DERS ADI Nesne Tabanlı Programlama I
ÜNİTE ADI Nesne Tabanlı Programlamanın Temelleri
ÜNİTE NO 1
YAZAR Dr. MUHAMMET DAMAR

GİRİŞ

Hızla dönüşen ve gelişen bilişim teknolojileri beraberinde yazılım teknolojilerinin ve programlama tekniklerinin de gelişip dönüşmesine sebep olmuştur. Bu dönüşüm sürecinde öne çıkan gelişmelerden birisi de nesne yönelimli programlama tekniğidir. 1960'lı yıllarda başlayan nesneye yönelimli programlama paradigması, 1990'larda hız kazanarak birçok alanda yaygın hale gelmeye başlamıştır. Aslında 1990'lı yıllar sonrasında teknolojinin hızla gelişmesi ve bilgi sistemlerinin eğitim, bankacılık ve finans, sağlık gibi tüm sektörlerde yaygın olarak kullanılması yazılım geliştirme süreçlerinin gelişmesinde etkili olmuş ve önemli bir dönüşümü başlatmıştır.

Bu bölüm içerisinde size nesne yönelimli programlamanın giriş seviyesinde bilgiler sunulmaktadır. Bu bölümde, .Net Framework ortamı ile tanışacak ve C# programlama dili aracılığı ile ilk programınızı yazabilmeniz için gerekli pratik bilgiyi bulacaksınız.

Bu ders için hazırlanan bölümler nesne yönelimli programlama ile programcılığa başlayan programcı adaylarının giriş seviyesinde kodlama yapabilmesini ve Nesne Yönelimli Programlama konusunda geniş bir bakış açısına sahip olmalarını hedeflemektedir. Bölümler içindeki nesne yönelimli programlama mantığını eksiksiz yansıtması için C# programlama dili kullanılmıştır.

NESNE YÖNELİMLİ PROGRAMLAMA

Bir programlama dili, çeşitli çıktılar üreten bir dizi talimattan oluşan resmi bir dildir. Programlama dilleri, bilgisayar programlamada algoritmaları uygulamak için kullanılır. Nesne yönelimli programlama, bir grup değişken ve fonksiyonun nesne adı verilen tek bir birimde birleştirilmesiyle düzenlenen bir tür programlama yaklaşımıdır. Tasarlanan nesneler, her nesnenin gruplandırıldığı sınıflar halinde düzenlenir.

Simülasyon dilinin kısaltması olan Simula, ilk nesne yönelimli programlama dilidir. Simula ilk nesne yönelimli programlama dili olarak kabul edilirken, günümüzde nesneye yönelimli programlama ile birlikte birçok başka programlama dili kullanılmaktadır. Ancak bazı programlama dilleri nesneye yönelimli programlama paradigması ile diğerlerine göre daha iyi eşleşmektedir.

Örneğin, saf nesneye yönelimli programlama dilleri (Ruby, Scala, JADE, Emerald) olarak kabul edilen programlama dilleri, her şeyi nesne olarak ele almaktadır. Öte yandan Diğer programlama dilleri öncelikle Nesneye yönelimli programlama için tasarlanmıştır, fakat bazı prosedürel süreçler dahil edilmiştir. Bu tip programlama dillerine C#, Python ve Java örnek gösterilebilir. Günümüzde ise öne çıkan nesneye yönelimli programlama dilleri C#, Python, Ruby, Go, C++ ve Java olarak sıralanabilir.

Nesne yönelimli programlamanın yapısı veya yapı taşları şunları içerir:

- Sınıflar (Classes), bireysel nesneler, nitelikler ve yöntemler için plan görevi gören kullanıcı tanımlı veri türleridir.
- Nesneler (Objects), özel olarak tanımlanmış verilerle oluşturulan bir sınıfın örnekleridir. Nesneler, gerçek dünyadaki nesnelere veya soyut bir varlığa karşılık gelebilir.
- Yöntemler (Methods), bir nesnenin davranışlarını tanımlayan bir sınıf içinde tanımlanan işlevlerdir. Sınıf tanımlarında yer alan her yöntem, bir örnek nesneye başvuru ile başlar.
- Nitelikler (Attributes) sınıf şablonunda tanımlanır ve bir nesnenin durumunu temsil eder.

Nesneler, nitelikler alanında depolanan verilere sahip olacaktır. Sınıf nitelikleri sınıfın kendisine aittir.

Nesne Yönelimli Programlamanın Temel İlkeleri

Nesne Yönelimli Programlama Teorisinde dört temel özelliğin gerçekleştirilmesi zorunlu sayılmaktadır. Bu temel özelliklerden herhangi birini sağlamayan bir dil Nesne Yönelimli Programlama Dili olarak sayılmamaktadır. Bu dört özellik sırasıyla, Kapsülleme (Encapsulation), Soyutlama (Abstraction), Miras (Inheritance), Polimorfizm (Polymorphism), şeklindedir.

Kapsülleme veya Paketleme (Encapsulation)

Bu ilke, tüm önemli bilgilerin bir nesnenin içinde bulunduğunu ve yalnızca belirli bilgilerin açığa çıktığını belirtir. Kapsüllemenin anlamı, sınıfı oluşturan metot ve özelliklerin gerçekleştirme biçiminin, bu sınıfı kullanacak olan kullanıcılardan gizlenmiş olmasıdır.

Genel tanımıyla kullanıcı tarafından verilerin, sınıfların ve metotların ne kadarının görüntülenebileceği ve değiştirilebileceğinin sınırlarının konulmasını sağlar. Public (herkese açık), private (özel) ve protected (koruma altında) olmak üzere üç adet access modifier'dan (erişim dönüştürücüsü) bahsedilebilir. Erişim belirteçleri (access modifier) sayesinde kapsülleme çok daha kolay yapılmaktadır.

- Public (Genel): Nitelendiğinde ilgili unsurlar herkese açık olur. Başka pakette olsa bile, program içindeki, her kod onlara erişebilir. Bir değerin public olarak belirtilmesi; o değerin, kod içinde herhangi bir yerden erişilebilir durumda olmasını sağlamaktadır.
- Private (Özel): Bazı değişken, metot ya da sınıflara başka sınıftaki kodların erişmesini engellemek isteyebiliriz. Bunun için Private nitelemesini kullanırız. Private erişim belirteci, public belirteci karşıtı gibidir. Bir değerin private olarak tanımlanması demek, o değişkene sadece kendi sınıfı içinden ulaşılabileceği anlamına gelmektedir.
- Protected (Korunumlu): Bir sınıf içindeki değişken ve metotlara alt-sınıfların erişebilmesini, ama paket içindeki ya da program içindeki başka kodların erişmesini engellemek isteyebiliriz. Kod içinde bir değerin protected olarak tanımlanması; o değere, değerin bulunduğu sınıf ve ondan türetilen diğer sınıflar içinden erişileceğini göstermektedir.
- Internal (İçsel): Internal olarak tanımlanan bir değer; aynı program içerisinden erişilebilir; fakat farklı bir program içerisinden erişilemez durumdadır. Program içerisinde herhangi bir kısıtlaması yoktur.
- Protected Internal (İçsel Korunumlu): Protected internal olarak tanımlanmış değer, tanımlandığı class'ın içinden ve ondan türetilen sınıfların içinden erişilebilir durumdadır.

Soyutlama (Abstraction)

Nesneler, yalnızca diğer nesnelerin kullanımıyla ilgili dahili mekanizmaları ortaya çıkararak gereksiz uygulama kodlarını gizler. Alt sınıfların ortak özelliklerini ve işlevlerini taşıyan ancak henüz bir nesnesi olmayan bir üst sınıf oluşturmak istenirse bir soyut (abstract) üst sınıf oluşturulur. Soyut sınıfın yöntemleri alt sınıfları tarafından üzerine yazılmak üzere şablon olarak tanımlanabilir veya soyut metot olarak oluşturulabilir.

Miras (Inheritance)

Bir sınıftan başka bir sınıf türetirken aralarında bir alt-üst ilişkisi oluşturmayı ve bu sınıflar üzerinde ortak metotlar ve özellikler kullanılmasını sağlayan bir mekanizmadır. Hali hazırda var olan sınıfların üzerine başka sınıfların inşa edilmesini sağlar. Sınıflar, diğer sınıflardan gelen kodları yeniden kullanabilir.

Alt sınıfın nesneleri, türetildikleri temel sınıfa ait özellikleri alıyorsa, burada miras alma (inheritance) özelliği vardır denir. Bu anlamda, miras alma özellikli bir nesne yönelimli programlama dilinde, bir nesne sınıfından türetilen alt nesne sınıfına ait nesneler, üst sınıfın özelliklerini (properties) ve metodlarını (methods) aynen alırlar.

Polimorfizm (Polymorphism)

Nesneler, davranışları paylaşmak üzere tasarlanmıştır ve birden fazla biçim alabilirler. Polimorfizm yani çok biçimlilik Nesneye Yönelik Programlamada programlama dilinin farklı tip verileri ve sınıfları farklı şekilde işleme yeteneğini belirten özelliğidir. Daha açık ifade etmek gerekirse; metotları ve türetilmiş sınıfları yeniden tanımlama yeteneğidir.

Neden Nesne Yönelimli Programlama

Yazılım geliştirme süreçlerinde nesneleri kullanmak ve tasarlanan sitemi veri modeline ve fonksiyonlarına göre nesneler halinde tasarlamak bize pek çok fayda sağlayacaktır. Nesne yönelimli programlama mantığı, modülerlik, tekrar kullanılabilirlik, üretkenlik, güvenlik, esneklik, bakım gibi pek çok avantaj sağlar. Bu avantajlar kısaca şu şekildedir: Modülerlik (modularity), tekrar kullanılabilirlik (reusability), üretkenlik (productivity), kolayca yükseltilebilir ve ölçeklenebilir (easily upgradable and scalable), genişletilebilirlik (extensibility), arayüz tanımlama (ınterface descriptions), güvenlik (security), esneklik (flexibility), yazılımın bakımı (maintenance).

Nesne Yönelimli Programlama İçin Eleştiriler

Nesne yönelimli programlama modeli, geliştiriciler tarafından kimi zaman eleştirilmiştir. En büyük eleştiri nedeni, nesneye yönelimli programlamanın yazılım geliştirmenin veri bileşenini aşırı odaklanması ve hesaplama veya algoritmalara yeterince odaklanmamasıdır.

Nesneye yönelimli programlamaya alternatif yöntemler ise şu şekilde sıralanabilir; fonksiyonel programlama, yapılandırılmış veya modüler programlama, zorunlu programlama, bildirimsel programlama, mantıksal programlama.

Buradaki programlama paradigmalarında aslında kazanan yoktur. Her yöntemin programcı tarafından kullanılabilir iyi yönleri vardır. Bu programlama tecrübeniz ve proje ön görünüz ile sizin tercihinize kalmıştır. Bir projeye hangi kod stilinin en uygun olabileceğini o kadar iyi belirleyebilir, hatta birden fazla paradigmayı örtüştürüp de kullanabilirsiniz.

VİSUAL STUDIO 2019

C# ile Geliştirmek

C# ile programlamak için C# programları oluşturmanın bir yoluna ihtiyacımız olacaktır. Bunu bir komut

satırı derleyicisi, Visual Studio veya bir programlama düzenleyicisi için bir C# paketi ile yapabiliriz ki biz uygulamalarımız için Visual Studio 2019 geliştirme ortamı tercih edilmiştir. C# ve .NET platformunda geliştirilmiş çok çeşitli yazılımlar vardır: ofis uygulamaları, web uygulamaları, web siteleri, masaüstü uygulamaları, mobil uygulamalar, oyunlar ve diğerleri.
C#, farklı bilgisayar platformları ve mimarilerinde çeşitli üst düzey dillerin kullanımına izin veren yürütülebilir kod ve çalışma zamanı ortamından oluşan Ortak Dil Altyapısı (Common Language Infrastructure) için tasarlanmıştır.
Aşağıdaki nedenler C#'ı yaygın olarak kullanılan bir profesyonel dil yapar:

- Modern, genel amaçlı bir programlama dilidir.
- Nesne yönelimlidir.
- Bileşen odaklıdır.
- Öğrenmesi kolaydır.
- Yapılandırılmış bir dildir.
- Verimli programlar üretir.
- Farklı bilgisayar platformlarında derlenebilir.
- .Net Framework'ün bir parçasıdır.

Microsoft dışı platformlar hedeflendiğinizde de seçenekleriniz elbette ki mevcuttur. Mono projesi bu amacı gerçekleştirmekte, Linux, iOS ve Android'i hedefleyebilen bir C# ortamı sağlamaktadır.
.NET Runtime'ın çok işlemcili sunuculardan akıllı telefonlara ve mikro denetleyicilere kadar her şey üzerinde çalışan birkaç farklı çeşidi bulunmaktadır. Yararlı görevleri gerçekleştirmek için C# kodunuz .NET Base Class Library'deki (BCL) kodu kullanacaktır. BCL, birçok program için faydalı olabilecek sınıfları içerir ve aşağıdakiler için destek içerir:

- Ağ işlemlerini gerçekleştirmek
- G/Ç işlemlerini gerçekleştirme
- Güvenliği yönetme
- Küreselleştirici programlar
- Metni değiştirme
- Bir veritabanına erişme
- XML'i manipüle etme
- Olay günlüğü, izleme ve diğer tanılama işlemleriyle etkileşim kurma
- Yönetilmeyen kod kullanma
- Dinamik olarak kod oluşturma ve çağırma

Küreselleştirici programlar ifadesinde geçen küreselleşme kavramı, geliştiricilerin dünyanın farklı bölgelerinde kullanılabilecek uygulamalar yazmasına yardımcı olur. Uygulamanın birden çok dili, farklı tarih ve sayı biçimlerini vb. desteklemesine yardımcı olur. BCL'nin üzerinde, aşağıdakiler de dahil olmak üzere belirli uygulama veya hizmet türleri oluşturmayı hedefleyen özel kitaplıklar bulunur:

- Konsol uygulamaları,
- Windows Forms veya Windows Presentation Foundation (WPF) kullanan Windows GUI uygulamaları,
- ASP.NET (web) uygulamaları,
- Windows Hizmetleri,
- Windows Communication Foundation (WCF) kullanan hizmet odaklı uygulamalar,
- İş akışı etkin uygulamalar, Windows Workflow Foundation (WF),
- Windows 8 uygulamaları ve Windows Mobil Telefon uygulamaları.

Günümüzde C# en popüler programlama dillerinden biridir. Dünya çapında milyonlarca geliştirici tarafından kullanılmaktadır.
C#'ın Güçlü Programlama Özellikleri
C# yapıları geleneksel üst düzey dilleri yakından takip etse de, C ve C++ dir. Java ile güçlü bir benzerliği vardır, onu dünya çapında birçok programcıya sevdiren çok sayıda güçlü programlama özelliğine sahiptir.
C#'ın birkaç önemli özelliğinin listesi aşağıdadır:

- Boolean Koşulları,
- Otomatik Çöp Toplama,
- Standart Kütüphane,
- Montaj Versiyonu,
- Özellikler ve Olaylar,
- Delegeler ve Etkinlik Yönetimi,
- Kullanımı kolay Jenerikler,
- İndeksleyiciler Koşullu Derleme,
- Basit Çoklu Okuma,
- LINQ ve Lambda İfadeleri,
- Windows ile entegrasyon yapılabilmesi.

Visual Studio 2019 Kurulumu

Nesne Tabanlı Programlar yapmak ve bu programlar içerisinde Visaul Studio .Net C# dilini daha kolay kullanmak amacıyla Studio.NET yazılım geliştirme ortamı kullanılmaktadır. Sürüm 16.11 versiyonu kod geliştirme ve anlatımda kullanılacaktır.
Visual Studio ve .NET'i kullanarak masaüstü, web, mobil, oyun ve nesnelerin interneti için uygulamalar geliştirebilirsiniz. .NET uygulamalarını C#, F # veya Visual Basic dilinde yazabilirsiniz. Örneğin; Windows Azure, Office 365 ile Cloud (bulut) teknolojisi ile çalışacak ofis uygulamaları geliştirilebilir.
Visual Studio 2019 Programın kurulabilmesi ve verimli çalışabilmesi için sistem gereksinimleri, Visual Studio Enterprise 2019, Visual Studio Professional 2019, Visual Studio Community 2019, Visual Studio Team Foundation Server Office 2019 yazılımlarını destekler. C# Programlama dilinde sınıf tanımlamak class kelimesi kullanılır.
Sistem gereksinimleri bölümünde de ifade ettiğimiz gibi bilgisayarımız uygun konfigürasyonlara sahip olmalı ve bilgisayarımızın diskinde yeterli yerimizin bulunması gerekmektedir. Unutulmaması gereken nokta sadece 11,82 GB yer ile hareket edilmemesidir. Bilgisayarımızda .Net Studio 2019 kurulumu için en az 20-25 GB yeriniz muhakkak olmalıdır. Bu sayede programı geliştirme sürecinizde daha az sıkıntı yaşayacaksınızdır.
Visual Studio 2019 Geliştirme Ortamı
Visual Studio 2019 programının bilgisayara kurulum işlemi tamamlandıktan sonra, Başlat menüsü içinden Programlar bölümü altıdan Visual Studio 2019 programı seçilerek program çalıştırılır.
"Yeni bir proje oluştur" menüsü tercihi edilerek ikinci ekranımız açılır. Buradan amacımıza uygun projeyi seçebiliriz.

DERS ADI Nesne Tabanlı Programlama I
ÜNİTE ADI Sınıf Yapısına Giriş
ÜNİTE NO 2
YAZAR Öğr. Gör. İSA BİNGÖL

Nesne yönelimli programlama (Object orianted programming), bir programda sınıf (class) tanımlamaya ve bu sınıflardan yeni sınıflar ve nesneler (object) oluşmaya olanak tanıyan bir programlama tekniğidir. Nesne yönelimli programlama yaklaşımının en önemli unsurları sınıf, nesne ve metot olarak sıralanabilir. Nesne bir sınıftan oluşturulan değişkenlerdir. Sınıf bir nesnenin özelliklerini barındıran yapılardır. Metot ise bu nesnenin gerçekleştirdiği işlerdir.

Nesne yönelimli programlama yaklaşımı, gerçek hayatta karşılaşılan birçok şeyin bilgisayar ortamına aktarılabilmesini ilke edinmiştir. Nesne yönelimli programlama kapsamındaki sınıf, nesne veya metot gibi yapılar programcıya karşılaşılan bir nesneyi olabildiğince esnek şekilde modellemesini sağlar. Örneğin; motorlu araçları nesne yönelimli programlama yaklaşımına göre modellememizin istendiğini düşünelim. Motorlu araçlar genel bir kavramdır ve içerisinde otomobil, uçak, motosiklet ve kamyon gibi birden fazla motorlu aracı barındırır. İşte burada motorlu araçları sınıf, otomobil, uçak, motosiklet ve kamyon gibi araçları da birer nesne olarak düşünebiliriz. Metot ise bu araçların gerçekleştirebildikleri işleri ifade eder. Örneğin, her araç çalışabilir, durabilir, gidebilir, yolcu (yük) taşıyabilir. Araçların yapmış olduğu bu işlemler metot olarak adlandırılabilir.

Sınıf konusu nesne tabanlı programlamanın en önemli konularından biridir. Bu kavramı iyi bilmek, projelerde nesneler ve metotlar oluşturup kullanabilmek hem yapılan projenin daha profesyonel bir yapıya kavuşmasına hem de kod tekrarının önüne geçmeye imkan tanır.

SINIFLAR (CLASS)

Sınıflar, içerisinde tutulan bilgilerin ve özelliklerin daha düzenli ve sistematik bir yapıda tutulmasını sağlarlar. Aynı zamanda bir program yazarken sınıfları kullanmak, her bir nesne için ayrı ayrı kod yazmaya gerek kalmadan tek merkezden kontrol etmeye imkan tanır. Bu şekilde ilerde kod üzerinde bir değişiklik yapılmak istendiğinde tek bir yerden değişiklik yapmak yeterli olacaktır.

Giriş bölümünde de bahsedildiği üzere sınıflar nesnelerin özelliklerini ve gerçekleştireceği işleri barındıran yapılardır. Nesneler ise sınıflardan tanımlanmış değişkenlerdir. Bir sınıftan birden fazla nesne oluşturulabilir. Ve her nesnenin bir özelliği (properties) ve yapacağı bir işi yani metotu vardır.

Bu kavramları daha da somutlaştırarak yukarıda vermiş olduğumuz motorlu araçlar örneği üzerinden gidelim. Motorlu araçlar genel bir kavramdır ve içerisinde birçok aracı barındırabilir. Dolayısıyla motorlu araçlar bizim sınıfımızdır. Otomobil, otobüs, uçak gibi araçlar motorlu araç sınıfından türediği için bunların her biri de birer nesnedir. Bu nesnelerin kendilerine göre rengi, kapı sayısı, yakıt türü gibi özellikleri vardır. Ve son olarak bunların gitmek, yolcu taşımak gibi yaptığı işler vardır. Bunlara da metot denmektedir.

Sınıf Tanımlama

C#'da bir sınıf tanımlamak için class anahtar sözcüğü kullanılır. Class anahtar sözcüğünden sonra sınıf ismi belirtilir. Ardından küme parantezleri açılarak sınıfın özellikleri eklenir. Küme parantezleri arasında sabit, değişken ve metot gibi sınıfın öğeleri eklenir.

Değişken ve metot tanımlarken ilk başta bu değişken ya da metoda nerelerden erişileceğini belirlediğimiz bir erişim türü ekleriz. Eğer değişken ya da metoda sadece tanımlandığı sınıfın içinden erişilmek isteniyorsa erişim türü private (özel) olarak ayarlanmalıdır. Değişken ya da metoda sadece tanımlandığı sınıfın içinden değil de sınıfın dışından da erişilmek isteniyorsa erişim türü public (genel) olarak tanımlanmalıdır.

Projeye yeni bir sınıf eklemek için birden fazla yol kullanılabilir. İlk olarak üst menülerde yer alan Project menüsü ardından Add Class komutu kullanılabilir. Bir diğer yöntem ise Solution Explorer penceresinde yer alan proje adına sağ tıklayıp Add>Class seçeneğini kullanmaktır.

Bir sınıftan yeni bir nesne oluştururken new anahtar sözcüğü kullanılmaktadır. Bu sözcük tanımladığımız sınıfın bir kopyasını hafızada oluşturacaktır. Bir sınıftan istenildiği kadar nesne oluşturulabilir.

this Anahtar Sözcüğü

Bir metot çağrıldığı zaman bu metoda ilgili nesnenin bir referansı otomatik olarak aktarılır. Bu referans

ise this olarak adlandırılmaktadır. Oluşturmuş olduğumuz bir sınıfın üye elemanları ile metodumuzun parametreleri aynı isimle tanımlanmış olabilir. Dolayısıyla this anahtar sözcüğü bu ikisi arasındaki ayrımı yapmamızda bize yardımcı olabilmektedir.

Get ve Set Anahtar Sözcükleri

Bir sınıf içine eklenen özelliklerin erişim türlerinin public (genel) olarak tanımlanması o özelliklere direkt erişebilmeyi mümkün hale getirir. Bu da programlamada çok istediğimiz bir durum değildir. Ama bir metot yazarak bu özelliklere doğrudan erişmek yerine metot değişkenleri aracılığıyla erişebiliriz. Bu şekilde girilen değerlerin belirli aralıkta olup olmadığını kontrol ederek olası problemlerin önüne geçebiliriz. Bir özelliği görüntülemek için bir metot bu özelliğin değerini değiştirmek için ise başka bir metot yazabiliriz. Ama her özellik için iki ayrı metot yazmak yerine get ve set anahtar sözcükleri kullanarak da sanki iki ayrı metot bildirmiş gibi oluruz.

Bir özelliğin ya da değişkenin değeri öğrenilmeye çalışıldığında get bloğu kullanılır. Bu blok içine yazılan kodla o değişkenin değerini geri göndermelisiniz. Geri gönderme işlemi return komutuyla gerçekleştirilir. Bir özelliğin ya da değişkenin değeri değiştirilmeye çalışıldığında ise set bloğu kullanılır. Değişkenin değeri value isimli özel bir anahtar sözcük ile öğrenilir. Eğer set bloğu tanımlanmazsa o değer sadece okunabilir olur. Yine aynı şekilde eğer get bloğu tanımlanmazsa o değişkene sadece değer atanabilir ama okuma yapılamaz.

Sınıfların bir özelliği de bir sınıftan başka bir sınıf türetilmesidir. Yani tanımlanan bir A sınıfından bir B sınıfı türeterek B sınıfına A sınıfından aldığı özelliklerin yanında yeni özellikler de eklenebilmektedir. Bu olaya kalıtım (inheritance) denilmektedir. A sınıfından bir B sınıfı oluşturulduğunda B sınıfının A sınıfından türetildiğini göstermek için : işareti kullanılır (B:A).

DERS ADI Nesne Tabanlı Programlama I
ÜNİTE ADI Sınıf Türleri
ÜNİTE NO 3
YAZAR Dr. MUHAMMET DAMAR

GİRİŞ

Nesneye yönelimli programlama için sınıf yapısı bir önceki bölümde de detaylı olarak ortaya konulduğu gibi kritik önemdedir. Özellikle belleği etkin kullanmak, program içinde her şeyin sınıflar, sınıflara ait nesneler ve metotlar üzerinden yönlendirilmesi tüm bu süreç için bellek ortamından etkin bir şekilde faydalanılması nesne yönelimli programlamanın en önemli avantajı ve karakteristik özelliğidir.

Sınıf veri üyeleri (sabitler ve alanlar), işlev üyeleri (yöntemler, özellikler, olaylar, dizin oluşturucular, işleçler, örnek oluşturucular, Yıkıcılar ve statik oluşturucular) ve iç içe türler içerebilen bir veri yapısıdır. Sınıf türleri, türetilmiş bir sınıfın bir temel sınıfı genişletebileceği ve özelleştirileceği bir mekanizma olan devralmayı destekler.
C#'da beş farklı erişim değiştirici mevcuttur. Bunlar sırasıyla kamuya yani herkese açık kullanım (public), korumalı kullanım (protected), dahili kullanım (internal), korumalı dahili kullanım (protected internal), özel kullanım (private), şeklindedir.

Derleme birimlerinde veya ad alanlarında belirtilen türler olabilir public veya internal erişebilir. Varsayılan olarak internal erişim'dir. Sınıflarda belirtilen türlerin, public protected, internal protected, internal veya private erişimi olabilir. Varsayılan olarak private erişim' dir. Yapılarda belirtilen türlerin public, internal veya private erişimi olabilir. Varsayılan olarak private erişim'dir. Bir sınıfı deklare etmek (class_declaration) , isteğe bağlı olarak bir sınıf değiştiricileri dizisi içerebilir. Aynı değiştiricinin bir sınıf bildiriminde birden çok kez görünmesi için derleme zamanı hatası vardır. Bu değiştiriciler şu şekilde ifade edilebilir:

- : 'new'
- | 'public'
- | 'protected'
- | 'internal'
- | 'private'
- | 'abstract'
- | 'sealed'
- | 'static'
- | class_modifier_unsafe

Konu kapsamında statik sınıflar ve sınıf üyeleri, kısmi sınıflar (partial classes) ve iç içe sınıflar (nested classes) açıklanmaktadır. Ayrıca abstract, sealed ve static değiştiricileri aşağıda açıklanmaktadır.

SOYUT SINIFLAR (ABSTRACT CLASS)

Veri soyutlama, belirli ayrıntıları gizleme ve kullanıcıya yalnızca temel bilgileri gösterme işlemidir. Soyutlama, soyut sınıflar veya arayüzler ile gerçekleştirilebilir (Nesne Yönelimli Programlama II dersinizde detaylandırılacaktır.). abstract değiştirici, bir sınıfın tamamlanmamış olduğunu ve yalnızca temel sınıf olarak kullanılması amaçlanan olduğunu göstermek için kullanılır. Soyut anahtar kelime, sınıflar ve yöntemler için kullanılır.
Soyut sınıf: Nesneler oluşturmak için kullanılamayan kısıtlı bir sınıftır (ona erişmek için başka bir sınıftan miras alınması gerekir).
Soyut yöntem: Yalnızca soyut bir sınıfta kullanılabilir ve bir gövdesi yoktur. Gövde, türetilmiş sınıf tarafından sağlanır (kalıtsal).
Soyut bir sınıf doğrudan başlatılamaz ve bir derleme zamanı hatası, new işleci bir soyut sınıfta kullanılır. Derleme zamanı türleri soyut olan değişkenler ve değerler olması mümkün olsa da, bu tür değişkenler ve değerler, null soyut türlerden türetilmiş soyut olmayan sınıfların örneklerine başvuru ya da içermeli. Soyut bir sınıfa, soyut Üyeler içermesi için izin verilir (ancak gerekli değildir). Soyut bir sınıf korumalı olamaz.

MÜHÜRLÜ SINIFLAR (SEALED CLASS)

Nesneye yönelimli programlamanın en önemli özelliklerinden birisi olan bir sınıftan başka sınıflar üretebilme durumu, güvenlik gibi çeşitli sebepler ile istenmeyebilir. Bu sınıflardan türemenin istenmemesi durumunda sınıfın başına mühürlendiğini yani bu sınıftan türetme yapılamayacağını gösteren sealed anahtar sözcüğü kullanılır. Bu sayede de sabit özellikler ve metotlara sahip olan sınıflar elde edilebilir. Bu duruma uyulmadığında yani bir mühürlü sınıf, başka bir sınıfın temel sınıfı olarak belirtilmişse, derleme zamanı hatası ile karşılaşılmaktadır. Ayrıca mühürlü bir sınıf de soyut bir sınıf olamaz.

STATİK SINIFLAR VE ÜYELERİ

C#'da statik, somutlaştırılamayan bir şey anlamına gelir. Statik bir sınıfın nesnesini oluşturamazsınız ve bir nesne kullanarak statik üyelere erişemezsiniz.

C# sınıfları, değişkenler, metotlar, özellikler, operatörler, olaylar ve yapıcılar, static değiştirici anahtar sözcüğü kullanılarak statik olarak tanımlanabilir.

Statik bir sınıf, temelde statik olmayan bir sınıfla aynıdır, ancak bir fark ile; statik bir sınıf somutlaştırılamaz. Başka bir deyişle, sınıf türünde bir değişken oluşturmak için new anahtar sözcüğünü kullanamazsınız. Statik bir sınıf, sealed veya abstract değiştiricisini içeremez. Ancak, statik bir sınıf tarafından örneklenemez veya türetilmediği için, hem korumalı hem de soyut gibi davranır. Statik bir kurucu yalnızca bir kez çağrılır ve statik bir sınıf, programınızın bulunduğu uygulama etki alanının ömrü boyunca bellekte kalır.

Aşağıdaki listede, statik bir sınıfın ana özellikleri ifade edilmektedir:

- Statik bir sınıf yalnızca statik üye(ler) içerir.
- Örneklenemez, yani statik sınıfın bir örneği oluşturulamaz.
- Mühürlüdür ve mühürleme, sınıfların ve arayüzlerin izin verilen alt türleri üzerinde daha fazla kontrole sahip olmasını sağlar. Bu, hem genel etki alanı modellemesi hem de daha güvenli platform kitaplıkları oluşturmak için kullanışlıdır.
- Statik bir sınıf örnek oluşturucuları (instance constructors) içeremez.

Statik sınıflar mühürlenir ve bu nedenle miras alınamaz. Nesne dışında herhangi bir sınıftan miras alamazlar. Statik sınıflar, bir örnek oluşturucu içeremez; ancak, statik bir kurucu (static constructor) içerebilirler.

Statik Üyeler

Statik olmayan bir sınıf, statik yöntemler, alanlar, özellikler veya olaylar içerebilir. Statik üye, sınıfın hiçbir örneği oluşturulmamış olsa bile bir sınıfta çağrılabilir. Statik üyeye, örnek adıyla değil, her zaman sınıf adıyla erişilir. Sınıfın kaç örneğinin oluşturulduğuna bakılmaksızın, statik bir üyenin yalnızca bir kopyası vardır. Statik üyeler, statik üyeye ilk kez erişilmeden ve varsa statik kurucu çağrılmadan önce başlatılır. Statik olmayan sınıftaki statik üyeler; normal sınıf (statik olmayan sınıf), bir veya daha fazla statik yöntem, alan, özellik, olay ve diğer statik olmayan üyeler içerebilir. Statik olmayan bir sınıfı bazı statik üyelerle tanımlamak, bütün bir sınıfı statik olarak bildirmekten daha pratiktir.

Statik Alanlar

Statik olmayan bir sınıftaki statik alanlar, static anahtar sözcüğü kullanılarak tanımlanabilir. Statik olmayan bir sınıfın statik alanları tüm örnekler arasında paylaşılır.

Statik Metotlar

Statik olmayan bir sınıfta bir veya daha fazla statik yöntem tanımlayabilirsiniz. Statik yöntemler, bir nesne oluşturmadan çağrılabilir. Statik olmayan sınıfın bir nesnesini kullanarak statik yöntemleri çağıramazsınız. Statik yöntemler yalnızca diğer statik yöntemleri çağırabilir ve statik üyelere erişebilir. Statik yöntemlerde sınıfın statik olmayan üyelerine erişemezsiniz.

Statik Yapıcılar

Statik olmayan bir sınıf, parametresiz bir statik kurucu içerebilir. Statik anahtar kelime ile ve public, private ve protected gibi erişim değiştiricileri olmadan tanımlanabilir. Aşağıdaki örnek, statik oluşturucu ve örnek oluşturucu arasındaki farkı örnek kodlar üzerinden görülebilir.

KISMİ SINIF (PARTIAL CLASS)

Kısmi sınıf (partial class), tanımı iki veya daha fazla dosyada bulunan bir sınıftır. Her kaynak dosya, sınıfın bir bölümünü içerir ve uygulama derlendiğinde tüm parçalar birleştirilir. Parçalara ayrılan sınıfı (class) derleme aşamasından tek bir sınıf (class) haline getirme işi derleme sırasında otomatik olarak yapılmaktadır. Partial yani kısmi sınıf kullanımı ile bizim ekstra bir şey gerçekleştirmeden bu işi

yapmamıza imkan tanımaktadır. Metod, özellik, event gibi nesneleriniz normalde aynı sınıf (class) içerisinde yer alıyorsa bir yerden sonra karışıklıklara sebep oluyorsa yapmanız gereken partial anahtar kelimesini kullanmak ve parçalara ayırmaktır.
C#'da kısmi sınıfları kullanmanın en önemli avantajları arasında; büyük projeler üzerinde çalışırken, bir sınıfı ayrı dosyalara yaymak, birden fazla programcının aynı anda üzerinde çalışmasına olanak tanıması şeklinde ifade edilebilir.

İÇ İÇE SINIFLAR (NESTED CLASS)

Nesneye yönelik programlama dilleri kullanıcıya bir sınıf içerisinde başka bir sınıf tanımlama olanağı sağlar. Bu tür sınıflara nested class (gömülü sınıf yani iç içe sınıf) adı verilir. Gömülü sınıflar statik ve statik olmayan gömülü sınıflar olmak üzere ikiye ayrılırlar. Statik olmayan gömülü sınıflara inner class (iç sınıf) adı verilir. İç içe sınıfların mantıksal gruplanması, daha iyi kapsülleme ve kod okunabilirliği yönünden üç temel faydası vardır.
İç içe sınıf (nested class) veya iç sınıf (inner class), aşağıdaki örnekte gösterildiği gibi içeren veya dış sınıfa erişebilir. İç içe türler, devralınan özel (private) veya korumalı (protected) üyeler dahil, içeren türün özel (private) veya korumalı (protected) üyelerine erişebilir.
Başka bir sınıf veya yapı içinde tanımlanan bir türe (sınıf veya yapı), iç içe tür (nested type) denir. Aşağıda bir örnek gösterilmiştir. İç sınıf, konteyner sınıfının (container class), içindedir. Dolayısıyla iç sınıfa (inner class) iç içe sınıf (nested class) denir.

DERS ADI Nesne Tabanlı Programlama I
ÜNİTE ADI Nesne Yapısına Giriş
ÜNİTE NO 4
YAZAR Öğr. Gör. DAHA ORHAN

NESNE YÖNELİMLİ PROGRAMLAMA

Nesneleri kullanarak gerçekleştirilen programlama anlayışı yaklaşık olarak 50 yıldır var olsa da, nesneler son 20 yılda programlamanın vazgeçilmez bir öğesi haline gelmiştir. Günümüzde sıklıkla kullanılan nesne yönelimli programlama anlayışından bahsedebilmek için programlama dilinde aramamız gereken bir takım kavramlar bulunmaktadır. Bu kavramlar; Sınıf, Nesne, Kalıtım, Çok biçimlilik ve Kapsüllemedir.

Sınıf

Sınıf kavramını günlük hayatımızla ilişkilendirecek olursak, gerçek dünyada gözlemlediğiniz bazı nesnelerin soyut olarak ifade edilmesi veya basitleştirilmesi olarak tanımlayabiliriz. Nesneler ise bu sınıfların birer üyesidir. Bir sınıfı temel olarak iki bileşene ayırmamız mümkündür. Bunlar; nesneyi tanımlayan özellikler ve nesneyle ilişkilendirmek istediğimiz yöntemler veya eylemlerdir.

Bir nesneyle ilişkilendirmek ve kaydetmek istediğiniz veriler, sınıfın özelliklerini oluşturmaktadır. Sınıf özelliklerinin değerlerini değiştirerek nesnenin durumunu değiştirebilirsiniz. Bir nesnenin durumu, nesneyi tanımlamak için kullanılan özelliklerin değerleriyle belirlenir.

Sınıfta bulunan özellik değerlerinden herhangi biri değişirse, nesnenin durumu da değişir. Yani yeni bir nesne ortaya konulmuş olacak ve bellekte o sınıfa ait farklı bir nesne yer alacaktır. Bir sınıfa ait herhangi bir özelliğin değeri her değiştiğinde, nesnenin durumu da değişmektedir.

Bir nesneye ait durumu tanımlamak için özellik değerleri kullanılmaktadır. Sınıf metotları ise bu özelliklere etki etmektedir. Buradan hareketle metot kavramını, nesnelerin belirli işlemlerini gerçekleştirmeyi sağlayan fonksiyonlar olarak tanımlayabiliriz. Kısaca sınıf metotları, nesnenin gerçekleştirebileceği davranışları veya eylemleri belirlemektedir.

Metotlar genel olarak değer (parametre) alabilen, değer döndüren ve birden fazla değer ile işlem yapabilen yapıdadırlar. Metotları nesnelerle ilişkilendirmek istediğimiz eylemleri tanımlamak için kullanırız.

Günlük hayatta yaşam boyu yaptığımız planlamayı bilgisayar ortamında da gerçekleştirmemiz gerekiyor. Bu yapılan planlama işlemine bilgisayar ortamında ise algoritma denilmektedir. Projelerimize başlamadan önce algoritmalarımızı hazırlamak yapacağımız projelerin daha sağlıklı bir şekilde yürütülmesine imkan tanıyacaktır.

Bir sınıfta ne kadar özellik ve metot varsa, o özellik ve metotları kullanmak için o kadar kod yazmanız gerekmektedir. Bu yüzden sadeliği ön planda tutmanız ve sınıfınızın planlamasını hazırlıklı bir şekilde gerçekleştirmeniz gerekiyor. Bir sınıfa ait kodlar ne kadar genel olursa o kodların başka bir projede kullanımı da o kadar kolay olmaktadır. Bununla birlikte özellik ve metotlar bir nesnenin ihtiyaçlarını karşılayabilecek düzeyde ayarlanmalı ve projelerin ana amacı gözden kaçırılmamalıdır. Yani bir araba sınıfı için araç kullanıcılarının arabada arayacağı özellikler ile çevreye olan zararlarını gözlemlemek isteyen bir kurumun arayacağı özellikler arasında farklılıklar olabilir. Bu nedenle bir sınıf tasarladığınızda, yazdığınız kodu en aza indirmek ile sınıf için tasarım hedeflerini yerine getirmek arasında bir denge kurmanız gerekir.

Nesne

Bir sınıfa ait özellikler nesneler sayesinde kullanılmaktadır. Nesneler sınıflardan türetilirler. Tek başlarına bir anlam ifade etmeyen bu yapılar bir sınıfa ihtiyaç duymaktadırlar. Daha öncede belirttiğimiz gibi nesne ve sınıf kavramları birbiriyle çok ilişkilidir. Bir nesne aynı zamanda bir sınıfın örneği olarak da adlandırılır. Yani örnekleme, bir nesneyi tanımlamak için bir sınıfı kullanma eylemidir. Hazırladığımız programlarda gerçekten kullanabileceğimiz yapılar nesnelerdir.

Öncelikle nesnenin türetileceği sınıf adı yazılır ve bir boşluk bırakıldıktan sonra türetilecek olan nesnenin adı yazılmaktadır. Ardından eşittir sembolü kullanılarak New anahtar kelimesiyle birlikte sınıfın adı tekrardan yazılır. Böylelikle belirtilen sınıfa ait bir nesne türetilmiş olur. Burada kullanılan New anahtar kelimesi ile sınıftan nesne türetilmiştir. Burada kullanılan metoda Yapıcı Metot ve Constructor denilmektedir. Bu metotların detaylarına ilerleyen ünitelerde değinileceği için şimdilik bu

kadar bilgi sahibi olmanız yeterli görülmektedir.
Nesne türetme işlemi gerçekleştirilirken nesne adını belirlediğimiz alanda belirli bir yazım tarzı benimsememiz hazırladığınız kodların okunabilirliğini olumlu yönde etkileyecektir. Bununla beraber isimlendirme kurallarını da dikkate almanız gerekmektedir.
Bir nesnenin durumunu değiştirmekle ilişkili yazım kurallarına dikkat etmek önemlidir. Bir programlama dilinin yazım kuralları, dilin kullanımını yöneten kuralları ifade etmektedir. C# programlama dilinde bir nesnenin özelliğini kullanmak için genel sözdizimi yandaki gibidir: Nesne_adı.Özellik
Bu sözdizimi kullanıldıktan sonra Nesne_adı nesnesi için bellekte bir yer ayrılır. Artık Nesne_adı isimli nesne kullanılarak türetildiği sınıfta tanımlanan özellik ve metotlara erişilebilir. Türetildiği sınıfta tanımlanan özellik ve metodları görüntülemek için Nesne_adı yazdıktan sonra nokta (.) konulmalıdır.
Kod yazım kurallarını dikkate aldığımızda eşittir (=) operatörünün aktarma işleminde kullanıldığı da dikkatimizi çekmektedir. Bu operatör yardımıyla ifadenin sağ tarafında bulunan değer sol tarafta bulunan özellik içine aktarılmaktadır. Böylece bu özellikler kullanıcıya istenilen bir yöntemle iletilmektedir.
Kalıtım
Kalıtımdan bahsedebilmek için bir sınıf kendinden üstte bulunan sınıfın özelliklerini taşıması gerekmektedir. Böylelikle bir sınıfa ait özelliklerin bir veya daha fazlasını başka sınıf veya sınıflar içinde kullanabiliriz. Projelerimizde kalıtımı kullanabilmek için iki nokta üst üste (:) kullanmamız gerekmektedir. Bu işlem ":" operatöründen önce sınıfımızın ismini, sonra da kalıtım alınacak sınıfın ismini yazarak gerçekleştirilmektedir.
Çok Biçimlilik
Kalıtımla bir sınıftan alınan özellik olduğu gibi kullanılıyorken, çok biçimlilik işleminde bu özellik farklı şekillerde de kullanılabilmektedir. Bu işleme aynı zamanda Polimorfizm de denilmektedir. Çok biçimlilik durumunu örneklemeden önce birkaç teknik terimden kısaca bahsedilecektir.
Virtual: Bir sınıftan alınan kalıtımının değiştirilebilmesi için kullanılan komut.
Override: Bu komut ise kalıtım ile gelen metodun bulunduğu sınıftaki değer yerine başka bir değer kullanılarak işletilmesini sağlamaktadır.
Komutlar ve tanımlarından da anlaşılacağı üzere çok biçimlilik, kalıtımla gelen metotların farklı biçimlerde kullanılmasına imkan tanıyan bir özelliktir.
Kapsülleme
Kapsülleme nesne yönelimli programlamanın önemli özelliklerinden birisidir. Bazı durumlarda nesnelerin özelliklerini gizlemek zorunda olabiliriz. Yani daha öncede bahsedildiği gibi nesne adı yazıldıktan sonra nokta operatörü konularak açılan özellikler listesini kısıtlamak isteyebilirsiniz. Biraz daha açacak olursak, verileri programın diğer bölümleri tarafından yanlışlıkla yapılan değişikliklerden korumak için bir nesnenin içine yerleştirirsiniz. Bu şekilde yapılan bir nesne içindeki verileri gizleme işlemine kapsülleme denilmektedir.

DERS ADI Nesne Tabanlı Programlama I
ÜNİTE ADI Sınıf ve Nesne İlişkisi
ÜNİTE NO 5
YAZAR Öğr. Gör. İSHAK METEHAN SİS

GİRİŞ

Nesne tabanlı programlama, önceki ünitelerde de belirtildiği üzere yazılım tasarımını işlevler ve mantık yerine, sınıf ve nesneler etrafında düzenleyen bir programlama yaklaşımıdır. Nesne tabanlı programlama, programcıların işlemleri için gereken mantıktan ziyade işlem yapacağı nesnelere odaklanır. Nesne tabanlı programlamada büyük, karmaşık ve aktif olarak güncellenen veya bakımı yapılan programlar için çok uygundur. Bu yöntem işbirlikçi geliştirmeye de elverişlidir ve projeler böylece gruplandırılabilir.

Nesne tabanlı programlama yaklaşımı, gerçek hayatta var olan problemi çözmek üzere tasarlanacak modelin, sınıflarlar yardımıyla oluşturulan nesneler ve bu nesneler arasındaki ilişkilerden faydalanılarak kurulmasını ilke edinmiştir.

Sınıf ve nesne bu programlama dili modelinin en önemli iki yapı taşıdır. Kullanılacak nesneler sınıflar yardımıyla oluşturulmaktadır. Sınıflar, nesneleri oluşturmak için önceden oluşturulmuş bir kalıp veya ilk örnek (prototip) olarak da düşünülebilir. Bu yaklaşıma göre bir nesne, özellikleri birbiri ile benzemeyen nitelik ve davranışlardan meydana gelen bir veri alanı olarak tanımlanabilir.

Nesne, içinde veri ve bu veriler üzerinde işlem yapacak olan metotları (fonksiyon) bulunduran yazılım bileşenidir. Nesne herhangi bir uygulamada tekrar tekrar kullanılabilir. Nesneler, sınıfların içinde tanımlanır. Sınıfta, nesneyi oluşturan üye değişkenler ve metotlar açıkça tanımlanır.

SINIF VE NESNE KAVRAMLARI

Sınıf Kavramı

Sınıf (Class), nesne yönelimli programlamanın en önemli öğesidir. Sınıflar sayesinde program birkaç bölüme ayrılır ve karmaşıklıkları azalır (İTÜ/BİDB, 2021). Oluşturulan yöntemler ve özellikler bir sınıftadır ve bir sınıf birden çok kez kullanılabilir. Bir sınıf, içinde oluşturulacak nesnenin bir dizi üyesini içermelidir. Bu üyelere örnek olarak özellikler, metotlar, yapıcılar, olaylar sıralanabilir. Sınıf aynı zamanda nesne için bir şablon görevi görür. Bir başka ifadeyle sınıf, nesnenin durumuyla ilgili işlemleri ve nitelikleri tanımlar. Ortak niteliklere sahip nesnelere ait veriler ve işlemler bir sınıfta toplanır. Program içindeki nesneleri tanımlamak için bu tür bir sınıf yapısı sıklıkla kullanılır.

Sınıf kavramının en büyük faydası, bir kez kodlama yapılarak her zaman kullanılabilecek nesneler oluşturma olanağı sağlamasıdır. Bir sınıftan nesne sayıda nesne, örnek olarak oluşturulabilir. Oluşturulan tüm nesneler, daha önce sınıfta belirlenen özelliklere ve davranışlara sahip olacaktır.

Sınıfın hangi özelliklere sahip olacağı, yani hangi bilgileri saklayacakları, sınıfta tanımlanan alanlar (üye değişkenler) yardımı ile belirlenebilir. Sınıfların hangi davranışlara sahip olacağı veya hangi işlevleri yerine getirecekleri ise sınıf kapsamında tanımlanan metotlar (yöntemler) ile belirlenir (Uyanık, 2017).

Nesne Kavramı

Nesne tabanlı programlamada nesneler sınıflardan türetilir (Hüsem, 2018). Bu nedenle nesneler, sınıfların aksine canlıdır ve kimlikleri vardır. Aynı sınıftan üretilen iki nesnenin özellikleri benzerdir ancak bu özelliklerin değerleri farklı olabilir. Bir ifadeyi nesne yapmak için bellekte bir konum belirtmelidir. Sınıflar kullanılarak uygulamadaki nesneleri tanımlanır ve bu nesnelere mesaj gönderilir. Gönderilen mesaja göre yeni bir nesneye ihtiyaç duyulduğunda bu nesne oluşturulur. Yanıt mesajı sınıfın işidir.

Bir nesne, kendisi ve gerçekleştirebileceği işlemler hakkında bilgi depolar. Programın gerektirdiği tüm veriler nesne tarafından kaydedilir. Gerçekleştirilen işlemin türüne bağlı olarak, nesne farklı bilgiler içerir.

SINIF VE NESNE İLİŞKİSİ

Sınıf, soyut bir veri tipidir. Nesne ise sınıftan türetilen somutlaşan bir varlıktır. Sınıflar,

nesnelerde olması gereken özelliklerden durum ve davranışları içerir. Bu sınıftan oluşturulan örneğe (instance) de nesne denildiği önceki nesne konusunda bahsedilmişti. Dolayısıyla bir sınıftan birçok nesne üretilebilmektedir.

Sınıf ve nesne arasındaki ilişkide en önemli noktalardan birisi sadece aynı sınıftan oluşturulan nesnelerin tipleri aynı olabilmektedir. Örneğin bir bilgisayar sınıfından üretilen nesnenin özellikleri ile bir ev sınıfından üretilen nesnenin özellikleri birbirinden farklıdır.

Bu kısımda dikkat edilmesi gereken önemli bir nokta ise, aynı sınıftan üretilen nesneler aynı ortak özelliklere sahip olabilirler ancak bu özelliklerin değerleri farklıdır. Örnek olarak bir bilgisayar sınıfından üretilecek renk, marka, işlemci modeli veya ram miktarı özellikleri aynı olsa da bu özelliklerin değerleri birbirinden farklıdır.

Sınıf ve Nesne İlişkisine Örnekler

Buraya kadar öğrendiklerimizi pekiştirmek için uygulamalar geliştirerek ilerleyelim. Örnek uygulamaları geliştirmek için Visual Studio 2019 ortamında Windows Forms proje türü seçilerek C# programlama dili kullanılmıştır.

DERS ADI Nesne Tabanlı Programlama I
ÜNİTE ADI Alanlar ve Özellikler
ÜNİTE NO 6
YAZAR Öğr. Gör. GÖKHAN TUTAR

SINIFLARDA ALANLAR

Nesne tabanlı programlamada sınıflar soyut yapılardır. Yani başka bir yapı ile ilişkilendirilmedikleri sürece tek başlarına koddurlar. Tetikleme olmadıkça veri taşımaz ve çalışmazlar. Nesne tabanlı programlamada nesneyi oluşturan iki temel yapı vardır. Bunlardan birisi veri diğeri ise metottur. Veri bir değişenin değeridir. Yani ad, soyadı, yaş veya kilo gibi değerlerin bulunduğu yapıdır. Metot ise belirlenen işlem veya işlemleri gerçekleştiren kod bloklarıdır.

Nesne tabanlı programda alan, sınıf içerisinde tanımlanan değişkenlerdir. Alan kelimesi İngilizcede "fields" olarak geçmektedir. Sınıfta tanımlanan değişkenin tipi (string, integer gibi) fark etmeksizin bütün değişkenler alandır. Metotların içerinde tanımlanan değişkenler alan değildir. Alanlar birden fazla metotta kullanılmak için tanımlanır. Zaten sadece bir metotta kullanılacak değişken metottun içerisinde tanımlanmalıdır.

Alanlara Erişim Şekilleri

Alanlara hem sınıf içerisinden hem de sınıf dışından erişim verebilmek mümkündür. Yani sınıfın içerisinde bir metot alana ulaşabilmektedir. Alanları sınıf içerisinde tanımlanan değişkenler olarak tanımlamıştık. Ancak metot içerisinde tanımlanan değişkenler alan değildir.

Sınıf İçerisinden Alanlara Erişim

Alanlara sınıf içerisinden erişmek için "this" ifadesi kullanılmaktadır. This ifadesi İngilizcede "bu" anlamındadır. Sınıf içerisinden alanlara erişimde herhangi bir engel bulunmaktadır. Sınıfın içerisindeki bütün metotlar bulundukları sınıfın bütün alanlarına (fields) erişebilirler.

Sınıfın içerisinde tanımlanan alanın adı ile metodun içerisinde tanımlanan değişkenin adı aynı olabilir. Bu durumda kod her hangi bir hata vermeyecektir. Ancak sınıfın içerisinde tanımlanan alanı çağırmak için "this" ifadesi kullanılmalıdır. Eğer kullanılmazsa değer metot içerisindeki değişken olarak algılanır.

Alanlara sınıf içerisinden erişilebildiği gibi değerleri de değiştirilebilmektedir. Sınıf içerisinde bulunan bir alanın değerini değiştirmek için tıpkı erişmek için kullanılan "this" ifadesi kullanılmaktadır.

Sınıf Dışından Alanlara Erişim

Alanlara erişim sınıf içerisinden direk yapılmaktadır. Ancak sınıf dışından alana erişim için biraz daha farklıdır. Çünkü önceden de belirtildiği gibi sınıflar başlı başına kod parçalarıdır çağrılmadıkça çalışmazlar. Sınıfta bunulan alana erişmek için öncelikle o sınıf tanımlanmalıdır.

Alanların değerleri dışardan erişilerek değiştirilme imkânı da bulunmaktadır. Bunun için öncelikle sınıf tanımlanmalı ardından sınıftaki alana değer atanmalıdır.

Alanlarda yalnızca okunabilir veriler oluşturmakta mümkündür. Yani sınıf içerisinde tanımlanan alana tanımlanırken değer atanır daha sonra bu değerin değiştirilmesi engellenebilmektedir. Sınıfta bulunan bir alanı sadece okunabilir yapmak için alanın önüne "readonly" yazılması yeterlidir.

Sınıf içerisinde tanımlanan alan isimleri eşşiz olmak zorundadır. Alan isimleri eşşiz olmazsa program hata verip çalışmayacaktır.

SINIFLARDA ÖZELLİKLER

Özellikler, alanların değerlerini okumak, yazmak veya hesaplamak için esnek bir mekanizma sağlayan sınıfların bir üyesidir. Başka bir deyişle, özellikleri kullanarak özel alanlara erişebilir ve değerlerini ayarlayabiliriz. Özellikler her zaman genel veri üyeleridir. Özelliklerin İngilizcede "properties" olarak geçmektedir.

Özellikler, alanlara erişmek ve bunlara değer atamak için erişimciler olarak da bilinen "get" ve "set" yapılarını kullanırlar. Bir özelliğin get ve set bölümlerine veya bloklarına erişimciler denir. Bunlar, bir özelliğin erişilebilirliğini kısıtlamak için kullanışlıdır, set erişimcisi, bir özellikteki özel bir alana bir değer atayabileceğimizi belirtir ve set erişimcisi özelliği olmadan, salt okunur bir alan gibidir.

Nesne tabanlı programlamada özellikler verilere yetkili ve kontrollü erişim için büyük önem taşımaktadır. Özelliğe değer atanırken ayrı bir satırda yapılmak zorunda değildir. Sınıf tanımlanırken sınıfın özelliklerine değer atanabilir. Sınıf tanımlanırken özelliğe değer atamak için süslü parantezler "{}"

içerisinde yapılmalıdır. Süslü parantezler arasında önce özelliğin adı yazılı daha sonra da alanın değeri yazılır. Tabi alan string tipinde ise değer tırnak işaretleri arasında yazılmalıdır.

Salt Okunur Özellik Oluşturma

Salt okunur özellikler de oluşturulabilir. Salt okunur, bir özelliğin değerine erişilebilen ancak ona bir değer atanamayan anlamına gelir. Bir özelliğin "set" erişimcisi yoksa salt okunur bir özelliktir.

Sadece Yazılabilir Özellik Oluşturma

Sadece yazılabilir bir özellikler de oluşturulabilir. Sadece yazılabilir, bir özelliğe değer atanabilir ancak onun değerini alınamaz anlamına gelir. Bir özelliğin "get" erişimcisi yoksa sadece yazılabilir bir özelliktir.

Doğrulamalı Özellik Oluşturma

Özelliklerin diğer bir katkısı değer atanmadan önce bazı kontroller oluşturularak değerler doğrulanabilmektedir. Yani özelliğe değer atanmadan önce kontrol edilecek eğer değer kontrolden geçerse atanacak kontrolden geçemezse hata veya uyarı verdirilebilir.

Özelliklere sadece değer atanmadan önce değil aynı zaman değer çağrılmadan öncede kontroller eklenebilmektedir. Yani değer çağrılmadan önce gerekli kontroller yapılarak farklı bir değer döndürülebilmektedir.

Otomatik Eşlenen Özellik Oluşturma

Özellikler (properties) kullanılırken illa alanların (fields) beraberinde kullanılması zorunlu değildir. Yani özellikler tek başlarına kullanılarak değerler atanabilmektedir. Bunu yapılabilmesi için "get" ve "set" erişimcilerinde herhangi bir işlem yapmaya ihtiyaç yoktur. Bu şekilde özellik otomatik olarak eşlenerek değer atanır veya değere erişilebilir. Özelliklerin bu şekilde kullanımı diğer kullanımlara göre daha kısa ve hızlı yapılabilmektedir.

DERS ADI Nesne Tabanlı Programlama I
ÜNİTE ADI Erişim Belirleyiciler
ÜNİTE NO 7
YAZAR Öğr. Gör. DAHA ORHAN

ERİŞİM BELİRLEYİCİLER

Bazı durumlarda nesnelerin özellikleri ve sınıflar içinde yer alan metotların kullanımı özelleştirilmek istenebilir. Yani her özelliğe erişim durumu aynı olmayabilir. Bazı özelliklerin ve metotların alt sınıflarda veya diğer projelerde kullanılıp kullanılmayacağı ayarlanabilmektedir. İşte bu işlemleri gerçekleştirmemizi sağlayan yapı erişim belirleyicileridir. Aynı zamanda erişim belirleyicileri sayesinde sınıflara erişebilme durumu da ayarlanabilmektedir.

C# .Net içinde temel olarak kullanabileceğimiz 4 adet erişim belirleyicisi bulunmaktadır. Bu erişim belirleyicilerin birlikte kullanımı da söz konusudur. Örneğin protected ve internal erişim belirleyicilerin beraber kullanımını ifade protected internal erişim belirleyicisidir. Benzer şekilde private protected erişim belirleyicisi de private ve protected erişim belirleyicilerin birlikte kullanımı olacak şekilde ayarlanmıştır.

Erişim belirteçleri kullanılırken özellikle dikkat edilmesi gerek kriterler bulunmaktadır. Bu kriterler göz ardı edildiği takdirde erişim belirteçlerinin kullanımı karıştırılabilmektedir. Bunlar;

Namespace ve Enum erişim belirleyiciler ile nitelendirilemezler. Çünkü daimi olarak public tanımlıdırlar.

Sınıflar protected ya da private olarak nitelendirilemezler. Public veya internal olarak tanımlanabilmektedirler.

Türetilen sınıflarda türetildikleri sınıfların erişim kısıtlamaları geçerlidir. Kalıtım yoluyla alınan bu kısıtlamalar artırılabilir ancak azaltılamazlar.

Programlama dillerinin sözdizimine bağlı olarak erişim belirleyicilerle yapılan tanımlamalarda büyük küçük harf duyarlılığına dikkat edilmelidir.

Public

Public erişim belirleyicisini kullandığımızda, kısıtlama olmayan yani sınıf içi, sınıf dışı ve farklı projeler olmak üzere her yerden erişilebilir olmasını sağlamaktayız. Tabi public olarak tanımlanan değişkenin ya da metodun farklı projelerde (exe dosyalarında) kullanılabilmesi için başvuru olarak eklenmesi gerekmektedir. Public olarak tanımlanan bir özelliğe kendi sınıfı içinden de ulaşılabilmektedir. Bu durumu basit bir metot tanımlayarak gösterebiliriz.

Public erişim belirleyicisi hiçbir sınırlama getirmemektedir. Tasarım alanında çalışırken F7 fonksiyon tuşu ile kod editörüne geçilebilir.

Private

Erişim belirleyicileri içinde en kısıtlayıcı olan erişim belirleyicisi private erişim belirleyicisidir. Sadece tanımlandığı sınıf içinden erişilmesine izin vermektedir. Bununla birlikte bir özellik veya metot yazılırken herhangi bir erişim belirleyici bildirilmediği takdirde program o özellik veya metodu varsayılan olarak Private algılamaktadır. Ancak genellikle bu belirleyicinin yazılması tavsiye edilmektedir.

Private erişim belirleyicisi kullanılarak oluşturulan bir özelliğe farklı sınıflardan erişilemeyeceğini söyledik. Public erişim belirleyicisinde olduğu gibi farklı bir sınıf içinde yazmak istersek çalıştırma aşamasına geçmeden kod editöründe kırmızı bir alt çizgi ile program bizi uyarmaktadır.

Erişim belirleyicileri içinde en kısıtlayıcı olan erişim belirleyicisi private erişim belirleyicisidir. Private erişim belirleyicisi kullanılarak oluşturulan bir özelliğe farklı sınıflardan erişilememektedir. Private erişim belirleyicisi kullanılarak oluşturulan bir özelliğe farklı projelerden erişilememektedir.

Protected

Protected erişim belirleyicisi kullanılarak tanımlanan özellik ve metotlara aynı sınıf içinden erişilebilmektedir. Ancak farklı sınıflardan bu özellik ve metotlara erişim engellenmektedir. Sadece bu yönüyle Protected erişim belirleyicisini değerlendirecek olursak Private erişim belirleyicisinden bir farkı olmadığı düşünülebilir. Bu durumda kalıtım kavramı devreye girmektedir. Yani Protected erişim belirleyicisinin private erişim belirleyicisinden farkını anlayabilmek için kalıtım kavramını

kullanacağımız bir uygulama geliştirmemiz gerekmektedir.
Protected olarak tanımlanan özellik ve metotlara o sınıftan türetilmiş sınıflardan da erişebilmek mümkündür. Private erişim belirleyicisinde böyle bir durum söz konusu değildir.
Protected erişim belirleyicisi kullanılarak tanımlanan özellik ve metotlara aynı sınıf içinden erişilebilmektedir. Protected erişim belirleyicisi kullanılarak tanımlanan özellik ve metotlara farklı sınıf içinden erişilememektedir.
Internal
Tek bir proje üzerinde çalışırken internal erişim belirleyicisi ile public erişim belirleyicisi arasında bir fark bulunmamaktadır. Internal erişim belirleyicisi kullanılarak tanımlanan özellik ve metotlara aynı sınıftan, türetilen sınıflardan ve farklı sınıflardan erişim mümkündür. Buradaki fark iki projenin tek bir çözüm gezgini üzerinde çalışırken ortaya çıkmaktadır.
Türetilen sınıflarda internal ve public olarak tanımlanan özellikler arasında bir fark yoktur. Internal ve public erişim belirleyicilerinin tek bir proje içerisindeki farklı sınıflarda benzer şekilde çalışmaktadır. Ayrıca farklı projelerin beraber çalışabilmesi için başvuru olarak eklenmesi gerekmektedir. Erişim belirleyicisi yazılmayan veya erişim belirleyicisi internal olarak ayarlanan sınıfların farklı projelerden erişilmesine izin verilmemektedir.
Protected Internal
Protected internal erişim belirleyicisi, protected erişim belirleyicisiyle internal erişim belirleyicisinin OR (veya) operatörüyle birleştirilmesi gibi çalışmaktadır. Protected internal erişim belirleyicisiyle tanımlanan özellik veya metoda tanımlandığı sınıftan, türetildiği sınıflardan veya farklı sınıflardan tek proje içerisinde erişim mümkündür. Bu erişim belirleyicisi kullanılarak tanımlanan özelliğe farklı bir projede bulunan sınıf içinden erişilmesine izin verilmemektedir. Ancak farklı bir projede bulunan türetilmiş bir sınıf içinde erişim mümkündür.
Protected internal olarak tanımlanan bir özellik veya metoda farklı sınıfta türetilen bir nesne üzerinden erişmek mümkündür.

DERS ADI Nesne Tabanlı Programlama I
ÜNİTE ADI Metotlara Giriş
ÜNİTE NO 8
YAZAR Dr. Öğr. Üyesi MUHAMMED FATİH ALAEDDİNOĞLU

Her geçen gün gelişen program ve yazılımlarda kullanılan kod miktarı ve karmaşıklık önemli ölçüde artmaktadır. Bu durumların üstesinden gelebilmek için en etkin programlama yapılarından biri nesne tabanlı programlamalardır. Nesne tabanlı programlamada işlemlerin nesneler şeklinde ifade edilebilmesi büyük önem arz etmektedir. Ayrıca algoritmaların işleyiş adımlarının anlaşılabilir olması da ayrıca önem arz etmektedir. Bu durumlar için kullanılan en önemli yapı ise metotlardır. Metotları iyi bir şekilde kullanabilmek için hem yapılan işlemleri tanımlayan hem de akılda kalabilecek bir isimlendirme önemlidir. Bu şekilde yapılan program veya gerçekleştirilen yazılımda tanımlanan bir metot bir kez tanımlandıktan sonra çok kere aktif kullanılabilir. Ayrıca programın bütünündeki işlemlerde tutarlılık sağlanabilir. Metotların bir diğer faydası ise karşılaşılan problemleri çözmede kullanılan algoritmaların işleyişi kolaylaşır.
Metotlar yapı bakımından bir makine, alet veya fabrikaya benzemektedir. Yani metotlar bir makinenin işleyişine benzer şekilde ya bir işlemi geçekleştirir veya girdi olarak verilen verileri belirli işlemlerden geçirdikten sonra işe yarar durumlara çevirir. Metotlar amaçlarına ve işlevlerine göre çok çeşitli şekillerde kullanılabilmektedir. Metotları daha rahat kullanabilmek için yapılacak işlemleri tek bir amaç gözeterek gerçekleştirmek uygun olacaktır. Ancak birden fazla amaca hizmet eden metotlarda yazmak mümkündür.
Metotlar kullanım amaçlarına göre geri dönüş değerlerine sahip olabilmektedir. Yani metodun yapması gereken işlemi gerçekleştirmesinden sonra return ifadesi ile metodun sonucu olarak bir değer döndürmesi işlemidir. Bu dönüş değerleri string, int, bool veya daha değişik tiplerde olabilir. Metot eğer bir değer döndürmüyorsa void ifadesi kullanılır. Bu void ifadesi metodun işlem yaptığı ve sonuç olarak herhangi bir değere sahip olmadığı anlamına gelir.
Nesne tabanlı programlamaları gerçekleştirmek için kullanılacak en uygun platformlardan biri Visual Studio ortamıdır. Bu ortam sayesinde projelerin gerçekleştirilmesi için gerekli bir çok ayar hazır şekilde gelmekte ve yazılımı gerçekleştiren kişi veya kişiler kodlarına odaklanmaktadırlar. Bu platform içinde ise en yaygın ve esnek yapıya sahip dil ise C# programlama dilidir. C# dilinde birçok işlem için hazır sınıf ve metotlar bulunmaktadır. Bu yapılar sayesinde yazılımcılar kodlarını daha hızlı ve güvenli şekillerde gerçekleştirebileceklerdir.
Günümüzde yazılım alanı kendi başına özel bir sektör olmuştur. Bu sektörde en yaygın kullanılan programlama türlerinden biri de nesne tabanlı programlamadır. Eğer yazılımcı nesne tabanlı programlama ile yazılım geliştiriyorsa sınıf ve metot yapısına hâkim olmalıdır. Özellikle kişi metotların kullanımını iyi bir şekilde anlarsa hem problemleri tanımlamada hem algoritmaları uygulamada hem de düzenli ve güvenli kod yazmada başarılı olacaktır. Ayrıca metot yapısı sayesinde bir kez yaptığı işlemi tekrar düşünmek zorunda kalmaz ve daha kısa sürelerde yazılımını tamamlar.

METOTLARDA ÖZEL İFADE VE YAPILAR

"This" Anahtar Sözcüğü

This anatar sözcüğü global (genel) olarak tanımlanmış değişkenlerin referans ile metot içerisinde kullanılmasını sağlar. Yani sınıf içerisinde tanımlanan değişkenin sınıfa ait olduğunu ifade eden bir referans yapısı olarak tanımlanır. Böylece metot içerisinde benzer isimde tanımlanan değişkenler ayrılır.

Main Metodu

Bu metot projenin yani C# uygulamasının başlangıç noktasıdır. Yani geliştirilen program derlenmeye başladığı anda Main metodunu bularak bu kısımdan başlar. Program içerisinde yalnızca bir tane Main metodu vardır. Main metoduna ait aşırı yüklemeler (overloading) yapılabilir.

Program Sınıfı

Bu sınıf Visual Studio platformunda C# programı oluşturulurken otomatik oluşturulan bir sınıftır. Projenin ya da programın ilk olarak başladığı sınıf olarak genellikle içerisinde Main ana metoduyla birlikte otomatik oluşturulur.

Void ifadesi

Bu ifade oluşturulan metodun geri değer döndürmeyeceği anlamını taşır. Yani oluşturulan metot bir takım işlemler yaptıktan sonra çıkan sonuçlardan herhangi birinin metodun bir çıktısı olarak ifade edilmemesi olayıdır.

Return ifadesi

Bu ifade nesne tabanlı programlama da sıkça kullanılan ve yapılan işlemin ve oluşan değerin geri gönderilmesi anlamında kullanılabilir.

Aşırı yüklemeler(overloading)

Çoğu nesne tabanlı programlama dilinde oldukça fazla avantaj sunan ve C# dilinde ise çok önemli bir özellik olan bu yapı aynı isimde ancak farklı giriş parametreler ile tanımlanmış metotlardır.

METOTLARA ERİŞİM

Metotların kullanım amaçlarına göre erişim seviyelerinin belirlenebilmesi, bu yapının hem daha güvenli hem kullanım amacına daha hizmet etmesi açısından önemlidir. Metotlarda sıkça kullanılan "public, protected ve private" adlarında üç adet erişim belirleyicileri vardır. Bu erişim belirleyicileri(seviyeleri) metodun kullanım amacına göre değişkenlik göstermektedir. "Public" erişime sahip olan metotlara hem kendi sınıfları hem de başka sınıflardan erişmek mümkündür. "Public" ifadesi kullanılan metotlara diğer metot veya sınıflardan erişirken ya sınıftan bir nesne oluşturulur ya da "static" ifadesi eklenerek hem doğrudan (aynı sınıfta ise) hem de sınıf üzerinden erişmek mümkün olmaktadır. "Protected" erişim belirleyicilere sahip bir metoda erişmek için aynı sınıfta olduğu zaman "public" erişim ile aynı iken farklı sınıftan erişmek isteyen sınıfın, kullanılmak istenen "protected" metoda sahip sınıftan türetilmesi gerekmektedir. Son olarak ise "private" erişim belirleyiciye sahip metotların kullanımından bahsetmek gerekmektedir. Bu kullanım ise metodun yalnızca kendi sınıfı tarafından kullanımına izin vermekte başka sınıfların erişimini ise kısıtlamaktadır.

Metotlarda "Static" İfadesi

Metotların kullanımlarında dikkat edilmesi gereken önemli bir durum ise "static" ifadesinin kullanımıdır. Bu ifade metodun sınıf üzerinde direk erişilebilir hale gelmesi yani herhangi bir nesne tanımlamadan kullanımına izin vermektedir. Bu ifade sayesinde oluşturulan metot başka bir yerde(başka bir metot veya sınıf) kendi sınıfından yeni bir nesne tanımlanmaksızın kullanımına izin vermektedir. "Static" ifadesinin kullanım anacı ise sistemin hafızasını ve performansını en iyi şekilde kullanarak daha efektif yazılımlar gerçekleştirmektir.

DERS ADI Nesne Tabanlı Programlama I
ÜNİTE ADI Kurucu ve Statik Metotlar
ÜNİTE NO 9
YAZAR Dr. Öğr. Üyesi MUHAMMED FATİH ALAEDDİNOĞLU

METOTLARIN ÖNEMİ VE TANIMI

Metotlar, yazılımların okunurluğu,, algoritmik yapılar, anlaşılırlık ve güncellemeler ile ilgili işlemlerde kullanılmaktadır. Özellikle isminin doğru ve amacını belirtir şekillerde ifade edilmektedir. Metotlar, sistemlerin, projelerin ve otomasyonların etkin ve doğru şekillerde gerçekleşmesi için çok önemlidir. Her geçen gün yaygınlığı ve etkinliği artan yazılımlar problemin karmaşıklığına bağlı olarak çok kolay veya zor algoritmalar içerdiğini söylemek mümkündür. Ancak algoritmalar zorlaştıkça yapıya hâkimiyet ve müdahale de oldukça zorlaşmakta hatta imkânsız hale gelmektedir. Tam bu esnada ortaya koyulan algoritmalar için kodları basitleştirmek, kodları azaltmak, güvenli ve düzenli kod yazımını sağlamak, isimlendirmeler ile anlaşılır hale getirmek, erişim belirleyicileri ile yetkilendirme işlemlerini rahat bir şekilde yapmak çok büyük bir önem arz etmektedir. Bu durumlar için kullanılan en önemli yapı ise metotlardır. Metotları iyi bir şekilde kullanabilmek için hem yapılan işlemleri tanımlayan hem de akılda kalabilecek bir isimlendirme önemlidir. Bu şekilde yapılan program veya gerçekleştirilen yazılımda tanımlanan bir metot bir kez tanımlandıktan sonra çok kere aktif kullanılabilir. Ayrıca programın bütünündeki işlemlerde tutarlılık sağlanabilir. Metotların bir diğer faydası ise karşılaşılan problemleri çözmede kullanılan algoritmaların işleyişi kolaylaşır.

Metotlar yapı bakımından bir makine, alet veya fabrikaya benzemektedir. Yani metotlar bir makinenin işleyişine benzer şekilde ya bir işlemi geçekleştirir veya girdi olarak verilen verileri belirli işlemlerden geçirdikten sonra işe yarar durumlara çevirir. Metotlar amaçlarına ve işlevlerine göre çok çeşitli şekillerde kullanılabilmektedir. Metotları daha rahat kullanabilmek için yapılacak işlemleri tek bir amaç gözeterek gerçekleştirmek uygun olacaktır. Ancak birden fazla amaca hizmet eden metotlarda yazmak mümkündür.

Metotlar kullanım amaçlarına göre geri dönüş değerlerine sahip olabilmektedir. Yani metodun yapması gereken işlemi gerçekleştirmesinden sonra return ifadesi ile metodun sonucu olarak bir değer döndürmesi işlemidir. Bu dönüş değerleri string, int, bool veya daha değişik tiplerde olabilir. Metot eğer bir değer döndürmüyorsa void ifadesi kullanılır. Bu void ifadesi metodun işlem yaptığı ve sonuç olarak herhangi bir değere sahip olmadığı anlamına gelir.

Nesne tabanlı programlamaları gerçekleştirmek için kullanılacak en uygun platformlardan biri Visual Studio ortamıdır. Bu ortam sayesinde projelerin gerçekleştirilmesi için gerekli bir çok ayar hazır şekilde gelmekte ve yazılımı gerçekleştiren kişi veya kişiler kodlarına odaklanmaktadırlar. Bu platform içinde ise en yaygın ve esnek yapıya sahip dil ise C# programlama dilidir. C# dilinde birçok işlem için hazır sınıf ve metotlar bulunmaktadır. Bu yapılar sayesinde yazılımcılar kodlarını daha hızlı ve güvenli şekillerde gerçekleştirebileceklerdir.

Günümüzde yazılım alanı kendi başına özel bir sektör olmuştur. Bu sektörde en yaygın kullanılan programlama türlerinden biri de nesne tabanlı programlamadır. Eğer yazılımcı nesne tabanlı programlama ile yazılım geliştiriyorsa sınıf ve metot yapısına hâkim olmalıdır. Özellikle kişi metotların kullanımını iyi bir şekilde anlarsa hem problemleri tanımlamada hem algoritmaları uygulamada hem de düzenli ve güvenli kod yazmada başarılı olacaktır. Ayrıca metot yapısı sayesinde bir kez yaptığı işlemi tekrar düşünmek zorunda kalmaz ve daha kısa sürelerde yazılımını tamamlar.

METOTLARDA ÖZEL İFADE VE YAPILAR

this Anahtar Sözcüğü

This anatar sözcüğü global (genel) olarak tanımlanmış değişkenlerin referans ile metot içerisinde kullanılmasını sağlar. Yani sınıf içerisinde tanımlanan değişkenin sınıfa ait olduğunu ifade eden bir referans yapısı olarak tanımlanır. Böylece metot içerisinde benzer isimde tanımlanan değişkenler ayrılır.

Main Metodu

Bu metot projenin yani C# uygulamasının başlangıç noktasıdır. Yani geliştirilen program derlenmeye başladığı anda Main metodunu bularak bu kısımdan başlar. Program içerisinde yalnızca bir tane Main metodu vardır. Main metoduna ait aşırı yüklemeler (overloading) yapılabilir.

Program Sınıfı

Bu sınıf Visual Studio platformunda C# programı oluşturulurken otomatik oluşturulan bir sınıftır. Projenin ya da programın ilk olarak başladığı sınıf olarak genellikle içerisinde Main ana metoduyla birlikte otomatik oluşturulur.

Void ifadesi

Bu ifade oluşturulan metodun geri değer döndürmeyeceği anlamını taşır. Yani oluşturulan metot bir takım işlemler yaptıktan sonra çıkan sonuçlardan herhangi birinin metodun bir çıktısı olarak ifade edilmemesi olayıdır.

Return ifadesi

Bu ifade nesne tabanlı programlama da sıkça kullanılan ve yapılan işlemin ve oluşan değerin geri gönderilmesi anlamında kullanılabilir.

Aşırı yüklemeler(overloading)

Çoğu nesne tabanlı programlama dilinde oldukça fazla avantaj sunan ve C# dilinde ise çok önemli bir özellik olan bu yapı aynı isimde ancak farklı giriş parametreler ile tanımlanmış metotlardır.

METOTLARDA ERİŞİM BELİRLEYİCİLERİ

Metotların kullanım amaçlarına göre erişim seviyelerinin belirlenebilmesi, bu yapının hem daha güvenli hem kullanım amacına daha hizmet etmesi açısından önemlidir. Metotlarda sıkça kullanılan Public, protected ve private adlarında üç adet erişim belirleyicileri vardır. Bu erişim belirleyicileri(seviyeleri) metodun kullanım amacına göre değişkenlik göstermektedir. Public erişime sahip olan metotlara hem kendi sınıfları hem de başka sınıflardan erişmek mümkündür. Public ifadesi kullanılan metotlara diğer metot veya sınıflardan erişirken ya sınıftan bir nesne oluşturulur ya da statik ifadesi eklenerek hem doğrudan (aynı sınıfta ise) hem de sınıf üzerinden erişmek mümkün olmaktadır. Protected erişim belirleyicilere sahip bir metoda erişmek için aynı sınıfta olduğu zaman public erişim ile aynı iken farklı sınıftan erişmek isteyen sınıfın, kullanılmak istenen protected metoda sahip sınıftan türetilmesi gerekmektedir. Son olarak ise private erişim belirleyiciye sahip metotların kullanımından bahsetmek gerekmektedir. Bu kullanım ise metodun yalnızca kendi sınıfı tarafından kullanımına izin vermekte başka sınıfların erişimini ise kısıtlamaktadır.

Metotlarda Statik İfadesi

Metotların kullanımlarında dikkat edilmesi gereken önemli bir durum ise statik ifadesinin kullanımıdır. Bu ifade metodun sınıf üzerinde direk erişilebilir hale gelmesi yani herhangi bir nesne tanımlamadan kullanımına izin vermektedir. Bu ifade sayesinde oluşturulan metot başka bir yerde(başka bir metot veya sınıf) kendi sınıfından yeni bir nesne tanımlanmaksızın kullanımına izin vermektedir. Statik ifadesinin kullanım anacı ise sistemin hafızasını ve performansını en iyi şekilde kullanarak daha efektif yazılımlar gerçekleştirmektir.

DERS ADI Nesne Tabanlı Programlama I
ÜNİTE ADI İleri Düzey Metot İşlemleri
ÜNİTE NO 10
YAZAR Dr. Öğr. Üyesi MUHAMMED FATİH ALAEDDİNOĞLU

METOTLARIN ÖNEMİ VE TANIMI

Nesne tabanlı programlama için metotlar, oldukça sık başvurulan, esneklik sağlayan ve algoritmik açıdan çok büyük önem arz eder. Metotlar, sistemlerin, projelerin ve otomasyonların etkin ve doğru şekillerde gerçekleşmesi için kullanılır. Metotlar, yazılımların okunurluğu, algoritmik yapılar, anlaşılırlık ve güncellemeler ile ilgili işlemlerde kullanılmaktadır. Özellikle metot isminin doğru ve amacını belirtir şekillerde ifade edilmesi önemlidir. Metotlar, sistemlerin, projelerin ve otomasyonların etkin ve doğru şekillerde gerçekleşmesi için kodları basitleştirmek, mümkün olduğunca azaltmak, güvenli ve düzenli kod yazımını standartlaştırmak, kolay hatırlanabilir isimlendirmeler kullanmak, erişim belirleyicileri ile yetkilendirme işlemlerini belirtmekle en etkili şekillerde kullanılabilir. Bu şekilde yapılan program veya gerçekleştirilen yazılımda tanımlanan bir metot bir kez tanımlandıktan sonra çok kere aktif kullanılabilir. Ayrıca programın bütünündeki işlemlerde tutarlılık sağlanabilir. Metotların bir diğer faydası ise karşılaşılan problemleri çözmede kullanılan algoritmaların işleyişi kolaylaşır.

METOT PARAMETRELERİNDE VARSAYILAN DEĞER ATAMA İŞLEMLERİ

Her geçen gün yaygınlığı ve etkinliği artan yazılımlar problemin karmaşıklığına bağlı olarak çok kolay veya zor algoritmalar içerdiğini söylemek mümkündür. Bu sebeple algoritmalar zorlaştıkça yapıya hakimiyet ve müdahale de oldukça zorlaşmakta hatta imkansız hale gelmektedir. Bu sebeple yazılımcılar tarafından kullanılan metotlardaki esneklik, problemleri çözmek için kolaylıklar sağlar. Benzer şekilde metotlarda da giriş parametreleri yani değişkenlerin kullanımlarındaki esneklikler oldukça önemlidir. Her geçen gün önemi artan yazılım geliştirme, hemen hemen her iş için değişkenlik göstermekte ve farklı yapılar ile kolaylaştırılmaya çalışılmaktadır.

PARAMETRELERDE "REF" VE "OUT" ANAHTAR SÖZCÜKLERİNİN KULLANIMI

C# dilini geliştiren kişiler ise yazılımcıların işlerini kolaylaştırmak, gerekli kontroller sağlamak ve hafızayı etkin kullanmak için değişkenlerin, sınıfların ve giriş parametresi olabilecek herhangi bir yapının kullanımını en basit hale getirmek için bazı teknikler geliştirmektedirler. Bu bağlamda C# nesne tabanlı programlamada, nesne mantığına uygun, sınıf ve metotların kullanımını kolaylaştıracak yapılar bulunmaktadır. Bunlardan bir tanesi, değişken veya nesnelere ait default yani varsayılan değerleri arka planda tanımlamaktır. Bir diğeri ise metot giriş parametrelerindeki verilerin gerektiğinde değişmesini sağlayan “ref” ve “out” anahtar sözcükleridir. Bu anahtar sözcükleri değişkenin hafızadaki yerine giderek değişkenin bizzat kendisi ile işlem yapılmasını sağlar. Böylece değişken üzerine yapılan işlem sonrasında değişikliği görmek mümkün olur. Normal şartlarda metotlara dışarıdan giriş parametresi olarak verilen değişken, işlem sonrasında önceki değeri korumaktadır. Çünkü giriş parametresi olarak verilen değişkenin bir kopyası oluşturulmakta ve asıl değişkene dokunmamaktadır. Sonuç olarak asıl değişkenin değerinde herhangi bir değişiklik olmamaktadır.

Metotlar kullanım amaçlarına göre geri dönüş değerlerine sahip olabilmektedir. Ancak bir metodun tekbir geri dönüş değeri olduğu için "ref" ve "out" anahtar sözcükleriyle dönüş değerlerini artırmak mümkün olabilmektedir. Yani metodun yapması gereken işlemi gerçekleştirmesinden sonra return ifadesi ile metodun sonucu olarak bir değer döndürmesi işlemidir.

Çoğu nesne tabanlı programlama dilinde oldukça fazla avantaj sunan ve C# dilinde ise çok önemli bir özellik olan bu yapı aynı isimde ancak farklı giriş parametreler ile tanımlanmış metotlardır. Yani aynı isme sahip birden fazla metot aynı sınıf içinde farklı görevlere hizmet edecek şekillerde ifade edilebilir. Ancak dikkat edilmesi gereken çok önemli bir husus ise aynı isimde tanımlanan bu metotların bütün parametreleri aynı sıra ve aynı veri tiplerinde olamayacağıdır. Buna göre en az bir giriş parametresi diğerlerinden farklı olmalıdır.

AŞIRI YÜKLEMELER(OVERLOADING)

Çoğu nesne tabanlı programlama dilinde oldukça fazla avantaj sunan ve C# dilinde ise çok önemli bir özellik olan bu yapı aynı isimde ancak farklı giriş parametreler ile tanımlanmış metotlardır. Aşırı yüklemeler metotlarda sıkça başvurulan bir yapıdır. Yani aynı işi yapan fakat farklı parametreler ile

farklı durumlar için sonuçlar üreten metotlar geliştirebilmek yazılımcı açısından oldukça esneklik sağlamaktadır. C# programlama dilinde yazılımcıların hizmetine sunulmuş çok fazla aşırı yükleme ile oluşturulmuş hazır sınıf ve metot bulunmaktadır. Bu sınıf ve metotlar sayesinde yazılım geliştirmek çok daha kolay ve esnek olmaktadır.

Aşırı yüklemelerin en etkin kullanımlarından biri de yapıcı metotlardır. Bu metot türünü C# programla dilinde özel bir yeri olmakla birlikte standart aşırı yüklemelerden farklı olarak geri dönüş değeri ve dönüş tipi yoktur. Bu farklılıklardan başka sınıflara ait çoğu temel işlemler için kullanılmaktadır.

DERS ADI Nesne Tabanlı Programlama I
ÜNİTE ADI Diziler
ÜNİTE NO 11
YAZAR Öğr. Gör. ORHAN ÇELİKER

GİRİŞ

Diziler, programlamada sıklıkla kullanılan ve ihtiyaç duyulan veri yapılarındandır. Diziler verileri gruplayarak saklamak için kullanılır. Özellikle, aynı türden verilerin sıralanarak gruplandırılmasında kullanılan dizilerin eleman sayıları sabit olarak belirlenir. Söz konusu eleman sayısının değiştirilmesi istendiğinde ise Array.Resize metodu kullanılır. Sevilen kitaplar veya arabalardan oluşan elemanların tek bir değişkende toplanması dizilere örnek olarak verilebilir.

DİZİ TANIMLAMA

Diziler, her bir değer için ayrı değişken tanımlamak yerine birden çok değeri tek bir değişkende tutmak için kullanılır. Örneğin haftanın günlerinin bir değişkende tutulmak istendiği bir durumda her bir gün için gun1, gun2, gun3 gibi ayrı ayrı değişken tanımlanmalıdır. Ancak 7 elemanlı bir dizi kullanılarak tek bir değişkende haftanın tüm günleri saklanabilir. Diziler, C# programlama dilinde System.Array isim uzayı (namespace) altında yer alır. Birden çok değerin bir değişken üzerinde saklanabilmesi ve bu değerlerin sıralanarak gruplandırılabilmesi dizilerin başlıca kullanım amacıdır.

Dizi tanımlanırken birden fazla yöntem izlenebilir. Tanımlamada dizi değişkeninin veri tipi, dizi adı ve eleman sayısı belirtilir. Dizi tanımlama ve bildirimin aynı satırda yapıldığı söz dizimi için aşağıdaki gibi bir yapı kullanılabilir:

tip[ ] dizi adı = new tip[büyüklük]

DİZİYE DEĞER ATAMA

Bir dizi tanımlandıktan sonra öncelikle dizi değişkenine değer atanmalıdır. Dizi değişkenine değer atamak için dizi elemanının indeks değeri köşeli parantez içine yazılır ve istenilen değer atanır.

DİZİ KOPYALAMA

Bazı durumlarda bir dizinin içeriği başka bir diziye kopyalanabilir veya iki dizinin içerikleri birleştirilebilir. Bu tür durumlar için döngülerin yer aldığı algoritmalar kullanılabilir. Ancak yazılım geliştirme ortamlarının sunduğu avantaşlar sayesinde dizilerden kesit alma, sıralama, dizi birleştirme gibi karmaşık işlemler programlama dilinde hazır bulunan metot ve fonksiyonlar aracılığıyla kolaylıkla gerçekleştirilebilir.

C# programlama dilinde Copy(), CopyTo() ve Clone() metodları yardımıyla iki dizinin içerikleri birbirine kopyalanabilir veya diziler tek bir dizide birleştirilebilir. CopyTo() metodu kullanılarak bir dizinin içeriğini başka bir diziye kopyalamak için kopyası alınacak kaynak dizi, indeks numarası ve hedef dizi belirtilir.

DİZİYE ELEMAN EKLEME

Dizilere değer atama işlemi dizi tanımlanırken yapılabildiği gibi dizi tanımlandıktan sonra SetValue metodu kullanılarak da yapılabilir. Bazı durumlarda dizi tanımlandıktan sonra diziye yeni eleman ekleme ihtiyacı doğabilir. Bu tür durumlarda C# programlama dilinde Array sınıfının altında yer alan SetValue metodu kullanılır. Bu metot kullanılırken öncelikle dizi değişkeni adı yazılır ve parametre olarak eklenecek elemanın değeriyle indeks numarası bildirilir.

DİZİDEN ELEMAN SİLME

Dizilere yeni eleman eklemek gibi dizilerden mevcut eleman ya da elemanların silinmesi de programlamada sıkça karşılaşılan durumlardandır. Bu işlem için C# programlamada Array sınıfının altında yer alan Clear() metodu kullanılır. Bu metot sözdiziminde 3 parametre alır. Bunlardan ilki elemanları sıfırlanacak dizi değişkenini, ikincisi içerik silme işleminin başlayacağı elemanı ve üçüncüsü de içeriği silinecek eleman sayısını ifade eder. Bu metot kullanılırken aşağıdaki sözdizimine uyulması gerekmektedir:

Array.Clear(Dizi Adı, Başlangıç İndeks Numarası, Silinecek Eleman Sayısı

DİZİLERDE SIRALAMA

Geliştirilen yazılımlarda ihtiyaca göre dizi içeriğinin sıralanması gerekebilir. Dizilerin kullanımında sıralama işlemi oldukça önemlidir. Bu işlem için geliştiriciler, çeşitli fonksiyonlar veya metotlar kullanabilirler. Ancak C# programlama dili sıralama işlemi için Array sınıfının altında Sort() metodunu

sunmuştur. Bu metot tanımlanırken aşağıdaki sözdizimi kullanılır.
Array.Sort(Dizi Adı)

ÇOK BOYUTLU DİZİLER

Diziler tek boyutlu ve çok boyutlu olmak üzere iki gruba ayrılabilir. Tek boyutlu diziler tanımlanırken sadece eleman sayısı belirtilir. Bu ünitede şimdiye kadar ele alınan başlıklar tek boyutlu dizilere örnek verilebilir. Çok boyutlu diziler ise tek boyutlu dizilere benzer biçimde tanımlanır ve kullanılır. Ancak çok boyutlu dizi tanımlanırken virgül (,) operatörü kullanılarak dizinin boyutu belirtilir.

Çok boyutlu dizilerde 2 boyutlu diziler sıklıkla kullanılmaktadır. Çok boyutlu dizinin elemanlarına erişmek için ise iki veya daha fazla indeks kullanılmalıdır. Çok boyutlu diziler matris ve düzensiz diziler olmak üzere iki kategoride incelenebilir.

MATRİS DİZİLER

Dikdörtgensel diziler olarak da adlandırılan matris dizilerin her satırında eşit sayıda eleman bulunur. Aşağıda iki boyutlu bir matris dizinin sözdizimi gösterilmiştir.

Değişken Tipi[,] Dizi Adı = {{x,y},{z,t}}

DÜZENSİZ DİZİLER

Her bir satırı farklı uzunlukta olan çok boyutlu dizilere düzensiz diziler denmektedir. İç içe diziler olarak da adlandırılan düzensiz dizilerin her bir elemanının içerisinde başka bir dizinin elemanları tanımlanabilir. Düzensiz diziler tanımlanırken her bir boyut için bir çift köşeli parantez [ ] kullanılır. Örneğin iki boyutlu bir düzensiz dizi oluşturmak için kullanılan sözdizimi aşağıdaki gibidir.

Değişken Tipi [ ][ ] Dizi Adı = new DeğişkenTipi [Büyüklük][ ]

DERS ADI Nesne Tabanlı Programlama I
ÜNİTE ADI Koleksiyonlar I
ÜNİTE NO 12
YAZAR Öğr. Gör. ORHAN ÇELİKER

GİRİŞ

Koleksiyonlar, programlamada sıklıkla ihtiyaç duyulan yapılardan olup System.Collections isim uzayı altında yer alırlar. Bu yapılar içerisinde aynı türden veriler saklanabildiği gibi farklı türden veriler de saklanabilir. Bunun için Visual Studio.Net'in sunmuş olduğu çeşitli koleksiyon sınıfları bulunmaktadır. C++ dilinde konteyner, Java dilinde Collections Framework olarak adlandırılan bu sınıflar, C# dilinde ise koleksiyon olarak adlandırılmaktır.

Koleksiyonlar, amaç ve yapı bakımından dizilerle benzerlik gösterse de hem verilerin saklanması hem de saklanan verilere erişilmesi açısından dizilerden ayrıldığı belli başlı noktalar bulunmaktadır. Bunların başında ise içerdiği çok çeşitli sınıflar gelmektedir. Bu sınıflar aracılığıyla geliştiriciler, nesne grupları ve veriler üzerinde daha esnek bir şekilde çalışma imkânı bulurlar.

KOLEKSİYON TANIMI

Koleksiyonlar; C++, Java gibi diğer programlama dillerindeki kullanım amacına bakıldığında temel olarak aynı ya da farklı türde birden fazla verinin veya nesnenin gruplanıp saklanmasına ve bunlara erişim sağlanmasına olanak tanıyan bir sınıf olarak tanımlanabilir. Bu tanıma göre dizilerle benzerlik gösterdiği görülebilir ancak koleksiyonların dizilerden ayrıldığı temel farklılıklar bulunmaktadır. Bunlardan en önemlisi dizilerde aynı türden veriler saklanabilir, koleksiyonlarda ise farklı türden verilerin saklanması mümkündür.

System.Collections isim uzayının altında birçok ara yüz bulunmaktadır. Bu ara yüzler kullanılarak koleksiyon sınıfları için genel karakteristik tanımlaması yapılmaktadır. Örneğin ArrayList sınıfı koleksiyon sınıfları arasında sıklıkla kullanılır ve IList ara yüzünü uygulamaktadır. Bu yüzden bazı geliştiriciler tarafından liste tipi koleksiyon olarak adlandırılmaktadır. Aynı şekilde HashTable koleksiyon sınıfı ise IDictionary ara yüzünü kullanır ve bu da birçok geliştirici tarafından sözlük tipi koleksiyon olarak sınıflandırılır. Ara yüzlerle ilgili bilinmesi gereken bir başka nokta da koleksiyon sınıflarında ortak olan bazı özelliklerin uygulanan ara yüzler tarafından belirlenmesidir. Örneğin anahtar-değer çiftleri mantığı ile çalışan HashTable ve SortedList sınıfları IDictionary ara yüzünü uygulamaktadır.

C# programlama dilinin sunduğu hazır koleksiyon sınıflarının dışında geliştiriciler, ihtiyaçları doğrultusunda kendi yazdıkları farklı koleksiyon sınıflarını da oluşturabilirler. Bu koleksiyon sınıfları farklı ara yüzleri barındırabilir ve amaca göre özelleştirilebilir. Bu yüzden C# tarafından hazır olarak sunulan ya da geliştiricinin kendi oluşturduğu her koleksiyon türünün belli bir amaca göre tasarlandığı söylenebilir. Farklı amaçlar için geliştirilen koleksiyonlar farklı yapılara sahip olacağı için kendi içerisinde gruplandırılmıştır. Genel kabule göre Non Generic, Generic ve Specialized (özelleştirilmiş) olmak üzere koleksiyonlar 3 grupta incelenebilir. Bu ünitede non generic koleksiyonlara değinilecektir.

NON GENERIC KOLEKSİYONLAR

System.Collections isim uzayının altında yer alan non generic koleksiyon sınıflarının temel özelliği farklı türdeki verilerin boxing işlemi ile object türüne dönüştürülerek saklanabilmesidir. Bu başlık altında Non Generic koleksiyonlar arasında en sık kullanılan ArrayList, Hashtable ve SortedList sınıfları incelenecektir.

ArrayList Sınıfı

System.Collection isim uzayı altında yer alan ArrayList sınıfı, içerisine aktarılan değerlerin boxing işlemi ile object türüne dönüştürüldüğü ve eleman sayısı belirlenmeden tanımlama yapıldığı koleksiyon türüdür. Dizilerde farklı türde eleman barındırılamaması ve dizi boyutunun dinamik bir şekilde düzenlenememesi sebebiyle geliştiriciler tarafından ArrayList sınıfı sıklıkla tercih edilir.

ArrayList sınıfı içerisinde farklı türlerde değişkenler tutulabilir ve çalışma anında dizinin boyutu düzenlenebilir. Sağlanan bu esnekliklerin dışında C# programlama dilindeki kullanımı klasik dizilerle neredeyse aynıdır. ArrayList sınıfının genel sözdizimine yönelik kod örneği aşağıdaki gibidir:

```
ArrayList mevsimler = new ArrayList();
```

HashTable Sınıfı

System.Collections isim uzayı altında yer alan HashTable sınıfı, barındırdığı elamanlara ait değerlerin indeks numarası yerine benzersiz bir değer kullanılarak saklandığı ve bu şekilde erişim sağlandığı koleksiyon türüdür. Koleksiyon elemanlarına indeks numarası yerine farklı bir değer ile erişilmesi gerekebilir. Aynı şekilde koleksiyon elemanlarının benzersiz bir değere sahip olması ve koleksiyon içerisinde arama yapılırken bu benzersiz değerin kullanılması gerekebilir. Şu ana kadar anlatılan Array (dizi) ve ArrayList sınıflarında elemanlara erişmek için int tipinde bir indeks numarası kullanılmış ve eleman arama, ekleme veya çıkarma gibi özellikler bu indeks numarasına göre gerçekleştirilmiştir. İndeks numarası kullanımı az sayıda elemana sahip koleksiyonlarda kullanışlı olabilir. Ancak özellikle performansın önemli olduğu durumlarda beklentileri karşılamayabilir. İşte bu tür durumlarda HashTable sınıfının kullanımı tercih edilebilir.

Genellikle performansın ön planda olduğu ve koleksiyon elemanlarına erişim için indeks numarası yerine benzersiz bir değer (anahtar) yardımıyla erişmek gerektiğinde System.Collections isim uzayı altında bulunan HashTable sınıfı kullanılmaktadır. HashTable sınıfının ArrayList sınıfından temel farkı ise çalışma algoritmasıdır.

HashTable sınıfında kullanılan anahtar değerin benzersiz olması zorunludur. Yani bir anahtar değer sadece bir kere kullanılabilir. Bir anahtar değerin HashTable sınıfı içerisinde daha önceden kullanılıp kullanılmadığını öğrenmek için ContainsKey() metodu kullanılır. Bu metot parametre olarak aranması istenen anahtar değeri koleksiyon içinde tarar ve sonuç olarak bool tipinde bir değer döndürür. Eğer parametre olarak aranan değer HashTable içerisinde mevcutsa true, bu değer daha önce kullanılmamışsa false değeri geri döndürülür. Bu metoda benzer şekilde HashTable içerisinde arama yapan ve bir elemana ait değerin bulunup bulunmadığını tarayan metot ise ContainsValue() metodudur. Bu metot da ContainsKey() metoduna benzer şekilde aranan değer koleksiyonda yer alıyorsa true, yoksa false değeri döndürür.

SortedList Sınıfı

SortedList sınıfı, elemanlarına erişimin hem HashTable sınıfında olduğu gibi anahtar aracılığıyla hem de indeks numarası kullanılarak yapıldığı bir koleksiyon türüdür. System.Collections isim uzayının altında yer alan SortedList sınıfı, elemanlarının küçüklük/büyüklük değerine göre veya alfabetik olarak sıralanmasına olanak tanır. Bir başka ifadeyle elemanlarını sıralı bir şekilde saklayabilen ve indeks numarası kullanılarak da erişilebilen HashTable olarak tanımlanabilir. SortedList sınıfına eklenen elemanlar anahtar değerleri baz alınarak sıralanırlar. Burada koleksiyona eklenme sırasının veya ArrayList'teki gibi elemanın değerinin büyüklüğünün bir önemi yoktur, sıralama sadece anahtar değerine göre yapılır.

DERS ADI Nesne Tabanlı Programlama I
ÜNİTE ADI Koleksiyonlar II
ÜNİTE NO 13
YAZAR Öğr. Gör. ORHAN ÇELİKER

GİRİŞ

List, Stack, Queue, LinkedList ve Dictionary sınıfları generic koleksiyonlar içinde sıklıkla kullanılan yapılardır. Söz konusu bu koleksiyonlar System.Collections.Generic isim uzayının altında yer alırlar. Bu sınıfların kullanımları benzerlik göstermekle birlikte kullanım amaçları ve çalışma yapıları arasında ciddi farklılıklar bulunmaktadır. Özellikle yüksek performansın öncelikli olduğu durumlarda söz konusu koleksiyonlar arasında yapılan tercih kritik öneme sahiptir. Koleksiyonlarda tutulan öğelere erişim ve bu öğelerin yönetilmesi nesne yönelimli programlamanın öne çıkan konuları arasındadır. Özellikle verinin işlenmesi ve yönetilmesi noktasında List, Stack, Queue, LinkedList ve Dictionary yapıları Visual Studio.NET çatısı altında sıklıkla tercih edilmektedir.

GENERİC KOLEKSİYONLAR

Generic koleksiyonlar, aynı türden verileri saklayacak şekilde tasarlanan ve bu sayede veri tipi güvenliğini sağlayan koleksiyon türleridir. Non generic koleksiyonlardan farklı olarak generic koleksiyonlar, tıpkı dizilerde olduğu gibi aynı veri tipinde elemanların tutulmasına izin veriler ve bu koleksiyonlar System.Collections.Generic isim uzayının altında yer alırlar. Generic koleksiyonlar, non generic koleksiyonlarda olduğu gibi boxing işlemi ile herhangi bir dönüşümü tabi tutulmazlar. Dolayısıyla bu da non generic koleksiyonara göre daha performanslı çalışmasını sağlar.

List Sınıfı

List sınıfı, aynı türden verileri saklamak için kullanılır. Tıpkı dizi tanımlamada olduğu gibi veri tipi belirtilir ancak eleman sayısı ya da büyüklük belirtilmez. List sınıfı non generic bir koleksiyon olan ArrayList sınıfıyla birçok ortak metoda ve özelliği sahiptir. Ayrıca bu sınıf için ArrayList sınıfının generic karşılığı denilebilir. Ancak List tanımlanırken veri tipi belirtildiğinden dolayı veriler boxing kullanılarak herhangi bir dönüşüme tabi tutulmaz. Dolayısıyla ArrayList sınıfına göre daha performanslı çalışır.

List sınıfı System.Collections.Generic isim uzayının altında yer alır ve genel sözdizimi aşağıdaki gibidir.

List<Değişken Türü> Değişken Adı= new List<Değişken Türü>()

Sözdiziminden de görüldüğü gibi List sınıfı tanımlanırken koleksiyonun içereceği elemanların veri tipi < > operatörleri arasına yazılır. Böylece List'e eklenecek elemanların veri tipi belirlenir ve < > operatörleri arasına yazılan tip dışında elemanın eklenmesine izin vermez.

Stack Sınıfı

Koleksiyonlarla ilgili şimdiye kadar anlatılan sınıflarda basit veri ekleme, silme, sıralama ya da saklama gibi işlemler yapılmaktaydı. Koleksiyonlar sadece bu tür basit işlemleri gerçekleştirmek veya performansı artırmak amacıyla kullanılmamaktadır. System.Collections.Generic isim uzayının altında yer alan Stack (Yığın) sınıfı derleyici, sistem veya yapay zeka yazılımlarında sıklıkla kullanılan bir koleksiyon türüdür. Stack sınıfı, son giren ilk çıkar (last in first out -LIFO) yöntemini kullanır ve boyutu dinamik olarak büyüyebilen bir koleksiyondur. Stack sınıfının çalışma mantığını anlamak için öncelikle LIFO yönteminin anlaşılması gerekmektedir. Bu yönteme örnek olarak 20 kitabın veya herhangi bir nesnenin üst üste bir yığın şeklinde dizilmesi düşünülebilir. Bu yığına eklenen son kitap en üstte olacaktır ve son giren ilk çıkar yöntemine göre son kitap erişilebilen ilk eleman olacaktır. Yığına eklenen ilk kitap ise yığının yani Stack koleksiyonunun son elemanı olacaktır.

Stack sınıfı, diğer generic koleksiyonların kullandığı birçok metodu ortak olarak kullanmaktadır. Bu metotlar daha önce anlatıldığı için burada değinilmeyecektir. Ancak Stack sınıfını kullanırken bu sınıfa ait olan Push(), Pop() ve Peek() metotlarının bilinmesi gerekmektedir.

Queue Sınıfı

Queue (kuyruk), System.Collections.Generic isim uzayının altında yer alan bir koleksiyon türü olup genelde ağ trafiği yönetimi, simülasyonlar ve veri tabanı işlemlerinde kullanılmaktadır. Stack sınıfı, son giren ilk çıkar (LIFO) yöntemine göre çalışmaktayken, Queue sınıfı ilk giren ilk çıkar (first in first out -

FIFO) çalışma yöntemini kullanır. Yani Queue koleksiyonuna eklenen ilk elemana ilk sırada erişim sağlanır.

LinkedList Sınıfı

LinkedList (bağlı liste) sınıfı System.Collections.Generic isim uzayı altında yer alır ve elemanlarının birbirlerine bir link şeklinde bağlı olduğu koleksiyon türüdür.

LinkedList sınıfı kullanılırken AddFirst() metodu yardımıyla listenin ilk alanına eleman eklenir. Benzer şekilde listenin son alanına eleman eklemek için AddLast() metodu kullanılır. Bu koleksiyonun elemanlarına erişirken First ve Last özellikleri kullanılır. First özelliği ile ilk elemana erişim sağlanırken Last ile son elemana erişilir.

Dictionary Sınıfı

Dictionary (sözlük) sınıfı içerdiği elemanların bir anahtar yardımıyla saklandığı koleksiyon türü olup System.Collections.Generic isim uzayının altında yer alır. Tıpkı HashTable sınıfında olduğu gibi Dictionary sınıfının elemanlarına benzersiz anahtarlar kullanılarak erişim sağlanır. Bu koleksiyonda bildirilen tüm anahtaralar kesinlikle birbirinden farklı olmalıdır, anahtarlara atanan değerler ise birden fazla kullanılabilir.

Dictionary sınıfını genel söz dizimi aşağıdaki gibidir.

Dictionary <Anahtar Değişken Tipi, Eleman Değişken Tipi> Koleksiyon Adı = new Dictionary <Anahtar Değişken Tipi, Eleman Değişken Tipi>()

DERS ADI Nesne Tabanlı Programlama I
ÜNİTE ADI Değişmezler, Sabitler ve Numaralandırma
ÜNİTE NO 14
YAZAR Dr. MUHAMMET DAMAR

GİRİŞ

Bu bölüm içerisinde değişmezlerden (literals), sabitlerden (constants) ve numaralandırma (enumarations) kavramlarından bahsedilecek ve Visual Studio 2019 ortamında C# programlama diliyle kullanımı açıklanacaktır. Bu bölümü tamamladıktan sonra projelerinizde ihtiyaçlarınıza bağlı değişmezler, sabitler ve numaralandırma mantığı ile programlama yapabileceksiniz.

Değişmezlere (literals), sabitlere (constants) ve numaralandırmalara (enumerations) bu bölüm içerisinde üzerinde duracağımız kavramlar olacaktır. Nesne yönelimli bir programlama dilleri olan, C#, Java, F#, C++ gibi dillerde proje ihtiyaçlarına bağlı sıklıkla kullanılan değişkenlerdir. Değişmezlere, sabitlere ve numaralandırmalar derleme zamanı oluşturulur ve değiştirilememeleri en önemli özellikleri olarak öne çıkmaktadır.
Sabitler, tamsayı sabiti, kayan sabit, karakter sabiti veya dize değişmezi gibi temel veri türlerinden herhangi biri olabilir. Ayrıca numaralandırma sabitleri de vardır. Program yürütme sırasında değişkenlerin değerleri değişse de, değişmezler ve sabitler değişmeyen veri öğelerini içerir. Aşağıda sırasıyla bu kon başlıkları Visual Sudio 2019 ortamında C# programlama dili ile uygulamalar sırasıyla gösterilmektedir.

Değişmezler (Literals), sabit değerler konu başlığı altında değerlendireceğimiz bir diğer konu değişmezler yani literallerdir. Değişmez, değişkenler tarafından kullanılan bir değerdir. Değerler tamsayı, kayan nokta veya dize vb. olabilir.

Değişmezler aşağıdaki türlerden olabilir:
- Tamsayı Değişmezler (Integer Literals): Tamsayı türünün değişmez değeri, tamsayı değişmezi olarak bilinir. Sekizli, ondalık, ikili veya onaltılık sabit olabilir.
- Kayan Noktalı Değişmezler (Floating-point Literals): Bir tamsayı kısmı, bir ondalık nokta, bir kesir kısmı ve bir üs kısmı olan değişmez değer, kayan nokta değişmezi olarak bilinir. Bunlar, ondalık biçimde veya üstel biçimde temsil edilebilir.
- Karakter Değişmezler (Character Literals): Karakter veri türleri için değişmez değerleri tek tırnak gösterim, evrensel kod ile gösterim ve özel gösterim olmak üzere üç şekilde ifade edilebilir.
- Dize Değişmezleri (String Literals): Çift tırnak (“”) içine alınmış veya @“” ile başlayan değişmez değerler, String değişmezleri olarak bilinir.
- Boş - Değersiz Değişmezler (Null Literals): Null türünü belirten değişmez değerdir. Ayrıca, null herhangi bir başvuru türüne sığabilir. Bu nedenle polimorfizm için çok iyi bir örnektir.
- Mantıksal Değişmezler (Boolean Literals): Değişmezleri için yalnızca iki değere izin verilir, yani true ve false.

Sabitler (Constants), derleme zamanında bilinen ve programın ömrü boyunca değişmeyen sabit değerlerdir. Bir sınıfın veya yapının içerebileceği üye türlerinden birisidir. Çoğu zaman kodunuzda aynı sabit değeri tekrar tekrar kullanmanız gerekir. Örneğin, bir dizi geometrik hesaplamanın pi değerini kullanması gerektiğinde, kodunuzdaki değişmeyecek bir değer olan pi değerini tekrarlamak yerine, bir sabit kullanarak bu durumu engelleyebiliriz.

Numaralandırmalar ise sabitler ile çalışmanın kolay bir yolunu sağlar atanan sabit değerleri adlar ile ilişkilendirir. Program yürütme sırasında değişkenlere atanan değerler değişebilirken, değişmezler ve sabitlere depolanan değerler ise sabit kalır. Bu ünitede değişmezler, sabitler ve numaralandırmalar ile ilgili konular verilen örnekler ile anlatılacaktır.

Numaralandırma (Enumerations), belli sözcüklerin, belli tamsayıları temsili durumlarında kullanılan bir yapıdır. Değişkenlerin alabileceği değerlerin sabit olduğu durumlarda kullanılır. Bir numaralandırma aşağıdaki türlerden herhangi birinden türetilebilir: byte, sbyte, short, ushort, int, uint, long, ulong. Enum için varsayılan türümüz int' dir ve enum tanımında tür belirtilerek yukarıda ifade ettiğimiz türlerden bir ile türetilebilir.