Nesne Tabanlı Programlama

Ders Hakkında

Ders Hedefi

1. Nesne Tabanlı Programlamanın Temelleri
   Bölüm Hedef
   1.1. Neden Nesne Teknolojisi?
   1.2. Nesne Nedir?
   1.3. Nesne Özellikleri
   1.4. Nesneye Yönelik Programlamanın Aşamaları
   1.5. Nesneler Arası İlişkiler
   1.6. Nesne Tabanlı Programlama Unsurları
   Bölüm Özeti
   Değerlendirme Soruları

2. Sınıf ve Nesneler
   Bölüm Hedefi
   2.1. Sınıf Tanımlamak
   2.2. Sınıf Yaşam Süresi(Scope)
   2.3. Üye Erişim Kontrolü(private,public, protected)
   2.4. Sınıfa ait nesnelerin ilklendirilmesi(initialization: constructors)
   2.5. Kurucu Aşırı Yüklenmesi (Overloaded Constructors)
   2.6. Bileşim Sınıflar(Composition)
   2.7. “this” referansının kullanılması
   2.8. Veri Soyutlanması ve Bilgi Saklanması (Data Abstraction, Information Hiding)
   2.9. Arayüzler
   Bölüm Özeti
   Değerlendirme Soruları

3. Miras(Kalıtım) ve Çok Biçimlilik
   Bölüm Hedefi
   3.1. Taban Sınıf, Türemiş Sınıf
   3.2. Çok Biçimlilik
   3.3. Operatör Aşırı Yüklenmesi
   Bölüm Özeti
   Değerlendirme Soruları

4. İstisna Yönetimi
   Bölüm Hedefi
   4.1. “try catch finaly” deyimi
   4.2. İstisna Türleri
   Bölüm Özeti
   Değerlendirme Soruları

5. Veri Tipleri Üzerine İleri Bakış
   Bölüm Hedefi
   5.1. Değer veri tipleri

5.2. Referans veri tipleri
5.3. Organizasyon yapısı
5.4. Structure organizasyon yapısı
5.5. Class organizasyon yapısı
5.6. ByVal – ByRef

Bölüm Özeti

Değerlendirme Soruları

6. UML

Bölüm Hedefi

6.1. Use Case Diagram
6.2. Class Diagram
6.3. Object Diagram
6.4. Sequence Diagram
6.5. Collaboration Diagrams
6.6. Statechart Diagram
6.7. Activity Diagram
6.8. Component diagram

Bölüm Özeti

Değerlendirme Soruları

7. Tasarım Desenleri

Bölüm Hedefi

7.1. Abstract Factory+
7.2. Adapter+
7.3. Command
7.4. Composite
7.5. Decorator
7.6. Factory Method
7.7. Iterator
7.8. Observer
7.9. Singleton
7.10. State
7.11. Strategy
7.12. Template Method

Bölüm Özeti

Değerlendirme Soruları

Nesne Tabanlı Programlama

Ders Hakkında

Ders Hedefi

1. Nesne Tabanlı Programlamanın Temelleri

   Bölüm Hedef

- Nesne tabanlı programlama konusunu kavrayabilme
- Nesne kavramını kavrayabilme
- Nesne özelliklerini kavrayabilme
- Nesne davranış biçimini anlayabilme
- Nesnelerin barındırdığı bilgileri anlayabilme
- Nesneye yönelik programlama aşamalarını kavrayabilme
- Nesneler arası ilişkiler konusunu kavrayabilme
- Nesneye yönelik programlamanın unsurları konusunu kavrayabilme

1.1. Neden Nesne Teknolojisi?

Yazılımların karmaşıklığı ve boyutları sürekli artıyor, ancak belli bir nitelik düzeyi korumak için gereken bakımın maliyeti zaman ve çaba olarak daha da hızlı artıyordu. NYP'yi bu soruna karşı bir çözüm haline getiren başlıca özelliği, yazılımda birimselliği (modularity) benimsemesidir.

NYP ayrıca, bilgi gizleme (information hiding), veri soyutlama (data abstraction), çok biçimlilik (polymorphism) ve kalıtım (inheritance) gibi yazılımın bakımını ve aynı yazılım üzerinde birden fazla kişinin çalışmasını kolaylaştıran kavramları da yazılım literatürüne kazandırmıştır. Sağladığı bu avantajlardan dolayı, NYP günümüzde geniş çaplı yazılım projelerinde yaygın olarak kullanılmaktadır.

NYP'nın altında yatan birimselliğin ana fikri, her bilgisayar programının (izlence), etkileşim içerisinde olan birimler veya nesneler kümesinden oluştuğu varsayımıdır. Bu nesnelerin her biri, kendi içerisinde veri işleyebilir ve diğer nesneler ile çift yönlü veri alışverişinde bulunabilir. Hâlbuki NYP'den önce var olan tek yaklaşımda (Yordamsal programlama), programlar sadece bir komut dizisi veya birer işlev (fonksiyon) kümesi olarak görülmektedir.

1.2. Bir Yazılımın Kalite Ölçeği

Kullanıcı Açısından;

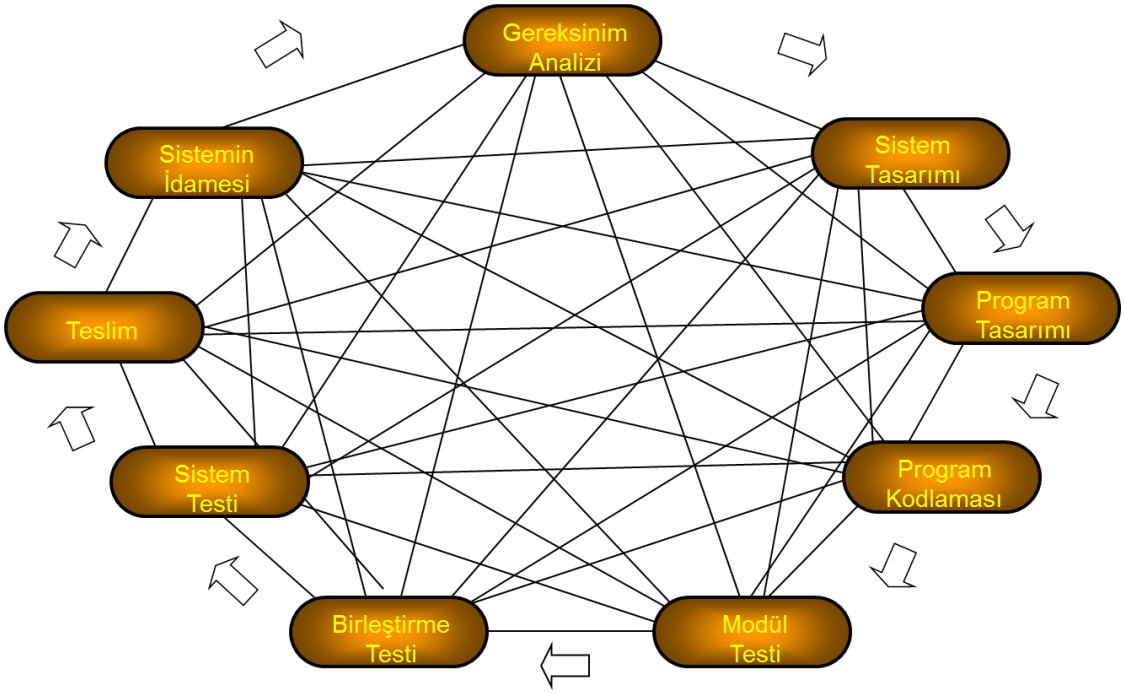
- Bir program doğru çalışmalıdır. Kullanılabilir ve kullanışlı olmalıdır.
- Bir program en hızlı performansta çalışmalıdır.
- Bir program sistem kaynaklarını çok fazla yormamalıdır. (İşlemci, bellek, disk, ağ kapasitesi)
- Güvenilir olmalıdır
- Program kolay güncellenebilmelidir.
- İyi bir programın yeterli dokümanı olmalıdır.

Yazılımcı açısından

- Kaynak kodlar okunabilir ve anlaşılabilir olmalıdır
- Yeni ihtiyaçları karşılamak için program kolay güncellenebilir ve bakımı kolay yapılabilir olmalıdır.
- Bir hata programın diğer parçalarını etkilememlidir.
- Programın modülleri başka projelerde kullanılabilir olmalıdır.
- Bir yazılım projesi teslim tarihinden önce sonlanmış olması gerekir

- İyi bir yazılımın yeterli dokümantasyonu yapılmalıdır.

1.3. Yazılım Geliştirme Süreci



Planlama

Personel ve donanım gereksinimlerinin çıkarıldığı, fizibilite çalışmasının yapıldığı ve proje planının oluşturulduğu aşamadır.

Analiz

Yazılım işlevleri ile sistem gereksinimlerinin ayrıntılı olarak çıkarıldığı aşamadır.
Temel olarak mevcut sistemde var olan işler incelenir, temel sorunlar ortaya çıkarılır ve yazılımın çözümleyebilecekleri vurgulanır**.**

Tasarım

Belirlenen gereksinimlere yanıt verecek yazılım sisteminin temel yapısının oluşturulduğu aşamadır. İki grupta incelenir.
Mantıksal tasarım; önerilen sistemin yapısı anlatılır
Fiziksel tasarım; yazılımı içeren bileşenler ve bunların ayrıntıları.

Gerçekleştirim

Tasarımı biten yazılımın, kodlama, test etme ve kurulum çalışmalarının yapıldığı aşamadır.

Bakım

İşletime alınan yazılım ile ilgili olarak, hata giderme ve yeni eklentiler yapma aşamasıdır. Bu aşama yazılımın tüm yaşamı boyunca sürer.
Yukarıda belirtilen adımlar yazılım yaşam döngüsünün çekirdek süreçleri (core processes) olarak tanımlanır
Bu süreçlerin gerçekleştirilmesi amacıyla;

Belirtim yöntemleri ; bir çekirdek sürece ilişkin Süreç modelleri kullanılır. Belirtim yöntemleri ve süreç modelleri bira raya gelerek metodolojileri oluşturur**.**

Bir çekirdek sürece ilişkin işlevleri yerine getirmek amacıyla kullanılan yöntemler "belirtim yöntemleri" olarak anılmaktadır. Farklı işlevler için kullanılan belirtim yöntemleri üç sınıfa ayrılabilir.

- Süreç Akışı İçin Kullanılan Belirtim Yöntemleri

  Süreçler arası ilişkilerin ve iletişimin gösterildiği yöntemler (Veri Akış Şemaları, Yapısal Şemalar, Nesne/Sınıf Şemaları).
- Süreç Tanımlama Yöntemleri

  Süreçlerin iç işleyişini göstermek için kullanılan yöntemler (Düz Metin, Algoritma, Karar Tabloları, Karar Ağaçları, Anlatım Dili).
- Veri Tanımlama Yöntemleri

  Süreçler tarafından kullanılan verilerin tanımlanması için kullanılan yöntemler (Nesne İlişki Modeli, Veri Tabanı Tabloları, Veri Sözlüğü).

1.4. Nesne Nedir?

Sözlük anlamı “belli bir ağırlığı ve hacmi, rengi olan her türlü canlı ve cansız varlık” olan nesne çevremizi saran dokunabildiğimiz, görebildiğimiz her şeydir.

Nesne tabanlı programlama gerçek dünyayı örnek almaktadır. Tıpkı gerçek dünyadaki gibi nesneler vardır ve birbirleri arasında etkileşim içerisindedirler.

Nesne, içinde veri ve bu veriler üzerinde işlem yapacak olan metotları (fonksiyon) bulunduran yazılım bileşenidir. Nesne bu tanıma uygun olarak, kendi işlevselliğini de içinde taşır. Nesneler her uygulamada tekrar tekrar kullanılabilir.

1.5. Nesne Özellikleri

Gerçek hayatta olduğu gibi nesne tabanlı programlamada da her bir nesnenin özellikleri vardır.
Nesnelerin özelliklerinin tutukları değerler nesnenin o andaki durumu hakkında bilgi verirler.
Bu değerler değiştikçe nesnenin durumu da değişikliğe uğrar.
Nesnelerin özellikleri genellikle nesnenin içinde ya da dışında oluşan bir olayla değişir.

1.5.1. Nesne Davranış Biçimi

Çevremizde ne kadar çok nesne var. Çiçekler, böcekler, insanlar, arabalar, otobüsler, kuşlar, taşlar, kayalar... Bu liste o kadar kabarık ki saymaya kalkmak bile sonsuzlukla çarpışmak gibi bir şey olsa gerek. Her nesnenin belli bir takım özellikleri, işlevsellikleri ve hatta amaçları var. Bazı nesneler bir araya gelerek başka yeni nesnelerin doğmasına neden olurlarken onlara bir takım ortak özelliklerini de veriyorlar.

Bir bilgisayar nesnesini göz önüne alalım. Bilgisayar, monitör, ram, hard disk, mouse, klavye vb. gibi nesnelerden oluşur. Bu parçaların görevi farklıdır. Örneğin, monitör verilerin ekranda görünmesini sağlar, ama veri depolama görevi yoktur. Veri depolama görevi hard disk biriminin görevidir.

Bilgisayarın çalışması, verilerin ekranda görünmesi, veri depolanması ise nesne davranışlardır. Verilerin ekranda gösterilmesi monitör nesnesinin bir davranışıdır.

1.5.2. Nesnenin barındırdığı Bilgiler

Her bir nesne, kendisi hakkında ve yerine getireceği işlem yani eylem hakkında bilgi barındırır. Sistemin gereksinim duyduğu tüm veriler bir nesnenin özellikleri içinde yerleşmiştir. Bazı nesnelerin az verisi vardır; hatta bazılarının hiç yoktur. Bazıları ise büyük miktarda veriye sahiptir. Bu olay nesnenin gerçekleştireceği işleme bağlıdır.

Nesne Özellikleri

- Nesneler o andaki durumları ile ilgili bilgiye sahiptir.
- Bilginin her bir parçasına özellik denilir.
- İşlemlere bağlı olarak, farklı nesneler farklı özellikleri saklarlar. Örneğin, bankanın ATM nesnesi, içerdeki para, kart tanıma, ATM kodu, ve bunun gibi özelliklere sahip olacaktır. Kalem, kalem gövdesinin uzunluğu, kalan mürekkep, ucun kalınlığı, vb. özelliklere sahiptir. Bir otomobil nesnesi ise daha farklı özelliklere sahip olacaktır. Örneğin, otomobilin hızı, ağırlığı, yakıt tüketimi vb. sayılabilir.

1.6. Nesneye Yönelik Programlamanın Aşamaları

Nesneye dayalı programlama belirli bir sıraya göre yapılır. Bu sıra içinde nesneler tasarlanır, yaratılır ve ana program içinde bu nesneler kullanılır. Problemi oluşturan nesneler belirlenir. Bu nesneler somut ya da soyut varlıklar olabilir. Bu nesneler kodlanarak tanımlanır.

Ana programda bu nesneler kullanılır. Nesne, ana programda aldığı mesaja göre davranır. Ya kendisi bir mesaj üretir ya da bir başka mesaj gelinceye kadar bekler.

Nesneler Belirlenir → Nesneler Kodlanarak Tanımlanır → Nesneler Ana Program içinde kullanılır.

1.7. Nesneler Arası İlişkiler

Nesneler, daima diğer nesnelerle ilişki içerisindedirler. Bu ilişki nesneler arası iletişim şeklinde olabildiği gibi, yapısal ilişkilerde kurabilirler. Öyle ki bir nesne başka bir nesnenin parçası olabileceği gibi, birden çok nesnenin birleşiminden de oluşabilir. Örneğin bir araba tekerlekler, kapı, motor, koltuk gibi pek çok nesnenin bir araya gelmesiyle oluşmuş bir nesnedir.

Nesneye yönelimli bir uygulamada tüm veriler ve kodlamalar nesneler üzerinden yapılırlar. Bu nedenle iş birliği söz konusudur. Nesnelerin birbirleriyle işbirliği yapabilmesi için birbirleriyle ilişki içerisinde olmaları gerekmektedir. Nesneler işbirliğini aralarında mesaj iletimiyle yaparlar. Mesajı alan nesne bu mesajı nasıl yorumlayacağını bilmekle yükümlüdür. Mesajlar genellikle nesnenin bir eylemini gerçekleştirmesi için gerekli veriyi taşırlar.

Nesneler arasındaki ilişkiyi 3 başlık altında toplayabiliriz.

1. Genelleştirme: Genelleştirme (generalization), bir sınıfa ait nesne ile o nesnenin ebeveyn sınıfına (parent class) ait nesne arasındaki kalıtım (inheritance) ilişkisidir. Bu ilişki "Kalıtım" başlığı altında incelenmiştir.
2. Bağımlılık: Bağımlılık (dependency) ya da bir başka deyişle kullanma (using), iki nesnenin birbirine olan bağımlılığıdır. Sözü geçen iki nesne arasında geçici ilişki vardır. Bir nesnenin diğer bir nesnenin kaynaklarını geçici olarak kullanmasıanlamındadır. Burada kullanan nesne istemci (client) diğer nesne ise sağlayıcı (supplier) olarak adlandırIlır. İki türlü bağımlılık vardır.

   1. Bir nesnenin diğer bir nesneyi parametre olarak kullanması (dependency as a parameter) durumudur.

   2. Bağımlıolunacak nesneyi yaratarak (dependency as an instantiation) kullanması durumudur.

```
public class Kitap
{
    public int sayfa;
    public string[] içerik;
}
public class Kişi
{
    public string adı;
    public string soyadı;
    public void KitapOku()
    {
        Kitap kitap = new Kitap(); // dependency as an instantiation
        if (kitap.sayfa > 100)
        {
            // (Kişi Kitabı kullanıyor)
        }
    }
}
public class Kişi
{
    public string adı;
    public string soyadı;
    public void KitapOku(Kitap kitap)// dependency as a parameter
    {
        if (kitap.sayfa > 100)
        {
            // (Kişi Kitabı kullnıyor)
        }
    }
}
```

3. İşbirliği : İşbirliği (association), iki farklı sınıfa ait nesneler arasındaki sürekli ilişkidir. Bu ilişkide, bir nesne diğer nesneyi yaratır, kullanır ve öldürür. İlişkinin sürekli olabilmesi için diğer nesne alan (field) olarak tanımlanır. Kullanan nesnenin hangisi olduğunu belirtmek için yöneltilmiş işbirliği (directed association) kullanılır. Yöneltilmiş işbirliğinde, yaratılıp ve öldürülen nesneden diğer nesneye bilgi akışı (navigability) vardır. İki türü vardır;

1. Herkese açık işbirliği (public association): Bu ilişkide yukarıda sözü edilen yaratılan nesneye dışarıdan erişime izin verilir. Yani değer nesneye ilişkin alan public görünürlüğü ile tanımlanır.

2. Özel işbirliği (private association) ya da bileşim (composition): Bu ilişkide yukarıda sözü edilen yaratılan nesneye dışarıdan erişime izin verilmez. Yalnızca yaratan nesne tarafından kullanılır. Bu ilişki türüne içerme (containment) adı da verilir.

```
public class Kitap
{
public int sayfa;
public string[] içerik;
}
public class Kişi1
{
public string adı;
public string soyadı;
public Kitap kitap = new Kitap();
// public association to Kitap
// kitabın başka nesneler tarafından kullanmasına
// izin veriliyor.
public Kitap KitabınıVer()
{
returnkitap;
}
}
public class Kişi2
{
public string adı;
public string soyadı;
private Kitap kitap = new Kitap();
// private association (composition) to Kitap
// kitabın başka nesneler tarafından kullanmasına izin
// verilmiyor.
public void KitabınaÇalış()
{
}
public string KitabındanBirSatırOku()
{
}


}
//...
Kişi1 ali=new Kişi1();
Kitapalininkitabı=ali.KitabınıVer();
//public association: alinin kitabına dışarıdan ulaşılıyor
Kişi2 veli=new Kişi2();
veli.KitabınaÇalış();
stringbisatır= veli.KitabındanBirSatırOku();
//composition:veli, kitabına çalıştırılıyor
```

4. Bütünleşme : Bütünleşme (aggregation), iki farklı sınıfa ait nesneler arasındaki sürekli olan ilişkidir. İşbirliğinden farklı olarak bu ilişkide, bir nesne diğer sınıfa ait başka yerde yaratılmış ya da hazır olan nesneyi kullanır. Burada tanımlanan ilişki işbirliğine (public association) benzer. Tek fark kullanılacak nesnenin kullanan nesnenin iradesi dışında yaratılmış olmasıdır. Aşağıdaki örnekte kitap olmadan kişi yaratılamaz. Yani kitap kişinin bir parçasıdır.

```
public class Kitap {
public int sayfa;
public string[] içerik;
}
public class Kişi {
public string adı;
public string soyadı;
public Kitap kitap; // 1. Kitap için referans tanımlanıyor
```

```
publicKişi(Kitap k) //2. Kitap olmadan kişi yaratılamıyor
{
kitap=k;
}
public void KitabınaÇalış(){
}
public string KitabındanBirSatırOku(){
}

}
//...
Kitapmatematikkitabı = new Kitap();
Kişimatematikkitaplıali = new Kişi(matematikkitabı);
//aggregation
matematikkitaplıali.KitabınaÇalış();
stringbisatır = matematikkitaplıali.KitabındanBirSatırOku();
// ali, matematik kitabına çalıştırılıyor
```

1.8. Nesne Tabanlı Programlama Unsurları

Nesne tabanlı programlamada temel 3 unsur vardır. Bunlar;

- Kapsülleme (Encapsulation)
- Kalıtım (Inheritance)
- Çok biçimlilik (Polymorphism)

Bölüm Özeti
Değerlendirme Soruları

2. Sınıf ve Nesneler

Bölüm Hedefi

2.1. Sınıf Tanımlamak

Bir sınıf aynı durum ve davranışa sahip olan nesneler kümesini tanımlayan ve sunan bir veri soyutlamasıdır. Yani, aynı karakteristik özelliklere sahip olan nesneler kümesi aynı sınıfa aittir. Bir sınıf bir nesnenin içini tanımlayan ayrıntılı tasarım olarak görülebilir, bir nesne bir sınıfın örneğidir.

Bir sınıf üye değişkenlerin ve üye işletmenlerin bildirildiği kodları içerir. Örneğin kitap sınıfı yazar, ödünç alan, ve iade tarihi gibi üye değişkenleri ve ödünç alan ve döndürülen üye yöntemlerini bildirir. Bir sınıf oluşturulduktan sonra, o sınıfın örneği olan nesneler oluşturabilir ve kullanabilirsiniz.
Pratikte, sınıflar programlarda tekrar kullanılabilirlik getirirken, nesneler yapışma ve modülerlik getirir. Her nesne bazı iyi tanımlanmış davranış ve veriye sahiptir ve bir nesneyi diğer nesneleri etkilemeden değiştirebilirsiniz. Bir sınıfı bildirerek, o sınıftan bir örnek yaratmak için aynı kodlar tekrar kullanılabilir.

2.2. Sınıf Yaşam Süresi(Scope)

2.3. Üye Erişim Kontrolü(private,public, protected)

2.4. Sınıfa ait nesnelerin ilklendirilmesi(initialization: constructors)

Her nesnenin durumu üye değişkenleri ile tutulur. Üye değişkenler sıradan değişken gibi bir sınıfın gövdesi içinde şöyle tanımlanır (fakat yöntem gerçekleştirmede değil-yöntem ile bildirilen değişkenler yerel değişken gibi davranır) :

VeriTürü degiskenAdi;

Her üye değişken, eğer açıkça ilklendirilmedi ise, nesne yaratıldığında kendi varsayılan değeri ile ilklendirilir (tamsayılar, kayan-noktalı, ve karakter ilkel data türleri için 0, bool veri türleri için false, ve ilgi veri türleri için null -örneğin String, dizge yada herhangi nesne ilgisi). Fakat , üye değişkenleri açıkça şu şekilde ilklendirmek de mümkün: VeriTürü degiskenAdi = ilkDeger;
Eğer isterseniz, üye değişkenleri yapılandırıcılar içinde de ilklendirebilirsiniz. Ayrıca açıkça ilklendirme şöyle de mümkün: class Daire{

```
        int originX;
        int originY;
        float yaricap;
        {
                originX = 0;
                originY = 0;
                yaricap = 1;
        }
...
}
```

Yukarıdaki ilklerdirme yöntemlerinin hepsi, bir değişken ilklendirmek için ya direk olarak kodun içinde, yada değer üreten bir anlatım yazarak sağlayabilir (fonksiyon çağırımı ve nesne yaratma anlatımları içerir).
Üye değişken bildirimleri ve ilklendirmeleri bir sınıfta yöntem bildirimleri arasında herhangi bir yerde yapılabilir, ve ilklendirmenin sırası ilklendirildikleri sıra ile belirlenir. Fakat değişkenlerin, en azından varsayılan değer ile herhangi bir yöntem veya yapılandırıcı çağrılmadan ilklendirilecekleri garantidir. Bu yüzden yapılandırıcı içindeki ilklendirme varsayılan ilklendirmeden önce yer alır, bildirimdeki ilklendirmeler ve ilklendirme bloklari içindeki açık ilklendirmeler yapılır.

2.5. Kurucu Aşırı Yüklenmesi (Overloaded Constructors)
2.6. Bileşim Sınıflar(Composition)
2.7. "this" referansının kullanılması
2.8. Veri Soyutlanması ve Bilgi Saklanması (Data Abstraction, Information Hiding)
2.9. Arayüzler
Bölüm Özeti
Değerlendirme Soruları

3. Miras(Kalıtım) ve Çok Biçimlilik
Bölüm Hedefi
3.1. Taban Sınıf, Türemiş Sınıf