Nesne Tabanlı Programlama
Bölüm 7
Soyutlama

Dr. Öğr. Üyesi Murat TAŞYÜREK (kayubmprogramlama1@gmail.com)

13 Kasım 2023

Kayseri Üniversitesi, Bilgisayar Mühendisliği Bölümü

Ders İçeriği

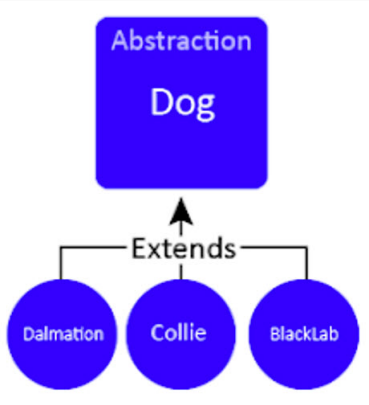
- Soyutlama
- Abstract Class
- Interfaces

Soyutlama

- Abstraction, OOP (Object Oriented Programming-Nesne Tabanlı Programlama) içerisindeki önemli kavramlardan birisidir.
- C# dilinde soyutlama; diğer Object Oriented dillerde olduğu gibi iç detayları gizleyerek sadece işlevleri göstermeye denir.
- Soyutlama bir temel oluşturma ve ilerde değiştirmeye mecbur bırakma işlemi olarakta tanımlanabilir.
- Soyutlama için **Abstract** Class ve **Interface** kullanılır.

Abstract Class

- **Abstract** sınıflar sınıf hiyerarşisinde genellikle temel sınıf (base class) tanımlamak için kullanılan sınıflardır.



Abstract Class

- C# dilinde bir sınıfı abstract yapmak için **abstract** keywordünü kullanırız.
- **Abstract** sınıflar en az bir tane abstract metod bulundurması gerekir.
- **class, method, property, index**, ve **event abstract** olarak tanımlanabilir.

```
0 references
abstract class Dog
{
    0 references
    public abstract void GetName();
}
```

Abstract Class Özellikleri

- Abstract sınıfları genel olarak inheritance (kalıtım) uygularken kullanırız.
- new anahtar sözcüğü ile nesneleri oluşturulamaz.
- İçerisinde değişken ve metod bulundurabilir.
- Abstract sınıflardan türetilen sınıfların abstract metodları implement etmesi zorunludur. Diğer metodları override etmeden de kullanabilir.
- Constructor ve destructor bulundurabilirler.
- Static olarak tanımlanamazlar.
- Türetilmiş sınıfta base sınıfta abstract olarak tanımı olan metot **override** edilerek tanımlanmak zorundadır.

Abstract Class

- Bir sınıf yalnızca bir **abstract** sınıftan kalıtım alabilir.
- C#'da çoklu kalıtım (multiple inheritance) (aynı anda iki abstract sınıftan kalıtım) desteklenmez.
- **Abstract** olmayan metodları da bulundurabilir.
- Abstract sınıf ve türütilen sınıf arasında **Is-A** ilişkisi vardır.
- Örneğin Mercedes ve BMW arabalarımız olduğunu düşünelim. Aslında burada Mercedes ve BMW adında iki tane türetilmiş sınıf ve araba taında base (abtract) sınıf tanımlanır. Deriz ki Mercedes is araba (Mercedes te bir arabadır). BMW'de bir arabadır. Bu tür ilişkilerin olduğu durumlara kuraların, değişkenleri vb. zorunlukların belirlenmesi için Abstract sınıflar kullanılır.

Örnek

- Memur, Üst Düzey yönetici işci vb. bir çok çalışanın olduğu için bir sistem tasarlayınız.
- Memur da Üst düzey yötenici de çalışandır. Yani **Is-a** ilişkisi vardır. Bu durumda calısan adında bir **abstract** sınıf tanımlanması gerekir.
- Ancak sistemde zam oranları çalışan türüne göre değişmeli. Bu durumda **abstract** sınıfında metot isimleri abstract olmalı.
- Memur ve Üst düzey yönetici için iki adet sınıf tanımlanmalı ve **abstract** sınıfından türetilmeli.
- Kodlayalım

Abstract Class

```
7 rererences
abstract class Calisan
{
    private string isim, soyisim, Unvan;
    private double m_Maas;
    2 references
    protected void SetBilgiler(string isim, string soyisim, string Unvan)
    {
        this.isim = isim;
        this.soyisim = soyisim;
        this.Unvan = Unvan;
    }
    4 references
    public double getMaas()
    {
        return m_Maas;
    }
    6 references
    protected void setMaas(double maas)
    {
        this.m_Maas = maas;
    }
    4 references
    public abstract void zamYap();
    4 references
    public abstract void zamZap(double zamOrani);
    6 references
    public override string ToString()
    {
        string bilgiler = String.Format(" İsim:{0}\n Soyisim:{1}:\n Unvan:{2}\n Maas:{3:0.00}\n",
            isim, soyisim, Unvan, m_Maas);
        return bilgiler;
    }
}
```

Abstract class

Abstract metot

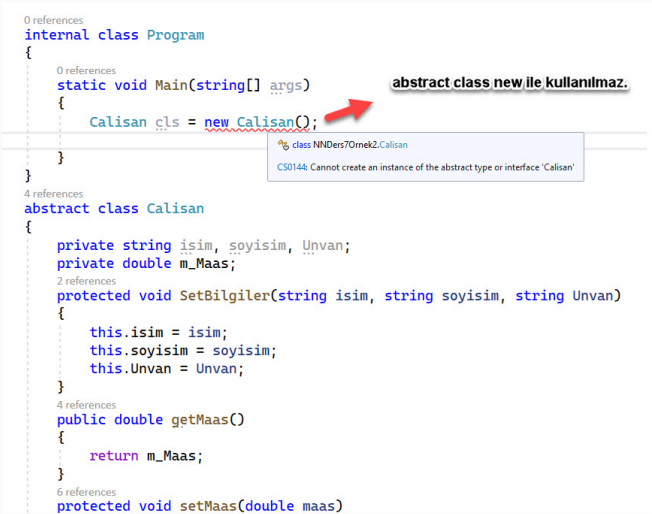
Memur Class

```
3 rererences
class Memur : Calisan
{
    1 reference
    public Memur(string isim, string soyisim, string Unvan, double baslangicMaasi)
    {
        base.SetBilgiler(isim, soyisim, Unvan);
        base.setMaas(baslangicMaasi);
    }
    //yıllık %10 zam yaptık.
    2 references
    public override void zamYap()
    {
        base.setMaas(getMaas() * 1.10);
    }
    2 references
    public override void zamZap(double zamOrani)
    {
        base.setMaas(getMaas() * zamOrani);
    }
}
```

Üst Düzey Yönetici Class

```
3 references
class UstDuzeyYonetici : Calisan
{
    1 reference
    public UstDuzeyYonetici(string isim, string soyisim, string Unvan, double baslangicMaasi)
    {
        base.SetBilgiler(isim, soyisim, Unvan);
        base.setMaas(baslangicMaasi);
    }

    //yıllık %20 zam yaptık.
    2 references
    public override void zamYap()
    {
        base.setMaas(getMaas() * 1.20);
    }
    2 references
    public override void zamZap(double zamOrani)
    {
        base.setMaas(getMaas() * (zamOrani+0.1));
    }
}
```

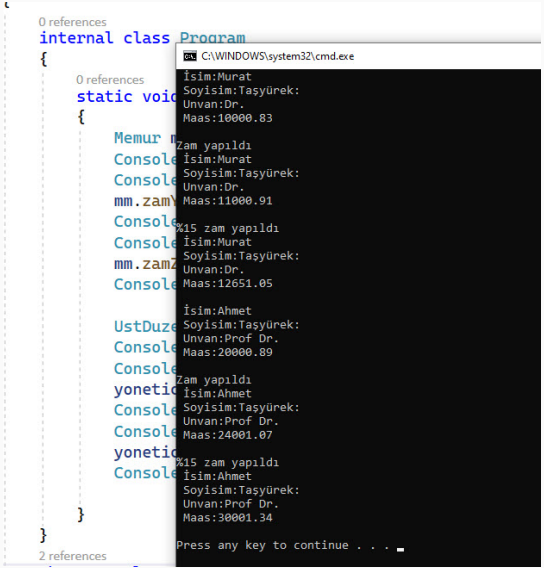
Hatalı Kullanım



Source Code

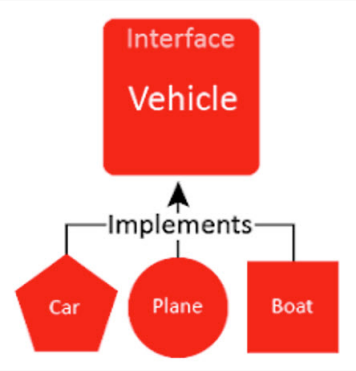
```
namespace NNDers7Ornek2
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            Memur mm = new Memur("Murat", "Taşyürek", "Dr.", 10000.83);
            Console.WriteLine(mm.ToString());
            Console.WriteLine("Zam yapıldı");
            mm.zamYap();
            Console.WriteLine(mm.ToString());
            Console.WriteLine("%15 zam yapıldı");
            mm.zamZap(1.15);
            Console.WriteLine(mm.ToString());

            UstDuzeyYonetici yonetici = new UstDuzeyYonetici("Ahmet", "Taşyürek", "Prof Dr.", 20000.89);
            Console.WriteLine(yonetici.ToString());
            Console.WriteLine("Zam yapıldı");
            yonetici.zamYap();
            Console.WriteLine(yonetici.ToString());
            Console.WriteLine("%15 zam yapıldı");
            yonetici.zamZap(1.15);
            Console.WriteLine(yonetici.ToString());

        }
    }
    2 references
    abstract class Calisan ...
    3 references
    class Memur ...
    3 references
```

Çıktı


0 references
internal class Program
{
0 references
static voi
{
Memur
Console
Console
mm.zam
Console
Console
mm.zam
Console
UstDuze
Console
Console
yoneti
Console
Console
yoneti
Console
}
}
2 references
C:\WINDOWS\system32\cmd.exe
İsim:Murat
Soyisim:Taşyürek:
Unvan:Dr.
Maas:10000.83
Zam yapıldı
İsim:Murat
Soyisim:Taşyürek:
Unvan:Dr.
Maas:11000.91
%15 zam yapıldı
İsim:Murat
Soyisim:Taşyürek:
Unvan:Dr.
Maas:12651.05
İsim:Ahmet
Soyisim:Taşyürek:
Unvan:Prof Dr.
Maas:20000.89
Zam yapıldı
İsim:Ahmet
Soyisim:Taşyürek:
Unvan:Prof Dr.
Maas:24001.07
%15 zam yapıldı
İsim:Ahmet
Soyisim:Taşyürek:
Unvan:Prof Dr.
Maas:30001.34
Press any key to continue . . .

Interface

- Sınıflara yol göstermek ve/veya rehberlik yapmak için oluşturulan, kendisinden kalıtım yoluyla oluşturulan bir sınıfa doldurulması zorunlu olan özelliklerin aktarılmasını sağlayan yapı OOP paradigmasında **Interface** olarak adlandırılır.



Interface

- Bir sınıf birden fazla **interface**'den kalıtım yoluyla türetilebilir.
- **Interface** yapıları rehber olarak tanımlandığı için içerisinde metot oluşumları dışında kod blokları bulunmaz.
- **Interface** yapılarında bulunan tüm özellikler **public** olarak kabul edilir.
- Bir interface başka bir interface tarafından türetilebilir.
- Sınıflar miras aldığı interface içerisindeki bulunan tüm özellikleri **implement** etmek zorundadır.
- Interface yapıları kullanarak nesneler oluşturulamaz.
- Türetilmiş sınıfta metot direk tanımlanarak kullanılmak zorundadır(override kelimesi kullanılmaz)

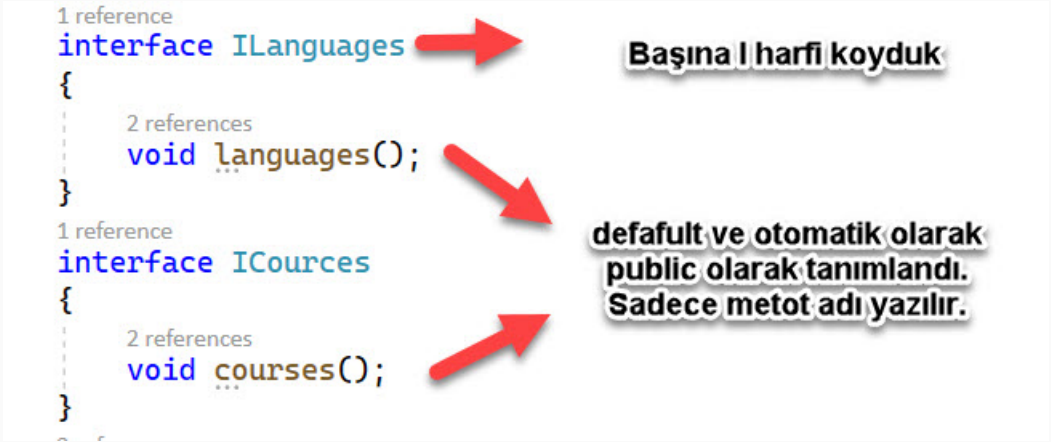
Interface vs Abstract Class

Abstract Class	Interface
Kurucu metot içerebilir	Kurucu metot içermez
Statik üyeler içerebilir	Statik üye içermez
Farklı türlerde erişim belirleyicisi (private, protectecd ...) içerebilir	Interface tanımlanan her metot public olarak tanımlanır
Sınıfın ait olduğu modeli belirtmek için kullanılır (Is a)	Sınıfın yapabileceği yetenekleri belirtmek için kullanılır (can do)
Bir sınıf sadece bir tane abstract sınıftan kalıtım (inherit) alabilir	Bir sınıf birden fazla iterface'den inherit edilebilir
Eğer birden çok sınıf aynı türden ve ortak davranış sergiliyorda bu durumda abstract class (base class) oluşturmak gerekir	Eğer birçok sınıf yalnızca ortak metotları kullanıyor ise bu durumda interface tanımlamak mantıklı olacaktır
Abstract class metot, feature, property, indexer vb. içerebilir.	İnterface sadece metot imzası içerir

Örnek

- Kurs veren merkezlerde iki ayrı fonksiyon verme zorunluğu bulunduğunu varsayalım.
- Bunlardan birisi verdiği programlama dillerini languages isminde fonksiyon ile ekrana yazdırmalı diğeri ise verdiği kursları cources olarak ekrana yazdırmalı.
- iki adet interface tanımlayalım ve bunlardan bir tane sınıf oluşturup kodlayalım.
- İpucu Interface tanımlarken daha sonra karıştırmamak için genellikle başına **I** harfi eklenir.

Interface


1 reference
interface ILanguages
{
2 references
void languages();
}
1 reference
interface ICources
{
2 references
void courses();
}
Başına I harfi koyduk
defafult ve otomatik olarak
public olarak tanımlandı.
Sadece metot adı yazılır.

Implemented Class

```
2 references
public class KursMerkezi : ILanguages, ICources
{
    2 references
    public void courses()
    {
        ArrayList listCurces = new ArrayList();

        listCurces.Add("System Design");
        listCurces.Add("Fork Python");
        listCurces.Add("Fork Java");
        Console.WriteLine("\nCourses provided by :"+this.GetType()) ;
        foreach (var elements in listCurces)
        {
            Console.WriteLine(elements);
        }
    }

    2 references
    public void languages()
    {
        ArrayList listCoruces = new ArrayList();
        listCoruces.Add("C++");
        listCoruces.Add("C#");
        listCoruces.Add("Java");

        Console.WriteLine("Languages provided by :" + this.GetType());
        foreach (var elements in listCoruces)
        {
            Console.WriteLine(elements);
        }
    }

}
```

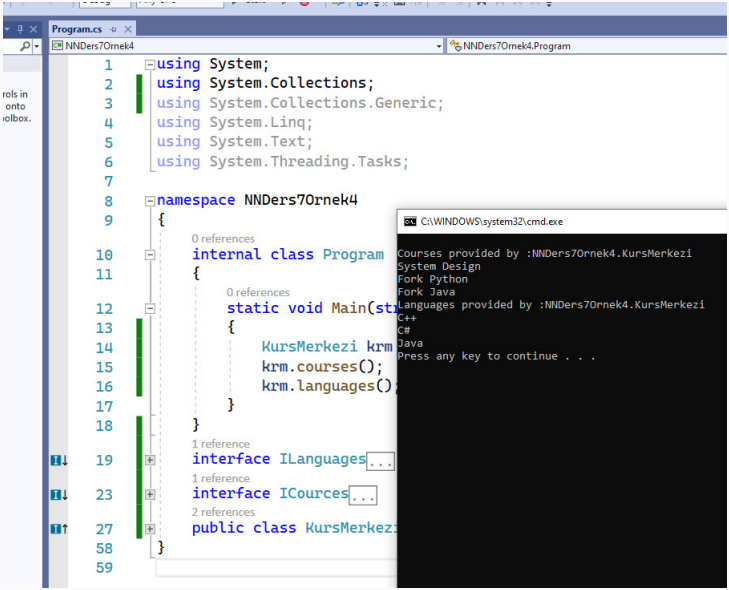
çoklu kalıtım örneği

tanımlamak zorunlu ve public tanımlamak gerekiyor ancak override etmiyoruz.

Source Code

```
namespace NNDers7Ornek4
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            KursMerkezi krm = new KursMerkezi();
            krm.courses();
            krm.languages();
        }
    }
    1 reference
    interface ILanguages...
    1 reference
    interface ICources...
    2 references
    public class KursMerkezi ...
}
```

Çıktı


Program.cs
NNDers7Ornek4
NNDers7Ornek4.Program
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace NNDers7Ornek4
{
0 references
internal class Program
{
0 references
static void Main(st
{
KursMerkezi krm
krm.courses();
krm.languages();
}
}
1 reference
interface ILanguages...
1 reference
interface ICources...
2 references
public class KursMerkez
}
C:\WINDOWS\system32\cmd.exe
Courses provided by :NNDers7Ornek4.KursMerkezi
System Design
Fork Python
Fork Java
Languages provided by :NNDers7Ornek4.KursMerkezi
C++
C#
Java
Press any key to continue . . .

Örnek

- Abstract class interface'den türetilebilir.
- Bu sayede metotları interface'de toplanabilir.
- Karmaşık projelerde metotlar bir interfacede diğer özellik ve özniteliklerde abtract sınıf içerisinde toplanabilir.
- Az önceki örneği iterface üzerinden türetilen abstract sınıf ile yapalım.

Interface ve Abstract Class

```
1 reference
interface ILanguagues
{
    3 references
    void Languages();
}
1 reference
interface ICources
{
    2 references
    void Courses();
}
1 reference
public abstract class KursMerkezi : ILanguagues, ICources
{
    2 references
    public void Courses()
    {
        ArrayList list = new ArrayList();
        list.Add("System Design");
        list.Add("Fork Java");
        Console.WriteLine("Cources");
        foreach (var el in list)
        {
            Console.WriteLine(el);
        }
    }

    3 references
    public abstract void Languages();

}
```

Class

```
3 references
public class GelismisKursMerkezi : KursMerkezi
{
    3 references
    public override void Languages()
    {
        ArrayList list = new ArrayList();
        list.Add("C#");
        list.Add("Java");
        Console.WriteLine("Languages");
        foreach (var el in list)
        {
            Console.WriteLine(el);
        }
    }
    1 reference
    public GelismisKursMerkezi()
    {
        this.Courses();
        this.Languages();
    }
}
```

Source Code

```
0 references
internal class Program
{
    0 references
    static void Main(string[] args)
    {
        GelismisKursMerkezi krm = new GelismisKursMerkezi();
    }
}
1 reference
interface ILanguagues...
1 reference
interface ICources...
1 reference
public abstract class KursMerkezi ...
3 references
public class GelismisKursMerkezi ...
```

Çıktı

```
0 references
internal class Program
{
    0 references
    static void Main(string[] args)
    {
        GelismisKursMerkezi krm = new GelismisKursMerkezi();
    }
}
1 reference
interface ILanguagues...
1 reference
interface ICources...
1 reference
public abstract class KursMer
3 references
public class GelismisKursMerk
```

```
C:\WINDOWS\system32\cmd.exe
Cources
System Design
Fork Java
Languages
C#
Java
Press any key to continue . . .
```

Interface kullanım

- Örneğin veritabanı sistemleri için bir kütüphane veya kütüphaneler oluşturmak istiyorsunuz.
- Kod yazan kişiler veya bu kütüphaneyi kullanan kişiler şunu bilir ve şunu bekler.
- Veritabanı ister MsSQL olsun isterse Oracle olsun nesne türü ve kütüphanesi değişse de nasıl bağlandığının içeriği değişse de veritabına bağlanma fonskiyonu ve komut çalıştırma fonksiyonu aynıdır.
- Bu sayede kod yazarken her kütüphaneyi ayrı ayrı öğrenmeniz gerek kalmaz. Birini bilirseniz diğeri de aynısıdır.
- Bu gibi kolaylıklar sağlamak ve yapılar oluşturmak için Interface kullanır. Örnekte sunulan veritabanı kütüphanleri de interface kullanılarak gerçekleştirilmiştir.

Örnek

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.Data.SqlClient;
7  using System.Data.OracleClient;
8
9  namespace NNDers7Ornek7
10 {
       0 references
11     internal class Program
12     {
           0 references
13         static void Main(string[] args)
14         {
15
16             OracleCommand oracleCmd = new OracleCommand();
17             oracleCmd.ExecuteNonQuery();
18
19             SqlCommand sqlCmd = new SqlCommand();
20             sqlCmd.ExecuteNonQuery();
21         }
22     }
23 }
```

Interface sayesinde aynı metot isimleri ile kullanıyor. Biz sadece aslında gidip nesnenin türünü değiştiriyoruz.