YMM1223

Nesne Yönelimli Programlama I

KONU: TEMEL KAVRAMLAR

Doç. Dr. Abdullah ELEN

Bandırma Onyedi Eylül Üniversitesi
Mühendislik ve Doğa Bilimleri Fakültesi, Yazılım Mühendisliği

aelen@bandirma.edu.tr
https://www.elenium.net

Belge No: YMM1223/01
Revizyon: R-001
Yayın Tarihi: 18.05.2023

İçindekiler

- Giriş
- Sınıf ve Nesne Kavramları
- Encapsulation (Kapsülleme)
- Inheritance (Kalıtım)
- Polymorphism (Çok biçimlilik)
- Abstraction (Soyutlama)

Giriş: NYP

Introduction to Object-oriented Programming

Giriş: NYP

- ✓Nesne Yönelimli Programlama (NYP) mantıksal işlemlerden ziyade, nesnelere (object) ve nesneler üzerinde işlemlere odaklanan programlama dili modelidir.
- ✓NYP'de programlar, nesnelerin birbirileriyle etkileşime geçmeleri sağlanmasıyla tasarlanır.
- ✓Java, C++, C#, Python, PHP, JavaScript, Ruby, Perl, Smalltalk, Objective-C gibi diller başlıca nesne yönelimli programlama dilleridir.

Giriş: NYP

- ✓NYP teorisinde, dört temel özelliğin gerçekleştirilmesi zorunlu sayılmıştır.
- ✓Bu özelliklerden herhangi birinin eksik olması durumunda, bu dil saf NYP sayılmamıştır.
- ✓Bu temel özellikler şöyledir:
  - **Encapsulation** (*Kapsülleme*)
  - **Inheritance** (*Kalıtım*)
  - **Polymorphism** (*Çok biçimlilik*)
  - **Abstraction** (*Soyutlama*)

Sınıf ve Nesne Kavramları

Class and Object Concepts

Sınıf ve **Nesne** Kavramları

- ✓ Sınıf (Class), kullanıcı tanımlı bir veri türüdür.
- ✓ Sınıfın bir örneğini oluşturarak erişilebilen ve kullanılabilen veri üyeleri ve üye işlevlerden oluşur.
- ✓ Bir türdeki tüm nesneler için ortak olan özellikler veya yöntemler kümesini temsil eder.
- ✓ Sınıf, bir nesnenin planı gibidir.
- ✓ Örnek: Araba Sınıfını düşünün.
  - ✓ Farklı isim ve markalara sahip birçok araba olabilir, ancak hepsinin bazı ortak özellikleri olacaktır, örneğin hepsinin 4 tekerleği, Hız Limiti, Kilometre aralığı vb. kilometre onların özellikleridir.

Nesne ve Sınıflar

- ✓ Aşağıda bir sınıf örneği görülmektedir.
- ✓ "Ogrenci" sınıfı için öğrencinin adı, bölümü ve doğum tarihi gibi özellikleri atanmıştır.

```
1 başvuru
class Ogrenci
{
    1 başvuru
    public string Adi { get; set; }
    1 başvuru
    public string Bolumu { get; set; }
    1 başvuru
    public DateTime DogumTarihi { get; set; }

    0 başvuru
    public Ogrenci(string adi, string bolumu, DateTime dogTarihi)
    {
        Adi = adi;
        Bolumu = bolumu;
        DogumTarihi = dogTarihi;
    }
}
```

- ✓ Aşağıdaki görüntüde nesne oluşturma örneği görülmektedir.
- ✓ Sınıfta belirlenen özellikler için veri girişi yapılmıştır.

```
0 başvuru
static void Main(string[] args)
{
    Ogrenci ogr1 = new Ogrenci("Ahmet",
        "Yazılım Mühendisliği", new DateTime(1993, 1, 15));

    Ogrenci ogr2 = new Ogrenci("Ayşe",
        "Yazılım Mühendisliği", new DateTime(1993, 5, 18));
}
```

Nesne ve Sınıflar

- ✓ Nesne, NYP'nin temel bir birimidir ve gerçek dünyadaki varlıkları temsil eder.
- ✓ Bir nesne, bir Sınıfın örneğidir.
- ✓ Bir sınıf tanımlandığında, bellek ayrılmaz, ancak örneklendiğinde (yani bir nesne oluşturulduğunda) bellek ayrılır.
- ✓ Bir nesnenin bir kimliği, durumu ve davranışı vardır.
- ✓ Her nesne, verileri işlemek için veri ve kod içerir.
- ✓ Nesneler, birbirlerinin verilerinin veya kodunun ayrıntılarını bilmek zorunda kalmadan etkileşime girebilir, kabul edilen mesajın türünü ve nesneler tarafından döndürülen yanıtın türünü bilmek yeterlidir.
- ✓ Örneğin, "Köpek" renk, ırk, havlama, uyku ve yemek gibi bazı özelliklere sahip gerçek hayattaki bir nesnedir.

Kapsülleme

Encapsulation

Kapsülleme

- ✓ Nesne yönelimli programlamanın ilk prensibi kapsülleme (encapsulation) olarak adlandırılır.
- ✓ Bu özellik, dilin nesne kullanıcısından gereksiz uygulama ayrıntılarını saklar.
- ✓ Oluşturulan bir sınıf içerisinde kullanıcının işlemlerini daha kolay gerçekleştirebilmesi için bazı işlemler birleştirilerek tek bir işlem gibi gösterilir. Bu birleştirme işlemine kapsülleme denir.
- ✓ Erişim belirteçleri (***access modifier***) sayesinde kapsülleme çok daha kolay yapılmaktadır.
- ✓ Erişim belirteçleri, oluşturulan sınıf veya sınıf içindeki elemanların erişim seviyelerini belirlemek için kullanılan anahtar kelimeler grubuna verilen isimdir.
- ✓ Metotlar ve değişkenler bir anahtar sözcük ile önceden belirlenen sınırlar dahilinde kullanılabilir.

Kapsülleme

- ✓ Bu anahtar kelimeler şu şekilde sıralanabilir:
- ✓ **Public:** Sistemdeki bütün sınıfların erişebilmesini sağlar. Yalnızca aynı proje içinden değil, diğer projelerden de erişim sağlanabilir.
- ✓ **Private:** Bir "özellik (property)"in veya "metod"un sadece tanımlandığı sınıftan erişilebilmesini sağlar. Oluşturulan sınıf veya yapıların "public" olması açık bir şekilde belirtilmez ise, derleyici tarafından "private" olarak belirlenir.
- ✓ **Internal:** Aynı derleyici (assembly) içinde bulunan tüm sınıflardan erişim sağlanır.
- ✓ **Protected:** Sadece tanımlandığı sınıfın içinde ve o sınıftan türetilmiş diğer sınıfların içinde erişilebilir.

Kapsülleme

- ✓ Kapsülleme "***private***" değişkenlerin metotlar gibi kullanılmasına yardımcı olur.
- ✓ Yalnızca okuma (*Read-only*) işleminin yanısıra okuma-yazma (*Read-write*) işleminin yapılmasını sağlar.
- ✓ Adı, Bölümü, D.Tarihi ve HarcUcreti alanlarından oluşan **Ogrenci** adlı bir sınıfın; **HarcUcreti** alanı kapsüllenip, birisi nesneye değer atamasını kontrol eden **SetHarcUcreti** metodu, diğeri de atanan değeri geriye döndüren **GetHarcUcreti** metotlarıyla kapsülleme uygulaması:

```
1 başvuru
class Ogrenci
{
    public string Adi;
    public string Bolumu;
    public DateTime DogumTarihi;

    private double harcUcreti; // HarcUcreti alanı kapsülleniyor

    // Constructor (Yapıcı)
    0 başvuru
    public Ogrenci() { }

    0 başvuru
    public void SetHarcUcreti(double tutar)
    {
        // Harç ücreti sıfırdan büyük olması durumunda HarcUcreti alanına atama yapılır.
        if (tutar > 0)
            harcUcreti = tutar;
        else // Aksi durumunda kullanıcıya hata mesajı verilir.
            Console.WriteLine("Girilen tutar geçersiz!");
    }

    0 başvuru
    public double GetHarcUcreti()
    {
        return harcUcreti;
    }
}
```

Kalıtım

Inheritance

Kalıtım

- ✓ Nesne Yönelimli Programlama dillerindeki ana prensiplerinden biri **Kalıtım** ya da **Miras** (*Inheritance*) kavramıdır.
- ✓ Bir sınıfın özelliklerinin ve metotlarının başka sınıflara aktarılarak işlevinin artırılmasını sağlar.
- ✓ Oluşturulan ve genel özellikler içeren ilk sınıfa, temel sınıf (*base class*), ondan miras alınarak özelleştirilen alt sınıflara türetilmiş sınıflar (*derived class*) denir.
- ✓ Kalıtım, birbirlerine benzeyen sınıfları tek tek yazmak yerine, ortak üyeleri belirleyerek bir temel sınıf oluşturmak ve geri kalan sınıfları bu temel sınıftan türetmek gibi pratik bir yol sunar.

Kalıtım

- ✓ Miras kavramı kapsama anlamında düşünülmemelidir.
- ✓ Örneğin, bir ***Pencere*** sınıfı bir de ***Oda*** sınıfı olduğu düşünülürse, oda ve pencere sınıfı arasında türetme ilişkisi kurulamaz.
- ✓ Çünkü ***Pencere*** sınıfı, ***Oda*** sınıfının genel özelliklerini taşımamaktadır.
- ✓ Örnekte görüldüğü gibi, bir "**Hayvan**" temel sınıfından "**Kedi**" sınıfını türetilirse, **Hayvan** sınıfındaki metotlar **Kedi** sınıfında da kullanılabilir.

https://www.programiz.com/csharp-programming/inheritance

```
// base class
1 başvuru
public class Hayvan
{
    public string name;

    0 başvuru
    public void Display()
    {
        Console.WriteLine("I am an animal");
    }
}

// derived class of Hayvan
0 başvuru
public class Kedi : Hayvan
{
    0 başvuru
    public void GetName()
    {
        Console.WriteLine("My name is " + base.name);
    }
}
```

Çok Biçimlilik

Polymorphism

Çok Biçimlilik

- ✓Kalıtım yoluyla temel bir sınıftan başka bir sınıf türetilebileceğinden bahsetmiştik.
- ✓Türetilen yeni sınıf içerisinde temel sınıfın kendisine özel bazı nesneler kullanılmak istenilebilir.
- ✓Polimorfizm sayesinde bu nesneler türetilen sınıflar içerisinde yeniden düzenlenebilir.
- ✓Başka bir deyişle oluşturulan nesnelerin aynı işi farklı yollarla yapması olarak kabul edilebilir.
- ✓Böylelikle bir metot çağrıldığında birden fazla ve farklı sonuçlar oluşur.

Çok Biçimlilik

- ✓Polimorfizmi kullanabilmek için bir adet temel sınıf (*base class*) ve bu sınıftan türemiş sınıfların (*derived class*) olması gerekir.
- ✓Temel sınıfta oluşturulan bir metot diğer sınıfların hepsinde kullanılabilir ama bu metotların içerikleri birbirinden farklı olur.
- ✓Türemiş sınıflarda kullanılmak istenen metot, temel sınıfta sanal (*virtual*) olarak tanımlanır.
- ✓Daha sonra türemiş sınıflar içerisinde tanımlanırken temel sınıftaki metot ezilir (*override*) ve yeni hali oluşturulur.

Çok Biçimlilik

✓ **Örnek:**

- Polimorfizmde temel sınıf (***Akort***) bir orkestra şefi olarak düşünülebilir.
- Şefin orkestrasına yönelttiği akort et emri ise şef sınıfının metodu olur.
- Temel sınıftan türemiş ***gitar*** ve ***keman*** sınıfları akort et emrini aynı şekilde algılar fakat farklı uygularlar.

```
7 başvuru
public class Akort
{
    3 başvuru
    public virtual void AkortEt()
    {
        Console.WriteLine("Çalgılar akort edilsin!");
    }
}

1 başvuru
public class Gitar : Akort
{
    2 başvuru
    public override void AkortEt()
    {
        Console.WriteLine("Gitar akort edildi.");
    }
}

1 başvuru
class Keman : Akort
{
    2 başvuru
    public override void AkortEt()
    {
        Console.WriteLine("Keman akort edildi.");
    }
}
```

Soyutlama

Abstraction

Soyutlama

- ✓ Soyutlama (*Abstraction*) detayları saklamak ve sadece gösterilmesi istenen bilgileri kullanıcıya göstermek amacıyla kullanılır.
- ✓ Soyutlama, hem soyutlama sınıfları (*Abstract class*) ile hem de arayüzler (*interface*) ile yapılır.
- ✓ Abstract anahtar kelimesi sınıflar ve metotlar için kullanılır.
  - **Abstract Class:** Soyut sınıflardan örnek nesne (*instance object*) oluşturulamaz. Soyut sınıfa erişmek için başka bir sınıftan kalıtım alınması gerekir.
  - **Abstract Method:** Sadece soyut sınıf içerisinde kullanılır. Metot içerisine herhangi bir şey yazılmaz. Soyut bir sınıf, hem soyut metotlara hem de diğer metotlara sahip olabilir.

Soyutlama

- ✓Yanda, Animal adında bir abstract class oluşturduk.
- ✓Bu sınıfın içerisinde bir AnimalSound() abstract metodu ve Sleep() adında değer döndürmeyen normal metot oluşturduk.
- ✓Abstract metotlar sadece soyut sınıflar içerisinde kullanılabilirler demiştik.
- ✓Bu metotlar mirasçı sınıflarda *override* edilmek zorundadırlar.
- ✓Yapacakları işlemler *override* edildikleri sınıfta kodlanmalıdır.

```
// Abstract class
1 başvuru
public abstract class Animal
{
    // Abstract method (does not have a body)
    1 başvuru
    public abstract void AnimalSound();

    // Regular method
    0 başvuru
    public void Sleep()
    {
        Console.WriteLine("Zzz");
    }
}

// Derived class (inherit from Animal)
0 başvuru
class Cat : Animal
{
    1 başvuru
    public override void AnimalSound()
    {
        // The body of animalSound() is provided here
        Console.WriteLine("The cat says: mioww!");
    }
}
```

Kaynaklar

Bu ders notu hazırlanırken aşağıda belirtilen kaynaklardan yararlanılmıştır.

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	