T.C.
MİLLÎ EĞİTİM BAKANLIĞI

BİLİŞİM TEKNOLOJİLERİ

NESNE TABANLI PROGRAMLAMADA TEMSİLCİLER VE OLAYLAR

Ankara, 2013

- Bu modül, mesleki ve teknik eğitim okul/kurumlarında uygulanan Çerçeve Öğretim Programlarında yer alan yeterlikleri kazandırmaya yönelik olarak öğrencilere rehberlik etmek amacıyla hazırlanmış bireysel öğrenme materyalidir.
- Millî Eğitim Bakanlığınca ücretsiz olarak verilmiştir.
- PARA İLE SATILMAZ.

İÇİNDEKİLER

AÇIKLAMALAR

ALAN	**Bilişim Teknolojileri**
DAL/MESLEK	**Veri tabanı Programcılığı**
MODÜLÜN ADI	**Nesne Tabanlı Programlamada Temsilciler ve Olaylar**
MODÜLÜN TANIMI	Bu modül program dilinin temel türleri arasında olan temsilci ve olay kavramlarının öğrenildiği bir öğrenme materyalidir.
SÜRE	40/24
ÖN KOŞUL	"Nesne Tabanlı Programlamada Bileşenler Oluşturma" modülünü tamamlamış olmak
YETERLİK	Program akışını kesmek ve olayları işlemek
MODÜLÜN AMACI	**Genel Amaç** Bu modül ile gerekli ortam sağlandığında; program akışını kesebilecek ve olayları uygulayabileceksiniz. **Amaçlar** **1.**Temsilcileri(delegates) kullanabileceksiniz. **2.**Bildirimleri olaylarla (events) etkinleştirebileceksiniz.
EĞİTİM ÖĞRETİM ORTAMLARI VE DONANIMLARI	**Ortam:** Bilgisayar laboratuvarı **Donanım:** Bilgisayar, programlama yazılımı
ÖLÇME VE DEĞERLENDİRME	Modül içinde yer alan her öğrenme faaliyetinden sonra verilen ölçme araçları ile kendinizi değerlendireceksiniz. Öğretmen modül sonunda ölçme aracı (çoktan seçmeli test, doğru-yanlış testi, boşluk doldurma, eşleştirme vb.) kullanarak modül uygulamaları ile kazandığınız bilgi ve becerileri ölçerek sizi değerlendirecektir.

GİRİŞ

Sevgili Öğrenci,

Okul yaşantınızda öğreneceğiniz her konu, yaptığınız her uygulama ve tamamladığınız her modül bilgi dağarcığınızı geliştirecek ve ileride atılacağınız iş yaşantınızda size başarı olarak geri dönecektir. Eğitim sürecinde daha özverili çalışır ve çalışma disiplinini kazanırsanız başarılı olmamanız için hiçbir neden yoktur.

Bu modülde .Net platformunun temel iki anahtarı olan temsilciler (delegate) ve olaylar (event) konusunu uygulamaları ile birlikte öğreneceksiniz. .NET çatısının hemen hemen her noktasında sıklıkla karşılaştığınız bu iki temel tipi, kendi yazılım tasarımlarınız içinde kullanma beceresine sahip olacaksınız. Aynı zamanda programlarınızda, fonksiyonellik ve esneklik adına oldukça iyi çözümler geliştirebilmenizi sağlayacaktır.

Temsilciler ve olaylar nesne tabanlı programlama dilinin ileri seviye kavramlarındandır. Ancak her programcının bilmesi gereken konularının arasında yer alır. Uygulama geliştirirken karşılaştığımız bir takım senaryolarda, çalışma zamanında program akışını değiştirme ve işleyebilme gereklilikleri ortaya çıkabilir. Bu gibi durumlarda daha fazla fonksiyonel ve daha fazla esnek tasarımları yapabiliyor olmak bir programcı için her zaman avantajlıdır. Temsilciler ve olaylar gibi ileri seviye kavramlar size üstün programlama tekniği, kullanım kolaylığı ve artan bir verimlilik sunar.

ÖĞRENME FAALİYETİ–1

AMAÇ

Temsilcileri sınıf içinde uygun şekilde tanımlayıp kullanabileceksiniz.

ARAŞTIRMA

Bu faaliyet öncesinde yapmanız gereken öncelikli araştırmalar şunlardır.

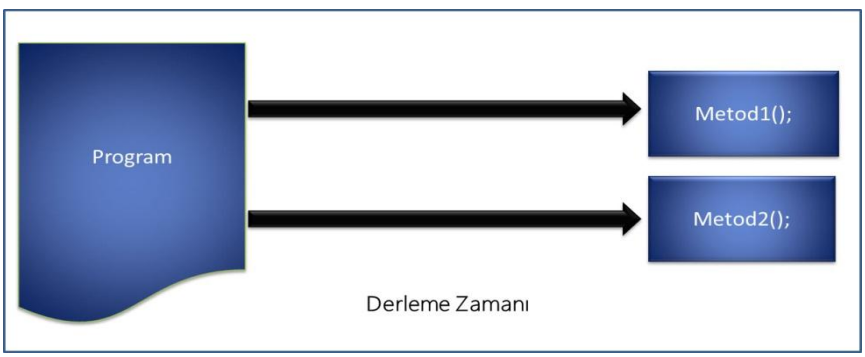
- Metod imzası nedir? Tanımlanmasındaki amaçları araştırınız.
- Arayüz (Interface) kullanımının programınıza kattığı kullanım esnekliğini ve avantajlarını araştırınız (Sınıfta tartışınız.).
- "Derleme zamanı" ve "çalışma zamanı" kavramlarını araştırınız.

1. TEMSİLCİLER

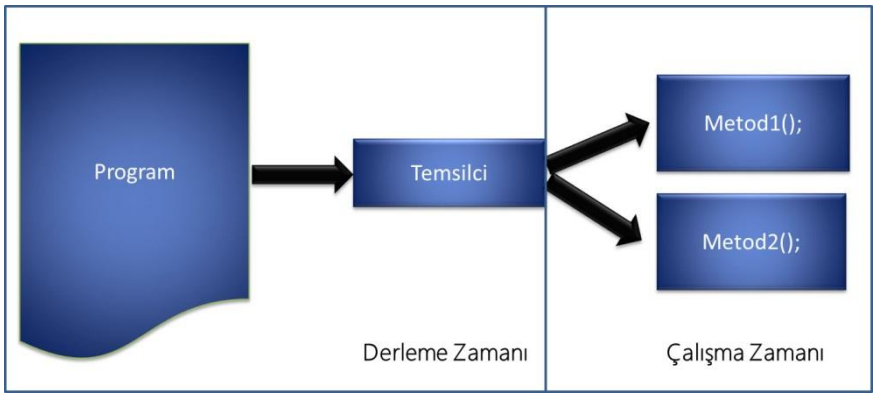
Program içerisinde bir veya daha fazla metodu gösteren (işaret eden), referans türünden nesnelere temsilci (delegate) adı verilir. Başka bir deyişle metotların referansını tutan türler olarak da tanımlayabilirsiniz. Temsilcinin kelime anlamı "aracılık yapan"dır. Temsilci .NET için bir metoda aracılık yapmaktadır.

Referans bilgisi, nesnenin bellekteki yerini gösteren bir adres değişkenidir. Alt düzeyli dillerde bu değişken işaretçi (pointer) olarak da bilinir. Temsilcinin taşıdığı referans bilgisi nesneyi değil, bir metodun bellekte oluşturulduğu yerin başlangıç adresini işaret eder.

Söz konusu metodu, temsilci vasıtasıyla referansı üzerinden çalışma zamanında görevini yerine getirmek üzere çağırabiliriz. İlk bakışta programınız içinde metodu direkt olarak çağırmak yerine, neden temsilci kullanma gereği duyulduğunu düşünebilirsiniz. Bu sorunun cevabı için aşağıdaki şekilleri inceleyiniz.



Resim 1.1: Temsilci kullanılmadan yapılmış metod çağrımları



Resim 1.2: Temsilci vasıtasıyla yapılmış metod çağrımları

Aslında buradaki kritik kelimenin "derleme zamanı" değil "çalışma zamanı" olduğunu söyleyebiliriz. Program yazımı aşamanızda belirlediğiniz bir metodu ne zaman çağırmanız gerektiğine rahatlıkla karar verebilir, kodunuz içinde sorunsuzca çağrımı yapabilirsiniz. Bütün bu süreci derleme aşamasında düşündüğünüz anlamına gelir. Bu kullanım resim 1.1'de gösterilmiştir. Ancak bazı senaryolarda hangi metodun işletileceği kararı, programınızın çalışma zamanında ele alınmasını gerektirebilir. Bir temsilci tanımlaması ile derleme zamanı için bir metodun çalıştırılacağı bilgisini programınıza vermiş olursunuz. İşaret edeceği metodu çalışma zamanında belirleyebilir ve dinamik olarak değiştirebilirsiniz.

- **Temsilciler neden kullanılır**

Bir temsilci ile sadece bir tek metodu işaret edebileceğiniz gibi, birden fazla metot için tanımlanmış ve oluşturulmuş temsilcileri de kullanabilirsiniz. Bir temsilcinin tanımlanması, onu kullanmak için yeterli değildir elbette. Her şeyden önce bir amacınız olmalıdır. Bir temsilciyi çalışma zamanında oluşturabilir ve kullanabilirsiniz. Diğer yandan, tek bir temsilcide birden fazla temsilciyi toplayarak bu temsilcilerin işaret ettiği, tüm metotları tek bir seferde çalıştırma lüksüne de sahipsiniz. Ancak temsilciler gerçek anlamda iki amaçla kullanılırlar. Bunlardan birincisi olaylardır (events).

Olaylar .NET için tıpkı temsilciler gibi temel kavramların arasında yer alır. Temsilciler olayların desteklenmesinde önemli bir rol oynar. Diğer yandan, bir metodun çalışma zamanında, hangi metotların çalıştırılacağına karar vermesi gerektiği durumlarda kullanılır.

Elbette söz konusu bu amaçları, herhangi bir temsilciye ihtiyaç duymadan da gerçekleştirebiliriz. Ancak temsilcileri kullanmadığınızda, size sağladığı üstün programlama tekniği, kullanım kolaylığı ve artan verimliliği de göz ardı etmiş olursunuz.

1.1. Temsilcileri (Delegates) Tanımlama

C# programlama dilinde metotların referansını tutan özel bir tip geliştirilmiştir. Bu tip "delegate" anahtar kelimesi ile tanımlanır.

- Yazım kuralı

erişim_belirteci delegate *metodun_geri_dönüş_tipi* temsilci_adı *(Parametre_listesi)*

Yazım kuralı dikkatlice incelendiğinde bir temsilci tanımlamasının metot yazım formuna olan benzerliği dikkatinizi çekebilir. Tek fark öncesinde "delegate" anahtar kelimesinin kullanılmasıdır.

Her veri tipinde olduğu gibi temsilcide bir erişim belirtecine (private, protected vb.) sahiptir. Geri dönüş tipi temsilcinin işaret etmek istediği metotların döndürdüğü değerin tipidir. Parametreler, temsil edilen metodun almak isteyebileceği parametre listesidir.

Tanımlama aşamasında dikkat edilmesi gereken temsilci nesneleri ile göstermek yani işaret etmek istediğimiz metotlar temsilcinin imzasına sahip olmalıdır. Bildiğiniz gibi metot imzaları, metodun geri dönüş tipi ve aldığı parametreler ile belirlenmektedir.

Örnek 1.1.

```
public delegate int toplamTemsilcim(int x, int y);
```

Yukarıda iki parametreli ve parametre tipleri ile geri dönüş tipi tamsayı olan "toplamTemsilcim" isimli bir temsilci tanımlaması yapılmıştır.

Örnek 1.2.

```
private delegate void getirTemsilcim(string kelime);
```

Yukarıda geri dönüş değeri olmayan, string türünden tek parametreli "getirTemsilcim" isimli bir temsilci tanımlaması yapılmıştır.

Örnek 1.3.

```
private delegate string editHandler(params string[] liste);
```

Yukarıda geri dönüş değeri "string" olan ve "string" türünden değişken sayıda parametre alan "editHandler" isimli bir temsilci tanımlaması yapılmıştır.

Temsilciler aslında bir metot tanımlarlar fakat bunu uygulamazlar. İşte bu özellikleri ile arayüzlerdeki metot tanılamalarına benzerler.

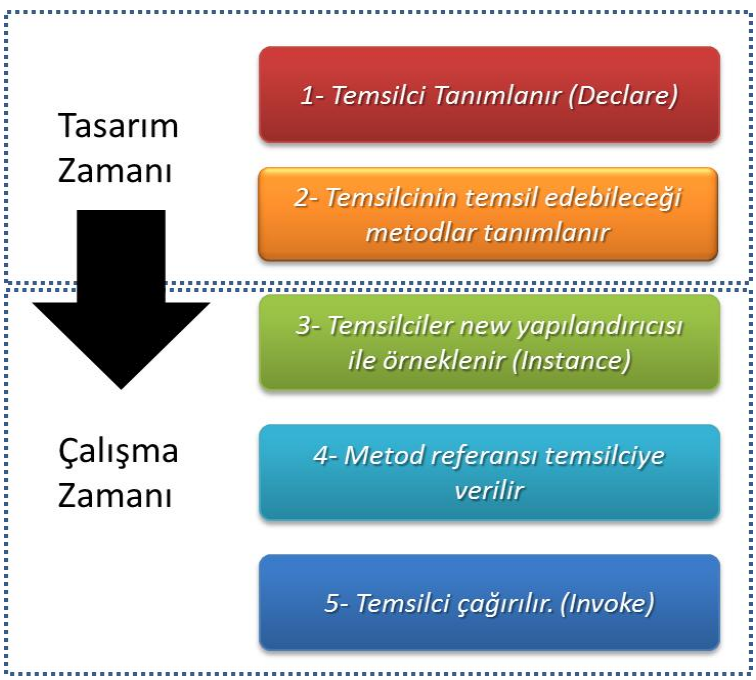
1.2. Temsilcileri Kullanma

Programlarınızda temsilciler kullanmak istediğinizde, öncelikle bu temsilcinin tanımı yapılır. Bir temsilci tanımlandığında, aslında işaret edebileceği metotların imzalarını da belirlemiş olur. Dolayısıyla, bir temsilci sadece tanımladığı metot imzasına uygun metotlar için kullanılabilir.

Temsilci tanımları tasarım zamanında yapılır. Bir temsilciyi, bir metodu işaret etmesi için kullanmak istediğimizde ise, çalışma zamanında onu "new" yapılandırıcısı ile oluşturur ve işaret etmesini istediğimiz metodu ona parametre olarak veririz.

Temsilci kullanımı 3 aşamada yapılır.
1. Tanımlama (Declare).
2. Örneklenme (Instance)
3. Çağırma (Invoke)

- Tanımlama aşaması : Temsilci referans alacağı metotların imzalarına uygun şekilde "delegate" anahtarı ile tanımlanır.
- Örnekleme : Tanımlaması yapılan temsilci artık bir veri türü olarak kabul edilebilir. Temsilci adı üzerinden "new" anahtarı kelimesi ile örneklemesi yapılır.
- Çağırma : Temsilciyi çağırma işlemi metod çağrımından farksızdır. Temsilci imzasına uygun şekilde parametreleri verilir, geri dönüş türüne uygun bir nesneye, dönen değeri aktarılabilir.



Resim 1.3: Temsilerin kullanım süreçleri

Temsilci kullanım süreçlerinin bir kısmı derleyici tarafından derleme zamanında, bir kısmı işletim sistemi tarfından çalışma zamanında ele alınır. Resim 1.2.1'de kullanım süreçlerinin hangi zamanlarda işletildiğini göstermektedir.

Örnek 1.4. Temsilcilerin bir program içinde nasıl ele alındıklarını ve bir metodu nasıl çalıştırdıkları görülmektedir.

```
namespace DelegateTest1
{
   class Program
   {
      // Tanımlama (Declaration)
      public delegate void TestHandler();

      public static void Metod1()
      {
         Console.WriteLine("Temsilci ile çağırıldım ...");
      }

      static void Main(string[] args)
      {
         // Örnekleme (Instantiation)
         TestHandler test = new TestHandler(Metod1);

         // Çağırma (Invocation)
         test();
      }
   }
}
```

Yukarıdaki program kodu temsilcilerin kullanım aşamalarına göre incelendiğinde;

1-Tanımlama aşaması: public delegate void TestHandler();

Hiçbir parametresi ve geri dönüş değeri olmayan metotları işaret edecek "TestHandler" adında temsilci tanımlaması (declaration) yapılır.

2-Örneklenme aşaması: TestHandler test = new TestHandler(Metod1);

Main() bloğu altında ilk olarak TestHandler tipinde "test" adında bir nesne "new" yapılandırıcısı ile örneklenir. Örnekleme (Instance) aşamasında nesnenin "Metod1" isimli yöntemi işaret edeceği belirtilir.

3-Çağırma aşaması: test();

Son olarak "test" isimli temsilci işaret ettiği metodu çalıştırmak üzere çağırımı (Invoke) yapılmıştır.

Program çalıştırıldığında oluşan ekran çıktısı Resim 1.4'teki gibi olacaktır.

Resim 1.4: Program kodu ekran çıktısı

Örnek 1.5. Aşağıdaki örnek programımızda "Topla" ve "Fark" adında iki farklı metodun tek bir temsilci üzerinden ayrı zamanlarda çalıştırılmaktadır.

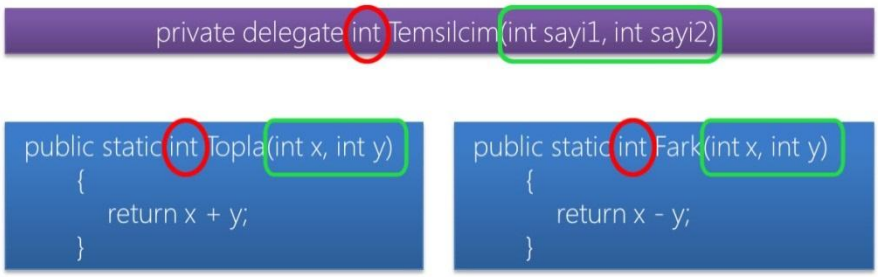
```
namespace DelegateTest2
{
   class Program
   {
      public static int Topla(int x, int y)
      {
         return x + y;
      }

      public static int Fark(int x, int y)
      {
         return x - y;
      }

      // Tanımlama
      private delegate int Temsilcim(int sayi1, int sayi2);

      static void Main(string[] args)
      {
         // Temsilci örneklemesi "Topla()" metodunu işaret eder.
         Temsilcim isTemsil = new Temsilcim(Topla);
         // Temsilci örneklemesi üzerinden metodun çağırılması
         int toplamSonuc = isTemsil (8, 5);
         Console.WriteLine("Sonuc :{0}", toplamSonuc);
         // Temsilci örneklemesi "Fark()" metodunu işaret etmektedir
         t = Fark;
         // Temsilci örneklemesi üzerinden metodun çağırılması
         int farkSonuc = isTemsil (8, 5);
         Console.WriteLine("Sonuc :{0}", farkSonuc);
         Console.ReadLine();
      }
   }
}
```

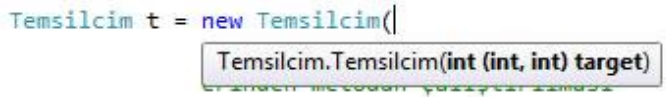
Yukarıdaki örneğimizde "Topla" ve "Fark" metotlarını referans alacak "Temsilcim" adında tek bir temsilci tanımlaması yapılmıştır. Yazım kuralı gereği tanımlama aşamasında temsilcimizin çalışma anında çağıracağı metotlar ile aynı imzaya sahip olduğuna dikkat ediniz. Bu durum resim 1.5'te gösterilmiştir.



Resim 1.5: Temsilci ve metot imzaları

Metot imzası ile temsilci imzası arasında uyuşmazlık halinde " *....' has the wrong return type*" şeklinde bir derleme zamanı hatası alabilirsiniz.

Kodlama sürecinde örneklemeyi yazarken Resim 1.6'daki görüntü dikkatinizi çekmiştir.



Resim 1.6: Temsilci örneklemesine ait yapılandırıcı parametreleri

Temsilcim tipindeki **t** nesnesi "new" anahtarı ile oluşturulurken temsilcinin sahip olduğu imza türünden bir metot talepte bulunduğu dikkate alınmalıdır.

Çağırma aşamasında sonuç değişkenlerimize;

```
int toplamSonuc = isTemsil (8, 5);  // Sonuc : 13
int farkSonuc = isTemsil (8, 5);      // Sonuc :3
```

şeklinde aynı atamalar yapılmış olmasına karşın çalışma zamanında farklı değerlerin üretildiği görülmektedir. Bunun sebebi "farkSonuc" değişkenine temsilciden dönen değer verilmeden önce **t = Fark;** satırı ile temsilcimiz artık "Topla" metodu yerine "Fark" metodunu referans alması ve çalıştırmasıdır.

Örnek 1.6. Bir önceki örnekte bulunan "Topla" ve "Fark" metotlarımızı ayrı bir sınıf altında toplayalım. Metot çağrımlarını temsilci üzerinden gerçekleştirelim.

Öncelikle "TestClass" isimli bir sınıf oluşturunuz ve söz konusu metotları yazınız.

```
namespace DelegateTest3
{
   class TestClass
   {
      public int Topla(int x, int y)
      {
         return x + y;
      }

      public int Fark(int x, int y)
      {
         return x - y;
      }
   }
}
```

Sırada TestClass sınıfını kullanacak program kodlarını yazınız.

```
namespace DelegateTest3
{
   class Program
   {
      private delegate int Temsilcim(int sayi1, int sayi2);

      static void Main(string[] args)
      {
         // TestClass sınıfından test nesnesinin örneklenmesi
         TestClass test = new TestClass();
         // Temsilci örneklemesi üzerinden metodun çağırılması
         Temsilcim t = new Temsilcim(test.Topla);
         // Temsilci üzerinden metodun çalıştırılması
         int toplamSonuc = t(8, 5);
         Console.WriteLine("Sonuc :{0}", toplamSonuc);
         // Temsilci Fark() metodunu işaret etmektedir
         t = test.Fark;
         // İşaret edilen metodun çalıştırılması
         int farkSonuc = t(8, 5);
         Console.WriteLine("Sonuc :{0}", farkSonuc);
      }
   }
}
```

TestClass sınıfına ait "Topla" ve "Fark" metotlarının kullanımı için sınıfa ait nesne örneğine ihtiyaç duyar. Bir temsilci, kendisiyle ilişkili, statik, statik olmayan veya başka bir sınıf örneğinden gelen yeni bir metotla kullanılabilir.

Referans türleri aynı zamanda **"null"** değerler taşıyabilir.

```
Temsilcim t = new Temsilcim(test.Topla);
t=null;   // Geçerli
int sonuç = t(8,5);   // Temsilci "null" olduktan sonra geçersiz
```

Yukarıda bulunan kod örneğinde Temsilcim tipindeki **t** nesnesine null değer ataması yapılmıştır. Null değer taşıyan temsilci, herhangi bir metodu işaret etmez. Bu durumda olan temsilcinin çağırılması çalışma zamanında "*Object reference not set to an instance of an object*." hatasını verecektir.

Şimdiye kadar yaptığınız örneklerde temsilcinin tek bir metodu referans aldığını gördünüz. Bu tip kullanım şekline "Single-Cast" (Tekli Çağrı) adı verilir. Ancak programlarımızda bazen, tek bir temsilciye birden fazla temsilci ekleyerek, birden fazla metodu tek bir temsilci ile çalıştırmak isteyebiliriz. Bu durumda "Multi-Cast" (Çoklu Çağrı) temsilciler tanımlarız. Çoklu çağrı kullanım şeklinde yapmanız gereken metot eklemek ise "+=" metot çıkarmak ise "-=" operatörlerini kullanmaktır. Temsilci çağrımı yapıldığında kendi referans listesinde tuttuğu tüm metotların çağrımını sırayla gerçekleştirecektir.

Örnek 1.7. Çoklu çağrı için temsilciler üzerinde "+=" ve "-=" operatörlerinin kullanımı aşağıda gösterilmiştir.

```
namespace MultiCastDelegate
{
   class Program
   {
      static void Metod1()
      {
         Console.WriteLine("Metod1 çalışıyor...");
      }
      static void Metod2()
      {
         Console.WriteLine("Metod2 çalışıyor...");
      }
      static void Metod3()
      {
         Console.WriteLine("Metod3 çalışıyor...");
      }
```

```
        private delegate void Temsilcim();

        static void Main(string[] args)
        {
            Temsilcim t = new Temsilcim(Metod1);
            t(); // Temsilci Metod1'i referans alır.
            Console.WriteLine("**********************");
            t += Metod2; // Temsilci listesine Metod2 eklenmektedir.
            t += Metod3; // Temsilci listesine Metod3 eklenmektedir.
            t(); // Temsilci Metod1+Metod2+Metod3 çağırır.
            Console.WriteLine("**********************");
            t -= Metod2; // Temsilci listesinden Metod2 çıkarılmaktadır.
            t(); // Temsilci Metod1+Metod3 çağırır.
            Console.WriteLine("**********************");
            t -= Metod1; // Temsilci listesinden Metod1 çıkarılmaktadır.
            t(); // Temsilci yalnızca Metod3'i çağırır.
            Console.ReadLine();
        }
    }
}
```

Örneğe ait program kodu çalıştırıldığında resim 1.7'de görülen ekran çıktısını verecektir.

```
Metod1 çalışıyor...
**********
Metod1 çalışıyor...
Metod2 çalışıyor...
Metod3 çalışıyor...
**********
Metod1 çalışıyor...
Metod3 çalışıyor...
**********
Metod3 çalışıyor...
```

Resim 1.7: Program testi sonucu

Örnek 1.8. Bir otomasyon projesi için bir cihazın aç / kapa işlemini tek fonksiyonla gerçekleştirme ihtiyacımız bulunmaktadır. Bu fonksiyonu kodlayınız.

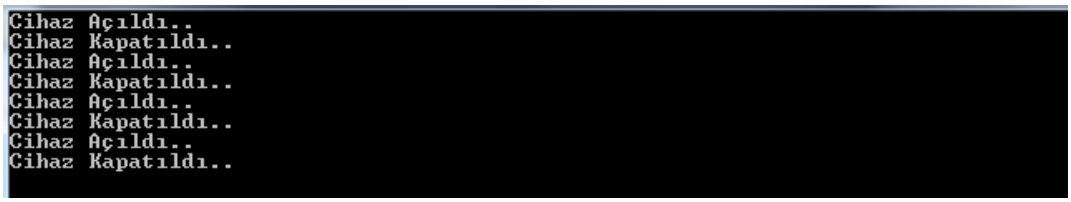
```
namespace DelegateTest4
{
    public class Cihaz
    {
        private delegate void Temsilcim();
        Temsilcim faaliyet;
```

```
        public Cihaz()
        {   // Kurucu metod altında temsilci Ac metoduna işaret ediyor.
            faaliyet=new Temsilcim(Ac);
        }
         private void Ac()
        {   //Cihaz açılmasının ardından temsilci Kapat metodunu işaret ediyor.
            Console.WriteLine("Cihaz Açıldı..");
            faaliyet=Kapat;
        }
        private void Kapat()
        {  // Cihaz kapatılmasının ardından temsilci Ac metodunu işaret ediyor.
            Console.WriteLine("Cihaz Kapatıldı..");
            faaliyet=Ac;
        }
        public void Kontrol()
        {  //Sadece kontrol metodu temsilciyi çağırabilir.
            faaliyet();
        }
    }
}
```

Cihaz sınıfımız içinde bulunan "Ac" ve "Kapat" metodları private erişim belirtecine sahiptir. Sınıfa özgü bu iki metodu "Kontrol" adında bir metod harekete geçirecektir. Bunun için "faaliyet" temsilcisi kurucu metod altında örneklenmekte ve "Kontrol" metodu altında çağrılmaktadır. Cihaz açık konumda iken temsilci "Kapat metodunu, kapalı konumda iken "Ac" metodunu işaret etmektedir.

Sınıfımızı test etmek için aşağıdaki kodları yazınız ve çalışmasını izleyiniz.

```
namespace DelegateTest4
{
   class Program
   {
      static void Main(string[] args)
      {
         Cihaz ch=new Cihaz();
         for(int j=0;j<=7;j++)
         {   // Cihazın kontrol fonksiyonu çağrılıyor.
            ch.Kontrol();
         }
         Console.ReadLine();
      }
   }
}
```



Resim 1.8: Program testi sonucu

Temsilciler (delegates) çok kanallı programlamada (multi threading), asenkron (asynchronous) mimarilerde (Polling, Callback, WaitHandle gibi) ve son olarak olay tabanlı (event based) kodlamada kullanılmaktadır. Modülünüzün sunduğu bilgiler ışığında söz konusu kullanım alanlarında temsilcilerin oynadığı rolü, programlama tekniklerini, ve sunduğu avantajları derinlemesine incelemeniz sizi daha ileriye taşıyacaktır.

UYGULAMA FAALİYETİ

Aşağıdaki işlem basamaklarını takip ederek faaliyeti gerçekleştiriniz.

İşlem Basamakları	Öneriler
➢ Bir temsilci türü tanımlayınız. ➢ Temsilcinin çağıracağı türden 3 farklı metot yazınız. ➢ Temsilcinin bir örneğini oluşturunuz. ➢ Temsilciyi kullanarak metotları sıra ile çağırınız.	➢ Temsilciniz geri dönüş tipi ve parametreli olmalıdır. ➢ Metotların çalışmalarını sağlıklı izleyebilmek için mesaj verdirmesini sağlayınız. ➢ Tekli ve çoklu çağrımlarda bulununuz.
➢ Sözcük (string) dizisini, her çağrıldığında önce A-Z, sonra Z-A sıralamasını yaptıracak bir temsilci metodu oluşturunuz.	➢ Örnek 1.8'den faydalanabilirsiniz. ➢ Metot çağrımları temsilcisi üzerinden gerçekleştirilecektir. ➢ Sıralama yapacak olan metotların geri dönüş tipi void olmalıdır. ➢ Sıralanan dizi metodun "ref" anahtar kelimesi kullanılan bir parametresi ile elde edilecektir. ➢ Sıralanmamış diziyi ekrana yazdırınız.

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki cümlelerde boş bırakılan yerlere doğru sözcükleri yazınız.

1. Temsilciler …….. referansını gösteren türlerdir.
2. Referans bilgisi nesnenin bellekteki ……. değişkenidir.
3. Temsilcileri kullanmadan yapılan metot çağrımları…… zamanında gerçekleşir.
4. Temsilciler kullanılarak yapılan metot çağrımları ……. zamanında gerçekleşir.
5. Bir temsilci …… anahtar kelimesi ile tanımlanır.

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki öğrenme faaliyetine geçiniz.

ÖĞRENME FAALİYETİ–2

AMAÇ

Olayları sınıf içinde uygun şekilde tanımlayıp kullanabileceksiniz.

ARAŞTIRMA

- Temsilcilerin program akışını değiştirmede oynadığı rolü araştırınız.

2. OLAYLAR

Varlıklar doğaları gereği kendileri ile ilgili oluşan durumları olay olarak algılar. Olay meydana geldiğinde bunu bildirmek isteyebilir. Bunu diğer varlıklarla etkileşime geçerek veya kendi varlıklarında başka bir davranışı tetikleyerek sağlar.

Neticede programlarımız içindeki nesnelerimizin gerçeklikteki varlıklara karşılık geldiğini düşünecek olursanız olayların ve buna bağlı bildirimlerin de sınıflarımız tarafından işlenebilmesi mümkün olmalıdır.

Olaylar (event) sınıfın (class) içinde oluşan bir durumun meydana geldiğini bildiren otomatik bildirgelerdir. .Net Framework'te olaylar önemli faaliyetleri yakalamanıza ve durumu işlemek için temsilcisi ile birlikte sınıflarınıza destek verir.

Olayların sınıflarınız içinde ele alınması 3 aşamadan meydana gelmektedir.

- Olay tanımlama
- Olaya abone olma
- Olayı çağırma

2.1. Olay (Event) Tanımlama

Olaylar bir sınıfın üyesi olup "event" anahtar kelimesi ile tanımlanırlar.

- Yazım kuralı;

erişim_belirteci event olay_temsilcisi olay_adı

Olay onun çalışmasını destekleyecek bir temsilci ile birlikte tanımlanmaktadır. Söz konusu temsilcimiz olay meydana geldiğinde temsilcisinin işaret ettiği metodu çağırarak bildirimde bulunurlar. Bir olayın (event) tanımlanabilmesi için mutlaka bir temsilci tipi ile eşleştirilmesi gerekmektedir.

Örnek 2.1: Ürün sınıfında stok tükenmesi olay tanımlamasına ait örnektir.

```
class Urun
   {
      private delegate void StokTemsilcisi();
      public event StokTemsilcisi StokBitti;
      ......
   }
```

İlk satırda olayın işlenmesini sağlayacak temsilcinin tanımlaması yapılmıştır. "StokTemsilci" geri dönüş tipi ve parametresi olmayan bir metodu referans alabilecek şekilde düzenlenmiştir. Ardından temsilcisi ile birlikte çalışacak olan "StokBitti" olayının tanımlaması yapılmıştır.

Programlarınız için de kullanmak istediğiniz sınıflarınızda bu türden kodlama ile rahatlıkla olay tanımlamalarını gerçekleştirebilirsiniz.

2.2. Olaya Abone Olma

Aslında bir olayın tetiklenmesi, bir istisna (exception) nesne örneğinin ortama fırlatılmasına (throw) benzetilebilir. Tek fark ortama fırlatılan istisnaların "catch" blokları ile yakalanabiliyor olmasıdır. Olaylarda durum farklıdır. Ortada bir catch bloku yoktur. Bunun yerine bir abone (subscriber) vardır. Basit olarak olayın tetiklenmesi sonucu çalıştırılacak olay metodunun bulunduğu nesne örneğini abone olarak düşünebiliriz. Bir başka deyişle olayı yakalayıp değerlendirecek olan nesne, olayın sahibi olan nesnenin ilgili olayına (event) abone olmaktadır. Bu durum resim 2.1'de görebilirsiniz. Dolayısıyla olayı tanımlayan ve tetikleyen nesneyi yayımcı (publisher) olarak da düşünebilirsiniz.

Resim 2.1: Olay metodunun yayımcı nesneye abone olması

Burada söz konusu olan abone etme işlemi aslında olayın (event) "+= " operatörü yardımıyla program nesne örneği içerisinde yüklenmesidir. Olaylar tanımlanırken hep bir temsilci (delegate) tipi yardımıyla oluşturulurlar. Dolayısıyla abone olan sınıf içerisinde yayımcı sınıf nesnesi için ilgili olay += operatör ile yüklendiğinde, temsilciye (delegate) parametre olarak verilen metot referansı yayımcı nesnesine bildirilir. Böylece yayımcı nesne örneği içerisinde ilgili olay tetiklendiğinde program içinde hangi metodun çağrılacağı bilinmektedir.

Örnek 2.2. "StokAzaldı" isimli olayın abone sınıf tarafından olay metoduna bağlayan program örneğidir.

```
public class Urun
   {
      public delegate void StokTemsilcisi();
      public event StokTemsilcisi StokAzaldi;
      ……
      ……
   }

class Program
   {
```

```
static void Main(string[] args)
{
// Urun sınıfından kitap nesnesinin örneklenmesi
Urun Kitap = new Urun();
// Kitap nesnesine dair stok azalması meydana geldiğinde
// Kitap_StokAzaldi() isimli olay metodunun temsilciye referans edilmesi
Kitap.StokAzaldi += new Urun.StokTemsilcisi(Kitap_StokAzaldi);
}
static void Kitap_StokAzaldi()
{
// Olay meydana geldiğince çalışacak olan metod
}
}
```

Temsilcilerin çoklu çağrıyı (multicasting) desteklediğini düşünürseniz bir tek olaya dilediğiniz kadar olay metodu bildirebilirsiniz.

Örnek 2.3. Kitap nesnemiz stoklarda tükenme noktasına geldiğinde tedarikçimize sipariş verilmesi, mail gönderilmesi ve rapor yazması istenmektedir.

```
class Program
   {
      static void Main(string[] args)
      {
         Urun Kitap = new Urun();
         Kitap.StokAzaldi += new Urun.StokTemsilcisi(Siparis_Ver);
         Kitap.StokAzaldi += new Urun.StokTemsilcisi(Mail_Gonder);
         Kitap.StokAzaldi += new Urun.StokTemsilcisi(Rapor_Yaz);
         ……
      }
      static void Siparis_Ver()
      {
         // Sipariş kayıtlarına geçilecek...
      }
      static void Mail_Gonder()
      {
         // Tedarikçiye mail gönderilecek...
      }
      static void Rapor_Yaz()
      {
         // Rapor yazdırılacak...
      }
      ……
   }
```

2.3. Abonelik İptali

Olay aboneliğini iptal etmek, temsilciyi bağlandığı olay metodundan ayırmak anlamına gelir. Temsilciler ünitesinden hatırlayacağınız üzere "-=" operatörü ile daha önce abone edilmiş olay metodunuzu temsilcinin referans listesinden kaldırabilirsiniz.

2.4. Olay Çağırma

Olayları çağırmak temsilcileri çağırmaktan farksızdır. Çağrımı yaptığınız zaman olay tanımına ilişkilendirilmiş temsilci üzerinden abone olan tüm metotlar sırayla çağrılır. Burada dikkat edilmesi gereken husus, çağrım yapılmadan önce olay temsilcisinin herhangi bir olay metodunu işaret ettiğinden emin olmaktır. Temsilcilerin herhangi bir metodu işaret etmediklerinde "null" değere sahip olduğunu unutmayınız. Aksi halde çalışma zamanında

"NullReferenceException" hatası alırsınız. Olay çağrımından önce "null" değer karşılaştırmasının yapılması gerekmektedir.

```
if (StokAzaldi != null) // Olay Temsilcisi boş değilse...
{
   StokAzaldi(); // Olayı çağır.
}
```

Olayın tetiklenmesi yalnızca olay tanımının yapıldığı sınıf içinde gerçekleştirilebilir.

Örnek 2.4. Stok ambarlarında tutulan ürünlerin hareketlerini takip edecek bir proje geliştirmeniz istenmektedir. Buna göre herhangi bir ürünün stok miktarının belirli bir değerin altına düşmesi sonrasında kullanıcınıza uyarı verecek şekilde tasarlanacaktır. Programı konsol ortamında gerçekleştiriniz.

Konsol uygulamanız içerisinde yer alan "program" sınıfınız abone (subscriber) olacak. Ürün sınıfı içerisinde tanımlayacağımız olay (event), stoktaki ürün sayısı 10 değerinin altına düştüğünde tetiklenecek şekilde tasarlanacaktır. Bir olay tanımlanırken mutlaka bir temsilcinin olması gerektiğinden bahsetmiştik. Dolayısıyla bir de temsilci (delegate) tipi geliştirmemiz gerekecektir. Söz konusu temsilci tipini ve Ürün sınıfını aşağıdaki gibi tasarlayabilirsiniz.

```
namespace StokTakip
{
   public class Urun
   {
     // Parametresi ve geri dönüş tipi olmayan olay temsilcisinin tanımlanması
       public delegate void StokAzaldiEventHandler();

     // Olay temsilcisi üzerinden event tanımlanması
       public event StokAzaldiEventHandler StokAzaldi;
```

```
        private int _id;
        public int Id
        {
            get { return _id; }
            set { _id = value; }
        }
        private string _ad;
        public string Ad
        {
            get { return _ad; }
            set { _ad = value; }
        }
        private double _birimFiyat;
        public double BirimFiyat
        {
            get { return _birimFiyat; }
            set { _birimFiyat = value; }
        }
        private int _stokMiktari;
        public int StokMiktari
        {
            get { return _stokMiktari; }
            set
            {
                _stokMiktari = value;
                if (value < 10 && StokAzaldi != null)
                {  // StokMiktarı 10'dan küçükse ve StokAzaldı olayına bağlı temsilci null
                   // değer içermiyorsa StokAzaldı olayını çağır
                   StokAzaldi();
                }
            }
        }
        public Urun(int id, string adi, double fiyati, int stokSayisi)
        {   // Kurucu metod (Constructor)
            Id = id;
            Ad = adi;
            BirimFiyat = fiyati;
            StokMiktari = stokSayisi;
        }
        public override string ToString()
        {  // .ToString metoduna ait görev tanımının yeniden yapılması
            return this.Ad+" için stok miktarı "+this.StokMiktari.ToString();
        }
    }
}
```

İlk olarak "StokAzaldiHandler" tipinden bir temsilci (delegate) tanımlıyoruz..Net içerisinde var olan isimlendirme standartlarına göre niteliklere ait sınıf adlarının "Attribute" kelimesi ile, istisna (exception) tiplerinin "Exception" kelimesi ile, özellikle olaylar ile ilişkili temsilcilerinde çoğunlukla "EventHandler" kelimesi ile bittiğini görürsünüz. Bu nedenle olay ile ilişkili temsilcimizi "StokAzaldiEventHandler" olarak isimlendirdik. "StokAzaldiEventHandler" isimli temsilci tipi, geriye değer döndürmeyen ve parametre almayan metotları işaret edebilecek şekilde tasarlanmıştır.

```
public delegate void StokAzaldiEventHandler();
```

Ardından UrunAzaldi isimli bir olay (event) tanımlanmıştır.

```
public event StokAzaldiEventHandler StokAzaldi;
```

Dikkat edilecek olursa "event" anahtar kelimesinden sonra "StokAzaldiEventHandler" isimli temsilci tipi gelmektedir. Son olarak da olayın adı yer alır. Böylece söz konusu olay için çalıştırılabilecek olay metotlarının yapısını "StokAzaldiEventHandler" isimli temsilcinin işaret edeceği belirtilmiş olur. Sırada, ilgili olay metodun nasıl ve nerede tetikleneceğinin belirlenmesi gerekmektedir. Bunun için "StokMiktari" isimli özelliğin set bloğunda aşağıdaki kod parçası kullanılmıştır.

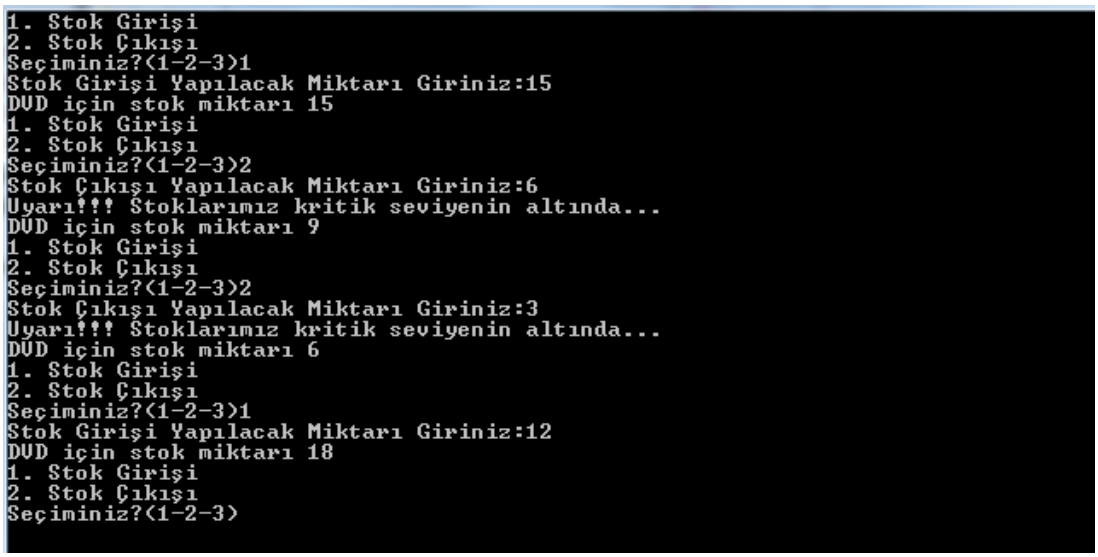
```
public int StokMiktari
    {
      get { return _stokMiktari; }
      set
      {
        _stokMiktari = value;
        if (value < 10 && StokAzaldi != null)
        {
          StokAzaldi();
        }
      }
    }
```

Burada stok miktarı eğer 10 rakamının altındaysa ve "StokAzaldi" olayı null değere eşit değilse "StokAzaldi" isimli bir metot çağrısı yapılmaktadır. "StokAzaldi" olayının null olmaması bir şekilde += operatörü ile yüklendiği anlamına gelmektedir. Yani başka bir nesne bu olaya kendisini abone (subscribe to) etmiştir. Bu durumda söz konusu olay metodunun buradaki set bloku içerisinden çağırılması gerekir. Bu iş için yine olayı sanki bir metot gibi çağırmak yeterli olacaktır.

Projemiz gereği ürün nesnelerimizi tüketecek olan program sınıfını aşağıdaki gibi tasarlayabilirsiniz.

```
namespace StokTakip
{
   class Program
   {
      static void Main(string[] args)
      {   // Urun sınıfından dvd nesnesinin örneklenmesidir.
         Urun dvd = new Urun(10001, "DVD", 1.20, 0);
          // Stoklarda bulunan dvd azaldığında çalışacak olayın metoda bağlanır.
         dvd.StokAzaldi += new Urun.StokAzaldiEventHandler(dvd_StokAzaldi);
         while (true)
         {
            Console.WriteLine("1. Stok Girişi");
            Console.WriteLine("2. Stok Çıkışı");
            Console.Write ("Seçiminiz?(1-2-3)");
            int secim = Convert.ToInt32(Console.ReadLine());
            if (secim == 1)
            {  // Stoğumuza urun girişi yapılmak istenmesi durumunda
               // urun miktarını verilen sayıda arttırma işlemidir.
               Console.WriteLine("Stok Girişi Yapılacak Miktarı Giriniz:");
               int miktar = Convert.ToInt32(Console.ReadLine());
               dvd.StokMiktari += miktar;
               Console.WriteLine(dvd.ToString());// Urun hakkında bilgi verir.
            }
            else if (secim == 2)
            {  // Stoğumuzdan urun tüketimi yapılmak istenmesi durumunda
               // urun miktarını verilen sayıda azaltma işlemidir.
               Console.Write("Stok Çıkışı Yapılacak Miktarı Giriniz:");
               int miktar = Convert.ToInt32(Console.ReadLine());
               dvd.StokMiktari -= miktar;
               Console.Write(dvd.ToString());// Urun hakkında bilgi verir.
            }
            else if (secim == 3)
            {  // Programın çıkış seçeneğidir.
               break;
            }
         }
      }
      static void dvd_StokAzaldi()
      {   // Stok kritik seviyenin altına düştüğünde
         // olayın çalıştıracağı metod gövdesidir.
         Console.WriteLine("Uyarı!!! Stoklarımız kritik seviyenin altında...");
      }
   }
}
```

Main metodu içerisinde "Urun" sınıfına ait bir nesne örneklendikten sonra "StokAzaldi" isimli olay, += operatörü ile yüklenmektedir. Burada Visual Studio kullanılıyorsa += işaretinden sonra iki kez "tab" tuşuna basmak yeterli olacaktır. Bu durumda Visual Studio otomatik olarak bir olay metodu oluşturacaktır. Örneğimizde bu olay metodu "dvd_StokAzaldi" adıyla anılmaktadır. Program çalıştırıldıktan sonra girilen örnek veriler sonucu oluşan çıktı ekranı Resim 2.2'de gösterilmiştir.



Resim 2.2: Stok takip programı ekran çıktısı

Olay metodu içerisinde bir de ürünün o anki miktarının ne olduğunu öğrenebilmek veya birden fazla ürün nesne örneğini aynı olay metoduna bağlayacaksa (ki bu mümkündür), olay metodu içerisinde hangi "Urun" nesne örneğinin ilgili olayın sahibi olduğunu tespit edebilmek isteyebilirsiniz. İşte bu gibi gereksinime benzer ihtiyaçlar, .Net içerisinde var olan tüm olaylar için de geçerlidir. O nedenle önceden tanımlanmış olan tüm olaylar aslında standart olarak iki parametre almaktadır. İlk parametre olayı tetikleyen nesne örneğine ait referansın yakalanması için kullanılırken, ikinci parametre olay metodu içerisine bilgi aktarmak maksadıyla ele alınır. Bu standart bir olay temsilcisinin işaret edeceği metodun parametrik yapısıdır. Tahmin edeceğiniz gibi ilk parametre "object" tipindendir. İkinci parametre ise genellikle "EventArgs" gibi kelimeler ile biten özel bir sınıftır. Kendi örneğimizi göz önüne aldığımızda öncelikli olarak, olay metoduna özel bilgilerin aktarılmasını sağlayacak şekilde bir tipin geliştirilmesi gerekmektedir.

Projemize "StokAzaldiEventArgs" adında bir sınıf ekleyip aşağıdaki tasarımı gerçekleştiriniz.

```
namespace StokTakip
{
   class StokAzaldiEventArgs:EventArgs
   {
      private int guncelStokMiktari;

      public int GuncelStokMiktari
      {
         get { return guncelStokMiktari; }
         set { guncelStokMiktari = value; }
      }

      public StokAzaldiEventArgs(int gStk)
      {
         GuncelStokMiktari = gStk;
      }
   }
}
```

"StokAzaldiEventArgs" isimli tipin tek yaptığı, güncel stok miktarını ilgili olay metodu içerisine taşımaktır. Bu tip asıl "StokAzaldi" olayı için anlamlıdır. Olay argümanlarını taşıyacak kendi tiplerimizi geliştirdiğimizde bunların "EventArgs" tipinden türetilmesi bir zorunluluk değildir ancak bir gelenektir. Amaç aynen isimlendirme kurallarında olduğu gibi kodun standardize edilmesidir. Nitekim .Net içerisindeki tüm olay argüman tipleri, bir şekilde "EventArgs" sınıfından türemektedir. Bu şekilde olay metoduna bilgi taşıyabileceğimiz bir tip tanımladıktan sonra temsilcinin de aşağıdaki gibi değiştirilmesi gerekmektedir.

```
public delegate void StokAzaldiEventHandler(object sender, StokAzaldiEventArgs args);
```

Elbette temsilcide yapılan değişikliklerin olayın tetiklendiği yerde adapte edilmesi gerekmektedir. Bu amaçla "StokMiktari" özelliğinin set bloğu aşağıdaki gibi değiştirilmelidir.

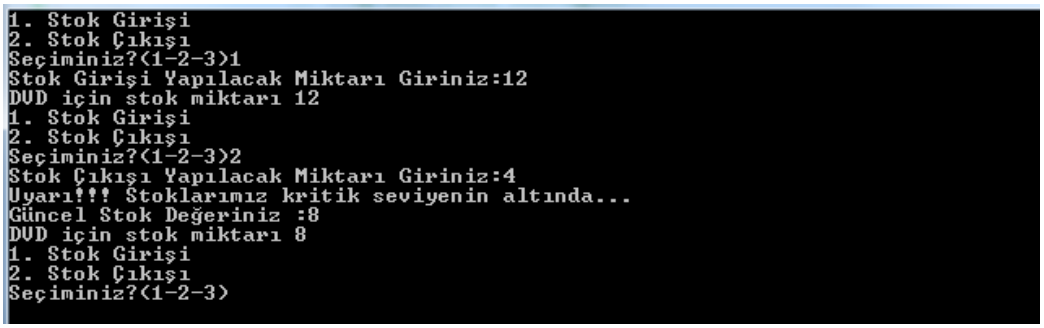
```
public int StokMiktari
    {
      get { return _stokMiktari; }
      set
      {   _stokMiktari = value;
        if (value < 10 && StokAzaldi != null)

        {
          StokAzaldi(this, new StokAzaldiEventArgs(value));
        }
      }
```

Dikkat edilecek olursa ilk parametreye “this” anahtar kelimesi getirilmiştir. Hatırlayacağınız gibi ilk parametrenin, olayı tetikleyen nesne referansını taşıdığını belirtmiştik. İşte buradaki “this” anahtar kelimesi çalışma zamanındaki (run-time) nesne referansının alınıp ilgili olay metoduna gönderilmesini sağlamaktadır. İkinci parametrede ise “StokAzaldiEventArgs” tipinden bir nesne örneği oluşturulmakta ve güncel stok miktarının değeri “value” anahtar kelimesi ile yapıcı metoduna gönderilmektedir. Tahmin edeceğiniz üzere nesne örneği de, olay metodu içerisinde ele alınabilecektir. Artık program içerisinde kodlarımızı da aşağıdaki gibi düzenlemeniz gerekmektedir.

```
dvd.StokAzaldi += new Urun.StokAzaldiEventHandler(dvd_StokAzaldi);
……
……
static void dvd_StokAzaldi(object sender, StokAzaldiEventArgs args)
{
    Console.WriteLine("Uyarı!!! Stoklarımız kritik seviyenin altında...");
    Console.WriteLine("Güncel Stok Değeriniz :" + args.GuncelStokMiktari.ToString());
}
```

Uygulamayı bu haliyle çalıştırdığımızda aşağıdaki ekran görüntüsünde yer alan sonuçları elde ederiz.



Resim 2.3: Stok takip programı ekran çıktısı

Birden fazla Urun nesnesini aynı StokAzaldi olay metoduna yönlendirme şansına sahibiz.

```
Urun dvd = new Urun(10001, "DVD", 1.20, 0);
Urun cd = new Urun(10002, "CD", 1, 0);
dvd.StokAzaldi += new Urun.StokAzaldiEventHandler(dvd_StokAzaldi);
cd.StokAzaldi+=new Urun.StokAzaldiEventHandler(dvd_StokAzaldi);
```

Bu şekilde bir ihtiyaç özellikle dinamik olarak oluşturulan kontrollerin bir olay metoduna bağlanarak ele alınması gibi durumlarda kullanılmaktadır. Böylece birden fazla olay metodunu düşünmek yerine tek bir merkez metottan kontrol ve yönetim işlemlerini gerçekleştirebilirsiniz.

Söz konusu olay metodunun hangi nesne tarafından çağırıldığının bilgisini parametrelerinden ilki olan “object” tipindeki “sender” nesnesi tarafından elde edilebilir.

```
static void dvd_StokAzaldi(object sender, StokAzaldiEventArgs args)
{
   Urun urn = (Urun)sender;
   Console.WriteLine("Uyarı!!! "+urn.Ad+" kritik seviyenin altında...");
   Console.WriteLine("Güncel Stok Değeriniz :" + args.GuncelStokMiktari.ToString());
}
```

Dikkat edilmesi gereken önemli noktalardan birisi de olay metodu içerisinde "sender" isimli parametre değişkeninin nasıl kullanıldığıdır. "sender" isimli değişken dönüşüm işlemine tabi tutularak bir "Urun" nesne örneğine dönüştürülmekte ve kullanılmaktadır. Burada elbette ki akla şu soru gelebilir. "Urun" nesne örneğine dönüştürme işlemi yapıldıktan sonra güncel stok miktarı gibi veriler de elde edebilir bu nedenle "StokAzaldiEventArgs" gibi tipleri geliştirmeye ihtiyacımız var mıdır? Aslında bu bir anlamda doğru olsa da bir argüman tipinin var olması, olay metodu içerisine gerçekten de ne aktarmak istediğimizi belirten bir kodlama yolu ve standardı sağlamaktadır. Diğer taraftan gereksiz dönüşüm (casting) işlemlerinin de önüne geçilmiş olacaktır. Bunların dışında nesne örneği üzerinden elde edilemeyen ancak argümanlar yardımıyla ele alınabilecek bazı verilerin yayıncı nesne (publisher object) içerisinden sadece olay metoduna aktarılması da sağlanabilir.

Örnek 2.5. Programınız içinde yer alan özel varlık nesnelerinin herhangi bir özelliği değiştiğinde kullanıcıyı haberdar edecek ve değişen özelliğin adını bildirecek programı yazınız.

Örneğimiz için "Dosya" isimli bir sınıfımız olsun. Buradaki amaç dosya sınıfından örneklenmiş nesnenin bir özelliği değiştiğinde söz konusu olayın yanında değişen özelliğin ismini taşıyacak bir parametresi ile birlikte tetiklenmesidir Bunun için ilk olarak olay parametresinin taşınmasını sağlayacak "BildireventArgs" isimli "EventArgs" türünden sınıfı oluşturunuz.

```
namespace Entity
{
   class BildirEventArgs:EventArgs
   {
      private string _propertyName;
      public string PropertyName
      {
         get{ return _propertyName; }
         set{ _propertyName=value; }
      }

      public BildirEventArgs(string propName)
      {
         _propertyName=propName;
      }
   }
}
```

Sınıflarda özellik (property) değişimi set bloklarında yapılmaktadır. Dosya sınıfına ait her bir özelliğin "set" blokunda söz konusu değişimin olduğunu olay çağrımını yaparak gerçekleştiriniz.

```
namespace Entity
{
   class Dosya
   {
      public delegate void BildirEventHandler(object sender, BildirEventArgs args);

      public event  BildirEventHandler DosyaBilgisiDegisti;

      private void Bildir(string propName)
      {
         // Olay yanında parametresi "özellik isimi" ile çağırılır.
         DosyaBilgisiDegisti(this,new BildirEventArgs(propName));
      }
      private string _dosyaAdi;
      public string DosyaAdi
      {
         get {return _dosyaAdi;}
```

```
            set
            {
                if(_dosyaAdi!=value)
                {
                    _dosyaAdi=value;
                    Bildir("DosyaAdi");
                }
            }
        }

        private double _boyut;

        public double Boyut
        {
            get {return _boyut;}
            set
            {
                if(_boyut!=value)
                {
                    if(_boyut!=value)
                    {
                        _boyut=value;
                        Bildir("Boyut");
                    }
                }
            }
        }

        private DateTime _tarih;
        public DateTime Tarih
        {
            get {return _tarih;}
            set
            {
                _tarih=value;
                Bildir("Tarih");
            }
        }
    }
}
```

Dosya sınıfını test etmek için kullanacağınız program sınıfını gerçekleştiriniz.

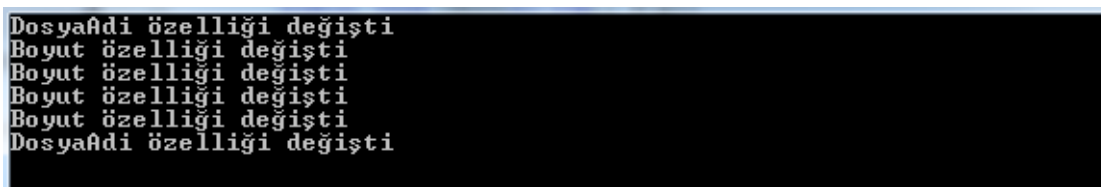
```
namespace Entity
{
   class Program
   {
      static void Main(string[] args)
      {
        Dosya dosyam=new Dosya();
        dosyam.DosyaBilgisiDegisti+=new Dosya.BildirEventHandler(DegisimOldu);

         dosyam.DosyaAdi="Dosya1";
         dosyam.Boyut=12;
         dosyam.Tarih=new DateTime(2012,5,10);

         dosyam.Boyut=13;
         dosyam.Boyut=14;
         dosyam.DosyaAdi="Dosya2";
         Console.ReadLine();
      }

      public static void DegisimOldu(object sender,BildirEventArgs args)
      {
         Console.WriteLine(args.PropertyName +" özelliği değişti");
      }
   }
}
```

Programınınızı çalıştırdığınızda nesnenin herhangi bir özelliğine değer girdiğinizde olay metodu devreye girmekte ve olay parametresini bildirmektedir. Sonuç ekranı Resim 2.4'te gösterildiği gibidir.



Resim 2.4: Program test sonucu

Olaylar ile yapacağınız bu tip çözümler özellikle veri tabanı varlıklarımız üzerinden gelen değişikliklerin algılanmasında ve kullanıcı tarafına iletilmesi gibi veri bağlı (data binding) senaryolarda oldukça sık kullanılmaktadır.

UYGULAMA FAALİYETİ

Aşağıdaki işlem basamaklarını takip ederek faaliyeti gerçekleştiriniz.

İşlem Basamakları	Öneriler
➢ Ürün listesinin tutulduğu bir sınıf tasarlayınız. ➢ Ürün eklendiğinde tetiklenecek bir olay tanımlayınız. ➢ Konsolda yer alan program sınıfınızda olay dinlemesini yapınız. ➢ Listeye her ürün eklendiğinde olay metodu içinde toplam tutarı hesaplayan programı yazınız.	➢ Program içinde "urun" ve "sepet" isimli iki sınıf oluşturulacaktır. ➢ "Sepet" urun listesinin tutulduğu sınıftır. ➢ Ürün bilgileri klavyeden girilecektir. ➢ Her olay gerçekleştiğinde toplam tutar ekrana mesaj olarak yazdırılacaktır.
➢ Bir makinenin çalışma anında sıcaklık değeri 1000 derecenin üzerine çıktığında makineyi kapatacak programı yazınız.	➢ Program içinde makine sınıfı oluşturulacaktır. ➢ Sıcaklık değeri program içinde rastgele (random) üretilecektir. ➢ Sıcaklık özelliği 1000 derecenin üstüne çıktığında olay çağırılacaktır. ➢ Program tarafından dinlenen olay yakalandığında makinenin kapat fonksiyonu çalıştırılacaktır.

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki cümlelerin başında boş bırakılan parantezlere, cümlelerde verilen bilgiler doğru ise D, yanlış ise Y yazınız.

1. ( ) Olaylar tanımlandığı sınıflar üzerinden tetiklenir.
2. ( ) Bir tek olay metodunu yalnızca bir tek olaya bildirebilirsiniz.
3. ( ) Bir tek olaya dilediğiniz kadar olay metodu bağlanabilir.

Aşağıdaki cümlelerde boş bırakılan yerlere doğru sözcükleri yazınız.

4. …….. sınıfın içinde bir durumun meydana geldiğini bildiren otomatik bildirgelerdir.
5. Abone etme işlemi …….. operatörü ile yapılmaktadır.
6. Abonelik iptali ……... operatörü ile yapılmaktadır.
7. Olay çağrımı yapılmadan önce ……..… değer ile karşılaştırılır.

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise "Modül Değerlendirme"ye geçiniz.

MODÜL DEĞERLENDİRME

Aşağıdaki cümlelerde boş bırakılan yerlere doğru sözcükleri yazınız.

1. Temsilcilere metot bildirimi yapmak için …….. operatörü kullanılır.
2. Temsilcinin metot zincirine ekleme yapmak için ……operatörü kullanılır.
3. Temsilcinin metot zincirinden metot çıkarmak için ….operatörü kullanılır.
4. Temsilcinin referans listesini boşaltmak için …… anahtar kelimesi kullanılır.
5. Olay tanımı ……... ile birlikte yapılır.
6. Olayı dinleyen ve olay metoduna sahip olan sınıflara …….. sınıflar denilmektedir.

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki modüle geçmek için öğretmeninize başvurunuz.

CEVAP ANAHTARLARI

ÖĞRENME FAALİYETİ-1'İN CEVAP ANAHTARI

1	**metod**
2	**adres**
3	**derleme zamanı**
4	**çalışma zamanı**
5	**delegate**

ÖĞRENME FAALİYETİ-2'NİN CEVAP ANAHTARI

1	**Doğru**
2	**Yanlış**
3	**Doğru**
4	**Olaylar**
5	**+=**
6	**-=**
7	**null**

MODÜL DEĞERLENDİRMENİN CEVAP ANAHTARI

1	**=**
2	**+=**
3	**-=**
4	**null**
5	**temsilci**
6	**abone**

KAYNAKÇA

- Michaelis, Marc, **Essential C# 4.0,** Addison-Wesley /US, 2010.
- Sharp, John, **Microsoft Visual C# 2008**, Microsoft Press / US, 2008.
- Herbert Schildt, **C# 4.0 Complete Reference**, Osborne / US, 2010.