Nesne Tabanlı Programlama
Bölüm 9
Çok biçimlilik (Polymorphism)

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

4 Aralık 2023

Kayseri Üniversitesi, Bilgisayar Mühendisliği Bölümü

Ders İçeriği

- Çok biçimlilik (Polymorphism)
- Polymorphism Çeşitleri
- Örnekler

Polymorphism

- **Polimorfizm (Polymorphism)**, "bir ad birçok biçim" anlamına gelen Yunanca bir kelimedir. Başka bir deyişle, bir nesnenin birçok biçimi vardır veya birden çok işlevi olan bir adı vardır.
- **"Poli"** çok anlamına gelir ve **"morf"** biçimler anlamına gelir. **Polimorfizm**, bir sınıfa aynı ada sahip birden çok uygulamaya sahip olma yeteneği sağlar.
- Kapsülleme ve kalıtımdan sonra Nesne Yönelimli Programlamanın temel ilkelerinden biridir.

Polymorphism

- **Çok biçimlilik (Polymorphism)** kavramı birçok türe sahip olmak anlamına gelmektedir.
- OOP gerçek hayatı modellediğinden dolayı gerçek hayat üzerinden incelersek; bir kişinin birden fazla görevi yapması, misal erkek olan biri, bir koca, bir çalışan veya bir baba olabilir.
- Yani aynı kişi farklı davranışlara sahip olabilir ve farklı görevlere sahip olabilir. OOP ise bir nesne birden fazla nesne türü olabilir ve farklı fonskiyonları olabilir.

Polymorphism

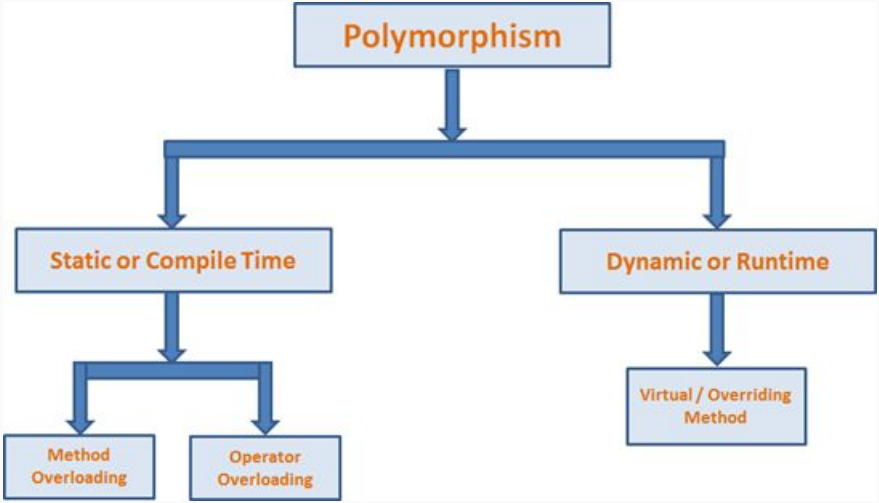
Gerçek hayattan **polimorfizm** örnekleri aşağıdaki gibi sıralanbilir.

- **Her öğrencinin** okuduğu okuldan **mezun olma** koşuluna sahip olması fakat bir üniversite mezuniyeti için gereken koşullar ile bir lise mezuniyeti arasında farklılık olması.
- Programlama dilleri açısından yazılan kodun ilk başta bir **derlenmesi** gerekmektedir fakat bu derleme işlemi c# içerisinde ayrı java ve C++ içerisinde farklı olması.
- Tüm taşıma araçları için **sürmek** eylemi kullanılmasına rağmen bir otomobil kullanımı ile bisiklet kullanımı arasında farklar olması.

Polymorphism avantajları

- Birden çok veri türünü depolamak için tek değişken kullanılabilir.
- Farklı işlemler arasındaki bağlantıyı azaltır.
- Kodun tekrar kullanılması sağlanır.
- Yapılan programda hataları ayıklamak daha kolay olur.

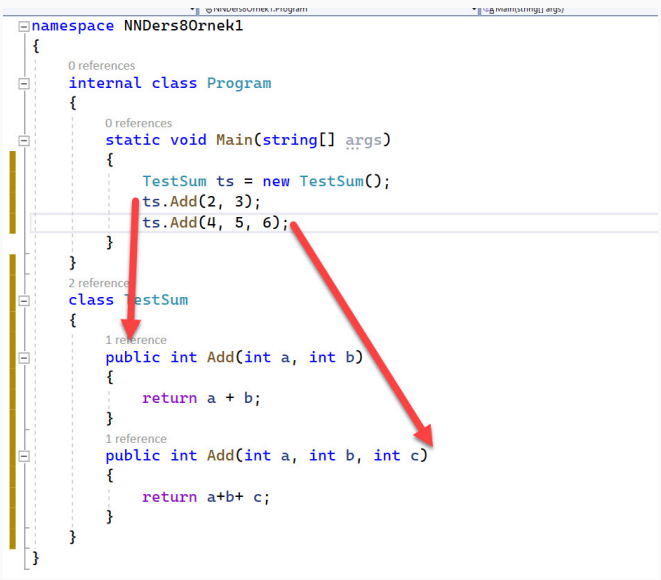
Polymorphism Çeşitleri



Örnek

```
namespace NNDers8Ornek1
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            TestSum ts = new TestSum();
            ts.Add(2, 3);
            ts.Add(4, 5, 6);
        }
    }
    2 references
    class TestSum
    {
        1 reference
        public int Add(int a, int b)
        {
            return a + b;
        }
        1 reference
        public int Add(int a, int b, int c)
        {
            return a+b+ c;
        }
    }
}
```

Örnek



```
namespace NNDers8Ornek1
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            TestSum ts = new TestSum();
            ts.Add(2, 3);
            ts.Add(4, 5, 6);
        }
    }
    2 references
    class TestSum
    {
        1 reference
        public int Add(int a, int b)
        {
            return a + b;
        }
        1 reference
        public int Add(int a, int b, int c)
        {
            return a+b+ c;
        }
    }
}
```

Run time Polymorphism Örnek

- Kullanıcı rolleri için bütün kullanıcıların showInfo yeteneğine sahip olmak zorunda olduğunu bir sistem (can do)
- Normal kullanıcı, premium kullanıcı gibi bütün kullanıcıların bir kullanıcı olduğu bir sistem (is a)
- Kullanıcılar oluşturulduktan sonra kullanıcı türünden bağımsız olarak kullanıcının bilgilerinin dinamik olarak sunulduğu bir sistem
- Kodlayalım.

Interface, class

```
interface IUser
{
    4 references
    void showInfo();
}
8 references
class User : IUser
{
    private String name;

    3 references
    public User(String name)
    {
        this.name = name;
    }

    4 references
    public virtual void showInfo()
    {
        Console.WriteLine("Kullanıcı Adı : " + name);
    }
}
```

Derived Class

```
3 references
class PremiumUser : User
{
    private double price;

    1 reference
    public PremiumUser(String name, double price) : base(name)
    {
        this.price = price;
    }

    4 references
    public override void showInfo()
    {
        base.showInfo();
        Console.WriteLine("Ödenen ücret : " + price);
    }
}
```

Class

```
1 reference
interface IUser...
8 references
class User ...
3 references
class PremiumUser ...
3 references
class NormalUser : User
{
    1 reference
    public NormalUser(String name) : base(name)
    {

    }
}
```

Source Code

```
0 references
internal class Program
{
    3 references
    public static void ShowUserInformation(User usr)
    {
        usr.showInfo();
    }
    0 references
    static void Main(string[] args)
    {
        User user = new User("Kullanıcı");
        PremiumUser user1 = new PremiumUser("Premium Kullanıcı", 250);
        NormalUser user2 = new NormalUser("Normal Kullanıcı");

        Console.WriteLine("--Kullanıcı Bilgileri--");
        ShowUserInformation(user);
        Console.WriteLine("\n--Premium Kullanıcı Bilgileri--");
        ShowUserInformation(user1);
        Console.WriteLine("\n--Normal Kullanıcı Bilgileri--");
        ShowUserInformation(user2);
    }

}
```

Çıktı

```
3 references
public static void ShowUserInformation(User usr)
{
    usr.showInfo();
}
0 references
static void Main(string[] args)
{
    User user = new User("Kullanıcı");
    PremiumUser user1 = new PremiumUser("Premium Kullanıcı", 250);
    NormalUser user2 = new NormalUser("Normal Kullanıcı");

    Console.WriteLine("--Kullanıcı Bilgileri--");
    ShowUserInformation(user);
    Console.WriteLine("\n--Premium Kullanıcı Bilgileri--");
    ShowUserInformation(user1);
    Console.WriteLine("\n--Normal Kullanıcı Bilgileri--");
    ShowUserInformation(user2);
}
```

```
eference
nterface IUser...
```

```
C:\WINDOWS\system32\cmd.exe
--Kullanıcı Bilgileri--
Kullanıcı Adı : Kullanıcı

--Premium Kullanıcı Bilgileri--
Kullanıcı Adı : Premium Kullanıcı
Ödenen Ücret : 250

--Normal Kullanıcı Bilgileri--
Kullanıcı Adı : Normal Kullanıcı
Press any key to continue . . .
```

Polymorphism

- Örnektede görüldüğü üzere teknik olarak; bir üst sınıf referansının tüm alt sınıf nesnelerini tutabilmesidir.
- Yani alt sınıftakiler de aslında üst sınıftakiler gibi davranır (Is-A) ilişkisi.
- **Polymorphism**, bir üst sınıf referansı ile alt sınıftaki nesnelerin kullanılabilmesine olanak sağlar.
- Buradaki avantaj, bir işlemi gerçekleştirirken hangi sınıfa ait nesne ile işlem gerçekleştirdiğimizi bilmemize gerek kalmamasıdır.
- **Polymorphism**, aynı zamanda nesnelerin birbirlerini tanımadan haberleşmeleri olarak da tanımlanır.

Casting

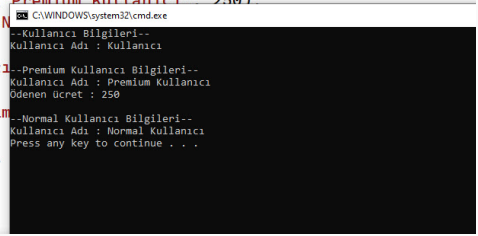
- **Upcasting** yukarı çevrim anlamına gelmektedir, **downcasting** ise aşağı çevrim anlamına gelmektedir.
- **Upcasting** alt sınıftan oluşturulmuş bir nesneyi üst sınıftan oluşturulmuş bir nesneye çevirmektir.
- **Downcasting** ise üst sınıftan oluşturulmuş bir nesneyi alt sınıftan oluşturulmuş bir nesneye çevirme işine verilen addır.

Upcasting

```
0 references
internal class Program
{
    3 references
    public static void ShowUserInformation(User usr)
    {
        usr.showInfo();
    }
    0 references
    static void Main(string[] args)
    {
        User user = new User("Kullanıcı");
        User user1 = new PremiumUser("Premium Kullanıcı", 250);
        User user2 = new NormalUser("Normal Kullanıcı");

        Console.WriteLine("--Kullanıcı Bilgileri--");
        ShowUserInformation(user);
        Console.WriteLine("\n--Premium Kullanıcı Bilgileri--");
        ShowUserInformation(user1);
        Console.WriteLine("\n--Normal Kullanıcı Bilgileri--");
        ShowUserInformation(user2);
    }

}
```

Upcasting sonuç

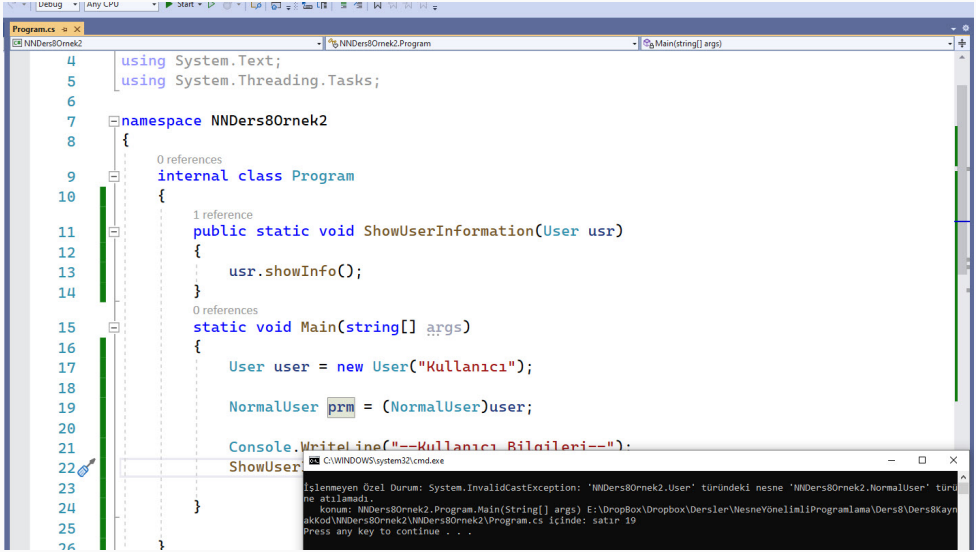
```
0 references
internal class Program
{
    3 references
    public static void ShowUserInformation(User usr)
    {
        usr.showInfo();
    }
    0 references
    static void Main(string[] args)
    {
        User user = new User("Kullanıcı");
        User user1 = new PremiumUser("Premium Kullanıcı", 250);
        User user2 = new NormalUser("N

        Console.WriteLine("--Kullanıcı
        ShowUserInformation(user);
        Console.WriteLine("\n--Premium
        ShowUserInformation(user1);
        Console.WriteLine("\n--Normal
        ShowUserInformation(user2);
    }
```



Downcasting

```
using System.Threading.Tasks;

namespace NNDers8Ornek2
{
    0 references
    internal class Program
    {
        1 reference
        public static void ShowUserInformation(User usr)
        {
            usr.showInfo();
        }
        0 references
        static void Main(string[] args)
        {
            User user = new User("Kullanıcı");

            NormalUser prm = (NormalUser)user;

            Console.WriteLine("--Kullanıcı Bilgileri--");
            ShowUserInformation(prm);


        }

    }
```

Downcasting hatalı sonuç



Downcasting

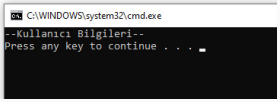
- Burada yapılan işlemde aslıdna **Polymorphism**'dir. Bir nesnenin başka bir nesne gibi davranmasını istiyoruz.
- Kodsal olarak bir hata yok. Ancak bu **run time polymorphism** olduğundan dolayı çalışırken hata veriyor.
- **Downcasting** işlemi her zaman başarılı olmayabilir. Bu nedenle öncelikle işlemin başarılı olup olmadığı kontrol edilmelidir.
- Bunun için de **as** terimi ile downcasting yapılarak nesne kontrol edilmelidir.

Downcasting

```
1 reference
public static void ShowUserInformation(User usr)
{
    usr.showInfo();
}
0 references
static void Main(string[] args)
{
    User user = new User("Kullanıcı");

    NormalUser prm = user as NormalUser;

    Console.WriteLine("--Kullanıcı Bilgileri--");
    if (prm != null)
    {
        ShowUserInformation(prm);
    }
}
```

Downcasting sonuç

```
0 references
internal class Program
{
    1 reference
    public static void ShowUserInformation(User usr)
    {
        usr.showInfo();
    }
    0 references
    static void Main(string[] args)
    {
        User user = new User("Kullanıcı");

        NormalUser prm = user as NormalUser;

        Console.WriteLine("--Kullanıcı Bilgileri--");
        if (prm != null)
        {
            ShowUserInformation(prm);
        }
    }
```



Dynamic/runtime polymorphism örnek

- Dinamik/çalışma zamanı polimorfizmi aynı zamanda geç bağlanma olarak da bilinir.
- Burada yöntem adı ve yöntem imzası (parametre sayısı ve parametre türü aynı olmalı ve farklı bir uygulamaya sahip olabilir).
- Yöntem geçersiz kılma, dinamik polimorfizmin bir örneğidir.
- Temel sıfıf olarak Shape isminde bir sınıf oluşturualım ve Area isminde bir metodu olsun.
- Circle, Square ve Rectangle sınıflarını Shape sınıfından oluşturup gerçek uygulamlarda Dynamic/runtime polymorphism nasıl çalıştığını inceleyelim.

Shape, Circle

```
4 references
public class Shape
{
    4 references
    public virtual double Area()
    {
        return 0;
    }

}
2 references
public class Circle : Shape
{
    2 references
    public double Radius { get; set; }
    1 reference
    public Circle(double Radius)
    {
        this.Radius = Radius;
    }
    2 references
    public override double Area()
    {
        return (3.14) * Math.Pow(Radius, 2);
    }
}
```

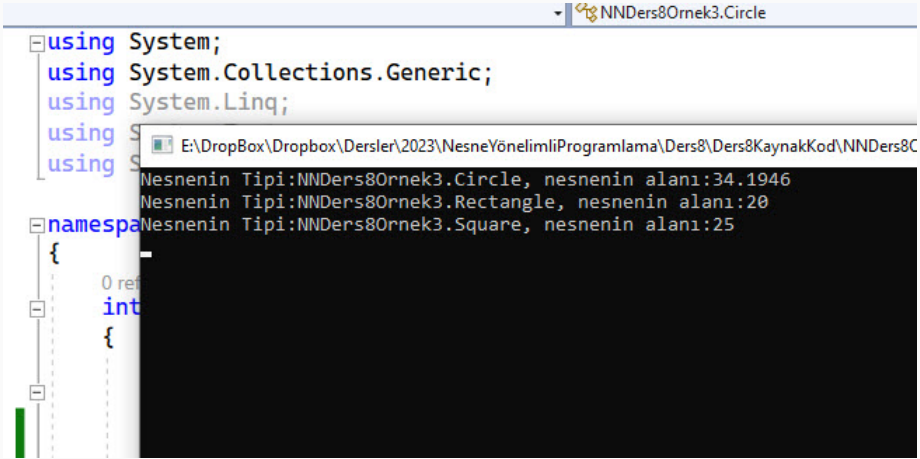
Square, Rectangle

```
2 references
public class Square : Shape
{
    2 references
    public double Length { get; set; }
    1 reference
    public Square(double Length)
    {
        this.Length = Length;
    }
    2 references
    public override double Area()
    {
        return Math.Pow(Length, 2);
    }
}
2 references
public class Rectangle : Shape
{
    2 references
    public double Height { get; set; }
    2 references
    public double Width { get; set; }
    1 reference
    public Rectangle(double Height, double Width)
    {
        this.Width = Width;
        this.Height = Height;
    }
    2 references
    public override double Area()
    {
        return Height * Width;
    }
}
```

Dynamic/runtime polymorphism kodlama

```
0 references
internal class Program
{
    0 references
    static void Main(string[] args)
    {
        // Polymorphism olarak adlandırılır. Listedeki öğelerin Shape türünde olması gerekirken
        //Circle, Rectangle, Square olarak gönderilmesini sağladık ve bunlarda Shape gibi davranıyor.
        var sekiller = new List<Shape>
            {
            new Circle(3.3),
            new Rectangle(4,5),
            new Square(5)
        };
        // Burada ise dinamik Polymorphism çalışır. temel sınıftaki Area metodu değil.
        // Türetilmiş sınıftaki Area metodu çağrılır.
        // Burada kod yazmanın güzelliği hangi sıftan olduğu ile iglenmeden doğru sonucun elde edilmesi.
        foreach (var shape in sekiller)
        {
            Console.WriteLine("Nesnenin Tipi:" + shape.GetType() + ", nesnenin alanı:" + shape.Area());
        }
        Console.ReadKey();
    }
}
```

Dynamic/runtime polymorphism sonuç



Ödev

- Asp.Net Web Application veya Windows Form ile Satranç oyunu tasarlayınız.
- Oyundaki taşların hareket etme yeteneklerini Interface kullanarak gerçekleştiriniz.
- Oyundaki taşların hepsini ortak sınıflardan türetiniz. Ödevleriniz özgün olsun (aynı ödevler sıfır puan).
- Kodlamanızı yaparken **Polymorphism** nasıl kullandığınız açıklayarak kodlayınız.
- Tasarladığınız programda taşın üzerine tıklayınca ne olduğunu ve hangi yönlere hareket edebileceğini göstersin.
- 11 Aralık 07.59'a kadar gönderenler 10+2 puan, finala kadar gönderenler 10 puan (kayubmprogramlama1@gmail.com).