktır.
- Nesneye dayalı tasarım yapılırken yıllar içinde oluşan deneyimlerin yöntem haline dönüştürülmesi ile oluşturulan tasarım kalıplarından (*design patterns*) yararlanılır.

  Bu derste de tasarım aşamasında GRASP kalıpları ve yaygın biçimde kabul gören GoF kalıpları tanıtılacaktır.
- Öğrencilerin bu dersten yararlanabilmeleri için nesneye dayalı programlama (OOP) yöntemini ve bu yöntemi destekleyen dillerden birini (C++, Java, C#) bilmeleri gerekmektedir.

Temel Kavramlar:

Nesneye Dayalı Çözümleme (*Analysis*): Problem (uygulama) domenindeki, yani gerçek dünyadaki sınıflar veya nesneler (kavramlar) belirlenip tanımlanır.

Bu aşamada amaç problemi çözmek değil, **anlamaktır**.

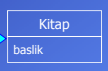
Örnek: Kütüphane otomasyonundaki kavramlar: Kütüphane, Kitap, Üye, Görevli vs.

Nesneye Dayalı Tasarım (*Design*): Yazılım (çözüm) domenindeki , yani bilgisayardaki sınıflar (ve nesneler) oluşturulur. Burada sınıfların içerikleri (özellik ve davranış) ve sınıflar arası ilişkiler tam olarak tanımlanır.

Tasarım tamamlandıktan sonra bir programlama dili (C++, Java, C#) kullanılarak kodlama yapılır.

Aşağıdaki şekilde küçük bir örnek üzerinde sınıfların değişik domenlerdeki biçimleri gösterilmiştir.

Uygulama domeni



Uygulama domeni modeli

Yazılım domeni modeli

```
public class Kitap{
private String baslik;
public void sayfaGit(int no)
{...}
}
```


Genel Bir Örnek:

Zar Oyunu: Oyuncu iki zar atar. Zarların üste gelen yüzeylerindeki sayıların toplamı 7 ise oyuncu kazanır, aksi durumda kaybeder.

Bu yazılımın gerçeklenmesinde aşağıdaki temel aşamalardan geçilir:

1. **İsteklerin (*Requirements*) Belirlenmesi, Kullanım Senaryolarının (*Use Case*) Yazılması**

Yazılım geliştirmenin ilk aşaması isteklerin (*requirements*) belirlenmesidir.

İstekleri belirlemek için kullanılan en geçerli yöntem, kullanım senaryoları (*use case*) yöntemidir.

Bu yöntemde tasarımı yapılan sistem ile kullanıcıları arasında gerçekleşebilecek tüm olaylar numaralandırılarak adım adım yazılır.

Örnek:

Ana senaryo:

1. Oyuncu iki zarı yuvarlar.
2. Sistem zarların üstündeki değerleri ve toplamlarını gösterir.
3. Oyun sona erer.

Alternatif akışlar:

2.a. Üste gelen değerlerin toplamı 7'dir. Sistem oyuncuya kazandığını bildirir.

2.b. Üste gelen değerlerin toplamı 7'den farklıdır. Sistem oyuncuya kaybettiğini bildirir.

2. Uygulama Domeninde Modelleme (Analiz)

Uygulamayı (gerçeklenecek olan sistemi) oluşturan kavramlar (sınıflar), bunların özellikleri ve aralarındaki ilişkiler belirlenir.

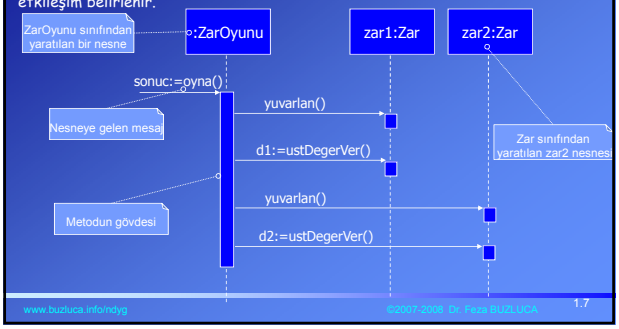
Bu aşamada elde edilen sınıflar gerçek dünyadaki sınıflardır. Oluşturulan modelin amacı problemi çözmek değil, anlamaktır.

3. Yazılım (Çözüm) Domeninde Modelleme (Tasarım)

Bu aşamada tasarım yapılarak sistemin sorumlulukları tasarım kalıplarının yardımıyla uygun yazılım sınıflarına atanır.

Tasarım sonucu iki farklı diyagram ile ifade edilir.

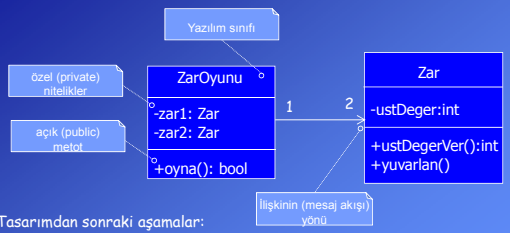
a. Etkileşim Diyagramı: Sınıfların (nesnelerin) davranışları ve aralarındaki etkileşim belirlenir.



b. Sınıf Diyagramı

Sınıfların sahip olacakları üyeler (özellikler ve davranışlar) programda yer alacağı şekilde belirlenir.

Bu aşamadaki model problemin çözümünü oluşturur.



Tasarımdan sonraki aşamalar:

4. Kodlama

5. Sınama

Bu örnek ile kısaca açıklanan konular dersin ilerleyen bölümlerinde ayrıntılı olarak anlatılacaktır.

Nesneye Dayalı Analiz ve Tasarım NEDEN Gerekli?
Yazılım Dünyasındaki Problemler:

- Donanım maliyetleri azalırken yazılım maliyetleri artmaktadır.
- Yazılımların boyutları ve karmaşıklığı artmaktadır.
- Yazılımların bakım maliyetleri çok yüksektir.
- Donanım problemleri ile çok az karşılaşılırken yazılım hataları sıklaşmaktadır.

Yazılım Geliştirme Aşamalarının Maliyetleri:

İsteklerin Çözümlenmesi (Requirements):	%3
Tasarım:	%8
Kodlama (Programlama):	%7
Sınama:	%15
Bakım:	**%67 (Maliyeti çok yüksek, neden?)**

Çözüm:

- Yazılım geliştirme: hem bir bilim dalı hem de bir sanat.
- Kolay ve kesin bir reçete yok. Sezgiler ve deneyim önemli
- Aşağıdaki unsurlar doğru şekilde kullanıldıklarında işler kolaylaşıyor, başarı olasılığı yükseliyor.

- Uygun yazılım geliştirme süreçleri:
  - Yinelemeli (*iterative*) ve evrimsel (*evolutionary*) yöntemler
  - Tümleştirilmiş geliştirme süreci (*The Unified Process* – UP)
- Programlama ve modelleme yöntemleri
  - Nesneye Dayalı Yöntem
- Yardımcı araçlar
  - UML (*The Unified Modeling Language*)
  - Yazılım Geliştirme Programları
- Tasarım kalıpları (*Design Patterns*)

Bir Yazılımın Kalitesi

- Bir yazılım istenen işi doğru olarak yapmalıdır (*useful*).
- Program kolay kullanılabilir olmalıdır (*usable*).
- Program uygulamanın gerektirdiği kadar hızlı çalışmalıdır.
- Program, sistem kaynaklarını (işlemci, bellek, disk, iletişim ağı vb.) gerektiğinden daha fazla kullanmamalıdır.
- Yazılım sağlam olmalıdır.
- Yazılımı güncellemek kolay olmalıdır.

kullanıcı

- Yazılımın bakımı (değişen isteklere uyum ve hata giderme) kolay olmalıdır.
- Yazılım projesi yeterli bir süre içinde tamamlanmalıdır.
- Yazılımın modülleri yeni projelerde tekrar kullanılabilmeli (*reusable*).
- Yazılımı geliştirme maliyeti düşük tutulabilmelidir.

Yazılım geliştiren

Programlama dilini bilmek yeterli değil. Programlama dilinin desteklediği yöntemleri iyi kullanabilmek gerekir.

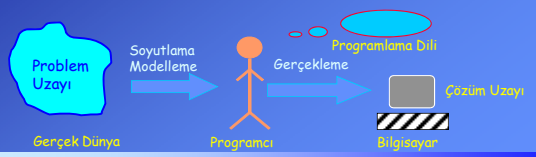
"Çekiç sahibi olmak kişiyi mimar yapmaz."

Nesneye Dayalı Yöntemin Yararı:

Programlama Nedir?

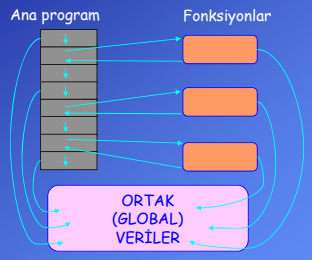
İnsanlar günlük hayatta kullandıkları konuşma dilleri ile çeşitli kavramları birbirlerine anlatmaya çalışırlar.
Benzer şekilde bilgisayar programcıları da çözülmesi gereken problemlerle ilgili kavram ve varlıkları, kullandıkları programlama dili ile bilgisayarda ifade etmeye çalışırlar.

Programlama, yaşadığımız gerçek dünyadaki problemlere ilişkin çözümlerin bilgisayarda ifade edilmesidir.
Bunu yapabilmek için, kodlamaya geçilmeden önce tasarım aşamasında, problemi oluşturan varlıkların bilgisayarda ifade edilebilecek şekilde modellerinin oluşturulması gerekmektedir.
Nesneye Dayalı Yöntem bu modellerin oluşturulmasında ve gerektiğinde sistemin güncellenmesinde diğer yöntemlere göre bazı avantajlar sağlar.



İşleve Dayalı (*Procedural*) Programlama Yöntemi

Basic, Fortran, Pascal, C gibi programlama dillerinin desteklediği bu yöntemde öncelikle gerçeklenmek istenen sistemin yapması gereken iş belirlenir.
Büyük boyutlu ve karmaşık işler, daha küçük ve basit işlevlere (fonksiyon) bölünerek gerçeklenirler.
Gerçek dünyanın modellenmesi (soyutlama) fonksiyonlar ile yapılır.



İşleve Dayalı Programlama Yönteminin Değerlendirmesi

- Karmaşık bir problemi daha basit fonksiyonlara bölmek programlama işini kolaylaştırmaktadır.

  Ancak, yazılımların karmaşıklılığı sadece boyutlarına bağlı değildir. Birimler arası ilişkiler ve bilgi akışı karmaşıklığa neden olabilir.
- Gerçek dünyadaki sistemler sadece fonksiyonlardan oluşmaz. Sistemin gerçeğe yakın bir modelini bilgisayarda oluşturmak zordur.
- Tasarım aşamasında verilerin göz ardı edilip fonksiyonlara ağırlık verilmesi hatalar nedeniyle verilerin bozulma olasılığını arttırır. Veri gizleme (*data hiding*) olanağı kısıtlıdır.
- Programcılar kendi veri tiplerini yaratamazlar
- Programı güncellemek gerektiğinde, yeni öğeler eklemek ve eski fonksiyonları yeni eklenen unsurlar için de kullanmak zordur.

İşleve dayalı yöntemi de kullanarak kaliteli programlar yazmak mümkündür. Ancak nesneye dayalı yöntem kaliteli programların oluşturulması için programcılara daha çok olanak sağlamaktadır ve yukarıda açıklanan sakıncaları önleyecek mekanizmalara sahiptir.

Nesneye Dayalı (*Object-Oriented*) Programlama Yöntemi

- Gerçek dünya nesnelerden oluşmaktadır.
- Çözülmek istenen problemi oluşturan nesneler, gerçek dünyadaki yapılarına benzer bir şekilde bilgisayarda modellenmelidir.
- Nesnelerin yapıları iki bölümden oluşmaktadır: **1.** *Nitelikler* (özellikler ya da durum bilgileri), **2.** *Davranışlar* (yetenekler, sorumluluklar)
- Nesneler belli bir **sorumluluğu** yerine getirmek üzere tasarlanırlar

Tasarım yapılırken sistemin işlevi değil, sistemi oluşturan **varlıklar** esas alınır. Bu nedenle tasarım yapılırken sorulması gereken soru,
"*Bu sistem ne iş yapar?*" değil,
"***Bu sistem hangi nesnelerden oluşur?***" olmalıdır.

Hangi unsurların nesne olarak modellenebilir:

- İnsan kaynakları ile ilgili bir programda; memur, işçi, müdür, genel müdür.
- Grafik programında; nokta, çizgi, çember, silindir.
- Matematiksel işlemler yapan programda; karmaşık sayılar, matris.
- Kullanıcı arayüzü programında; pencere, menü, çerçeve.

Nesne Örneği: Grafik Programındaki nokta

Düzlemdeki bir noktanın özellikleri; x-y koordinatlarıdır.

Davranışları ise, noktanın düzlemde yer değiştirmesi, renginin değişmesi, ekranda görünmesi ve ekranda kaybolmasıdır.

Buna göre, örnek olarak düşünülen Nokta modeli şu bölümlerden oluşacaktır:

x ve y koordinatları için iki adet tamsayı değişken: **x** , **y**

Noktanın koordinatlarını değiştirerek düzlemde yer değiştirmesini etmesini sağlayan fonksiyon: **hareket** ,

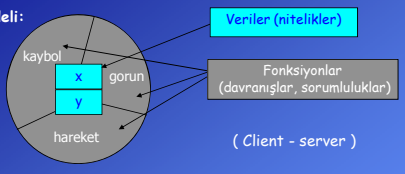
Noktanın ekranda görünmesini sağlayan bir fonksiyon: **gorun** ,

Noktanın ekrandan silinmesini sağlayan bir fonksiyon: **kaybol** .

Model bir defa oluşturulduktan sonra, ana programda bu modelden bir çok nesne yaratılabilir.

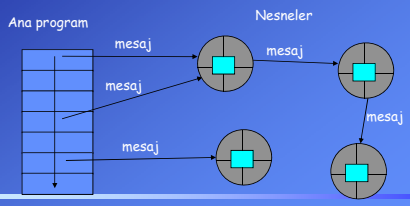
```
Nokta nokta = new Nokta();        // Nokta sınıfında bir nesne yaratıldı
:
nokta.hareket(50,30);             // nokta nesnesine mesaj gönderiliyor
nokta.gorun();
```


Bir Nesne Modeli:



(Client - server)

Nesneye Dayalı Bir Programın Yapısı:



Nesneye Dayalı Programlama Kavramlarını Kısaca Anımsatma

Sınıf Üyelerine Erişimin Denetlenmesi (Açık/Kapalı Prensibi)

Sınıfı oluşturan programcı sınıf yapısının doğru çalışmasından sorumludur.

Sınıfı nesne tanımlamak için kullanan programcı sınıfın iç yapısını bilmek zorunda değildir (kapalı). Bilmesi gereken tek şey nesnelere nasıl mesaj yollanacağıdır (açık).

Bu özellikleri sağlamak için, sınıfı yazan programcı, sınıfın bazı üyelerini gizleyerek (*data hiding*) onlara sınıf dışından erişilmesini engelleyebilir. Bir sınıfın içindeki üyelere erişimi denetleyen üç farklı etiket bulunmaktadır: **public** (açık), **private** (özel) ve **protected** (korumalı).

Bir sınıfın içindeki açık elemanlar o sınıfın dışarıya verdiği hizmetleri (*services*) tanımlarlar. Bunların tümüne sınıfın **arayüzü** (*interface*) denir.

Özel üyeler sınıfın **gerçeklenmesi**yle (*implementation*) ilgili olduklarından sınıfın kullanıcılarını ilgilendirmezler.

Bir sınıf iyi hizmet verebilecek kadar açık,

Kullanıcıyı ayrıntılar ile uğraştırmayacak ve sınıfın zarar görmesine izin vermeyecek kadar kapalı olmalıdır.

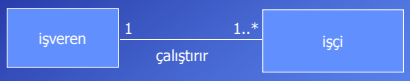
Sınıflar (Nesneler) Arasındaki İlişkiler (*Relations*)

Bir uygulamanın yapısal analizi büyük ölçüde gerekli sınıfların ve sınıflar arasındaki ilişkilerin belirlenmesine dayanır.

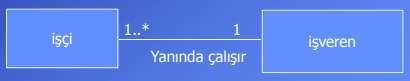
Nesneye dayalı dünyada sınıflar arasında; bağlantı (*association*), sahip olma (*aggregation*), -dan oluşma (*composition*), kalıtım (*inheritance*) gibi ilişkiler olur.

Bağlantı (*Association*)

Sınıflar arasında hizmet alma/vermeye dayalı ilişkidir.



Çoğunlukla çift yönlüdür. Tek yönlü olduğu durumlarda ok kullanılır.



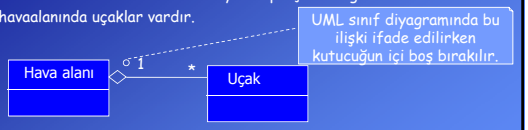
Nesneler arasındaki bağlantıya bağ (link) denir.

İç içe nesneler: Nesnelerin başka sınıfların üyesi olması
Sahip olma ilişkisi (*Aggregation / Compositon*)

Sahip olma ilişkisinin (*has a relation*) iki ayrı türü vardır:

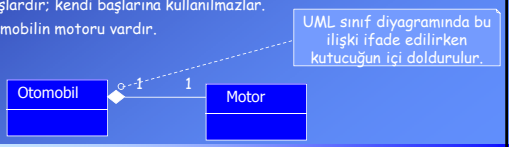
• Toplama, bir araya getirme (*Aggregation*): İçerilen nesneler (alt parçalar) kendi başlarına da kullanılırlar. Sadece o nesneye ait parçalar değillerdir.

Örneğin havaalanında uçaklar vardır.



• Meydana gelme (*Composition*): Alt parçalar o nesneyi meydana getirmek için oluşturulmuşlardır; kendi başlarına kullanılmazlar.

Örneğin otomobilin motoru vardır.



Kalıtım (*Inhertiance*), *Generalization / Specialization*

Varolan genel bir sınıftan daha özel (ek niteliklere sahip) sınıflar türetilebilir.

Bu türetimde üst sınıfın ayrıntılarının bilinmesine ve kaynak kodunun elde edilmesine gerek yoktur.

Yararları:

1. Tekrar kullanılabilirlik (*reusability*).

Sistemin genel kısımları önce kodlanır. Daha özel kısımlar genel kısımlardan türetilir. Ortak özelliklerin yeniden yazılmasına gerek kalmaz.

2. Çok şekillilik (*polymorphism*) ile birlikte kullanıldığında, aynı sınıftan türeyen farklı varlıkların onlara mesaj gönderen nesnelere aynı varlıklar gibi görünmesini sağlar.

Yandaki örnek diyagramda Müdür bir öğretmendir (*is a relation*).

Müdür öğretmenin tüm özelliklerine sahiptir, ayrıca ek özelliklere sahiptir.

Üst sınıfın (öğretmen) istenen özellikleri alt sınıfta (müdür) değiştirilebilir (*overriding*).

Çok Şekillilik (*Polymorphism*)

Aynı mesaja farklı sınıflardan yaratılmış olan nesneler farklı tepkiler verirler.

Mesajı gönderen taraf bu mesajı hangi sınıftan bir nesneye gönderdiğini bilmek zorunda değildir.

```
public class Teacher{                          // Base class
  private String name;
  private int numOfStudents;
  public Teacher(String name, int nos){...}    // Constructor of base
  public void print() {
    System.out.println("Name:"+name);
    System.out.println("Num of Students:"+ numOfStudents );
  }
}

class Principal extends Teacher{               // Derived class
  private String SchoolName;
  public Principal(String name, int nos, String SName){...};
  public void print(){                                  // polymorphic method
    super.print();
    System.out.println("School Name:" + SchoolName );
  }
}
```



```
void show (Teacher  teacher)
{
  teacher.print();      // hangi print
}

// Test amaçlı program
public class Example {
  public static void main(String[] args)
 {
  Teacher t1 = new Teacher("Teacher 1",50);
  Principal p1 = new Principal ("Principal 1",40,"School");
  show(t1);
  show(p1);
 }
}
```

Yandaki örnekte show fonksiyonu Teacher sınıfından türeyen tüm sınıfların nesneleri üzerinde işlem yapabilir.

Çok şekillilik esneklik sağlar. Örnek olarak ileride Teacher sınıfından türetilerek stajyer öğretmenleri tanımlamak üzere InternTeacher adlı yeni bir sınıf sisteme katılsa show fonksiyonunda bir değişiklik yapmaya gerek olmaz.

Nesneye Dayalı Yöntemin Değerlendirmesi:

• Gerçek dünya nesnelerden oluştuğundan bu yöntem ile sistemin daha gerçekçi bir modeli oluşturulabilir. Program daha anlaşılır olur.

• Nesne modellerinin (sınıfların) içindeki veriler sadece üye fonksiyonların erişebileceği şekilde düzenlenebilirler. Veri saklama (*data hiding*) adı verilen bu özellik sayesinde verilerin herhangi bir fonksiyon tarafından bozulması önlenir.

• Programcılar kendi veri tiplerini yaratabilirler.

• Bir nesne modeli oluşturduktan sonra bu modeli çeşitli şekillerde (*aggregation, inheritance*) defalarca kullanmak mümkündür (*reusability*).

• Programları güncellemek daha kolaydır.

• Programların bakımını yapmak daha kolaydır.

• Nesneye dayalı yöntem takım çalışmaları için uygundur.